TYMSHARE

A COMPUTER AT YOUR FINGERTIPS

REFERENCE MANUAL FOR THE TIME-SHARING EXECUTIVE

By

Verne E. Van Vlear

April 12, 1967

TABLE OF CONTENTS

1.0 INTRODUCTION

The Time-sharing System, (TSS), is a system for making a single computer provide simultaneous, continuously supervisable computing power to a number of authorized users. By "simultaneous" we of course mean apparently simultaneous; it is the computer's capacity for performing tasks at tremendously high speed that gives the illusion of simultaneity. TSS is furthermore designed principally for those users who, in order to obtain fruitful results, need to guide their programs more or less continuously through the course of its execution. One example of such application is in the checking-out of a program -- wherein, at each successive catastrophe, the user is called upon to make an amendment and try again.

TSS provides the following facilities:

1. Mutual protection of the users against one another.

2. Optional partial removal of this protection so that users can communicate with one another via the computer.

3. A more-or-less equal division of computing time between the current users.

4. Software packages necessary to permit one program to control others -- with overall control by the user via peripheral equipment.

5. Software packages which permit communications between computer and peripherals without regard to the latter's special physical peculiarities.

6. A filing system for preserving user's program documentation.

7. Response to a number of requests that arise naturally in the course of a user's connection with the system.

The preliminary medium by which computer and user communicate is at present the teletype console, which allows input, user to computer, from a keyboard and output, computer to user, via a type-head. Paper tape, punched card and magnetic tape are also available but are generally more

cumbersome and unsuitable for primary input. These same devices, along with a high speed printer, are available for output but are not normally used during regular TSS service. Cathode ray displays and other graphical display devices are also being developed and are likely to render manageable the input and output of graphical data.

2.0 ACCESS TO THE COMPUTING FACILITIES

In order to establish a link between a teletype console and the computer, follow one of the procedures described in the "Tymshare Reference Manual" under "LOG IN", depending upon the type of teletype terminal equipment. It should be noted here that there is no direct link between keyboard and typehead. Any such link which appears to exist is established through intermediate software. The TSS software, of course, arranges that the typehead respond to keyboard input in a manner appropriate to the occasion which usually involves echoing back to the typehead, any characters input from the keyboard.

Following the "LOG IN", a printed response "carriage return, line feed, -" indicates that the user has been connected to a built-in software package, called the Executive Program, hereinafter referred to as "Exec" which awaits further teletype input in the form of stylized English language requests or commands. Via these commands the Exec will provide sufficient services to get the user into contact with all the other TSS facilities. It is, itself, primarily a medium for specifying memory requirements handling user's program documentation and establishing communications between different teletypes.

3.0 EXEC COMMAND AND FILE NAME RECOGNITION

It is appropriate here to describe the way in which Exec Commands and Input File Names are recognized. After sufficient characters have been typed in to distinguish the intended command or name from all others, succeeding characters that agree with the name string will continue to be taken from the teletype input buffer. If an alphabetic character is encountered that does not agree with the string an error is assumed and an appropriate diagnostic is given. All characters in the input buffer will be deleted. The first non-alphabetic character (carriage return, space, comma, digit, etc.) that does not agree with the selected string will act as a terminator and will be left in the buffer.

If the command "COPY" and the file names /INPUT/ and /OUTPUT/ are unique, the following string:

```
COPY /INPUT/ TO /OUTPUT/
```

could be abbreviated to:

```
C/I T /OUTPUT/
```

or COPY /INPUT/ TO TELETYPE could be abbreviated to:

```
C/I,T
```

Note the optional use of the "," or the T to act as the separator between file names.

Note that most Exec commands await a terminating Cr which is the user's confirmation that the command is to be executed. The command is not implemented until this confirmation is received and until then may be aborted by pressing usually any other character or, certainly, by pressing the "escape" key.

Exec commands that must be followed by additional input from the user (such as the COPY command that is immediately followed by the input file name) will provide some help to the user if he follows the command with a carriage return. The Exec will respond with a message that indicates the next input from the user. For example, if the command COPY is followed by a carriage return, the Exec will type:

```
FROM FILE:
```

If the user makes an error, the Exec will respond with

an appropriate error message indicating where in the sequence of input the user went wrong, as follows:

```
ERROR, TYPE FROM FILE:
```

It will be left to the user to discover the various help and error messages provided by the Exec.

4.0 FILES

Before describing the Exec's command language it would be appropriate to discuss the nature of files and the manner in which they are referenced.

Files are the primary means by which the user establishes continuity between one computer run and the next (a "run" being that sequence of activities, mutual to the computer and a user, between "LOG IN" and the next EXIT or LOGOUT command -- see sec. 5.1). A file is any named block of information which the user finds is convenient to regard as a single entity; the commonest example of a file is just a program. To provide a check against inappropriate use, files created by the Exec and TSS subsystems are classified, according to the nature of the information in them, into one of five types -- with each of which is associated a type number. This type number is carried along with the information content and is checked whenever the file is referenced by an Exec command (or any other of the TSS facilities which reference files). If the file is found to be of a type inappropriate to the context the command is not executed and an error is indicated.

The file types are:

1. Core Image - The information in this originates from specified segments of core memory.

2. Binary - The information has the form of an assembled but unloaded program.

3. Symbolic - The information is of a form which can be readily listed on some printing device.

4. Dump - Comprises all the information in memory necessary to restart the user from his current situation, i.e., the situation at the time of creation of the dump file.

5. Subsystem - Comprises up to eight 2K blocks which can be read into shared memory. The information originates from core memory and is normally executable as an assembled and loaded program.

Files of types 1, 4 and 5 originate from information in core. Before names have been explicitly assigned to them, type 1, "Core Image" files are referred to by their bounding core addresses; the whereabouts of a type 4, "Dump" file, is implicit in its nature, while type 5, "Subsystem" files are specified by delivering the pseudo-relabeling of the pages containing the information to the command which attaches a name to them.

The information in type 3, "Symbolic" files may come directly from paper tape, teletype or cards and in such a case is referred to by using the name of the corresponding physical medium, viz.,

PAPER TAPE
TELETYPE
PRINTER
CARDS

These names are built into the system and are always appropriately recognized. Another built-in "file" name is

NOTHING

which always contains precisely nothing and whose function is to act as an infinite sink in which limitless unwanted output can be lost.

A more common source of symbolic files is the output from a subsystem, notably the text editor, QED.

Type 2, "Binary" files may originate from paper tape or cards, but, more commonly, arise as the output from the machine language assembly subsystem, ARPAS, and as the data output of a program.

Until the actual process of output from the subsystem occurs, identification of the information is handled by the said subsystem and is usually implicit since the subsystems can usually handle only one file at a time. However, when the information is ejected into a contest involving many other blocks of information of a similar kind some explicit identification must be attached to it.

4.1 File Naming

The names which the user is free to invent (although with some restrictions) and assign to files are of three types:

1. Slashed names
2. Unslashed names
3. Quoted names

Slashed names are reserved for files that are on the disc, quoted names are reserved for files that are on magnetic tape, while unslashed names may be of any type. Tape files may be created and used only by the class of users that are assigned "peripheral" status so that the general use of quoted file names is also restricted to the peripheral class of users. By the use of the command RENAME, to be described later, slashed and quoted file names may be renamed into unslashed names.

When reference is made to an unslashed file name, the Exec will consider the name to be fully delivered as soon as it has received sufficient characters to distinguish the name from all others currently defined by the user. This also applies to slashed and quoted names when the file is used for input. Note that a new name can never be introduced in its unslashed form, and that slashed and quoted names must be typed in their entirety when the name is used for an output file. See Section 3.0.

4.2 Accessing Other User's Files

The naming system described is adequate to reference all the files belonging to the current user, in whose name the Exec was entered. However, to refer to files belonging to another user, it is necessary to augment the file name by that user's account number and name. For example:

```
(B2 JONES) /@FILE1/
```

The access that any other user may have to each of Jones' files is in the hands of Jones, himself. Jones may declare that a member of the public at large has read-only access to the file by placing a control character or the "@" character, as shown above, in the file name. It is also possible to refer to a file belonging to another user in the same account without indicating the account number. For example:

```
(JONES) /@FILE1/
```

5.0 THE EXECUTIVE COMMAND LANGUAGE

This section will describe the functions available to the user through the executive program. These functions are initiated by special commands recognized by the executive program. The commands are divided into the following six logical areas for ease of reference.

1. Entering and Leaving TSS
2. Allocation of Memory
3. Interaction of Teletypes
4. Creating and Manipulating Files
5. "System" and Tape Functions
6. Miscellaneous Functions

5.1 Entering and Leaving TSS

Before the Exec will execute any of the possible commands which may be given to it, the prospective user must make himself known by executing the "LOG IN" procedure. The procedure is explained in the "Tymshare Reference Manual". This brings into core the user's complete file directory. See Section 5.4 for commands to control the handling of files.

To leave the TSS system, the user may give the commands EXIT or LOGOUT or he may simply hang up. For a description of LOGOUT and what happens when the user hangs up, see the "Tymshare Reference Manual". EXIT is similar to LOGOUT except that the user's file directory is NOT written back on the disc with the EXIT command. This means that the user would lose any new files created since logging in or since the last file DELETE or since giving the command WRITE FD (see section 5.4).

5.2 Commands Relating to the Allocation of Memory

The commands described in this section are:

STATUS
UNUSED MEMORY
RELEASE
KILL PROGRAM
RESET
PMT

*SMT
*RSMT

*These commands require the user to have System Status.

STATUS

Types the status of user's memory. The format is as follows:

```
PROGRAM:  nn nn - -/- - - -
s.s.n: nn - - -/- - ss ss
M.S. 30K, U.M. xxK
```

where nn is the relabeling byte from the program memory table,
ss is the relabeling byte from the shared memory table,
s.s.n. is the sub-system name,
xx is the number (in thousands of words) of unused memory in the user's virtual memory.

The M.S. is the current Machine Size (virtual) for the user. The dashes indicate pages in the relabeling words that are not currently being used. Note that the program line corresponds to the two words of program relabeling kept in an exec table and the "s.s.n." line corresponds to the two words of sub-system relabeling kept in the exec table. The pages indicated by "nn" are swapped. There may be other pages currently assigned to the user as listed in his PMT (see PMT command) that are not shown by the Status typeout since his program is not currently relabeled over them.

UNUSED MEMORY

```
nnKCr
```

where nn is the number of K (1024) blocks of words of the user's total memory allocation remaining unaccessed. The user is assigned 32K upon logging in. He has no control of the one page (2K) immediately assigned as his temporary storage (T.S.) block.

The following three commands return parts of the memory currently assigned to the user to the pool of unused memory retained by the Monitor. Any information in the memory so released is irretreivable.

RELEASE

Releases the blocks (up-to-eight) of memory assigned to the subsystem the user was last using.

KILL PROGRAM

Releases the blocks (up-to-eight) of memory listed by the STATUS command under the heading PROGRAM.

RESET

Releases all the memory assigned to the user except the one block used by the Exec for temporary storage. (The T.S. block)

PMT

Types the user's current Program Memory Table in the following format:

```
    aa DRMPOS: bbb, cc (PAGE dd)
```

where aa is the pseudo relabeling byte number,
bb is the drum address (shifted right three places),
cc will type as RO for read only,
EX for exec page,
DR for drum.
dd (if typed) will be the real page number in memory.

SMT

Types the Shared Memory Table. The format is the same as that indicated under the command PMT, above.

5.3 Commands Relating to the Interaction of Teletypes

The commands described in this section are:

```
   USERS
   WHERE IS
   WHO IS ON
  #*SHUT DOWN
  #*UP
  #*ANSWER
  #*HANG UP
   *LETTER
```

#These commands require Operator or System Status. All of the above commands require at least Subsystem Status. In addition, the commands marked with # require console switch one to be toggled.

USERS
nn

Types the number of users (nn) currently logged on the system.

WHERE IS aa nnnnnnn
xx

By typing the Account Number of "aa" and the user's name for "nnnnnnn...", this command will type the teletype number "xx" that the user is on currently.

WHO IS ON
xx aa nnnnnnn..
xx sss

This command causes a complete list of the current TSS users to be typed where:

xx	teletype number
aa	user's account number
nnnn..	user's name
ssss	status of lines if the line is NOT completely idle (-1) and no one is logged on the line.

SHUT DOWN (toggle switch 1 required)

After the operator toggles console switch 1, the command will set a flag that initiates system shutdown. All lines that are not currently being used will be made unavailable.

UP (toggle switch 1 required)

After the operator toggles console switch 1, the automatic shut down flag described under SHUT DOWN is reset so that teletype lines are no longer unavailable. The operator must re-answer (by using the ANSWER command) all lines that have previously been made unavailable.

ANSWER (toggle switch 1 required)
K,m-n,...

This command enables selected teletype lines so that the users may make use of these lines. The operator may specify single numbers, indicated by "k", separated by commas, or a range of numbers where the range is separated by a dash, or any combination as indicated. Spaces are ignored and the string is terminated by a carriage return. If the line has already been enabled, the command will have no effect. Note that after the SHUT DOWN command has been issued, a line can be made available by this command but it will become unavailable after the user logs out.

HANG UP (toggle switch 1 required)
k,m-n,...

The command has two functions; it may be used to hang up a user while he is logged in (in this case the line will go ready again after the hang up operation has been completed unless the SHUT DOWN command has been used), or it may be used to make a line not available if no one is currently using the line. The format is exactly as described under ANSWER.

LETTER Cr
LETTER OFF/ON

LETTER n

This command has three functions and two formats. The second format, where a number n is typed after the command, is used to type a broadcast letter, where n is the letter number from one to six.

The first format is used to control the transmission of broadcast letter. It is used by typing a carriage return immediately after the command. If the response is LETTER OFF, then no one will receive the broadcast letters. The exec will not come back to the "-" response until all users currently on the system have finished receiving any letters addressed to them. If the operator desires, he may "escape" from this condition by typing the escape key. No harm is normally done except that it is possible that a user may receive the same letter twice. The operator must set the LETTER OFF condition before he can use the "Operator Program" to create new letters or cancel old ones. If the response is LETTER ON

then all users will start receiving broadcast letters. Any new letters created by the operator will not start being received by all users currently on the system.

5.4 Creating and Manipulating Files

The following Exec commands are available. They are described in the references indicated.

FILES	1,2
WRITE FD	1
DF	1
FD FOR	1
DELETE	1,2
RENAME	1,2
GO TO	See Below
PLACE	See Below
SAVE	See Below
DUMP	2
RECOVER	2
CONTINUE	2
COPY	2
*GFD	See Below
*REMOVE FILE	See Below
CREATION	1
CD FOR	1

*These commands require Operator or System Status in the user's control parameters.

GO TO (input file name)

The action is initially as for the PLACE command. However, after transferring the file to core, instead of a return to the Exec, there is a branch of control into the user's own environment at the starting address specified at the time of the file's creation. If a zero starting address or none at all was then given, the transfer is back to the Exec as for the PLACE command.

PLACE (input file name)

The contents of the named input file is transferred to the core addresses specifed at the time of its creation (by BRS 93 or the "SAVE" command. It is transferred into the user's current environment which is extended, as necessary to accommodate it.

The file name must be in the user's file directory. If it is not, a ? is printed, the name is forgotten and must be delivered anew. The file name must be terminated by a carriage return. The file must be a core-image (type 1) file (see section 4.0). If any of these conditions is not satisfied, the command is aborted -- as it is also if the attempted data transfer to core results in some transfer-error conditions arising.

SAVE

This command is typed in the following format:

```
SAVE bbb TO eee ON nnnnn Cr
OLD/NEW FILE Cr

    or (optionally)

OLD/NEW FILE Lf
STARTING LOCATION sss Cr
```

The contents of the specified range of core starting with "bbb" and ending with "eee" together with the starting location "sss" if provided, are preserved on the named output file "nnnnn".

The output file name must be of a form accepted by BRS 16. If it satisfies the conditions for a no-skip return from BRS 16, the name is ignored and another name must be provided. The name may be terminated by a carriage return, thus terminating the command and causing it to be executed, or a line feed, in which case a "starting address" (see also the "GO TO" command) must be typed in.

Each of the addresses bbb, eee, sss, whether core range limit or starting address, is interpreted as an octal number. The starting address, sss, must be terminated by a carriage return. Delivery of any other non-octal digit character, except rubout, aborts the address -- which must be retyped. The octal numbers, bbb and eee must all be terminated by a space, a comma, or a carriage return. Any other character aborts the command.

If a carriage return is typed immediately after the command "SAVE" the Exec will respond with "FIRST LOC". If a carriage return is typed immediately after "bbb", the Exec will respond with "LAST LOC".

If a carriage return is typed immediately after "eee" the Exec will respond with "TO FILE".

GFD aa nnnnn Cr

The command is used by the operator to get a file directory belonging to another user for special background or non-timesharing processing. The operator's own file directory and user number is replaced by that belonging to the account number "aa" and user name, "nnnnn" but the operator's account number and control parameters are retained.

REMOVE FILE nn, Cr

This command allows a user with System or Operator Status to remove an entry from a file directory without using the DELETE command. Since it may be possible to delete a file if the name contains leading spaces or other spurious characters, it may be required to use this command as a last resort. The command removes a file from the "in-core" directory by referring to the file name's position "nn" in the printed file directory. The command FILES must be used just before using this command in order to find the current relative position of the name. The file directory is NOT rewritten on the disc by this command.

References:

1. Reference Manual for the Time-Sharing System, Chapter 13, "Executive Commands Related to Files".

2. Tymshare Reference Manual

5.5 "System" and Tape Commands

The commands described here are:

REWIND
RLT
STN
PTN
POSITION TAPE
TAPE POSITION
*ABT
#SYSDP
#SYSLD
*LOOK

*These commands require operator user status.
#Require system status.

The commands controlling tape are system commands and only one user, normally the operator, will be mkaing use of the commands at a time.

REWIND Cr

This command frees up the tape, regardless of its current status and rewinds the tape. It is applied to the current tape number (0 or 1).

RLT Cr

This command releases the tape so that it is available for other users.

STN n Cr

Allows a user to set his own tape number, where "n" is 0 or 1.

PTN Cr
n Cr

Types a user's current tape number, where "n" is 0 or 1.

POSITION TAPE Cr

This command will cause a user's current tape to position to the beginning of the next file.

TAPE POSITION Cr

Types the current tape position as far as it is known by the Exec. This command does not check the actual position by reading tape.

ABT Cr

This command will abort any tape operation currently in progress. It may be used by the operator to stop run-away tape.

LOOK

This command is typed in the following format:

```
LOOK a,n Cr
a    bbbbbbbb
a+1  bbbbbbbb
etc.
```

This command allows an operator or system class user to display real memory addresses where "a" is the first location to be displayed (in octal) and "n" is the number of locations in decimal to be displayed. The format of the type out is as indicated in the example where "a" and "a+1" are the octal addresses and "b" represents the contents in octal.

The following two commands require a special system status by the user since they allow direct writing and reading at any location on the disc.

SYSLD

The command is typed in the format:

```
SYSLD a Cr
TO    b Cr
LOC   c Cr
```

This command allows a user to load his program memory from any location on the disc into any of his eight pages. "a" and "b" refer to his page numbers from 0 to 7 and "c" is either a real disc address or a number from 0 to 7 referring to disc 0 to 7, with the load (or dump, see below) starting at arm position 63 of the given disc. Also, "c" may be of the format "n.m" where "n" is the disc number described above and "m" is a number from 0 to 7 referring to

a relative page number of arm position 63. Note that the dump and load location using the specified disc format corresponds to the area of the disc addressed by the disc swap utility program using the console switch settings 0 to 7.

SYSDP

The command is typed in the following format:

```
SYSDP a Cr
TO    b Cr
LOC   c Cr
```

This command allows a user to dump his program memory onto any location on the disc from any selected pages of his eight pages of program relabeling. The nomenclature is the same as that described under SYSLD (see above).

5.6 Miscellaneous Commands

These fall into none of the preceding categories. They are described in the references indicated.

BRANCH	See Below
DATE	See Time-sharing System Ref. Manual
TIME	" " " " " "
"	" " " " " "
ACCOUNTING	See Below
PSP	" "
SETEXEC	" "
ENABLE/DISABLE	" "

BRANCH adr Cr

A transfer of control is made to the specified address "adr" in the user's own environmnet. The address (an octal number) must be terminated by a carriage return. Any other character aborts the command. If the user does not have the page containing the address under his relabeling, he will receive a memory trap. If he has a blank page (containing the illegal instruction HLT), he will receive an instruction trap.

SETEXEC nn Cr

This command is available only to users with one of the special status. These users may use the

command to set one of the following classes of executivity if the user's status parameters agree that the user is permitted to use this class. The class set is then propogated to any fork started by the system executive under the "GO TO" command.

"NN"	DESCRIPTION
1	Subsystem
0	Cancel status
-1	Subsystem & system
-2	System only

The various classes allow the user's program to issue special BRS's that are needed for system software but could cause great havoc to the TSS if used improperly. Debugging of programs which use these BRS's must be restricted to certain time periods so as not to disrupt T.S. operations.

PSP Cr

This command requires operator status or a higher status. It will type out with symbols the current system error counters. For a key to the symbols and their meaning, see the current Tymshare Monitor Manual.

ACCOUNTING n Cr

This command requires operator status or a higher status. After the Cr is typed, the following message will type:

TOGGLE SW. 1 Cr

The command will not execute until console switch 1 is toggled. It will then perform one of the following functions depending on n.

n = 0 Stops the accounting information from being punched on paper tape when users log out.

n=-1 Starts the punching of the accounting information on paper tape when users log out.

A number of special purpose TSS software aids, called "subsystems" can be requested simply by typing the name of the subsystem as a command. Two commands

allow the operator to ENABLE or DISABLE groups of subsystems so that various classes of users may or may not use the subsystem group. The commands are typed as follows:

```
ENABLE s or
DISABLE s
```

where s is the name of a subsystem in the group. The subsystems currently available are grouped as follows:

```
Group 1 - ARPAS, DDT
Group 2 - LISP, SNOBOL
```

The following subsystems have no group restrictions and are always available to all users:

```
BASIC
FTC
FOS
QED
CAL
FORTRAN
```

For details of any subsystem, the appropriate subsystem manual should be consulted.

TYMSHARE, INC.

OPERATIONS MANUAL

Prepared By:

Dave Brallier
Los Altos, California

Dean Marr
Los Angeles, California

January 6, 1967

SYSTEM LOAD FROM DISC

4-1-67

GENERAL: The timesharing system program is now stored on the disc as well as tape. There can be several versions (as well as several copies) on the disc at the same time. It is the Operator's responsibility to see that the correct version is loaded.

The new load procedure will dump as well as load; i.e., if a crash occurs, DSWAP3 will dump the crashed system on the disc and load the new system into core. (The dump can be bypassed by placing BP Switch 1 down.) The selection of the system to be loaded is controlled by break point switches 2, 3, and 4 on the console. The switches are tested octally per example 1:

EXAMPLE 1:

BP	BP	BP	BP
1	2	3	4

If the system to be loaded is on Disc 3, place BP #3 and BP 4 down. This is interpreted as an octal 3 and will load the system from Disc 3.

LOAD PROCEDURE:

1. Place DSWAP3 in the paper tape reader

2. Set BP switches to correspond to the system to be loaded. If a crashed system is not to be saved, put BP switch 1 down also.

3. Standard fill from paper tape. This consists of the following steps:

   a) IDLE The Run-Idle-Step switch is put in IDLE
   b) START Depress the Start button. This clears the P and C registers
   c) RUN The Run-Idle-Step switch is put in RUN
   d) FILL The paper tape switch is toggled. This reads in the paper tape. The HALT light will come on when the paper tape is completely read in, but if the ERROR light comes on, the paper tape must be reloaded. If the ERROR light comes on, let the loading go on to completion, for if the Run-Idle-Step switch is taken out of RUN and put into IDLE, the paper tape will run away. If this happens, press START (this stops the paper tape) and re-position the paper tape and then go back to Step 3a.

   This will execute the dump of the crashed system (if BP switch 1 is up) and load the new system into core. As stated in the beginning of the section, if the crashed system is not to be saved, BP switch 1 must be down.

NOTE: There are two phases to DSWAP3; the WRITE and READ phases. Which phase DSWAP3 is in is indicated by a 66 in the W buffer for a WRITE and a 26 for a READ. If a crashed system is to be saved, obviously the WRITE phase (writing the crashed system onto the disc) would occur first (a 66 in the W buffer), for if the new system was read into core first (a 26 in the W buffer), it would read in over the crashed system and destroy it.

If, for any reason DSWAP3 is aborted while saving a crash before it is fully executed, be careful to observe which phase, the WRITE (66), or the READ (26) the DSWAP3 is in. If it is in the WRITE phase at the time of the abortion, precede as before, but if DSWAP3 is in the READ phase, that means that the crashed system has been written on the disc and and the new system is partially read into core and therefore, the crashed system that was in core is now destroyed. In order not to write the new system that was partially read into core onto the disc and wipe out the crashed system already there, BP switch 1 must be put down when going through the preceding steps.

4. Prog. will stop at P=25

5. IDLE The Run-Idle-Step switch is put in IDLE. The BP switches are reset.

6. RUN The Run-Idle-Step switch is put in RUN

After the above 6 steps have been completed, the system will respond on Teletype 1 with the following:

1. 81-nE (mo-day-time):

   This is a request to have the month, day, and time entered after the colon as per the example that follows: 11-17-1530 CrLf

2. LAST START UP n/n n:nm CrLf

   Nothing is required of the Operator at this point

3. NO. OF USER:

   This is a request to have a number entered after the colon. The number should correspond with the number of channels to be answered, as per the following example: 15 CrLf

4. PAGES:nn CrLf

   n=The number of pages available to the individual users (n varies depending on the number of users). A page equals 2048 words. Nothing is required of the operator at this point.

5. PLEASE LOG IN!

This response will occur after the system searches the disc and builds a bit map. The time it takes to build a bit map is a function of the size of the disc, the number of users, and the number of files.

The response of the Operator to the "PLEASE LOG IN!" command depends upon one thing; is the crashed system that was dumped on the disc to be saved or not?

a) @1;Operator CrLf

The Operator logs in under this account number and user name if the crashed system is not to be saved.

b) @1; CRASH CrLf

The Operator logs in under this account number and user name if the crashed system is to be saved. The reason for this is to save all crashes under an identifying user name.

All the procedures that are done under @1;OPERATOR can be done under @1;CRASH.

NOTE: (a) and/or (b) mentioned above may be done between Steps 4 and 5 to speed things up.

6. READY (date) (time) CrLf

This is a response giving the date and time that the Operator got on the system. Nothing is required of the Operator at this point.

7. The system will reply with an Executive Dash (-) indicating that the computer is in the Executive mode and that it is ready to accept any command from the Operator.

8. Answer the Channels (See "Answer" Section).

CRASH SAVE

This is a procedure used to save the crashed system on the disc. The Operator is logged in under @1;CRASH.

1. ST^{C}S^{C}EX$_{\wedge}$-1 CrLf

   This command, known as Exec Status, is needed in order to do the following steps.

2. ST^{C}S^{C}LD$_{\wedge}$0 CrLf

   The computer responds with:

   TO The Operator types: 7 CrLf
   The computer responds with:
   LOC The Operator types: 0 CrLf

   The computer will effectively place the crashed system into core.

3. SAVE$_{\wedge}$0$_{\wedge}$37777$_{\wedge}$ON$_{\wedge}$ /FILE NAME/ CrLf

   The crashed system is put on a file. The file name should be in the form of /day-time-P reg. /

4. LOG OUT CrLf or DELETE$_{\wedge}$/FILE NAME/ CrLf

   The Operator types one of these two commands, in order to write the file directory of the crashed file on the disc. The reason for this is that if the system should crash again before the operator logs out or deletes a file in the normal run of things, the saved crash file would be lost.

5. ST^{C}S^{C}EX$_{\wedge}$-1 CrLf

   This step has to be done if the Operator logged out in the previous step. The Exec Status is needed to do the remaining steps.

6. RESET CrLf

   This command clears all programs out of memory

7. RECOVER$_{\wedge}$/JST/ CrLf

   The file JST, system J symbol table, is recovered from the disc and put into core.

8. ST^C S^C LD↑0 CrLf

The computer responds with:
TO The Operator types: 1 CrLf
The computer responds with:
LOC The Operator types: 0 CrLf

The crashed system is loaded into core

9. CONTINUE CrLf

This command will put the DDT Program into the operating system

10. WERISC

This command is followed by 18 linefeeds. It gives the location of each user number in reference to each channel at the time of the crash. The 1 in the WERIS + 1 is the channel number.

AUNNC

This command is given after the WERISC is finished and while still in DDT. It may be given on the same line as the last WERISC. The AUNNC gives the account number and user number in reference to the job number. The 1 in the AUNN + 1 is the job number.

There is no relationship between the 1 in the WERIS+1 and the 1 in the AUNN+1

If only a certain section is wanted, the command is typed with the first channel number of that section; i.e, WERIS + 17

Two Altmodes will put you back in the Exec

11. Go to the "SPS" Section

DISC LOAD

1. Mount the disc dump/load program on Unit 0

2. Mount the tape to be dumped on the disc on Unit 3

3. Set BP Switch 1

4. Standard FILL

   a) IDLE Put the Idle-Run-Step switch in IDLE
   b) START Depress Start button to clear P and C registers
   c) RUN Put the Idle-Run-Step switch in RUN
   d) TOGGLE MAG TAPE SWITCH This reads the disc/dump load program into core. The HALT light will come on when it is completely read into core. If the W buffer ERROR light comes on while the program is being read in, rewind tape and START over.

5. IDLE

6. START

7. BRU 207 - Enter 100207 in C Register

8. Computer will halt with 2010101 in C. Register

9. IDLE

10. RUN

11. Toggle BP switches 3, 4, 3

    The BP switches are programmed to act as a combination lock, to prevent the accidental loading of the disc. They must be used exactly as described or a HALT will occur. To recover from the error HALT go to Step 6.

See List of ERROR HALTS for
Disc Dump/Load

DISC DUMP

1. Mount the disc dump/load program on Unit 0

2. Mount scratch tape on Unit 3

3. Standard FILL

   a) IDLE
   b) START
   c) RUN
   d) TOGGLE MAG TAPE SWITCH. This reads in the disc dump/load program into core. The HALT light will come on when program is completely read into core. If the W buffer error light comes on, rewind tape and start over.

4. IDLE

5. START

6. BRU 207 - enter 100207 in the C register

7. Computer will halt with 2010101 in the C register

8. IDLE

9. RUN

10. Toggle BP switch 4, 3, 2

    The BP switches are programmed to act as a combination lock. They must be used exactly as described or an error halt will occur. To recover from the error, go to Step 5

See List of ERROR HALTS for
Disc Dump/Load

DISC DUMP/LOAD

ERROR LIST

P = 512	Tape not ready
P = 515	W buffer not ready
P = 531	W buffer staying busy

For any of the above errors, start load or dump again.

C = 2000001	Tape read errors
C = 2000002	Tape read errors
C = 2000003	The 1, 2, or 3 indicates the logical record within the physical record on which the error occurred.

When any one of these errors occur, it indicates that ten read errors have occurred on a logical record. The physical record on which the tenth error occured is designated by 1, 2, 3, corresponding to the three physical records on a logical record. If this happens, clean tape head and start again. Should this fail, clean tape head on other drive and try again on that drive. If still no success, PUNT.

C = 2000005	Seek time or search time error on disc controller
C = 2000006	Disc controller error
C = 2000007	W buffer error

For any of the above 3 errors, the following action is to be taken:

1. Go to IDLE
2. Press Controller Clear
3. Go to RUN

If a read error occurs, indicate in Log Book on what disc, track, and sector it occurred. Contact Center Manager.

CARD TO TAPE

1. Mount card to tape (CTT) program on Unit 0 and set density

2. Ready punched cards in card reader

3. IDLE

4. START

5. RUN

6. TOGGLE MAG TAPE SWITCH This loads the CTT program. The HALT light will come on when the program is loaded successfully. If an error occurs (W buffer error light) while loading the program, rewind the tape and load again

7. Take tape drive out of AUTO

8. Mount scratch tape an Unit 0 and set density

9. IDLE

10. START

11. BRU 200 - enter 100200 in the C register

12. RUN

13. E. O. F. When last card has been read, depress E. O. F. When the E. O. F. is depressed, the number of words copied to the tape will be typed out on the maintenance teletype

NOTE 1: The system requires that a dummy deck be placed on the tape as the last file. The dummy deck need consist of 1 card only. It must be added or the system will crash when an attempt is made to copy the tape to disc.

NOTE 2: Decks may be stacked. The only limit on the number of decks which can be stacked is the amount of tape.

NOTE 3: When any Reader error occurs, the READY light will go out. Until further notice, take the following action on any error condition:

1. Reset (clear) error condition
2. Terminate processing of that deck, i. e., depress E. O. F.
3. Restart that deck

TAPE TO CARD

1. Mount tape to card (TTC) program on Unit 0 and set density

2. Place blank cards in card punch

3. IDLE - Idle-Run-Step switch is put in IDLE

4. START - Press START button to clear P and C registers

5. RUN - Idle-Run-Step switch is put in RUN

6. TOGGLE MAG TAPE SWITCH This loads the TTC program. The HALT light will come on when the program is loaded successfully. If an error occurs (W buffer error light) while loading the program, rewind tape and load again

7. Take tape drive out of AUTO

8. Mount File Tape on Unit 0 and set density

9. IDLE

10. START

11. BRU 200 - Enter 100200 in the C register. The card punch will Cycle 1 card

12. RUN

13. FILE NUMBER - When the TTC program is ready to accept input of a file number from the maintenance teletype, the input light will come on. The number must be inputted as a two-digit octal number, i.e., 05 = 5th file on tape. The files must be inputted in ascending order, though they do not have to be in sequence.

SYSTEM TAPE COPY

1. Place "32k DUMP" paper tape program in paper tape reader
2. Mount system tape (Disc Dump) on Unit 0
3. IDLE
4. START
5. RUN
6. TOGGLE MAG TAPE SWITCH
7. When computer halts, take system tape out of AUTO
8. Mount scratch tape on Unit 0, set density and put in AUTO
9. IDLE
10. START
11. RUN
12. TOGGLE PAPER TAPE SWITCH
13. Watch W buffer for error
14. When copy is finished, load disc with copy and bring system up. If you can LOG IN, tape copy is good.

SAM OUTLINE

I SAM SYMBOL CHANGE

A. Print CST Table

-RECOVER↑/NRECSAM/ CrLfLf

-CONTINUE CrLf

DDT Lf
CST" . LfCr
CST+1":LfCr
CST+2"! LfCr
.
.
.
CST+11" 7 CrLf
CST+12" 8 LfCr
CST+13" 9 LfCr
.
.
.
CST+76" $ CrCrLf

B. Changing User Symbols (Still in DDT)

CHANGE;G LfCr
2:1, 3:2-57, 3:60, 12:100-111 LfCr
56:112-157, 33:160-177, 76:300-700 CrLf

240;G LfCr (This will run the program)
1830..:$$ 5 (2 Altmodes)

-DUMP↑/NEWRECSAM/ CrLfLf

-SAVE↑0↑TO↑3777↑ON↑/SAM/ CrLf
NEW FILE or OLD FILE LfLf

STARTING LOCATION 240 CrLfLf
.
-ST^CS^CEX↑-1 CrLfLf

-GO↑/SAM/ CrLf
1840 ..::$$$! 8

II CHANGE INCTIK

-RECOVER↑/NRECSAM/ CrLfLf

-CONTINUE CrLf
DDT CrLf
INCTIK/ *3120 16040 CrLf

INCTIK/ 16040 CrLf
(2 Altmodes)

-DUMP↑/NRECSAM/ CrLfLf

-SAVE↑0↑TO↑3777↑ON↑/SAM/ CrLf
OLD FILE or NEW FILE LfLf

STARTING LOCATION 240 CrLf

-ST^CS^CEX -1 CrLfLf

-GO↑/SAM/ CrLf
1840 ABV↑:... 8

SAM DESCRIPTION

SAM is a GO TO Program which periodically prints out the number of users on the system. The users on the system are represented by symbols. These symbols are the letters of the alphabet, numbers 1 through 9, and special characters such as *, ., $, etc. The procedure to change the symbols that represent a user is discussed in Section I.

As stated above, SAM prints out periodically. The time increment can be changed. This is discussed in Section II.

I SAM SYMBOL CHANGE

As stated above, symbols such as A, 9, $, etc., represent a user on the system. Each symbol in the SAM program is represented by an octal number. For example, 1B (B indicates that the number is octal) is equated to the character:, 44B is equated to the letter Y., etc. The octal number and what it is equated it is found in the CST Table. These octal numbers in the CST Table, along with the user numbers, are used by the SAM Program to print out a symbol for a particular user.

A. LISTING THE CST TABLE

-RECOVER /NRECSAM/ CrLfLf

This loads the recover file (24 type file) into core. In this case, the file name is NRECSAM, but this is not always so. As long as the recover file is the version you want to use, the name makes no difference.

-CONTINUE
DDT

You are now in DDT. This is the only language that can be used with a recover or 24 type file.

CST: . Lf
CST+1": Lf
CST+2"! Lf
.
.
.
CST+76"$

To list the CST Table the user types CST" and the computer will type a period (.) The octal number representinga period is 0. A Lf after the period will cause CST+1": to be printed out by the computer. 76 linefeeds will print out the complete CST TABLE.

One carriage return in place of a linefeed will put the user in a position to enter another command. This one carriage return will not give a linefeed, so the user is on the same line. But two carriage returns will give a linefeed and put the user on the following line. At this point, the user might want to know what symbol is equated to 12B, for example. He would do the following:

CST+12: 8 Lf or Cr

'8' is the symbol represented by 12B. After the '8', the user may do one of two things: 1) he may type linefeeds which would continue the print out of the table starting at CST+13, or 2) He may type a carriage return which would allow him to enter another command.

B. CHANGING USER SYMBOLS

If a symbol for a user is to be changed, or a symbol assigned to a new user, the following procedure is gone through:

-RECOVER$_{\wedge}$/NRECSAM/ CrLfLf

-CONTINUE CrLf
DDT

Determine what symbol is to be used and find its octal number equivalent in the CST Table. Also, determine the user number(s) to be assigned to the symbol.

CHNGE:G LfCr
Symbol Number:User Number, Cr or Lf

The user number need not be restricted to one, but a contiguous block may be put in; i.e., 2:1-57,. Any number of changes may be made on one line; i.e., 2:1-57, 3:60, 12:100-111, 56:200-500,. The change is terminated by a carriage return, or the changes are continued on to the next line by a linefeed.

There can only be one symbol assigned to a user number at one time. So the current change will replace the old symbol assignment for that user number.

If a user name is deleted and no user name reassigned to the user number, then that user number is assigned to the symbol 'quotes' ("). This is the symbol to which all unassigned user numbers are assigned.

NOTE: There must always be a comma (,) after the user number. Except at end of a line where a line feed serves the same purpose. Also, if a mistake is made, this is remedied by typing a ? immediately after the mistake. This would delete the entry with the mistake in it and give a carriage return and linefeed and put DDT in the command mode again. To precede with the changes, type CHANGE;G.

After the changes and new additions have been made, one of two things may be done after the carriage return: 1) The changes may be checked against the users on the air at the time by typing 240;G. This will cause SAM to run with the new changes. To get out of the running program, hit altmode once. This will put you back into the command mode. Hitting altmode twice will put in in the Exec. 2) Hit altmode twice and get back to the Exec.

Once back in the Exec, you are ready to dump the corrected recover file onto a new recover file.

-DUMP↑/NEWRECSAM/ CrLfLf

This dumps the recover file that you have made changes to in core to a new file called NEWRECSAM.

-SAVE↑0↑TO↑3777↑ON↑/SAM/ CrLf
NEW FILE LfLf

STARTING LOCATION 240 CrLfLf

-ST^CS^CEX -1 CrLfLf

-GO↑/SAM/ CrLf

The SAVE Command stores the core image of the recover file on a file called SAM. Two linefeeds after the NEW FILE (or OLD FILE) print out will cause STARTING LOCATION to be typed out. This is a command for the user to type in at what location he wants the GO TO program to begin. In this particular case, and in most others, the starting location is 240.

To run SAM Exec Status (ST^CS^CEX 1)must be set.

SECTION
II CHANGING SAM INCTIK

A. Description of SAM INCTIK

The SAM INCTIK is the time interval for the SAM type out. The INCTIK can be set from one second on up. (The usual time interval is 2 or 5 minutes). Since the timing of SAM is dependent on the real time clock, and the real time clock is dependent on the AC current, 60 cycles would equal one second. So 60 times the number of seconds and the result converted to octal would be the value entered fo the INCTIK. To convert to octal, do the following:

5 X 60 X 60 = 18,000

8|18000 = 0
8|2250 = 2
8|281 = 1 = 43120B
8|35 = 3
4 = 4

The 4, 3, 1, 2, and 0 are the remainders of the divisions. This is the octal number entered for the INCTIK to cause SAM to print out every five minutes.

To change the INCTIK the recover file or 24 type file that is to be changed must be loaded into core and the change made under DDT. This is done per the following example:

-RECOVER↑/NRECSAM/ CrLfLf

-CONTINUE CrLf
DDT CrLf
INCTIK/ *3120 16040 CrLf
INCTIK/ 16040

Typing INCTIK/ will cause the computer to print out the current value of the INCTIK, which in this case, is *3120. (Due to the configuration of the hardware, a 4 (four) will print out as an asterik (*)). The carriage will space over to the next tab stop. If the INCTIK is to be changed, the new value is entered here, which in this case, is 16040 on two minutes. To see if the new value of the INCTIK has been accepted, again type INCTIK/, as per above example, and the new value will be typed out.

If, while typing in a new value of the INCTIK, a mistake is made, type a question mark (?) and the computer will delete the value entered with the mistake and space forward to the next tab stop. You can now enter the new value.

After changing the INCTIK, you can either make further changes or go back to the Exec. To go back to the Exec, hit the altmode 2 or 3 times.

Once back in Exec, dump the file and create a GO TO file per the following example:

-DUMP↑/NRECSAM/ CrLfLf

-SAVE↑0↑TO↑3777↑ON↑/SAM/ CrLf
NEW FILE or OLD FILE LfLf

STARTING LOCATION 240 CrLfLf

-

OPER

DESCRIPTION

OPER is a utility program that contains 24 commands for the upkeep of the system. To use OPER, Exec Status must be set. OPER is a 21 type file or a GO TO program. To access the commands in OPER, the following must be done:

-ST^CS^CEX -1 Cr

-GO_OPER Cr

*

The asterik indicates that OPER is ready to accept any of the 24 valid commands. These commands, what they do, and how to use them, are described in the following sections.

```
* HELP Cr

THE VALID COMMANDS ARE:

HELP
UAD
LENGTH
TIME
SET DAY
RESET TIME
SET HOUR
FILES
CLEAR FILE
SIZE ACCOUNT
ACCOUNT
NAME
CANCEL ACCOU
CANCEL NAME
OVERFLOW
MAP
GARBAGE
POINTER
USERS
COUNT LETTER
REMOTE LETTE
LETTER
COPY RECORDS
CLEAR RECORD

*
```

As can be seen, the command HELP lists the valid commands that can be used under OPER. When it is finished the listing, the program comes back with the asterik. OPER is now ready to accept another command.

```
*UAD Cr

OUTPUT TO: PR Cr

3/24     22:15


*1  ␣Y␣E␣T                                    0  77777777
    UTILITIES␣S          77770001
    OPERATOR␣P           77770025
    SYS81␣T              77770037

*2  ␣T␣A␣B                                    0  77777777
    B␣ILL                      77
    JACK                      156
    A␣ND                 20000036
    JILL␣F                 200536

*3

.
.
.

TOTAL:  0:00.00 0:00

.....END OF JOB.....
```

The command UAD Cr will print out all of the active accounts. In the above example, the output was to the printer (PR) So all the information from "3/24 22:15" to "...END OF JOB..." will be printed out on the printer. When it is through printing the computer will output to the teletype "END JOB" and return an Executive Dash (-).

If one wishes to only print out one account, such as A5, then a linefeed after *UAD instead of a carriage return will allow one to do this. Instead of outputting to the printer, output to the teletype.

Example:

```
*UAD Lf

OUTPUT TO: T Cr

A5 Cr
A5&NU &T                    0  77777777
    JAMES&W      201
    BILL&R       202
```

TOTAL: 0:00.00 0:00

END JOB

-

After the carriage return (Cr) in the "OUTPUT TO:" request, the computer waits for an input of an account number, such as the A5 in the preceding example.

The printer designates a control letter with a delta before the letter, i.e, ΔN. The teletype designates a control letter with an ampersand before the letter, i.e., &N. For the atsign (@) accounts the printer will print out an asterik for the @.

As can be seen from the above examples, the operating system returns an Exec Dash after printing the UAD. To give anymore OPER commands, one must get back into OPER.

NOTE: See Appendix A for description of the UAD output.

```
*FILES Cr

OUTPUT TO: PR Cr

3/19  12:13

1    0:00.00 0:00  77777777
     101122  22000000  31656 /$/
     177012  21000000  10316 /▵GOP/
     102110  22000000  14100 /8SSY/
OVERFLOW:  1350
2    0:00.00 0:00  77777777
          .
          .
          .
1350 o:00.00 0:00
     701105  22000000  32631  /▵GT8/
OVERFLOW:  1
          .
          .
          .
TOTAL:  0:00.00

.................END OF JOB.........
```

The Operation of the FILES command is the same as for UAD; that is, a carriage return after FILES will print all of the files or a linefeed will only print out the ones selected.

With a linefeed (Lf) after FILES, one may select more than one file to print.

Example:

```
*FILES Lf

OUTPUT TO: T Cr

3/19  12:30

77
77 0:00.00 0:00 77777777
     631101  23000000  12343 /MUD/
     612066  23000000  36477  /QUI/

102
102  0:00.00 0:00  77777777
     605115  24000000  25776  /$/
```

To get out of the FILES command, hit almode a few times

*CLEAR FILE Lf

3/24 8:30

105
107
300

Altmode

-

The CLEAR FILE command deletes all files under a user number. After the date and time is printed out, the computer waits for the user number or numbers whose files are to be deleted.

*ACCOUNT Cr
LNP$_{1}$ --- P$_{1,2}$ Cr
TTTTTTTT,AAAAAAAA Cr
Alter Switch 1

NEW or OLD

*

The ACCOUNT command is used to enter new accounts, change account time parameters, change account parameters, or change account passwords. The ACCOUNT command requires four arguments:

Where L = Account Letter
N = Account Number
P = Password
T = Time Parameter

There must be a space between the Time parameter and the Account parameter.

Altering switch 1 enters the account into the system. Up to this point, one may altmode out of the ACCOUNT command. This is the only way to correct a mistake in input.

Whenever an account is entered or changed, it must be checked to see if entered correctly by doing a UAD for that account.

*

The Account Letter, L, can be any letter from A through Z and the special character @. The @ is restricted to internal Tymshare usage only.

The Account Number, N, can be any number from 1 through 8.

The Password, P, can consist of any combination of numbers, letters, and control characters (up to 12 characters)

The Time Parameter, T, controls the access time of the user. It is usually 24 hour access. Sometimes, parameters are as follows:

77777777 or -1	=	24 hour access
37700	=	10AM to 6 PM
37774	=	10 AM to 10 PM
1400	=	2 PM to 4 PM
77600377	=	4 PM to 8 AM
77600077	=	6 PM to 8 AM
1477	=	2 PM to 4 PM, 6 PM to 12 PM

6074	=	12 AM to 2 PM, 6 PM to 10 PM
300	=	4PM to 6 PM
1700	=	2 PM to 6 PM
377	=	4 PM to 12 PM

The time parameter is right justified

The Account parameter, A, is not used at the present time, but a zero (0) must be entered in order for the Account command to be executed

*NAME Cr
LNU,--- U,$_2$Cr
PPPPDDDD Cr
Alter Switch 1

NEW or OLD

*

The NAME command is used to enter a new user name into an account or to change the user parameter. The NAME command requires the following arguments:

Where L = Account Letter
N = Account Number
U = User Name
P = User Parameter
D = User Number

The user parameter must be typed only if it is other than zero (0). The user parameter/user number is right justified.

NOTE:D(user number) should be unique for each user. If it is not, both users will share the same file directory.

Altering Switch 1 enteres the user name, etc., into the system. To correct any mistakes an input, altmode out of the NAME command before altering Switch 1.

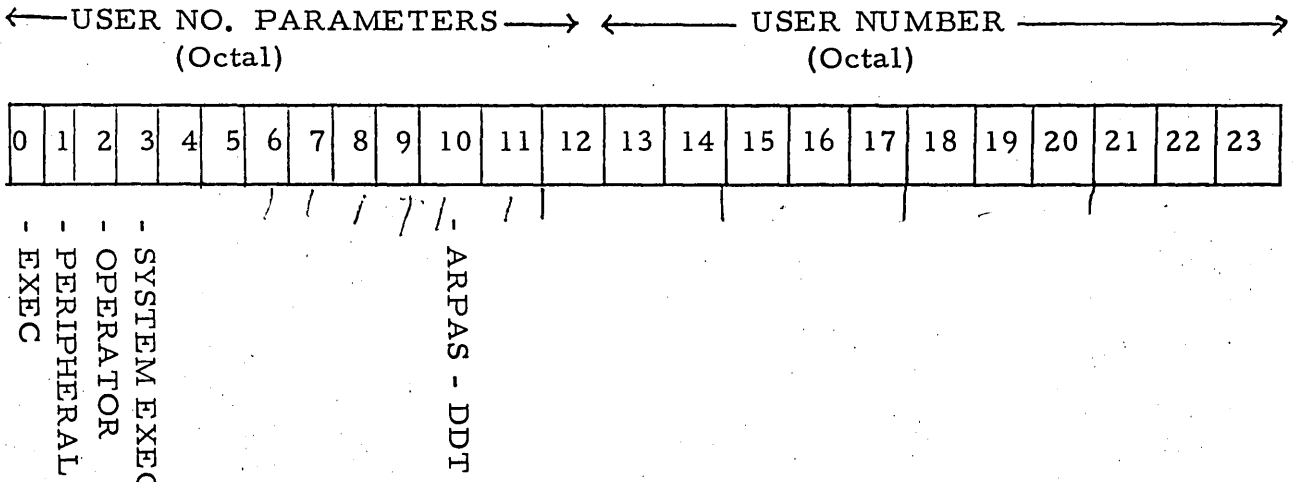
Whenever a user name is entered or changed, it must be checked to see if entered correctly by doing a UAD for that user name's account.

*
The Account letter, L, and the Account Number, N, must have been entered previously by the use of the ACCOUNT command.

The User Name, U, can consist of any combination (up to 12 characters) of numbers, letters, or control characters.

The User Parameter, P, will not be used in the majority of cases for outside users. It is generally restricted to internal use. The user parameters allow a person to have Exec, Operator, Peripherial, System Exec, or ARPAS-DDT status, all five, or any combination of the five.

The User Number, D, is necessary for all users. At the present time, it is an octal number from 1 to 777. Under this number is all the user's files. The number is different for every user.


USER NO. PARAMETERS (Octal)
USER NUMBER (Octal)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
- EXEC
- PERIPHERAL STATUS
- OPERATOR
- SYSTEM EXEC
- ARPAS - DDT

*CANCEL ACCOU Cr
LN Cr
Alter Switch 1

OLD

*

Where L = Account Letter
N = Account Number

Confusion may arise with this command, in that cancelling an account does not cancel the users in that account. If the account is to be cancelled completely, the user names should be cancelled first by using the CANCEL NAME command (See following section).

As in the other commands, one may altmode out of the CANCEL ACCOU command before altering Switch 1.

*CANCEL NAME Cr
LNU, --- U/2Cr
Alter Switch 1

OLD

*

Where L = Account Letter
N = Account Number
U = User Name

This command cancels the user name. But cancelling a user name does not eliminate the files that the user may have amassed. There are two ways to eliminate these files:

1. By using the DELETE command in Exec before cancelling the User Name.

2. By using the CLEAR FILE command in OPER, which is much quicker and which does not require logging in under the user nameand therefore, does not require that the user name still be in existence.

Whenever a cancellation is made, do a UAD for that particular user account to see if it is cancelled.

OVERFLOW

The OVERFLOW command is used to assign an overflow number to a user having used up all of his file space under his initial user number. The initial user number is still the identifying number for a user. The block of user numbers from 1237 to 1377 is to be used for overflows.

The system will not allow more than one overflow to be assigned to a user number. An overflow can not be assigned while the user who has that user number is logged in. Also, the system will not allow an overflow to be cancelled if there are any files under that overflow number.

The procedure for setting up an overflow is as follows:

```
* OVERFLOW Cr
DDDD, FFFF Cr
Alter Switch 1

*
```

Where D = User Number
F = Overflow Number

The procedure for cancelling an overflow is as follows:

```
*OVERFLOW Cr
DDDD, Cr
Alter Switch 1

*
```

If the information is not entered in the correct format, or a user is logged in to whom the overflow is to be assigned, or the files are not deleted from the overflow before cancellation, the system will come back with a ?.

USERS

This command lists and sorts all of the users by either User Number, Account Number, or User Name, depending on which column; 1, 2, or 3, respectively, it is sorted in.

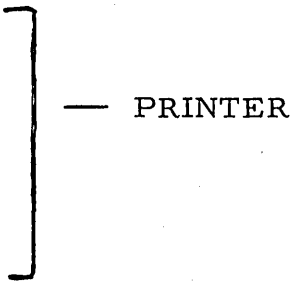
* USERS Cr

OUTPUT TO: PR Cr

SORT ON COL. (1,2, or 3): 1 Cr

3/25 17:48

1 *1 UTILITIES
2 *1 CRASH
3 *1 RAY
.
.
.
TOTAL: 0:00.00 0:00



.....END OF JOB.....

END JOB

-

POINTER

This command will indicate the last overflow number assigned, minus one. At this time, the validity of the pointer over 1000 octal is doubtful. So use the results of this command with caution.

*POINTER Cr

OVERFLOW POINTER at 1355

END JOB

-

FILES FROM TAPE TO DISC

To copy files from Tape to Disc, or from Disc to Tape, is known as sys-defining. To sys-define one must have peripheral and exec status; peripheral status in order to use the Tape drive, and exec status in order to use the "SDF..." command.

Files on a Tape have three (3) parameters:

1. File Number
2. File Type
3. File Length

The File Number is a number that designates the location of the file on the tape. These numbers are octal numbers, i.e., 1, 2, ...7, 10, ... There is no zero (0) File Number. The number of files on a Tape is only limited by the amount of Tape.

The File Type is a number that indicates what type of file:

1. 1 is a "go to" file
2. 2 is a binary file
3. 3 is a symbolic file
4. 4 is a recover file

The File Length is a number that indicates the length of the file in words., i.e., 44477, 25003, etc.

With every Tape that has files on it, there should be a file directory with it. This file directory should list all of the files and their parameters. The file directory is vital to the sys-defining.

MOUNT FILE TAPE on Unit 0, set density and put in Auto.

-ST^{C}S^{C}EX$_{\wedge}$-1 Cr

- POSITION Cr
  3

This command is used to position the Tape. It is always positioned at File 3. Though the tape is usually at file 1 when it is at the load point, sometimes this is not the case. This command is not used if there are only two files on the Tape. As stated above, the POSITION command positions the Tape at file 3. If there is no file 3, the Tape will run away and probably crash the system.

If while trying to position the tape, you get a tape wind error, TW⊏, it indicates that the tape is not on unit zero. The tape units are now hung up. They can not be used until the following steps are gone through:

Put Tape drive Unit 0.
and
-ST^{C}S^{C}REW Cr
This command will clear the system of the hang up, and position the tape at file 2.

-ST^{C}S^{C}DF$_{\wedge}$'<Tape file-name>'$_{\wedge}$AS$_{\wedge}$<File-number> , <File-Type> , <File-length> Cr

This command defines the tape file-name with the parameters after the AS. The same <Tape file-name> may be used over and over again, if it is copied to the disc before it is defined again with different parameters. If the <Tape file-name> is defined again with different parameters, the computer will type back "(ALREADY DEFINED)" after the Cr on the ST^{C}S^{C}DF command.

-COPY$_{\wedge}$'<Tape file-name>'$_{\wedge}$TO$_{\wedge}$/<File-name>/ Cr Lf
NEW FILE or OLD FILE Cr Lf

-

This command will copy the tape file, which has been defined with parameters of a file on the the tape, to a disc file.

After the second Cr the system will search the tape for the file number stipulated in the ST^{C}S^{C}DF command. When it has found the file, the system will give a carriage return and start writing the tape file to the disc file. When it is finished the system returnsan executive dash.

If the file cannot be found on the tape, this will be indicated by the tape unwinding and rewinding. To stop this, altmode out of the COPY command.

If the file is found, but it can't be read, (PE), position error, will be typed out on the TTY. (PE) will continue to be typed out until the file is read or until the COPY command is altmoded out of. If more than 5 or 6 (PE) occur, altmode out of the COPY command anyway. Try again. If (PE)'s still occur, try another tape drive. If it still doesn't work, the tape is bad or the tape units are in bad condition.

FILES FROM DISC TO TAPE

System - defining files from disc to tape is basically the same as from tape to disc, except for the copy command;

-COPY$_{\wedge}$/<File-name>/$_{\wedge}$To$_{\wedge}$'<Tape file-name>'Cr

In the SDF command, the parameters of a file on the tape to which a disc file is to be copied to is defined, and the copy command copies the disc file to that tape file.

In copying files to tape, two of the three tape file parameters must always agree with the file directory of the tape. These are the file number and the file length. The file type may change depending on the type of file to be copied to the tape.

To copy files to tape, there must always be at least two files. To create more files on a tape, do the following:

-COPY$_{\wedge}$/<file-name>/$_{\wedge}$TO$_{\wedge}$'<Tape file-name>'Cr
NEW FILE Cr

NOTE: The tape file-name has not been defined by a SDF command. The disc file is copied to the tape and a new tape file is created.

-FD: '<Tape file-name>'Cr
43, 3, 13171

This command will give the parameters of the new tape file. Enter this onto the tape file directory.

NOTE: Never use the same '<Tape file-name>' over again until the parameters of a particular file copied to tape has been determined by FD:.

SHUT

The shut command does the opposite of the answer command, though it does not selectively shut channels, but instead shuts all unused channels. If a user logs out after a shut command has been given, the channel he was using is shut.

-$ST^{C}S^{C}EX_{\wedge}$ -1 Cr

-SHUT Cr
alter B. P. switch 1

-

HANG

This command will cause the indicated channels to be disconnected from the computer. It has the same affect as if the dataphone was hung up or a log out occurred, except if a user has a $ file, his core at the time of the hang is dumped on to the $ file.

-ST^{C}S^{C}EX -1 Cr

-HANG N (or) N, N (or) N-N Cr
alter B. P. Switch 1

-

NOTE: This command only causes a Temporary Disconnect. It does not prevent the effected line (s) from being re-activated immediately.

GARBAGE FILES

Garbage files are files in a user's file directory that have gotten clobbed, or have been added to the user's file directory. A file with garbage files might look like the following:

```
-FI Cr

/DATA/          23,512
/AC/            22,1047
/PRT/           23,512
                   %@      23,23156
/SI/            22,2456
/NUT/           22,1056
<'A             330,2,703635
/QUS/           23,3675

-
```

The fourth and the seventh files are garbage files. To remove these files, determine the location of the file, i.e., the first garbage is the fourth file. Then do the following:

```
-ST^CS^CCL^4 Cr

-
```

This command will remove the garbage file that is the fourth one in the file directory. Take another files

```
-FI Cr

/DATA/          23,512
/AC/            22,1047
/PRT/           23,512
/SI/            22,2456
/NUT/           22,1056
<'A             330,2,703635
/QUS/           23,3675

-
```

The other garbage file is now sixth in the file directory. To remove it, do the same as with the previous one.

PROGRAM TITLE: DISC LOAD 22

ACC./USER NAME:

RESPONSIBLE PROGRAMMER: V. VAN ULEAR

STATUS REQUIRED

OPER.	EXEC1	EXEC2	PERH.	SUBSYS	ARPAS-DDT

PURPOSE: TO LOAD THIS DISC FROM A PREVIOUS DISC DUMP

SET UP INSTRUCTIONS: MOUNT DISC LOAD/DUMP PROGRAM ON UNIT 0. CLEAR REGISTERS AND GO TO RUN. PLACE B.P. SW 1 down and MAG TAPE. MOUNT THE TWO TAPES TO LOADED FROM ON THE UNIT THAT IS WRITTEN ON THE TAPE LABEL. PLACE BOTH TAPES IN AUTO.

OPERATING INSTRUCTIONS:

1. GO TO IDLE
2. CLEAR REGISTERS
3. LDA 5040
4. GO TO RUN (COMPUTER WILL HALT AT 2010101 in C)
5. CLEAR HALT (toggle RUN)
6. TOGGLE B.P. SW 2, 3, 4

ERROR RESTART PROCEDURE: TO STEP 1

REVISION DATE: REVISED BY:

PROGRAM TITLE: DISC DUMP # 2

ACC./USER NAME:

RESPONSIBLE PROGRAMMER: V. VAN VLEAR

STATUS REQUIRED

OPER.	EXEC1	EXEC2	PERH.	SUBSYS	ARPAS-DDT

PURPOSE: TO TAKE A DISC DUMP TO KEEP A BACKUP OF THE FILES CURRENTLY ON THE DISC.

SET UP INSTRUCTIONS: MOUNT DISC LOAD/DUMP PROGRAM ON UNIT 0. CLEAR REGISTERS AND GO TO RUN. PLACE B.P.SW 1 UP. AND MAG FILL. MOUNT THE TWO TAPES TO BE DUMPED ON UNIT 0 AND UNIT 1 and PLACE [illegible] BOTH OF THEM IN AUTO. THE FIRST 8 DISCS WILL GO ON TAPE, UNIT 0. THE SECOND 8 DISCS ON TAPE, UNIT 1.

OPERATING INSTRUCTIONS:

1. GO TO IDLE AND CLEAR REGISTERS
2. DSN [illegible] 240
3. GO TO RUN (COMPUTER WILL HALT AT 2010101 IN C)
4. Clear HALT (Toggle RUN)
5. TOGGLE B. P. SW. 2, 4, 2 (IN WHAT ORDER)

ERROR RESTART PROCEDURE: TO STEP 1

REVISION DATE: REVISED BY:

ERROR HALTS AND CORRECTIVE ACTION

DISC DUMP/LOAD #2

C Reg. Flag	Description	Corrective Action
2010101	Dump or Load ready to start	
2000001	LOAD only: Tape Read Retried 10 times	(a) Toggle Run Sw., tries 10 more times. (b) Set Console Sw.#4, Toggle Run Sw. Accepts tape record as read.
2000003	LOAD only; Disc Address read from tape is not valid for tape number.	If first read on tape then tape might be on wrong handler number, otherwise bad read. (c) Toggle Run Sw., Rereads tape. (d) Set Console Sw.#4, toggle Run Sw. Record not written on disc, next tape record read.
2000004	SKS 14000 error on disc (W Buffer not going ready)	WONT WORK WRONG UNIT SET (e) Toggle Run Sw., Tries disc instructions again. (f) Set Console Sw.#4, toggle run. DUMP: Accepts disc record (1 page) as read and writes on tape whatever is currently in disc buffer. (Probably lost 1 page of data) LOAD: Record is left on disc as written, probably with errors. (Depending upon type of disc failure.
2000005	SKS 10026 error on Disc (Seek or Search Time Error)	Take actions (e) or (f) above.
2000006	SKS 11026 error on disc (Disc Controller Error)	Take actions (e) or (f) above.
2000007	SKS 11000 error on disc (W Buffer Error, includes Disc Read Errors)	Take actions (e) or (f) above.
2077777	DUMP or LOAD complete on two tapes if 16 disc dump, on 1 tape if 8 disc dump. (Ready to proceed with 2nd two tapes if 32 disc dump)	

WIM error (P reg. at 454) (LOAD ONLY) can be bypassed by placing run switch in halt, pressing START and branching to 446.

P REGISTER LOOPS

626/627	Error in erasing tape.
644/646	Error in backing up tape on a retry.
656/660	Tape not ready.
661/662	W Buffer not ready on tape instruction.

1 page of 2

PROGRAM TITLE: TAPE COPY

ACC./USER NAME: O.I. OPERADOR

RESPONSIBLE PROGRAMMER: ANN

STATUS REQUIRED

(OPER.) EXEC1 EXEC2 PERH. SUBSYS ARPAS-DDT

PURPOSE: COPY FROM OR TO TAPEFILES.

SET UP INSTRUCTIONS: MOUNT TAPE

OPERATING INSTRUCTIONS:

1. POS
   NN
2. DF 'x' AS NN, X, NNNN *
3. COPY /x/ TO 'x' OR COPY 'x' TO /x/
   NEW FILE OR OLD FILE

* THIS IS IF YOU ARE USING A TAPE WITH DEFINED FILES

ERROR RESTART PROCEDURE: ALT MODE

REVISION DATE: _____ REVISED BY: _____

PROGRAM TITLE: RELEASE OF ARPAS & DDT

ACC,/USER NAME: @1 : OPERATOR OR @3 : TCSCSC

RESPONSIBLE PROGRAMMER:

STATUS REQUIRED

(OPER.) EXEC1 EXEC2 PERH. SUBSYS ARPAS-DDT

PURPOSE: RELEASE ARPAS & DDT FOR ALL USERS ON THE SYSTEM.

SET UP INSTRUCTIONS: —

OPERATING INSTRUCTIONS:

1. ENABLE, ARPAS C.R.

SYSTEM RESPOND WITH EXEC MODE.
AFTER TYPING: ARPAS DDT

ERROR RESTART PROCEDURE:

REVISION DATE: REVISED BY:

PROGRAM TITLE: LETTER

ACC./USER NAME: @L; OPERATOR OR @S:T[illegible]

RESPONSIBLE PROGRAMMER: VERNE

STATUS REQUIRED

OPER. EXEC EXEC2 PERH. SUBSYS ARPAS-DDT

PURPOSE: TO PUT A LETTER INTO THE SYSTEM FOR ALL USERS.

SET UP INSTRUCTIONS: GO /OPER/ PREVIOUS TO THIS IN EXEC MODE TYPE: LETTER C.R. IF IT RESPONDS WITH LETTER OFF. GO OPER, IF IT RESPONDS WITH: LETTER @N.. TYPE LETTER C.R. AGAIN AND YOU GET: LETTER OFF

OPERATING INSTRUCTIONS:

1. COUNT LETTER
   - 1. NUMBER
   - 2. NUMBER
   - 3. 0
   - 4. 0
   - 5. 0
   - 6. 0
2. LETTER C.R.
   LETTER NR. 1-6 : N C.R

ERROR RESTART PROCEDURE: ALT MODE GO OPER

REVISION DATE: _____ REVISED BY: _____

PROGRAM TITLE: LETTER

CONT. OPERATING INST:
TYPE TEXT OF LETTER. D^C *

3. IN EXEC TYPE: LETTER$_{\wedge}$N (FOR LETTER NR)
TEXT OFF LETTER (ERROR CHECK)

4. LETTER C.R.
LETTER ON

* WHEN TYPING TEXT OF LETTER, A SHIFT 8 WILL DELETE PRECEDING CHARACTER. A D^C WITHOUT ANY TEXT AFTER TYPING LETTER NR. WILL REMOVE A LETTER.

IF, WHEN SYSTEM RESPONDS WITH LETTER OFF AND $\wedge$ THE SYSTEM DOES NOT RETURN TO EXEC. TYPE ALT MODE KEY. THIS IS A BUG IN THE PROGRAM YET. ALSO PUT IN AN EXTRA C.R. AFTER $\wedge$ TYPING NR.

REVISION DATE _____ REVISED BY _____

EXEC COMMANDS (System 1.86)

All Users

LOGOUT	Allows user to logout
WRITE FD	Writes File Dir. on disc
RENAME	Renames a file
DATE	Types date and time
KILL PROGRAM	Kills program relabelling only
RESET	Returns all of user's memory
COPY	Copies file to file
FILES	Types file directory
FD FOR	Types selected file dir. entry
GO TO	Goes to a "GO TO" (type 1) file
PLACE	Places a "SAVE" type program (type 1) in core
SAVE	Saves program, creates GO TO or type 1 file
BRANCH	Branches into a program
DELETE	Deletes a file
TIME	Types real time used (and computer time*)
STATUS	Types user's relabelling status
MEMORY	Types unused user's memory
"	Causes typing to be ignored by EXEC
DUMP	Dumps all program, saves status
RECOVER	Recovers from a DUMP file (type 4)
CONTINUE	Continues running a Sub-system
RELEASE	Releases a subsystem
PMT	Prints a users Program Memory Table
EXIT	Allows a user to LOGOUT without writing File Dir.
SIZE	Sets Users Machine size
MAIL	Types all Mail in user's mail box.
SEND TO	Allows user to put letter in Mail box

Users with Sub-System Status and above.

USERS	Types number of users on system
WHERE IS	Gives teletype number for a user
WHO IS ON	Types users on system by Account and name
REWIND	Rewinds tape, resets tape logic
RLT	Releases tape
STN	Sets tape no.
PTN	Types tape no.
SETEXEC	Sets user status
POSITION TAPE	Positions tape
TAPE POSITION	Types current tape position
DF	Allows a file directory entry to be set up.
REMOVE FILE	Removes file from directory (without deleting)
PSP	Types error counters, etc.
CREATION	Types file directory with Creation Date & Access Count
LFCRE	Types Creation Date & access count of selected file
STORE	Stores a file on Mag. Tape (in backup format)
RETREIVE	Retrieves a file from mag. tape
DIRECTORY	Types File Directory for files in Backup Format

OPERATOR PROGRAM COMMANDS

HELP	Types list of commands
MAIL COUNT	Provides a list of Mail originators and total number of addressees.
COPY MAIL	Allows operator to copy selected mail to a file.
CANCEL MAIL	Allows operator to cancel mail by number
MAIL GARBAGE	Removes holes and null entries from Mail List
UAD	Ouputs User/Account Directory
LENGTH	Computes length of files by acoount
TIME	Provides time used by user number
SET DAY	Validates a user for a whole day
RESET TIME	Same as TIME but also clears time.
SET HOUR	Validates a user for selected hours
FILES	Outputs complete or selected File Directories
CLEAR FILE	Clears a file directory
SIZE ACCOUNT	Computes and provides maximum size of files by account.
ACCOUNT	Sets up or changes Account parameters
NAME	Sets up or changes a user's parameters
CANCEL ACCOUNT	Cancels an account directory
CANCEL NAME	Cancels a user entry in the A.U.D.
*OVERFLOW	Allows assignment of an overflow directory
*MAP	Builds system bit-map
GARBAGE	Removes garbage from overflow file directory
POINTER	Indicates current location for a new overflow file directory.
USERS	Provides a sorted list of users on the system
COUNT LETTERS	Counts the number of users YET to receive a all broadcast letters
REMOVE LETTER	Removes a broadcast letter from the system
LETTER	Allows a broadcast letter to be created.
COPY RECORDS	Allows the accounting records to be copied to a file.
CLEAR RECORDS	Same as COPY RECORDS but also clears records.

EXEC COMMANDS (cont.)

Users with Operator or System status

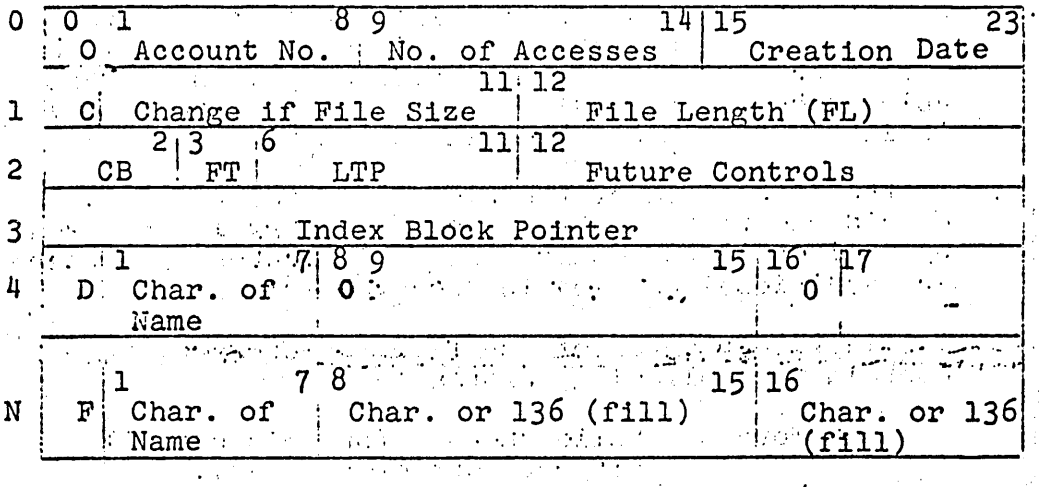
SHUT DOWN	Starts system shut down
UP	Cancels shut down
HANG UP	"Hangs up" selected teletype phone lines (DSS)
ANSWER	Ansers (or enables) Data subset
ACCOUNTING	Controls accounting to paper tape
LETTER	Types broadcast letters
ABT	Aborts tape operation (halts runaway)
GFD	Gets another user's file directory
ENABLE	Enables a subsystem group
DISABLE	Disables a subsystem group
LOOK	Looks at real core locations
SYSLD	Allows load from disc directly into user's core

System commands

RSMT	Reads in from RAD a SMT Page
SYSDP	Allows core to be dumped directly on disc.

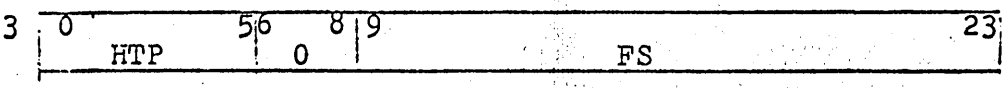
FILE DIRECTORY FORMAT ON DISC

1 Entry (Disc File)

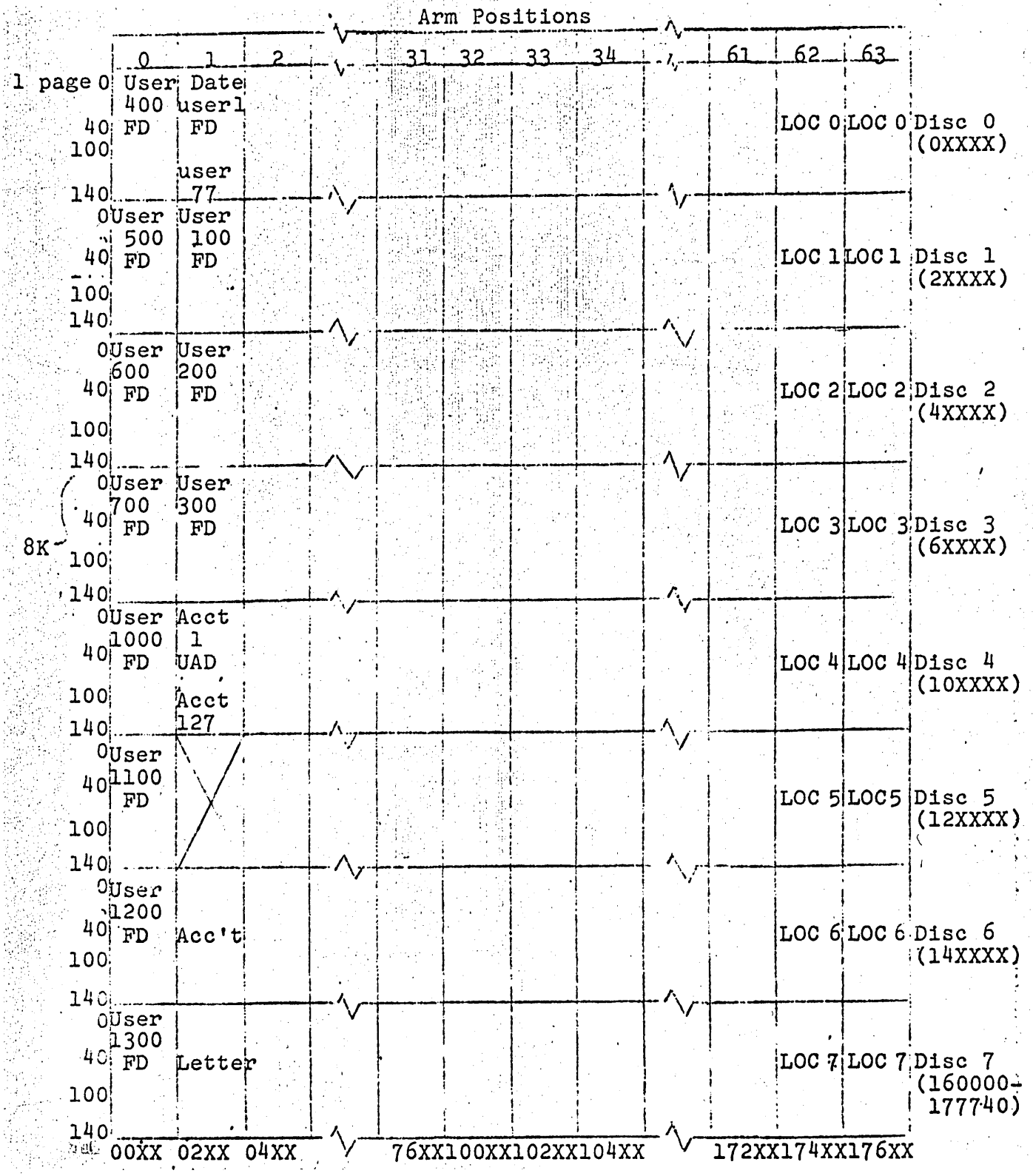


FT = File Type
LTP = Low Order Tape position
HTP = High Order Tape position
FS = Tape File Size
FL = File Length for disc Files
C = Change in file length (file length no longer valid)
CB = File Control Bits, 0=Tape file 4 = IGNORE FILE
2=Disc file
F = End of Entry Flag (1)

If Tape File, word #3 =



DISC MAP


Arm Positions
0 1 2 31 32 33 34 61 62 63
1 page
0
40
100
140
8K
User 400 FD
Date user1 FD
user 77
User 500 FD
User 100 FD
User 600 FD
User 200 FD
User 700 FD
User 300 FD
User 1000 FD
Acct 1 UAD
Acct 127
User 1100 FD
User 1200 FD
Acc't
User 1300 FD
Letter
LOC 0 LOC 0 Disc 0 (0XXXX)
LOC 1 LOC 1 Disc 1 (2XXXX)
LOC 2 LOC 2 Disc 2 (4XXXX)
LOC 3 LOC 3 Disc 3 (6XXXX)
LOC 4 LOC 4 Disc 4 (10XXXX)
LOC 5 LOC5 Disc 5 (12XXXX)
LOC 6 LOC 6 Disc 6 (14XXXX)
LOC 7 LOC 7 Disc 7 (160000-177740)
00XX 02XX 04XX
76XX100XX102XX104XX
172XX174XX176XX

After a crash is found + the lines are entered -

-LEN
- REC /MONST/
- PL /CRASH/
- CON
DDT

MONCR/		=	NOT MON 0
CLINT/		=	
MCRA/		=	
MCRB/		=	
MCRX/		=	
MCR0/		=	
MCRC/		=	
BSX/		320	(EXEC CRASH)
SSO1/		=	
SSO2/		=	
SSO3/		=	
JOB[	X		0 IF PHANTOM CRASH
UNIT[	Y		
FUNC[			
WORD3A/[			
RRL1[			
RRL2[			
RRL3[			
PWFL[			
:			
:			
NB1[2[			
error addressed[			
BLK31/		=	0
IDCL1/			-1
NRCL/			-1
PUCTR/		SHOULD BE 45	
PWFI/			0

To look at the bad address in the running system follow this procedure:

- R /MONDST/
- CON

DDT

ADDR2	=	W
RD2	=	X
TNIT2L	=	Y
TFIT2L	=	Z

- LOOK W 10
- LOOK X 10
- LOOK Y 6
- LOOK Z 6

All crashes on this system 1.85 V10 should stop with P = 21176. If this is not the case, record the P counter and the name of the crash and call me.

Error counters and error addresses

For read errors and disc errors we have
[illegible] of the addresses at which the hardware
failed. And unexpected ON and OFF teletype interrupts
We have a table of the channel that failed.
Any of the counters indicate an error. Type
out the appropriate table to find out where
the hardware failed.

Read errors: RDA (10 words)

Disc errors: IDER (10 words)

TTY ON interrupts: TNIAL (6 words)

TTY OFF interrupts: TFIAL (6 words)

There are extra ON interrupts if
TNIACTR > TNIAC. (TN > TNA)

There are extra, OFF interrupts if
TFICTR > TFIAC. (TF > TFA)

These errors may be checked in the running
system by typing -PSP.

APPENDIX A

GENERAL DESCRIPTION OF THE COMBINED FILE DIRECTORY

1. A user may have one or two file directory blocks on the disc; the second block is an overflow block. Each block consists of 128 words containing a variable number of file directory entries. Each entry is in the format pictured in (d).

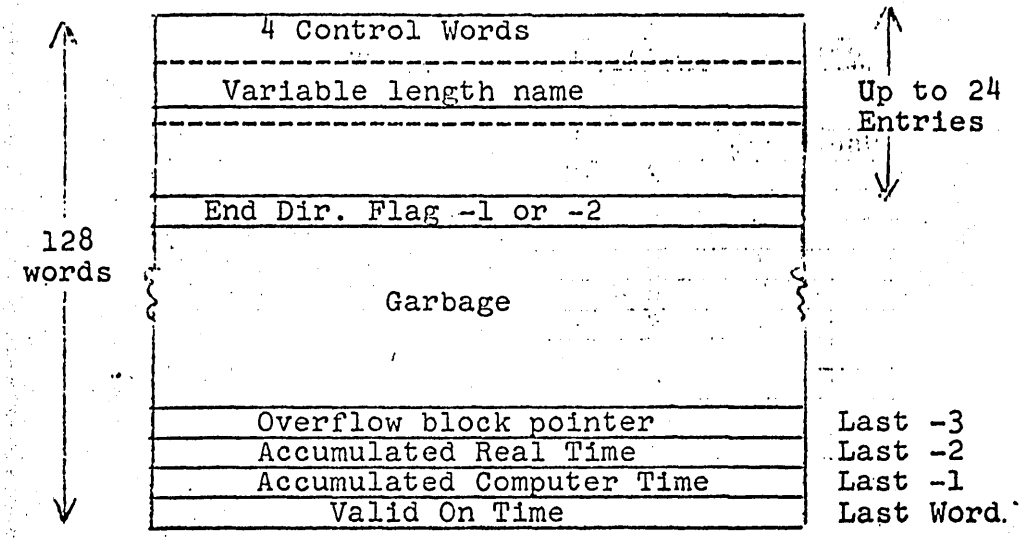
2. If the first word of the block is zero, the block considered to be empty. The last entry is followed by a -1 or -2 word where the -2 indicates that there are additional entries in the overflow block.

3. The last four words of the file directory block contain the following information:

Last Word	Valid on-time for this user (1 bit per hour of the day).
Last Word -1	Accumulated computer time used.
Last Word -2	Accumulated real time used.
Last word -3	Overflow block pointer.

4. In the case of an overflow block, the last three words are zero, and the overflow block pointer is a backward pointer to the first file directory block.

FILE DIRECTORY BLOCK



LOCATION OF FILE DIRECTORIES
(on 16 DISC system)

(All numbers in Octal)

A File Directory occupies two sectors or 200_8 words on the disc.

The Disc Location or address is composed of two parts; the first part is made from the low order two digits of the User no.

DOUBLE THE TWO LEAST SIGNIFICANT DIGITS TO OBTAIN PART 1 of the address.

OBTAIN PART TWO OF THE ADDRESS FROM THE FOLLOWING TABLE, USING THE REMAINING MOST SIGNIFICANT DIGITS:

0	200	(disc 0, arm position 1)
100	20200	(disc 1, arm position 1)
200	40200	(disc 2, arm position 1)
300	60200	(disc 3, arm position 1)
400	0	(disc 0, arm position 0)
500	20000	(disc 1, arm position 0)
600	40000	2
700	60000	3
1000	100000	4
1100	120000	5
1200	140000	6
1300	160000	7
1400	200000	10
1500	220000	11
1600	240000	12
1700	260000	13
2000	300000	14
2100	320000	15
2200	340000	16 } (overflow user numbers,
2300	360000	17 } not to be assigned)

EXAMPLES:

User number: 243
Double 43 += 106
from table, 200 = 40200
Disc address = 40306

User number: 2367
Double 67 = 156
from table, 2300 = 360000
Disc address + 360156

IMPORTANT DISC ADDRESSES
(16 Disc system)

Last user number: 2377 Disc adr.: 360176

Overflow user numbers start at 2100
Lowest overflow pointer without garbage collection: 2140

Accounting records: 140200
Account/User directory: 100200 (Disc 4, arm pos 1)
Broadcast Letter bit map: 160200 (allows 2048_{10} users,last=3777)
Broadcast Letters: #1 160210
#2 160212
#3 160214
#4 160216
#5 160220
#6 160222
(Maximum size of letter,383 characters)
Note: current letter bit map programs setup for maximum number of users of 1023_{10} which allows a LAST POSSIBLE USER NO. of 1777.)

Mail Box List: 120200
Mail Box mailæ: 120240
Maximum size of mail: 240 chars
(packed 3 chars. per word,fits into 64 words)

Future Expansion:

Account/User Directory for Accounts from P through Z and overflow goes at location: 200200
Mail box List moves to 160240 and additional mail at 160300.

DDT

TIME-SHARING DEBUGGING SYSTEM

REFERENCE MANUAL

L. Peter Deutsch

Butler W. Lampson

University of California, Berkeley

Document No. 30.40.10

Issued March 25, 1965
Revised September 3, 1965

Contract SD-185

Office of Secretary of Defense

Advanced Research Projects Agency

Washington 25, D.C.

TABLE OF CONTENTS

1.0. General

DDT is the debugging system for the SDS 930 Time-Sharing System. It has facilities for symbolic reference to and typeout of memory locations and central registers. Furthermore, it permits the use of literals in the same manner as in the assembler. It can also insert breakpoints into programs, perform a trace, and search programs for specified words and specified effective addresses. There is a command to facilitate program patching. Finally, DDT can load both absolute and relocatable files in the format produced by the assembler.

The system has a language for communication between DDT and its users. The basic components of this language are symbols, constants, and commands.

1.1. Symbols

A symbol is any string of six or fewer letters and digits containing at least one letter. All opcodes recognized by the assembler are built-in symbols. Other symbols are ;1, ;2, ;A, ;B, ;F, ;L, ;M, ;Q, ;X, and dot. Their meanings are explained below. Every symbol is given a value when it is introduced into the system. This value is a 24-bit integer; For most symbols it will be either an address in memory or the octal encoding of an operation code. Note that DDT makes no distinction between opcodes and symbols.

Examples:

ABC
AB124
12XYZ

The following are not symbols:

ABCDEFG
AB*CD

Symbols may be introduced to DDT in two basically different ways:

(A) They may be written out by the assembler and read in from the binary program file by DDT.

(B) They may be typed in and assigned values during debugging using the @, :, and < commands.

It is possible for the value of a symbol to be undefined. This may occur if a program is loaded which references an external symbol not defined in a previously loaded program. It may also occur if an undefined symbol is typed in an expression. In general, undefined symbols are legal input to DDT except when their values would be required immediately for the execution of a command. Thus, for example, the ;G command could not have an undefined symbol as its argument.

Undefined symbols may become defined in several ways. They may be defined by EXT directives in the assembler and read into DDT as part of a binary program. Alternatively, they may be defined by one of the three symbol definition commands available in DDT. When the definition occurs, the value of the symbol will be substituted in all the expressions in which the symbol has appeared.

If DDT gives a carriage return without a line feed after typing out the contents of a register, it means that the register contains an undefined symbol. The register is closed at once so that its contents cannot be erroneously changed.

The only restriction on this facility is that, as for the assembler, the undefined symbol must be the only thing in the address field of the word in which it appears.

Warning: If the program alters words containing undefined symbols, unpredictable errors will result. This is the only way in which the programmer can get into serious trouble by using an undefined symbol. Other incorrect uses of undefined symbols will be detected by DDT and will result in an error comment.

DDT keeps track of an undefined symbol by building a pointer chain through the address of the words containing the symbol. Thus, suppose that the symbol A is undefined and appears as follows

```
S1    LDA    A
       .
       .
S2    STA    A
       .
       .
S3    MRG    A
```

and nowhere else in the program. After loading, the symbol table entry for A will contain a flag indicating that it is undefined and a pointer to S1. The above locations will contain:

```
S1    LDA    S2
       .
       .
S2    STA    S3
       .
       .
S3    MRG    0
```

When the symbol is defined, DDT goes through the pointer chain and fills in the value. It recognizes the end of the pointer chain by a 0 address.

From this description it should be obvious what will happen if the pointer chain is destroyed. A probably consequence is

that a search down the pointer chain will not terminate. DDT does such searches whenever it prints an address, and also during some effective address searches. If the chain it is searching has more than 2048 links, it will print the symbol at its head, followed by (U) and continue. Fixing up an undefined symbol pointer chain which has been clobbered is an exercise which we leave to the reader.

1.2. Block Structure

A limited facility called the block structure facility is provided to simplify the referencing of local symbols which are defined in more than one program. Note that DDT's block structure has only a tenuous connection with the block structure of ALGOL. The block structure of a program is organized in the following manner: Every binary program file loaded by DDT constitutes a separate block. In addition, there is an intrinsic block called block zero. Any symbol input to DDT has a block number associated with it. It also has a type; it may be external or local. All operation codes and all symbols defined by < are external block zero symbols. When a binary file written by the assembler is loaded by DDT, it defines a new block, and all symbols defined during the assembly and written on the binary file are associated with that block. Any symbols which were declared to the assembler to be external are of external type. All other symbols are of local type.

1.3. Literals

A literal is a special kind of symbol recognized by DDT. The two characters ' =' signal the beginning of a literal, which is terminated by any of the characters which ordinarily terminate an expression. The literal is looked up in the literal table which is generated when the program blocks are loaded. If it is found, the address which has been assigned to it is the value of the symbol. If it does not appear in the literal table, a location is assigned to it at the current value of ;F and the address of this location is returned as the value. ;F is increased by 1. Exception: In patch mode, literals are saved and defined when the patch is completed since, otherwise, they would interfere with the patch.

When DDT types out a symbol whose value is an address in the literal table, it will type out in the same format in which it would be input; that is, as = followed by the value of the literal.

Do not use 0 as a literal.

1.4. Constants

A constant is any string of digits, possibly preceded by a % sign. The number represented by the string is evaluated, truncated to 24 bits and then used just like the value of a symbol. The radix in which the evaluation is performed is controlled by the O-D mode. The ;O command sets this radix to octal, ;D sets it to decimal. The preceding % changes the radix for the immediately following number.

The radix in which constants are typed out by DDT is also determined by the setting of the O-D mode. Further flexibility

is provided by the ;R command, which sets the radix to any specified value. If the radix is not octal, % causes the following number to be taken as octal.

1.5. Commands

A command is an order typed to DDT which instructs it to do something. The commands are listed and their functions ~~explained~~ given in ~~the table below.~~ appendix A

1.6. Expressions

An expression is a string of numbers or letter connected with blanks, + signs, and - signs. The first symbol or number may be preceded by a minus sign. Blank acts like plus, except that the following operand is truncated to 14 bits before being added to the accumulated value of the expression. The value of an expression is a 24-bit integer. A single symbol or constant may be an expression.

Examples:

	LDA	has the value	7600000	
	LDA 10	has the value	7600010	if the radix is octal
	LDA %10	has the value	7600012	
If	SYM is a symbol with the value		1212,	then
	SYM	has the value	1212	
	SYM 10	has the value	1222	
	LDA SYM	has the value	07601212	

If this last expression were put into a memory register and later executed by the program the effect would be to load the contents of SYM, register 1212, into the A register.

When DDT types out expressions, two mode switches control the format of the output. The C-S mode determines whether the

value will be typed as a constant (C), or as a symbolic expression (S). In the latter case, the opcode (if any) and the address will be put into symbolic form. If there is no recognizable opcode, the value will always be in constant form regardless of the setting of the C-S mode. A zero opcode (HLT) is not printed. The setting of this mode is controlled by the ;S and ;C commands. It is also controlled locally by the [and] commands.

The R-V mode controls the format in which addresses are typed. DDT types addresses when asked to open the previous or the next register, when it reports the results of word and address searches, and on breakpoints. In relative (R) mode, addresses are typed in symbolic form, i.e., as the largest defined symbol smaller than the address plus a constant if necessary. If the constant is bigger than 200 octal or, if the value of the symbol is less than the first location of the program, the entire address is typed as a constant. In absolute (V) mode, addresses are always typed as constants. The setting of this mode is controlled by the ;R and ;V commands.

1.7. The Open Register

One other major ingredient of the DDT language is the open register. Certain commands cause a register to be "opened". This means that its contents are typed out (except in enter mode, for which see the \ command), followed by a tab. Any expression the user types will then be inserted into

the open register in place of its current contents. After this insertion the register is closed at once. Note that the string LDA ABC= is a command, and does not cause LDA ABC to be entered into the current open register. The current location is given by the symbol "." (dot) which always has as its value the address of the last register opened, whether or not it is still open.

Note:

(1) Comma and star (for indirect addressing) may be used in expressions as they are used in the assembler; e.g., LDA* 0,2 has the value 27640000.

(2) DDT will respond to any illegal input with the character ? followed by a carriage return, after which it will behave as if nothing had been typed since the last tab or carriage return. The command ? also erases everything typed since the last tab or carriage return.

1.8. Memory Allocation and DDT

DDT may cause the system to assign memory to the user for use either by the system or by the user's program. System memory is used to hold the symbol table, which starts in block 7 and grows down. The symbol table contracts at the end of each load of a binary file and when symbols are killed; this contraction may cause memory to be released.

DDT grabs program memory only when it is required for loading a binary file or when a ;U (execute) command is given,

and the value of ;F is such that a new block is needed to hold the instruction to be executed. For executing an instruction, DDT requires location ;F, ;F+1 and ;F+2. Memory is never grabbed for examination of a register. Attempts to open locations not assigned will cause DDT to type ?. This means that upon initial entry to DDT no registers are available for examination. The easiest way to obtain memory for typing in a program is to execute a NOP, thus: NOP;U. This assigns a block containing the initial value of ;F, which is 200.

If an attempt to grab memory leads to a trap, DDT types (M) and abandons whatever it is doing. This can happen if the machine size is exceeded (Cf. Section 1.3. of the Executive Manual.).

2.0. DDT Commands

In the following descriptions of DDT commands, the string < E > will be used to denote an arbitrary expression which may be typed by the user. Unless otherwise indicated, the value of this expression is truncated to 14 bits before it is used by DDT.

2.1. Register Opening Commands

/ < E > / opens the register addressed by the value of the expression. DDT will give a tab, type an expression whose value is equal to the contents of the register, give another tab and await further commands. The precise form of the expression typed in this and most other commands is dependent on the setting of the S-C and R-V modes. If the user types in an expression, DDT will insert its value into the register. Typing another command closes the register, unless it is a type value or symbol definition command. Note that in a command that requires a preceding expression, the expression is regarded as part of the command and would not, for instance, be inserted into the open register. If another / is given as the next command with no preceding expression the contents of the register addressed by the expression typed by DDT are typed out. A further / repeats this process. Note, however, that the original register opened remains the open register; any changes made will go into that register.

carriage return

This command does not necessarily have any effect. If the specified conditions are present, however, any of the following actions may occur:

(1) If there is an open register, the register is closed.

(2) If DDT is in enter mode, it leaves it.

(3) If DDT is in patch mode, the patch is terminated (for a fuller description of this effect, see the patch command).

] This command has the same effect as /, except that the contents of the register opened are typed in symbolic form regardless of the setting of the S-C mode.

[This command has the same effect as /, except that the contents of the register opened are typed in constant form regardless of the setting of the S-C mode.

$ This command has the same effect as /, except that the contents of the register opened are typed as a signed integer regardless of the setting of the S-C mode.

" This command acts like /, except that the register constants are typed in ASCII.

line feed This command opens the register whose address is the current location plus one, i.e., the register after the one just opened. The output of DDT on this command is carriage return, register address (format controlled by the R-V mode), /, tab, value of contents, tab.

↑ This command opens the register whose address is the current location minus one, i.e., the previous register. The output is the same as for the line feed command.

Example:

```
ABC/      LDA     ALPHA                    (line feed)
ABC+1/    STA     BETA     STA    GAMMA    (line feed)
ABC+2/    LDB     DELTA    ↑
ABC+1/    STA     GAMMA
```

(This command opens the register whose address is the last 14 bits of the value of the last expression is typed. The output is the same as for line feed.

\ This command is the same as /, except that the contents of the register are not typed. DDT goes into enter mode, in which the contents of registers opened by line feed, ↑, or (are not typed. Any other command causes DDT to go out of enter mode. In particular, carriage return has this effect. When a register has been opened with \, DDT thinks that it has typed out the contents. The type value commands will, therefore, work on the contents of the register.

The type register in special mode characters [,], $ (type as a negative integer), " (type in ASCII) are also preserved by line feed, up arrow and (.

;\ This command suppresses typeout of register addresses during line feed, up arrow and (chains.

;/ Cancels the ;\ command.

2.2 Type Value Commands

= This command types the value of the last expression typed in constant form. It may appear in the form < E > =, in which case the value of the expression is typed. Otherwise, the expression referred to is the one most recently typed, either by DDT or by the user.

This command types the value of the last expression typed as a signed integer.

← This command types the value of the last expression typed in symbolic form.

' This command types the value of the last expression typed as three 8-bit characters.

@ This command types the address part of the last expression typed in symbolic form. If, for instance, the program has executed BRM X,

then X\@ will cause DDT to print the address of the BRM.

Example:

```
LDA=        7600000
LDA 10=     7600010
LDA ←       LDA
7600000 ←   LDA
-1=         77777777
-1#         -1
77777777    -1
10221043'   ABC
```

2.3 Symbol Definition Commands

: This command defines the value of the preceding symbol to be the current location. The symbol is local to the block which is primary when the : command is given.

@ This command defines the value of the preceding symbol to be the address of the last expression typed by DDT or the user. The symbol is local to the block which is primary when the command is given.

< > < (symbol) > defines the symbol to have the value of the immediately preceding expression, which must be typed by the user. The symbol is global and is associated with block zero.

2.4 Mode Changing Commands

% This command causes the immediately following number to be taken in the radix opposite to the normal one which is set by the O-D mode.

" This command generates a constant whose value is the octal encoding of the next three characters typed in the standard internal code. These characters will not be recognized as commands no matter what they are. Thus: "ABC=10221043.

;D (DECIMAL) This command changes the O-D mode to decimal. This mode determines the radix in which constants are typed out and read in.

;O (OCTAL) This command changes the O-D mode to octal.

< E > ;R (RADIX) sets the radix to the value of the expression.

;C (CONSTANT) This command changes the S-C mode to constant. This mode determines the format in which the values of expressions are typed out.

;S (SYMBOLIC) This command changes the S-C mode to symbolic.

;H (HOLLERITH) This command causes expressions to be typed out as described under the ' command. It may be reversed by ;C or ;S.

;R (RELATIVE) This command changes the R-V mode to relative. This mode determines the format for the output of addresses, both in symbolic expression and when generated by line feed and ↑.

;V (ABSOLUTE) This command changes the R-V mode to absolute.

2.5. Breakpoint Commands

! (BREAKPOINT) < E > ! sets the breakpoint at the address given by the value of the expression. The effect is that if the program executes the instruction at this address control returns to DDT, which will print the address and the contents of the A, B, and X registers and await further commands (see ;N, and ;P). The break occurs before execution of the instruction in the breakpoint location. The breakpoint is removed by a ! with no preceding expression.

;P (PROCEED) This command restarts the program after a break. The program executes the instruction at the breakpoint and goes on from there. The breakpoint is not removed unless this is

specifically done by ! so that, if the program arrives at this location again, another break will occur. If ;P is preceded by an expression, another break will not occur until the instruction at the breakpoint has been executed that many times.

;N (NEXT) This command executes the next instruction after the breakpoint and breaks again. The breakpoint is moved to the location to which the program will go after the second break. This program provides a trace facility in that repeated executions of ;N will provide a running print out of the contents of the significant internal registers, instruction by instruction. The function is essentially the same as that of the step switch on the console. If ;N is preceded by an expression, that many instructions will be executed before the next break occurs.

A ;N command follows the flow of control in the user's program. In particular, it will trace the execution of users' POPs. The execution of SYSPOPs, however, is not traced. In other words, a SYSPOP such as FAD (floating add) is regarded as one instruction by ;N.

When a proceed (;P or ;N) command is given, the following sequence of events takes place:

1) DDT computes the two locations to which the instruction being proceeded from may go (depending on whether or not it skips) and inserts BRS 10 instructions there, preserving the old contents.

2) Control is transferred to the location from which the proceed occurs. The instruction executes and the program proceeds to the next instruction, which because of step 1 will be a BRS 10 which returns control to DDT.

3) The two locations altered in step 1 are restored, the location at which the breakpoint (if any) was put is replaced with a BRS 10, and control is transferred to the next location of the program, which will be the one from which the BRS 10 of step 2 occurred. The program then executes until it arrives at the breakpoint location, which contains the BRS 10 inserted at the beginning of step 3.

From this description it should be clear that attempts to proceed through certain instructions will lead to disaster, and also that breakpoints which are encountered when the program is running in a fork will not do the right thing. In some cases, attempts to proceed through unreasonable instructions will cause the error comment

$ > >

but this cannot be counted on.

2.6. Input Commands

< E > ;Y causes DDT to give a tab and await a file name. On the specified file it expects to find a binary program. If the program is absolute it is read in. If it is relocatable it is read in and relocated at the location specified by the

expression preceding the ;Y command. If the expression is omitted, relocatable loading commences at location 200_8 and continues by beginning each program in the first available location after the preceding one. After reading is complete, the first location not used by the program is typed out. Any local symbols on the binary file are ignored.

;T This command is identical to ;Y except that it also reads symbols from the tape and adds them to DDT's symbol table. Any symbols on the tape will be recognized by DDT thereafter. Any literals used in the program will also be recognized. Furthermore, all the literals on binary files read in before the first ;E command is given will be consolidated into a single table. Duplicate space for identical literals will not be assigned in memory. Identical literals in different blocks will be assigned the same memory location when the ;E command is given.

The following two points should be noted in connection with ;Y and ;T commands.

1) The use of an expression before ;T or ;Y when the file is absolute is in error.
2) The block read in becomes the primary block.

2.7. Search Commands

;W (WORD SEARCH) < E > ;W searches memory between the limits ;1 and ;2 for words which match the value of the preceding expression when both are masked by the value of ;M. The addresses and contents of all such words are typed out.

;E (EFFECTIVE ADDRESS SEARCH) < E > ;E searches memory between the limits ;1 and ;2 for effective addresses equal to the value of the expression truncated to 14 bits. Indexing, if specified, is done with the value of X saved by DDT. Indirect address chains are followed to a depth of 64. The addresses and contents of all words found are typed out.

2.8. The Patch Command

) < E >) causes a patch to be inserted at the address specified by the value of the expression. DDT inserts in this location a branch to the current value of ;F. When the patch is done, ;F is updated. It then gives a carriage return and a) and waits for the user to type in the patch. Legal input consists of a series of expressions whose values are inserted in successive locations in memory. Each of these expressions should be terminated by a line feed, after which DDT will give a carriage return and) and await the next expression. The ↑ command may be given in place of the line feed and has its usual meaning, except that the contents of the previous location are not typed. Two other commands are legal in patch mode. They are:

(1) Colon, which may be used to define a local symbol with value equal to the current location.

(2) Carriage return, which terminates the patch. When the patch is terminated, DDT inserts in the next available location the original contents of the location at which the patch was inserted. It then inserts in the following two locations branch instructions to the first and second locations following the patch. This means that if the patch command is a skip instruction, the program will continue to operate correctly. Any other commands given in patch mode may cause unpredictable errors.

;I Is identical to the) command except that it puts the instruction being patched before the code inserted by the programmer instead of after.

2.9 Miscellaneous Commands

? This command erases everything typed since the last tab or carriage return. It is always legal.

;G (GO TO) < E > ;G restores the A, B, and X registers which were saved when DDT was entered (unless they have been modified) and transfers to the location specified by the value of the expression.

;K (KILL) Used alone, this command removes from DDT's symbol table all symbols defined by the program. DDT will type back --OK and wait for a confirming dot. All other characters will abort the command. Preceded by a symbol, it removes that symbol only from the table.

;L < E >, < E > ;L sets ;1 and ;2 (the lower and upper bounds for searches) to the values of the first and second expressions respectively.

;U (UNDEFINED) This command causes all undefined symbols to be listed.

< E > ;U causes the value of the expression to be executed as an instruction. If it is a branch, control goes to the location branched to. In all other cases control remains with DDT. A single carriage return is typed before execution of the instruction. If the instruction does not branch and does not skip, a $ and another carriage return are typed after its execution. If the instruction does skip, two dollar signs ($$) are typed followed by a carriage return.

;Z (ZERO) < E >, < E > ;Z sets to zero all locations between the value of the first expression and that of the second. ;Z alone releases all memory accessible to the user's program. DDT will type back --OK and wait for a confirming dot. Any other characters will abort the command. If this memory is returned, due to later access by DDT or a program, it will be cleared to zero.

2.10 Special Symbols

The value of "." is the current location, i.e., the address of the last register opened.

The following symbols refer to various special registers of the machine. Their value is the contents of these registers as saved by DDT. To change the contents of a register, a command of the form < E > ;A is used. This command sets the A register to the value of the expression. Whenever DDT executes a ;U, ;G, ;P, or ;N command, it restores the values of all machine registers. If any of these values have been changed by the user, it is the changed value which will be restored.

;A The value of this symbol is the contents of the A register.

;B The value of this symbol is the contents of the B register.

;X The value of this symbol is the contents of the X register.

;L The value of this symbol is the contents of the program counter. Note: There is never any reason for changing the value of this symbol.

The values of the following special symbols are used by DDT in certain commands or are available to the programmer for his general enlightenment. These values may be changed in the same way that the values of the symbols for the central registers of the machine may be changed.

;M The value of this symbol is the mask for word searches.

;1 The value of this symbol is the lower bound for word and effective address searches. It may also be set by the ;L command.

;2 The value of this symbol is the upper bound for word and effective address searches. It may also be set by using ;L.

;Q This symbol has a value equal to the value of the last expression typed by DDT or the user. It is useful, for instance, if the programmer wishes to add one to the contents of the open register; he need only type ;Q + 1.

;F The value of this symbol is the address of the lowest location in core not used by the program. New literals and patches are inserted starting at this address. Note: Like all other special symbols, ;F may be changed by the command < E > ;F. It is also updated as necessary by patches and literal definitions.

2.11. Panics

DDT recognizes four kinds of panic conditions:

(1) Illegal instruction panics from the user's program.

(2) Memory allocation exceeded panics from the user's program.

(3) Panics generated by pushing the rubout button.

(4) Panics generated by the execution of BRS 10 in the user's program.

For each of these conditions DDT prints out a message, the location of the instruction at which the panic occurred, and the contents of this location. The messages are as follows:

(1) Illegal instruction panic I > >

(2) Memory allocation exceeded M > >

(3) Rubout button panic PB > >

(4) BRS 10 panic P > >

If a memory allocation exceeded panic is caused by a transfer to an illegal location, the contents of the location causing the panic is not available and DDT, therefore, types a ?.

Two other panic conditions are possible in DDT.

(1) If the rubout button is pushed twice with no intervening typing by the user, control returns to the executive.

(2) If the rubout button is pushed while DDT is executing a command, execution and typeout are terminated and DDT types carriage return and bell and then awaits further commands.

APPENDIX A
DDT COMMANDS

/	open a register
CR	close register
]	open symbolic
[	open as constant
$	open as signed integer
"	following address: Open register and type contents in ASCII
lf	open next register
↑	open previous register
(	open register addressed by last expression typed
\	enter mode
=	type as constant
#	type as signed integer
←	type symbolic
'	type in ADCII
:	define symbol equal to current location
@	define symbol as address of last expression typed; type address as symbolic
<	define symbol as expression
%	switch radix of following number
"	take next three characters as ASCII text (not immediately preceded by an address)
!	insert breakpoint
)	patch
?	erase
;C	set typeout mode to constant

;D	change radix to decimal
;E	effective address search
;G	go to
;H	set typeout mode to ASCII
;I	insert patch following instruction
;K	kill symbols
;L	program counter; set bounds for searches
;N	next
;O	change radix to octal
;P	proceed
;R	set radix; set address typeout to relative
;S	set typeout mode to symbolic
;T	read binary with symbols
;U	type undefined symbols; execute preceding expression
;V	set address typeout to absolute
;W	word search
;Y	read binary without symbols
;Z	clear memory

DDT SPECIAL SYMBOLS

.	current location
;1	lower bound for search
;2	upper bound for search
;A	A register
;B	B register
;E	effective address search
;F	last location of program
;L	program counter

;G	go to
;H	set typeout mode to ASCII
;K	kill symbols
;L	program counter; set bounds for searches
;N	next
;O	change radix to octal
;P	proceed
;R	set radix; set address typeout to relative
;S	set typeout mode to symbolic
;T	read binary with symbols
;U	type undefined symbols; execute preceding expression
;V	set address typeout to absolute
;W	word search
;Y	read binary without symbols
;Z	clear memory

DDT SPECIAL SYMBOLS

.	current location
;1	lower bound for search
;2	upper bound for search
;A	A register
;B	B register
;F	last location of program
;L	program counter
;M	word search mask
;Q	last expression typed
;X	X register

;M	word search mask
;Q	last expression typed
;X	X register
;\	change to insert mode
;/	change back to open register mode
;[	set 0-D mode to type registers in octal
;]	" " " " " " " " symbolic

APPENDIX B

CONCISE GUIDE TO THE DDT MANUAL

ARPAS

January, 1967

TABLE OF CONTENTS

1.0. Introduction

An assembly program or assembler is a translator whose source language is assembly language and whose object program is in machine language. Assembly language is virtually a one-for-one representation of machine language written in a symbolic form. Its value comes from mnemonic representation of operations and from the ability of the assembler to perform address computations and to do space allocation.

The introduction serves to define most of the terminology used. It is assumed that the programmer is familiar with the basic operation of the SDS 940

1.1. Basic Description of the Assembler

The assembler is a two-pass assembler with subprogram, literal, and macro facility. Its output is in two formats depending on the nature of the assembly.

1.2. Symbols

Numbers may be represented symbolically in assembly language by symbols. Symbols are arbitrarily long strings of characters not forming a number. In the assembler only the first six characters of a symbol are significant. When a symbol is used to represent the memory address of a machine command or a datum, it is called a label.

---

* See the SDS 940 Computer Manual.

1.3. Instructions and Directives

Input to the assembler takes the form of a string of instructions and directives. Instructions are generative and are mnemonic representations of machine commands. Directives may or may not be generative and serve to facilitate the entry of data or to control the assembler.

1.4. Subprogram Facility

Often programs become quite large or fall into logical divisions which are frequently almost independent. In either case it is convenient to break the programs into pieces and assemble (and even debug) them separately. Separately assembled pieces of a program are called subprograms.

Before a program assembled in subprograms can be run it is necessary to link together the subprograms while loading them. The linking process is similar to the assembly process itself described in Section 1.7. The vehicle for linking is the external symbol.

While local symbols are used by the assembler to perform address and space allocation calculations, global or external symbols are passed on to the loader where they are similarly used.

1.5. Literals

Many data are placed in programs at assembly time. It is frequently convenient to refer to constants by value than by label. A literal is a symbolic reference to a datum by value.

The assembler has complete literal facility, i.e., any type of expression can be used in a literal.

1.6. Relocation

A relocatable program is one in which memory references have been computed relative to the first word or origin of the program. Thus, if a reference is to the nth word of a program, and if the program is loaded beginning at k, the loader must form the address n + k.

The operands of instructions are not always memory references. It is necessary to instruct the loader for each word of the program whether to relocate the operand. Relocation is determined automatically during assembly and transmitted to the loader by the relocation value R. Thus, if

R = 1, the operand is to be relocated

= 0, the operand is absolute.

The only difference between relocating machine commands and constants or data is that constants are allotted all 24 bits of the 930 word. The assembler accounts for this difference automatically.

It is possible to disable the relocation in the assembler and to do an absolute assembly. The assembler produces in this case a different output format which is self-loading.

1.7. Basic Assembly Procedure

In relocatable assembly during Pass I the operands are scanned for the presence of single symbols. If a single symbol is

present, a table of symbols is searched. If absent, the symbol is added to the table but marked as having no value. Labels are evaluated by assigning them the current value of the location counter, a word which points to the ultimate relative destination of the instruction. In case of previous occurrence, labels are marked as duplicate symbols. At the end of Pass I the symbol table is sorted. All symbols present having no value are assumed to be external. At the beginning of Pass II, a list of all external symbols, the external symbol usage table, is output on the binary output medium for use by the loader. The program is then assembled and output on the binary medium.

In absolute assembly the scan for single symbols is disabled. This has the effect of doing away with external symbols.

1.8. Notation

In the following pages, square brackets [ ] are used to indicate the presence of an optional term.

2.0 The Assembly Language

2.1 Character Set

The sets of characters recognized by the assembler are as follows:

(a) numeric

(1) octal 0-7

(2) decimal 0-9

(b) alphabetic A-Z

(c) alphanumeric 0-9, A-Z

(d) delimiting characters + - * / , ' ( ) = . $

(e) special characters : ; < > ? [ ] ← "

2.2 Input Records

Input records consist of lines of information (i.e., all character strings between carriage returns or on cards) or of parts of lines separated by semi-colons.*

Each non-blank input record is either an instruction, a directive, or a comment. A comment record begins with an asterisk. Blank records are ignored.

Directives and instructions are divided into four fields. The fields are, from left to right, the label field, operation field, operand field, and comment field.

The label field begins with the first character in the record and terminates -- as do all other fields -- on the first blank. All other fields begin with the first non-blank character after the termination field. The operation field contains the mnemonic operation code. Only the first four characters are recognized by the assembler. The operation codes are classified as follows:

---

*Programmers must accordingly avoid the use of the semi-colon in comments. Semi-colons enclosed in single quotes (4.2(c) or 6.2(a)) are taken literally and do not cause a carriage return.

(a) machine instructions

(b) directives

(c) instructions and programmed operators defined by the programmer for the particular program

2.3. Assembler Syntax

A program consists of a set of instructions and directives terminated by an END directive. Normally, programs are assembled relocatable. A program is assembled absolute if it begins with an ORG directive.

Instructions have a common syntax. Each directive, in general, has its own syntax. The syntax for instructions and directives will be considered in separate sections.

3.0. Instruction Syntax

Instructions fall into two classes.

3.1. Classification of Instructions

(a) Class 1 - normal instructions

Class 1 instructions have an operand field. For each instruction it is possible to specify:

(1) address (operand field) required/not required

(2) sign bit set/cleared*.

There are two subclasses of Class 1 instructions:

(1) subclass 0

The operand is stored in the machine command mod 2^{14}. This subclass contains orders having memory references.

(2) subclass 1

The operand is stored in the machine command mod 2^{9}. This subclass is used for shift orders. If indirect addressing is called for, any class 1 instruction is treated as a subclass 0 instruction.

Class 1 instructions have the following form:

[[$]label] ABC[*] [expression 1[, expression 2]] [comment]

For most class 1 instructions, expression 1 (operand) is required. Indirect addressing is signified by an asterisk following the operation code.

---

* This feature is intended to be used with system programmed operators (cf. Lichtenberger, W. W., Pirtle, M. W. & Sanders, W. J. Modifications to the SDS 930 Computer For the Implementation of Time-Sharing, Document No. 20.10.10, January 22, 1965.)

(b) Class 2 - complete or full word instructions

Class 2 instructions have no operand field. Indirect addressing is signified by an asterisk following the operation code. Class 2 instructions have the following form:

[$][label] ABC[*] [comment]

3.2. Label Field

A label, if present, identifies the instruction. An instruction will have a label normally if it is referred to elsewhere in the program, although it is not necessary that a symbol so defined be used. Symbols defined but not used are called nulls and are marked as such in the assembly listing.

If the same symbol appears in the label field of more than one instruction, it is marked as a duplicate each time it is used, and it is given the value at its most recent definition.

A $ preceding a label causes an identical external definition (cf. 6.8 (a)).

3.3. Operand Field

The operand field contains at most two arithmetic expressions (or a literal and one expression) used to evaluate the operand and tag of the machine command. The tag, if present, is always evaluated mod 2^3 and must be absolute.

3.4. Comment Field

The comment field is not processed by the assembler, but is listed as part of the assembler output.

4.0. Syntax of Expressions

Expressions are used in the operand field of most instructions and directives. In the following, V(x) will represent the numerical value of the expression x, and R(x) will represent the relocation value of x. U is used to mean unary operator, and B stands for binary operator. A subscript refers to a term or expression of indicated type, i.e., $term_d$ means a term of type d.

The evaluation of all expressions is made using the full word length of the machine (24 bits).

4.1. Operators

Expressions consist of terms connected by operators. The operators permitted are:

		Operator	Hierarchy
(a)	unary	+	4
		-	4
		(NOT)	4
(b)	relational	(LSS) <	3
		(GRT) >	3
		(EQU) =	3
(c)	binary	+	1
		-	1
		*	2
		/	2
		(AND)	2
		(OR)	1
		(EOR)	1

Parentheses are not allowed in expressions except to designate mnemonic operators. Relational operators give rise to a value 1 if the relation is true and 0 if false.

4.2. Constants

Constants are of three types:

(a) decimal integers: one or more decimal characters.

(b) octal integers: one or more octal characters terminated with the letter B.

(c) string: '1-4 characters (except ')'.

For any constant, R(constant) = 0.

4.3. Classification of Symbols

Symbols are classified by the assembler in the following way:

(a) local: defined by their use in the label field of instructions and in some directives.

$V(symbol_a)$ = V(location counter at definition)

$R(symbol_a)$ = 1 if relocatable assembly

= 0 if absolute assembly

(b) equated: defined by an EQU directive.

$V(symbol_b)$ = V(expression in operand field of EQU)

$R(symbol_b)$ = R(expression in operand field of EQU)

(c) *: current location counter symbol.

V(*) = current value of location counter

R(*) = 1 if relocatable assembly

= 0 if absolute assembly

(d) external: defined in another subprogram.

4.4. Terms

There are two types of terms:

(a) [U]constant

$V(term_a)$ = [U]V(constant)

$R(term_a)$ = 0

(b) $[U]symbol_{a,b,c}$

$V(term_b) = [U]V(symbol_{a,b,c})$

$R(term_b) = [U]R(symbol_{a,b,c})$

4.5. Expressions

There are three types of expressions:

(a) term

$V(expression_a) = V(term)$

$R(expression_a) = R(term)$

(b) expression $1_{a,b}$ B expression $2_{a,b}$

$V(expression_b) = V(expression\ 1_{a,b})$ B $V(expression\ 2_{a,b})$

$R(expression_b) = R(expression\ 1_{a,b})$ B $R(expression\ 2_{a,b})$

(c) $symbol_d$ (note: no unary operator permitted)

$V(expression_c)$ = temporarily the location of the symbol in an external symbol usage table; ultimately the value of the external symbol when known by the loader.

$R(expression_c)$ has no meaning

In an absolute assembly, expressions of type c are considered to be undefined symbols.

4.6. Evaluation of Expressions

Expressions are evaluated from left to right using operators of decreasing hierarchy.

Example: If A = 100
B = 200
C = -1 ,

then A+B*C/A = 98.

Example: If A = 54321_8
B = 44444_8
C = 00077_8 ,

then A(OR)B(AND)C = 54365_8.

4.7. Relocation Constraints

The following constraints apply to expression evaluation:

(a) No relocatable term (R = 1) may occur in conjunction with * or /, i.e., no relocatable symbol may multiply, be multiplied by, divide, or be divided by anything.

(b) $R(\text{expression}_{a,b})$ = 0 or 1. R may attain other values during evaluation.

5.0. Literals

Literals are of the form:

= any expression of any type

When encountering a literal, the assembler replaces the value of the expression by the location of that value in a table of literals constructed for each program. The literal table is appended to the program. Thus, it is dangerous to terminate a program with a record which labels a block of storage unless the record is a BSS or a BES directive.

6.0. Directives

The following directives are included in the assembly language:

Data Generation:	DATA
	TEXT
	ASC
Value Declaration:	EQU
	EXT
	OPD
	POPD
Assembler Control:	BES
	BSS
	ORG
	END
	LIST
	NOLIST
	PAGE
	SYTB
	BIN
	NOBIN
	IDENT
	DELSYM
Macro Generation:	MACRO
	ENDM
	RPT
	CRPT
	ENDR
	IF
	ENDF
	NARG
	NCHR

Since directives, in general, possess unique syntaxes, we consider each one in turn.

6.1. DATA Generate Data

[label] DATA $expression_1$, $expression_2$,...

The label is given the current value of the location counter. Each expression is then evaluated and the results assigned to sequential locations. The effect of the directive is to create a block of data, the first word of which may be labeled by a symbol. Note that values are assigned mod 2^{24}.

6.2. TEXT Generate Text

There are two forms for this directive.

(a) The first form creates text of unspecified length.

[label] TEXT 'THIS IS A SAMPLE.'

In this form all characters between the apostrophes are converted into 6-bit trimmed ASCII, packed four to a word, and assigned to sequential locations. The first word of the list may be identified by a label. Characters in the last word are left-justified, with remaining positions filled in by blanks (octal 00). This form will allow any text to be generated conveniently except that containing apostrophes.

(b) The second form creates text of a specified length.

[label] TEXT 5,THIS IS A SAMPLE.

In this form, all characters following the comma are packed and assigned as above. The operand field of the directive terminates when the specified number of words has been packed.

6.3. ASC Generate Text with Three Characters per Word.

This directive is identical to TEXT, except that 8 bits are given to each character.

6.4. BES Block Ending Symbol

[label] BES $\text{expression}_{a,b}$

BES reserves a block of storage for which the following location is labeled. The expression must be absolute, and it must have a value when BES is first encountered, i.e., symbols present must have been previously defined.

6.5. BSS Block Starting Symbol

[label] **BSS** expression$_{a,b}$

BSS reserves a block of storage for which the first word is labeled. The expression must be absolute, and it must have a value when BSS is first encountered, i.e., symbols present must have been previously defined.

6.6. EQU Equals

symbol EQU expression$_{a,b}$

The EQU directive causes the symbol in its label field to be given the value of the expression. The value is held in a full machine word (24 bits). The expression must have a value when EQU is first encountered, i.e., symbols present must have been previously defined. It is permissible to redefine by EQU any symbol previously defined by EQU. This ability is particularly useful in macros.

6.7. END End of Assembly

END [Starting location]

The END directive terminates the assembly. The optional expressior is used in absolute assemblies (cf. 8.2.)

6.8. EXT Define External Symbol

Symbols can be defined externally in three ways.

(a) $label opcode operand

The presence of the preceding $ causes the symbol in the label field of any instruction to be defined both locally and externally.

(b) label EXT

The (local) symbol in the label field is defined as an external symbol having the same value. The label must be defined locally somewhere in the program.

(c) symbol EXT expression$_{a,b}$

The symbol in the label field is defined as an external symbol whose value is given by the expression. This form is used for defining absolute external symbols, symbols which depend on combinations of other symbols, and symbols which are synonymous with local symbols. The EXT directive may be used at any position in the program.

6.9. ORG Program Origin

ORG expression$_{a,b}$

The use of ORG forces an absolute assembly. The location counter is initialized to the value of the expression. The expression must be absolute, and it must have a value when ORG is first encountered, i.e., symbols present must have been previously defined. An ORG must precede the first instruction in an absolute program.

6.10. OPD Operation Definition

Operation codes defined by the OPD directive take precedence over other operation codes. The form of OPD is:

opcode OPD cd,cl[,ar[,sc[,sb]]]

where: cd is the opcode as an arbitrary expression$_{a,b}$

cl is the class number (1 or 2)

ar signifies address required (0 or 1),

sc is the subclass number of class 1 (0 or 1), and

sb signifies that the sign bit of the machine command should be set.

The definition of a hypothetical system programmed operator LLA might appear as follows:

```
LLA  OPD  11000000B,1,1,0,1
```

and that for the machine command CLA would be:

```
CLA  OPD  04600001B,2
```

Missing expressions will be given the value zero and the associated conditions the consequent meanings.

6.11. POPD Programmed Operator Definition

The POPD directive is identical in effect to the OPD directive except for the following features:

(a) Bit 2 of the word programmed-operator bit) must be a 1.

(b) The loader will place a branch instruction in the transfer vector for programmed operators ($100\text{-}177_8$). The branch instruction will transfer to the value of the location counter when the POPD is encountered. Thus, the body of instructions constituting the programmed operator must follow the POPD. If the insertion of the transfer instructions is not desired, OPD should be used.

6.12. LIST List Program on Output Medium

```
LIST
```

The LIST directive enables the assembly listing in case it was previously disabled by the NOLIST directive. The assembler is initialized not to list.

6.13. NOLIST Disable Assembly Listing

NOLIST

The NOLIST directive disables the assembly listing. All errors encountered during assembly will be listed, however, regardless of any past use of NOLIST. NOLIST in conjunction with NOBIN can be used to provide a quick error scan of a program.

6.14. PAGE Skip to a New Page

PAGE

PAGE causes the assembler listing program to advance to a new page unless already there. The directive is used to improve listing format.

6.15. SYTB List Symbol Table

SYTB

SYTB causes the local symbol table to be listed. Symbols are listed in alphanumerical order along with their values as octal integers with leading zero suppression. Relocatable symbols are marked by a plus to the right of the value. Null and duplicate symbols are marked by N and D, respectively, between the symbol and its value.

6.16. BIN Output Binary Program

BIN

The BIN directive enables the output of binary program (loader input) in case it was previously disabled by the NOBIN directive. The assembler is normally initialized to do such output.

6.17. NOBIN Disable Binary Output

NOBIN

NOBIN disables binary output. The directive can be used in conjunction with NOLIST to provide a quick error scan of a program, or it may be used alone to provide a listing only. It may also be used in conjunction with BIN to produce only selected portions of binary output.

6.18. IDENT Subprogram Identification Marker

symbol IDENT

IDENT causes the symbol found in its label field to be output on the binary output medium along with a unique control word. IDENT may be used for editing purposes and for reference to a block of program. The loader ignores all IDENT records.

6.19. DELSYM Delete Punch-out of Symbol Table and Defined Operations

DELSYM

DELSYM inhibits the local symbol table and all programmer-defined operations from output on the binary medium for later debugging purposes. The effect of the directive is to shorten the binary output.

6.20. MACRO Define a Macro-operation

6.21. ENDM End a Macro Definition

```
name  MACRO  [dummy[,generated symbol,no. of generated symbols]]
      (body of macro)
      ENDM
```

The macro is a subprogram defined as shown above and referred to by name. Dummy symbols are used in the body of the macro for strings which are supplied as arguments in the operand field when the macro is called. The strings may be expressions, symbols, or parts of each. Strings may be concatenated by writing dummies or substrings separated by periods. Dummies are written as a subscripted symbol declared as dummy in the definition (e.g., DUM(2), DUM(XYZ), etc.). The subscripts may be expressions.

The special notation DUM() calls forth the entire argument string of the macro call. In addition, the symbol

DUM(expression1$expression2)

is expanded as the k^{th} character of the n^{th} argument, where:

$n = \text{expression}_{a,b}1$

$k = \text{expression}_{a,b}2$

Finally, if $j = \text{expression}_{a,b}3$, the j^{th} through k^{th} characters of the n^{th} dummy are called forth by

DUM(expression1$expression3,expression2)

Examples: The macro:

```
EXMPLE   MACRO     D
         LDA       D(1)
         ST.D(2)   D(ABC)      (ABC=4)
A        NARG
D(A)     BRU       CON.D(2)
         DATA      D()
         TEXT      'THIS IS AN EXAMPL.D(3$A)'
         DATA      D(4$2,A-1)
         ENDM
```

when called by:

```
         EXMPLE   XYZ,A,ABCDEF,ALOC,Z.
```

will expand to be:

```
         LDA       XYZ
         STA       ALOC
Z        BRU       CONA
         DATA      XYZ,A,ABCDEF,ALOC,Z
         TEXT      'THIS IS AN EXAMPLE'
         DATA      LOC
```

Symbols may be generated at each macro call. Such symbols are defined as a subscripted symbol declared as a generated symbol at definition. It is necessary to include the maximum number of such generated symbols in the definition. Generated symbols are not punched or printed (as in SYTB).

In calling a macro, argument strings are separated by commas. If a comma is desired to be in a string, the string may be parenthesized. When the macro is expanded, the outer parentheses are discarded. It should be noted that macro definitions may include calls to other macros. Similarly, macros may be defined within other definitions.

In using macros it is sometimes necessary to have a call or definition which exceeds the length of a line. Accordingly, a + in column 1 is a continuation mark.

6.22. RPT Repeat the Next Block

6.23. ENDR End RPT Block

RPT expression$_{a,b}$

Although usually most useful in connection with macros, the RPT directive may be used anywhere. The expression in the operand field must have a value when the RPT is encountered, and the value of this expression determines the number of times the body of code is repeated. If the expression is of value ≤ 0, the body is ignored.

6.24. CRPT Conditional Repeat

CRPT expression$_{a,b}$

CRPT differs from RPT in that the expression is revaluated after the ENDR is encountered. If the new value is > 0, the body is inserted again. This process will be endless unless, of course, the body contains some directive (like EQU) which changes the value of the expression.

6.25. IF Insert the Next Block If the Expression > 0

6.26. ELSF Else Insert the Next Block If the Expression > 0

6.27. ENDF End the If Block

IF expression$_{a,b}$

ELSF expression$_{a,b}$

IF permits the following block of instructions and/or directives to be inserted only once or ignored, depending on the value of the expression. In evaluating the expression, all undefined symbols are given the value -1. This happens only in an IF statement.

ELSF is an alternative IF statement. Every IF may be followed by one or more EKSFs. If the preceding IF or any preceding ELSFs are not obeyed, the expression in an ELSF statement is evaluated and treated as a new IF. In case the expression is true (value > 0), the following text is assembled, and any subsequent ELSFs are ignored.

The ENDF terminates the IF block and permits new IFs and ELSFs.

6.28. NARG Equals Number of Arguments

symbol NARG

The NARG is permitted only within the body of a macro and serves to equate the value of the symbol to the number of arguments used in the macro call. Its function is entirely similar to the EQU directive, and symbols defined by EQU may be redefined by NARG and vice-versa. NARG permits macros with varying numbers of arguments to be dealt with readily.

6.29. NCHR Equals Number of Characters

symbol NCHR expression

The NCHR is legal only within the body of a macro and serves to equate the value of the symbol to the number of characters in the associated expression. Its function is entirely similar to EQU, and symbols defined by EQU may be redefined by NCHR and vice-versa. NCHR is useful mainly when the expression consists of or contains one or more dummies, since the character count of a fixed expression can be determined before assembly and defined by an EQU.

6.30. Facility for Immediate Evaluation of an Expression and Conversion to a Digit String

($expression$_{a,b}$)

As an adjunct to the automatic symbol generation facility and for any other purpose for which it may be useful, the macro expander will replace the string ($expression$_{a,b}$) by its value as a string of decimal digits. The expression must have a value when first encountered.

Example: The macro

```
EXMPLE      MACRO     D,GZ,2
Z           NCHR      D(1)
Y           NARG
X.($Y+2)    LDA       D(1$3,Z)
G(1)        STA       W.($Y)
G(2)        STA       V.($Y+ABC)        (ABC=100)
            ENDM
```

when called the fifth time by:

```
            EXMPLE    ABCDEF
```

will expand to be:

```
X3          LDA       CDEF
GZØØØ9      STA       W1
GZØØ1Ø      STA       V1Ø1
```

7.0. Assembler Diagnostics

Diagnostic information is placed under the input record in the listing at the location of the offence. All erroneous records are listed even if the NOLIST directive has previously been used.

Errors detected are:

Error	Condition
D	Duplicate symbol
L	Illegal symbol in label field
M	Missing field in input record
O	Illegal or undefined Opcode
R	Relocation error in expression
S	Expression or other syntax error
U	Undefined symbol

Other messages which may be received are:

(a) SYMBOL TABLE FILLED. ERROR CHECK CONTINUES.

(b) LITERAL TABLE FILLED. ERROR CHECK CONTINUES.

(c) MUST ASSEMBLE ABSPGM ON PAPER TAPE.

(d) INPUT STACK OVERFLOW.

(e) INPUT STACK UNDERFLOW.

(f) INPUT BUFFER FULL.

(g) TOO MUCH MACRO RECURSION.

(h) TOO MUCH RPT RECURSION.

(i) TOO MANY ARGS IN MACRO.

(j) STRING STORE EXCEEDED.

(k) END OF FILE.

8.0. Assembler Binary Output Formats

8.1. Relocatable Output Format for Linking Loader

Binary output is divided into logical variable length records. Each record begins with a control word which defines its type. Bits 0, 1, and 2 normally signify the type. The first word of the binary output is a 3-bit register (cf. (a) below) whose single entry is an octal 4.

Bits 0, 1, and 2 (octal)		Meaning
0		Binary program follows
1		Programmed operator follows
200	Bits 0-8 are used.	End of program
201		Origin of literal table is in address field
3		OPD follows
4		External symbol definition(s) follows
5		Identification record follows
6		External symbol usage table follows
7		Symbol table follows

The remaining bits in the control word and the format of the record which follows are different for each type.

(a) 0 - Binary Program Follows

Bits 10-23 of the control word are added to the current value of the location counter. Binary program consists of groups of eight machine commands preceded by eight groups of three bits packed into a single word (the 3-bit register). Each group of three bits is associated

with a following instruction or control word and serves as a loading indicator for that instruction. The following indicators are used:

3-Bits (octal)	Meaning
0	Absolute address
1	Evaluate address from external symbol table mod 2^{14}
2	Relocate address mod 2^{14}
4	Abandon binary program format -- next word is a control word.
5	Evaluate word from external symbol table mod 2^{24}
6	Relocate word mod 2^{24}

(b) 1 - Programmed Operator Follows

Bits 2-8 of the control word determine the position of a transfer command which is placed by the loader in the programmed operator transfer vector ($100\text{-}177_8$). Bits 10-23 determine the address of the transfer command. Information following is binary program which follows the previous program, i.e., the location counter is unaffected by POPD.

(c) 200 - End of Program

No other bits in the control word are significant. The 200 record is a one-word record.

(d) 201 - Origin of Literal Table

The origin of the literal table is found in the address field.

(e) 3 - Defined Operation Follows

All OPDs are punched in the form of a standard symbol (cf. (f) below).

(f) 4 - External Symbol Definition(s) Follows

Each definition consists of a block of three words. The first two words contain the six characters of the symbol in ASCII, left-justified with trailing blanks. The third word contains the symbol value. Bit 12 of the second word signifies relocation of the external symbol value. Relocation of external symbols is performed mod 2^{24}. Each block of such definitions is terminated by a single word of all 1's.

(g) 5 - Identification Record Follows

The identification record consists of one block of three words. The format of the block is identical to that for each entry of (d), although only the six characters of the identification symbol are meaningful.

(h) 6 - External Symbol Usage Table Follows.

Each entry of the usage table is a three-word block of the same format as in (e) above.

(i) 7 - Symbol Table Follows

The format of the local symbol table is the same as in (e) above. The order of records is as follows:

[ident record]

[external symbol usage table] (if any)

[literal table origin]

{ binary program
  programmed operators
  external symbol definitions } in any arrangement

end of program

8.2. Absolute Assembly Output Format

For absolute programs, the assembler first punches a bootstrap loader and then program in blocks. Each block is begun when it is necessary to reset the location for loading.

The use of a POPD will cause a transfer to be placed in the POP transfer vector as in relocatable programs.

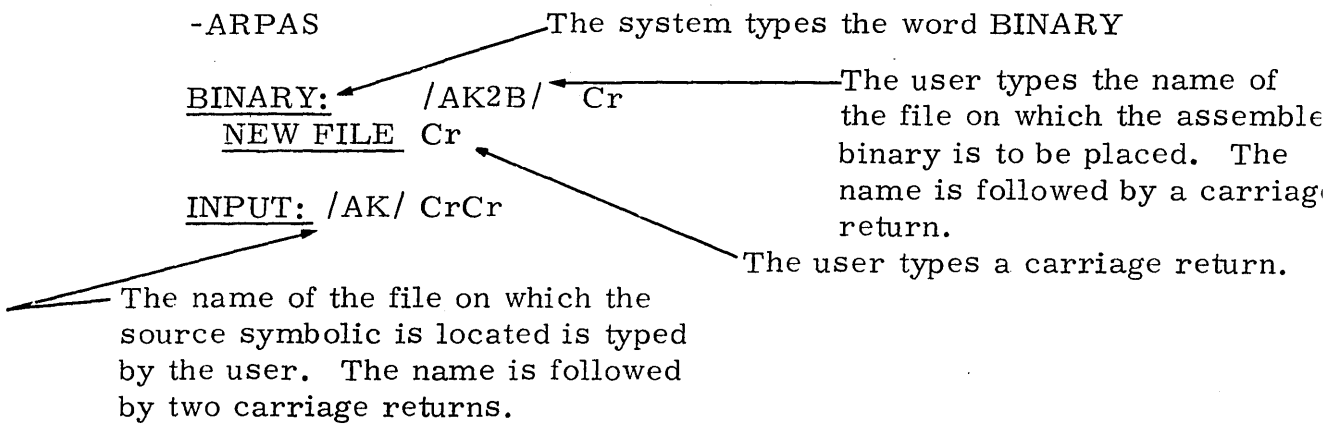
IDENT records will be punched as in relocatable programs but with a preceding word which causes the bootstrap loader to ignore the record.

Similarly, OPDs and the symbol table may be punched and ignored by the loader.

END with a blank operand field causes the loader to halt. In case the field is not blank, a transfer of control to the value of the expression is placed in location 1 and executed.

9.0 ASSEMBLER OPERATING INSTRUCTIONS

To call the assembler from the executive, the user types the word ARPAS.
Example:



Errors and null locations will be typed out at this time.

APPENDIX A

EXTENDED LIST OF INSTRUCTIONS

Mnemonic	Instruction Code	Function
Load/Store		
LDA	76	LOAD A
STA	35	STORE A
LDB	75	LOAD B
STB	36	STORE B
LDX	71	LOAD X
STX	37	STORE INDEX
EAX	77	COPY EFFECTIVE ADDRESS INTO INDEX
XMA	62	EXCHANGE M AND A
Arithmetic		
ADD	55	ADD M TO A
ADC	57	ADD WITH CARRY
ADM	63	ADD A TO M
MIN	61	MEMORY INCREMENT
SUB	54	SUBTRACT M FROM A
SUC	56	SUBTRACT WITH CARRY
MUL	64	MULTIPLY
DIV	65	DIVIDE
Logical		
ETR	14	EXTRACT (AND)
MRG	16	MERGE (OR)
EOR	17	EXCLUSIVE OR

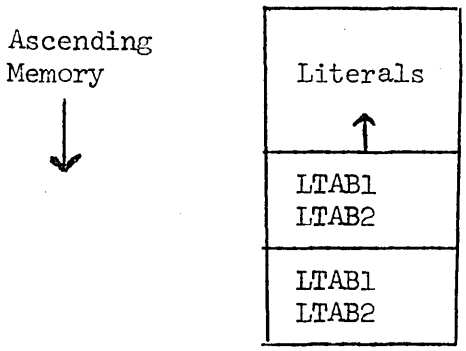
Mnemonic	Instruction Code	Function
Register Change		
CLA	0 46 00001	CLEAR A
CLB	0 46 00002	CLEAR B
CLAB	0 46 00003	CLEAR AB
CLX	2 46 00000	CLEAR X
CLEAR	2 46 00003	CLEAR A, B, AND X
CAB	0 46 00004	COPY A INTO B
CBA	0 46 00010	COPY B INTO A
XAB	0 46 00014	EXCHANGE A INTO B
BAC	0 46 00012	COPY B INTO A, CLEARING B
ABC	0 46 00005	COPY A INTO B, CLEARING A
CXA	0 46 00200	COPY X INTO A
CAX	0 46 00400	COPY A INTO X
XXA	0 46 00600	EXCHANGE X AND A
CBX	0 46 00020	COPY B INTO X
CXB	0 46 00040	COPY X INTO B
XXB	0 46 00060	EXCHANGE X AND B
STE	0 46 00122	STORE EXPONENT
LDE	0 46 00140	LOAD EXPONENT
XEE	0 46 00160	EXCHANGE EXPONENTS
CNA	0 46 01000	COPY NEGATIVE INTO A
Branch		
BRU	01	BRANCH UNCONDITIONALLY
BRX	41	INCREMENT INDEX AND BRANCH
BRM	43	MARK PLACE AND BRANCH
BRR	51	RETURN BRANCH

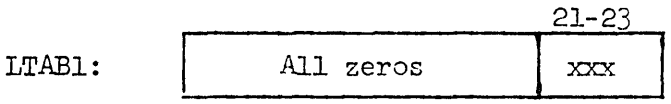
Mnemonic	Instruction Code	Function
Test/Skip		
SKS	40	SKIP IF SIGNAL NOT SET
SKE	50	SKIP IF A EQUALS M
SKG	73	SKIP IF A GREATER THAN M
SKR	60	REDUCE M, SKIP IF NEGATIVE
SKM	70	SKIP IF A = M ON B MASK
SKN	53	SKIP IF M NEGATIVE
SKA	72	SKIP IF M AND A DO NOT COMPARE ONES
SKB	52	SKIP IF M AND B DO NOT COMPARE ONES
SKD	74	DIFFERENCE EXPONENTS AND SKIP
Shift		
RSH	0 66 00xxx	RIGHT SHIFT AB
RCY	0 66 20xxx	RIGHT CYCLE AB
LRSH	0 66 24xxx	LOGICAL RIGHT SHIFT
LSH	0 67 00xxx	LEFT SHIFT AB
LCY	0 67 20xxx	LEFT CYCLE AB
NOD	0 67 10xxx	NORMALIZE AND DECREMENT X
Control		
HLT, ZRO	00	HALT
NOP	20	NO OPERATION
EXU	23	EXECUTE
Overflow		
ROV	0 02 20001	RESET OVERFLOW
REO	0 02 20010	RECORD EXPONENT OVERFLOW
OVT	0 40 20001	OVERFLOW TEST AND RESET

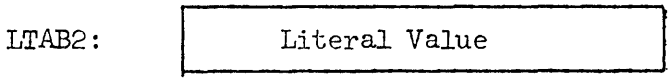
APPENDIX B

ASSEMBLER TABLE STRUCTURE

B.1. Literal Table

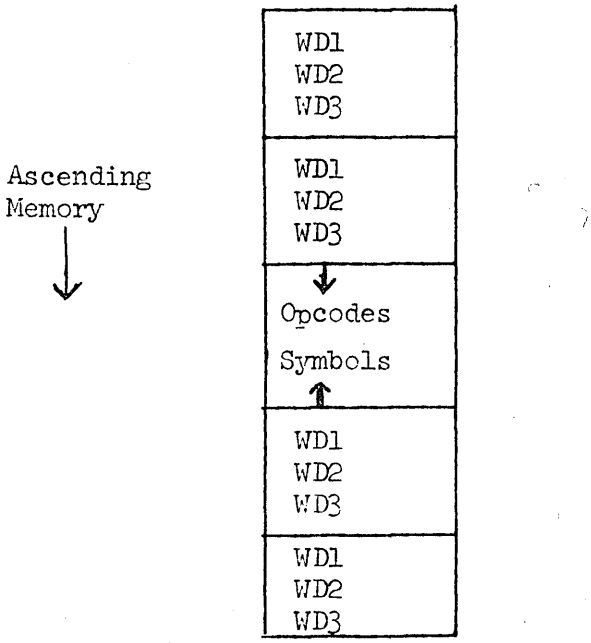




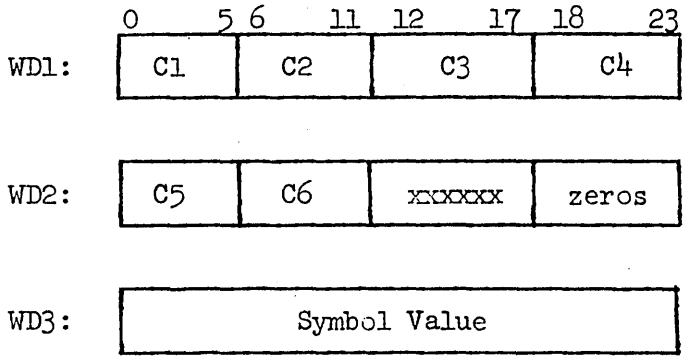


LTAB1 contains relocation information for the following value. Bits 21-23 have the same meaning as defined in Section 1.6. on relocation.

B.2. Symbol and Opcode Tables



(a) Symbols

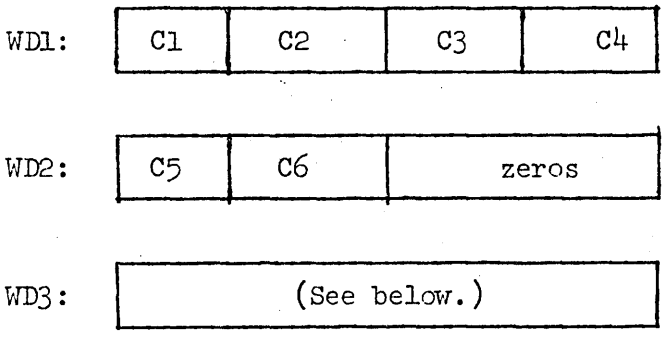


C1-C6 are the six significant characters of the symbol left-justified with trailing 77_8's.

In WD2 bits 12-17 are flags which have the following meanings (when set):

Bit	Meaning
12	Relocatable Symbol (R = 1)
13	Duplicate Symbol
14	External Symbol
15	Null Symbol
16	Generated Symbol
17	Equated Symbol

(b) Operation Codes



C1-C6 are the six significant characters of the opcode, left-justified with trailing 77_8's.

WD3 contains the binary operation code and certain other bits which serve to define the class and subclass of operation.

Bit	Meaning
0	Set: Class 2; Reset: Class 1 or Directive (see bit 1).

If bit 0 is set (class 2 instruction), all other bits comprise the machine command. Otherwise, they have the following meaning:

Bit	Meaning
1	Set: Directive; Reset: Class 1

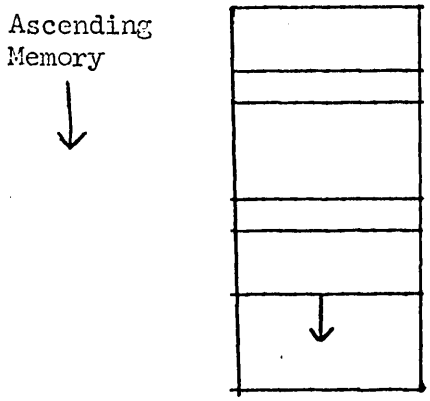
If bit 1 is set (directive), bits 10-23 contain the address of the entry point to the directive processor.

If bit 1 is reset (class 1 instruction), the following bits are significant:

Bit	Meaning
9	Sign bit to be set (system programmed operator)
19	Operand is required
23	Set: subclass 1; Reset: subclass 0

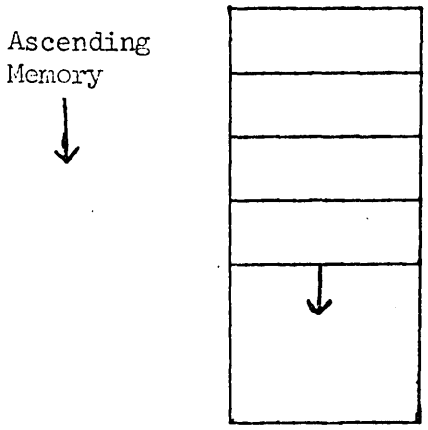
B.3. Macro and RPT Storage Area

(a) Macro Data Stack



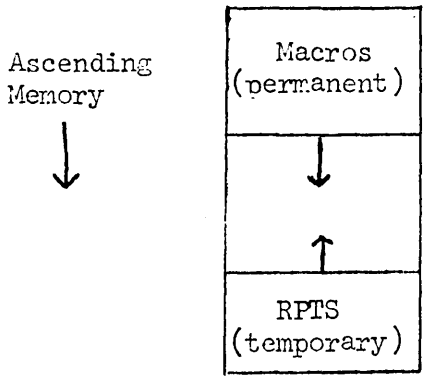
(Variable size blocks.)

(b) RPT Data Stack



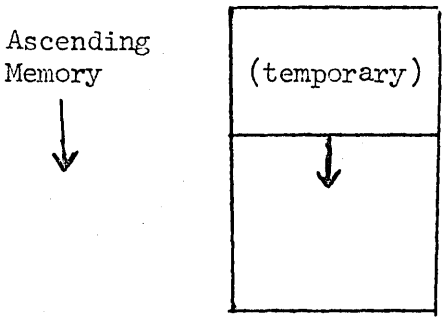
(Fixed size blocks.)

(c) Macro and RPT String Storage



(4 characters per word.)

(d) Macro Argument String Storage



APPENDIX C

ASSEMBLER INTERNAL CODE

In order to facilitate symbol sorting and logical operations, a non-standard, internal code is used throughout the assembler. This code is strictly internal, and is not transmitted outside. Binary output is in trimmed ASCII (cf. Appendix D).

Within the assembler the following equivalences hold:

Character	Code	Character	Code
0	00	0	36
1	01	P	37
2	02	Q	40
3	03	R	41
4	04	S	42
5	05	T	43
6	06	U	44
7	07	V	45
8	10	W	46
9	11	X	47
"	15	Y	50
:	16	Z	51
;	17	<	52
A	20	>	53
B	21	?	54
C	22	[	55
D	23	]	56
E	24	←	57
F	25	+	60
G	26	-	61
H	27	*	62
I	30	/	63
J	31	↑	64
K	32	,	65
L	33	=	66
M	34	.	67
N	35	(	70
		)	71
		'	72
		$	73

(all others) 77

APPENDIX D

TABLE OF TRIMMED ASCII CODE FOR THE SDS 930*

(NUMERIC ORDER)

0	SPACE	31	9	62	R
1	!	32	:	63	S
2	"	33	;	64	T
3	#	34	<	65	U
4	$	35	=	66	V
5	%	36	>	67	W
6	&	37	?	70	X
7	'	40	@	71	Y
10	(	41	A	72	Z
11	)	42	B	73	[
12	*	43	C	74	\
13	+	44	D	75	]
14	,	45	E	76	↑
15	-	46	F	77	←
16	.	47	G	144	EOT
17	/	50	H	145	WRU
20	0	51	I	146	RU
21	1	52	J	147	BELL
22	2	53	K	152	LF
23	3	54	L	155	CR
24	4	55	M		
25	5	56	N		
26	6	57	O		
27	7	60	P		
30	8	61	Q		
X					

*The Teletype characters enclosed in boxes cannot be handled by ARPAS and are converted to blanks when present.

TIME-SHARING SYSTEM

REFERENCE MANUAL

By

Ann Hardy
David Gardner
Verne Van Vlear

TABLE OF CONTENTS

Appendix A (cont.)

1.0 INTRODUCTORY

This manual describes the Berkeley Time-Sharing System as it was modified by Tymshare, Inc. The Berkeley Time-Sharing System is divided into three major parts: The monitor, the system executive, and the subsystems. Only the first two of these are discussed in detail in this manual. The manual attempts to describe exhaustively all the features of the monitor and the system executive, and, in addition, to give a number of implementation details.

We use the word monitor to refer to that portion of the system which is concerned with scheduling, input-output, interrupt processing, memory allocation and swapping, and the control of active programs. The system exec, on the other hand, is concerned with the command language by which the user controls the system from his teletype, the identification of users and specification of the limits of their access to the system, the control of the directory of symbolic file names and backup storage for these files, and other miscellaneous matters.

The next ten sections of this manual discuss various features of the monitor. The remaining sections deal with the executive.

2.0 THE SCHEDULER

The primary entities with which the time-sharing system is concerned are called forks. Each fork is an abstract object capable of executing machine instructions. At least one fork is associated with each active user, but a user may have many forks, each computing independently under his control. Also associated with each user is a temporary storage area called the TS block.

A fork is defined by its entry in the program active table (PAC table or PACT). This table contains all of the information required to specify the instantaneous state of the extended computer which the user is programming, except for that contained in the user's memory or in the system's permanent tables. The structure of a PACT entry is displayed at the end of this section together with brief notes about the significance of the various items. These matters will be explained in more detail in the following few sections. It will be observed that PACT contains locations for saving the program counter and the contents of the active A register. The B and X registers are saved in the TS block. It also contains two pseudo-relabeling registers for the user. A third one, which specifies the monitor map, is kept in the job tables. The matter of pseudo-relabeling is discussed in detail in section 5. There is a word called PTEST which determines the conditions under which the fork should be reactivated if it is not currently running. The panic table address in PTAB and the three pointers called PFORK, PDOWN and PPAR are discussed in section 3 on forks.

The word called PTAB contains in bits 3 through 8, the number of the job to which this fork belongs. The top of PQU contains information about the amount of time for which the fork is allowed to compute. before it is dismissed. Six bits of QR count the number of clock cycles remaining before the fork is dismissed, and three bits of QUTAB point to a table which specifies the length of time which the fork should be allowed to run when it is activated. All times in the discussion are measured in periods of the 60-cycle computer clock.

When a fork is activated the number in QR is put into TTIME. This number is the amount of time left in the fork's long time quantum. The length of a long quantum is tentatively going to be the same for all users. At the same time, the value in QTAB which is pointed to

There are two operations available to the user which are connected with the quantum overflow machinery. BRS 45 causes the user to be dismissed as though he had overflowed his quantum. BRS 57 guarantees to the user upon return at least 16 msec of uninterrupted computation. This feature is implemented by dismissing the user if less than 16 msec remain in his quantum.

Ordinarily, the code which is being executed at any particular instant is that belonging to the fork which is currently active. This situation may be disturbed, however, by the occurrence of interrupts from I/O devices. These interrupts cause the computer to enter system mode and are processed entirely independently of the currently running program. They never take direct action to disturb the running of this fork, although, they may set up conditions in memory which will cause some other fork to be activated when the presently running one is dismissed. Interrupt routines always run in system mode. Other code which may be running which may not belong to the fork currently active is dhe code of system programmed operators or BRS routines. These routines are not re-entrant and, therefore, should not be dismissed by the clock. To ensure that they will not be, the convention is established that the clock will not dismiss a program running in system mode. In order to guarantee that a user will not monopolize the machine by executing a large number of SYSPOPS, the user mode trap is turned on when the clock indicates that a fork is to be dismissed. The trap will occur and cause dismissal as soon as the fork returns to user mode.

The PACT word called PTEST contains the activation condition for a currently inactive fork. The condition for activation is contained in the six opcode bits of this word, while the address field normally contains the absolute address of a word to be tested for the specified condition. It is possible, however, for the address to hold a number indicating which program interrupt has occurred.

The following activation conditions are possible:

```
 0    Word greater than 0
 1    Word less than or equal to 0
 2    Word greater than or equal to 0
 3    Word less than or equal to teletype early
      warning
 4    Special test.  The address points to a special
      activation test routine.
 5    Interrupt occurred.  The address contains the
      number of the interrupt which occurred.
 6    Do not activate
 7    Special address =
      0   dead
      1   running
      2   BRS 31
      3   BRS 106
      4   Executive BRS
      5   BRS 109
      6   BRS 9 (User Program)
10    Do not activate
11    Word 20000000 = 0 (buffer ready)
12    Word less than 0
```

An executive program can dismiss itself explicitly by putting a queue number (0 to 3) in X and a dismissal condition in B and executing BRS 72. The address of a dismissal condition must be absolute.

There is normally one running fork in the system, i.e., a fork which is executing instructions, or will be executing instructions after the currently pending interrupts have been processed. An active fork (i.e., a PACT entry) which is not running is said to be dismissed, and is kept track of in one of two ways. 1) If it is dismissed with BRS 9, 31, 106 or 109 (see Section 3) it is said to be in limbo and is pointed to only by the PFORK, PDOWN, and PPAR of the neighboring forks in the fork structure. 2) If it has been dismissed for any other reason, it is on one of the schedule queues. There are four queues of dismissed programs. In order, they are:

```
QTI   Programs dismissed for teletype Input/Output
QIO   Programs dismissed for other I/O
QSQ   Programs dismissed for exceeding their short
      quantum
QQE   Programs dismissed for exceeding their long
      quantum.
```

Programs within the queues are chained together in PNEXT, and PNEXT for the last program in each queue points to the beginning of the next queue.

Whenever it is time to activate a new program, the old program is put on the end of the appropriate queue. The schedule then begins at QTI and scans through the queue structure looking for a program whose activation condition is satisfied. When one is found, it is removed from the queue structure and turned over to the swapper to be read in and run. If there are no programs which can be activated the scheduler simply continues scanning the queue structure.

Programs reactivated for various reasons having to do with forks (interrupts, escapes, panics) are put onto QIO with an immediate activation condition. They, therefore, take priority over all programs dismissed for quantum overflow.

There is a permanent entry on the teletype queue for an entity called the phantom user. The activation condition for this entry is a type 4 condition which tests for two possibllities:

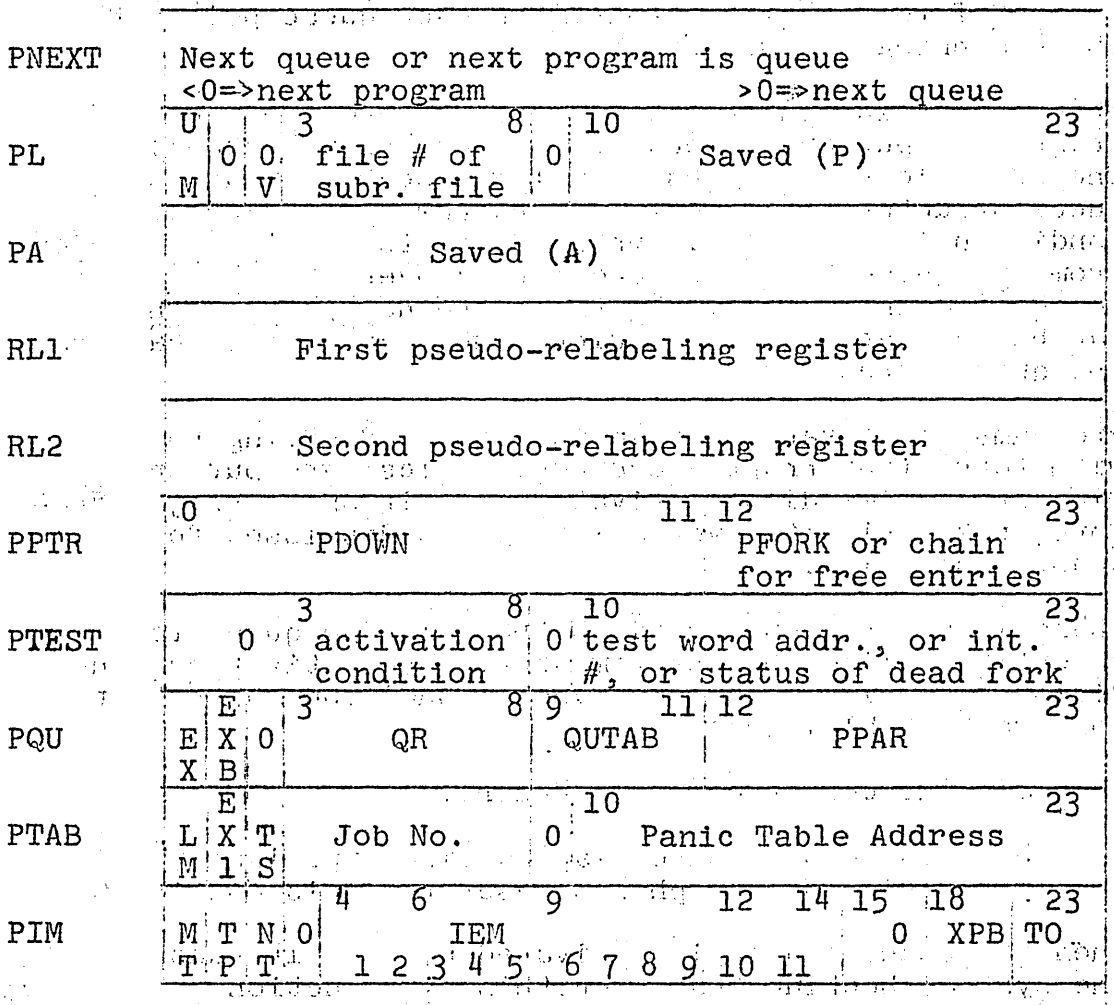
a) The cell PUCTR is non-zero,

b) ten seconds have elapsed since the last activation of the phantom user for this condition.

When the phantom user is activated by (b), it runs around the system checking that everything is functioning properly. In particular, it checks that the W-buffer has not been waiting for an interrupt for an unusual length of time, and that all teletype output is proceeding normally.

If the phantom user is activated by (a), it runs down the phantom user queue looking for things to do. A phantom user queue entry is displayed at the end of this section. It is essentially a very abbreviated PAC table entry. Such an entry is made when the system has some activity which it wants to carry out more or less independently of any user PAC table entry, tests for tape ready (on rewind) and card reader ready, and processing of escapes (an interrupt routine kind of activity, but too time-consuming). The second word of the entry is the activation condition. PUCTR contains the number of entries on the phantom user queue.

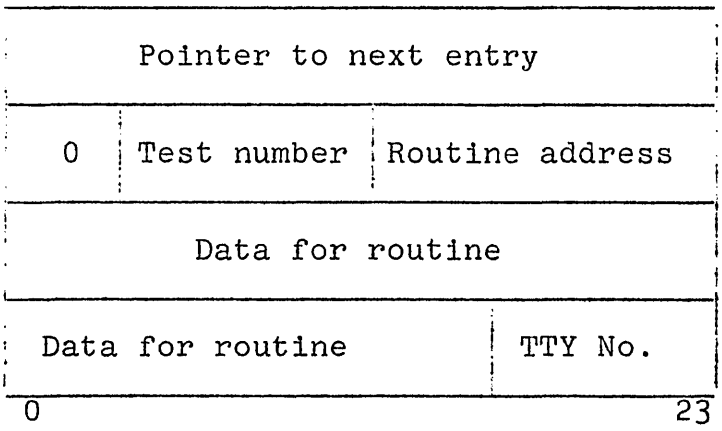
PAC TABLE - ONE PER FORK



UM= User Mode (1)
OV= Overflow
QUTAB= addr. for short quantum
EX=
QR= Amount of long quantum left
IEM= Interrupt Enabled Mask
LM= Local Memory
MT= Add no memory
EXB= Exec BRS
TP= Termination Pending
NT= Non-Terminability
TO= Time Out interrupt armed

IEM(11)= Interrupt on disc errors
IEM(1-4)= System interrupts
XPB= Index to PB in TS Block
EX1= Subsystem status
TS= TS block assigned

PHANTOM USER QUEUE ENTRY - TYMSHARE



PUCT - Phantom user queue.
FPULST - First free entry in PU queue.
PUBPTR - Pointer to first active entry. Last entry points to PUBPTR.
PUCTR - Number of PU entries.
PUEPTR - Last PU entry.

PUCTR1 - Entry counter uuring PU processing.
PUCPTR - Pointer to active entry during PU processing.
PUPAC - PACPTR of entry being processed by PU.

3.0 FORKS AND JOBS

3.1 Creation of Forks

A fork may create new, dependent, entries in the PAC table by executing BRS 9. This BRS takes its argument in the A register, which contains the address of a panic table, a seven-word table with the following format:

0 Program counter
1 A register
2 B register
3 X register
4 First relabeling register
5 Second relabeling register
6 Status

The status word may be:

-2 Dismissed for Input/Output
-1 Running
0 Dismissed on escape or BRS 10
1 Dismissed on illegal instruction panic
2 Dismissed on memory panic

The panic table address must not be the same for two dependent forks of the same fork, or overlap a page boundary. If it is, BRS 9 is illegal. The first six bits of the A register have the following significance:

0 Make fork executive if current fork is executive.
1 Set fork relabeling from panic table. Otherwise, use current relabeling.
2 Propogate escape assignment to fork (see BRS 90).
3 Make fork fixed memory. It is not allowed to obtain any more memory than it is started with.
4 Make fork local memory. New memory will be assigned to it independently of the controlling fork.
5 Make fork exec type 1 if current fork is exec.

When BRS 9 is executed, a new entry in the PAC table is created. This new fork is said to be a fork of the fork creating it, which is called the controlling fork. The fork is said to be lower in the hierarchy of forks than the controlling fork. The latter may itself be a fork of some still higher fork. A job may have, at most, eight forks including the exec. The A,B and X registers for the fork are set up from the current contents of the panic table. The address at which execution of the fork is to be started is also taken from the panic table. The relabeling registers are set up either from the current contents of the panic table or from the relabeling registers of the currently running program. An executive fork may change the relabeling as it pleases. A user fork is restricted to changing relabeling in the manner permitted by BRS 44. The status word is set to -1 by BRS 9. A fork number is assigned which is kept in PIM. This number is an index to the fork parameters kept in the TS block.

The fork structure is kept track of by pointers in PACT. For each fork PFORK points to the controlling fork, PDOWN to one of the subsidiary forks, and PPAR to a fork on the same level. All the subsidiary forks of a single fork are chained in a list. A complex situation is shown at the end of this section entitled "Fork Structure". The arrows indicate the various pointers.

If the fork executing a BRS 9 is a user fork, it is dismissed until the lower fork terminates. If it has exec status, it continues execution at the instruction after the BRS 9. The fork established by the BRS 9 begins execution at the location specified in the panic table and continues independently until it is terminated by a panic as described below. It is connected to its controlling fork in the following three ways:

1) The controlling fork may examine its state and control its operation with the following six instructions:

   BRS 30 reads the current status of a lower fork into the panic table. It does not influence the operation of the fork in any way.

BRS 31 causes the controlling fork to be dismissed until the lower fork causes a panic. When it does, the controlling fork is reactivated at the instruction following the BRS 31, and the panic table contains the status of the fork on its dismissal. The status is also put in X.

BRS 32 causes a lower fork to be unconditionally terminated and its status to be read into the panic table.

All of these instructions require the panic table address of the fork in A. They are illegal if this address is not that of a panic table for some fork.

BRS 31 and BRS 32 return the status word in the X register, as well as leaving it in the panic table. This makes it convenient to do an indexed jump with the contents of the status word. BRS 31 returns the panic table address in A.

BRS 106 causes the controlling fork to be dismissed until any subsidiary fork causes a panic. When it does, the controlling fork is reactivated at the following instruction with the panic table address in A, and the panic table contains the status of the fork at its dismissal.

BRS 107 causes BRS 30 to be executed for all subsidiary forks.

BRS 108 causes BRS 32 to be executed for all subsidiary forks.

2) If interrupt 3 is armed in the controlling fork, the termination of any subsidiary fork will cause that interrupt to occur. The interrupt takes precedence over a BRS 31. If the interrupt occurs and control is returned to a BRS 31 after processing the interrupt, the fork will be dismissed until the subsidiary fork specified by the restored (A) terminates.

3) The forks can share memory. The creating fork can, as already indicated, set the memory of the subsidiary fork when the latter is started. In addition, there is some interaction when the subsidiary fork attempts to acquire memory.

3.2 Memory Acquisition

If the fork addresses a block of memory which is not assigned to it, the following action is taken: A check is made to determine whether the machine size specified by the user has been exceeded. If so, a memory panic is generated. If the fork is fixed memory, a memory panic is also generated. Otherwise, a new block is assigned to the fork so that the illegal address becomes legal. For a local memory fork, a new block is always assigned. Otherwise, the following algorithm is used.

The number, n, of the relabeling byte for the block addressed by the instruction causing the memory trap is determined. A scan is made upwards through the fork structure to (and including) the first local memory fork. If all the forks encountered during this scan have Rn (the Nth relabeling byte) equal to 0, a new entry is created in PMT for a new block of user memory. The address of this entry is put into Rn for all the forks encountered during the scan.

If a fork with non-zero Rn is encountered, its Rn is propogated downward to all the forks between it and the fork causing the trap. If any fixed memory fork is encountered before a non-zero Rn is found, a memory panic occurs.

This arrangement permits a fork to be started with less memory than its controlling fork in order to minimize the amount of swapping required during its execution. If the fork later proves to require more memory, it can be reassigned the memory of the controlling fork in a natural way. It is, of course, possible to use this machinery in other ways, for instance to permit the user to acquire more than 16K of memory and to run different forks with non-overlapping or almost non-overlapping memory.

3.3 Panic Conditions

The three kinds of panic conditions which may cause

a fork to be terminated are listed in the description of the status word above. When any of these conditions occur, the PACT entry for the fork being terminated is returned to the free program list. The status of the fork is read into its panic table in the controlling fork. If the fork being terminated has a subsidiary fork, it too is terminated. This process will, of course, cause the termination of all the lower forks in the hierarchy.

The panic which returns a status word of zero is called a fork panic and may be caused by either of two conditions:

A) the escape button on the controlling teletype is pushed or an off interrupt occurred. This terminates some fork with a fork panic. A fork may declare that it is the one to be terminated by executing BRS 90. In the absence of such a declaration the highest user fork is terminated. When a fork is terminated in this way its controlling fork becomes the one to be terminated. If a user fork is terminated by escape, the teletype input buffer is cleared. If the controlling fork of the one terminated is executive, the output buffer is also cleared.

   If the fork which should be terminated by escape has armed interrupt 1, this interrupt will occur instead of a termination. The teletype buffers will not be affected. If there is only one fork active, control goes to the location EXECP in the executive. This consideration is of no concern to the user. Executive programs can turn the escape button off with BRS 46 and turn it back on with BRS 47. An escape occurring in the meantime will be stacked. A second one will be ignored. A program which is running with escape turned off is said to be non-terminable and cannot be terminated by a higher fork. BRS 26 skips if there is an escape pending.

   If two escapes occur within about .12 seconds, the entire fork structure will be cleared and the job left executing the top level executive fork. This device permits a user trapped in a malfunctioning lower fork to escape. Closely spaced escapes can be conveniently generated with the repeat button on the teletype. This

type of escape will cause a user to lose memory, and should be followed by a RESET. An off interrupt from the teletype is treated like a high-speed escape.

B) A BRS 10 may be executed in the lower fork. This condition can be distinguished from a panic caused by the escape button only by the fact that in the former case, the program counter in the panic table points to a word containing BRS 10.

As an extension of this machinery, there is one way in which several forks may be terminated at once by a lower fork. This may be done by BRS 73, which provides a count in the A register. A scan is made upward through the fork structure, decrementing this count by one each time a fork is passed. When the count goes to 0, the scan is terminated and all forks passed by are terminated. If an executive program is reached before the count is 0, then all the user programs below it are terminated.

The panic which returns a status word of 1 is caused by the execution of an illegal instruction in the fork. Illegal instructions are of two kinds:

1) Machine instructions which are privileged,
2) SYSPOPs which are forbidden to the user or which have been provided with unacceptable arguments.

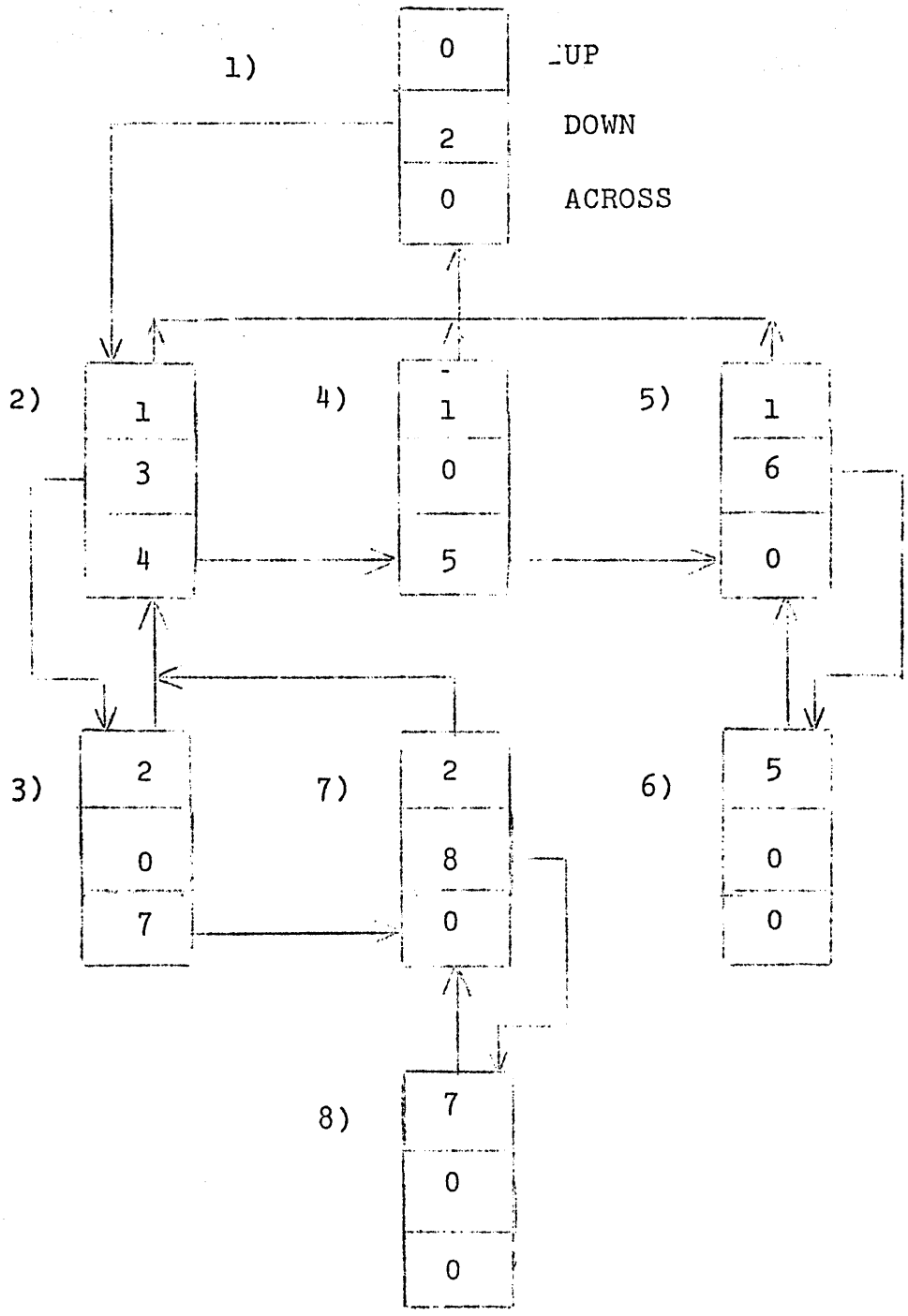
A status word of 2 is returned by a memory panic. This may be caused by an attempt to address more memory than is permitted by the machine size which the user has set, or by an attempt to store into a read-only page. If interrupt 2 is armed, it will occur instead of the memory panic.

3.4 Jobs

Every complete fork structure is associated with a job, which is the fundamental entity thought of as a user of the system, from the system's own point of view. The job number appears in the PAC table entry for every fork in the job's fork structure. In addition there are several tables indexed

by job number. These are displayed at the end of this section entitled "Job Tables", and indicate more or less what it is that is specifically associated with each job.

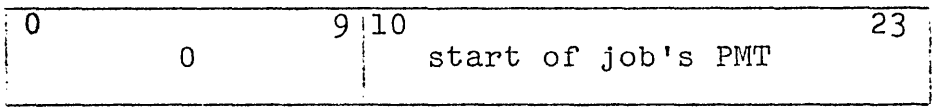
FORK STRUCTURE



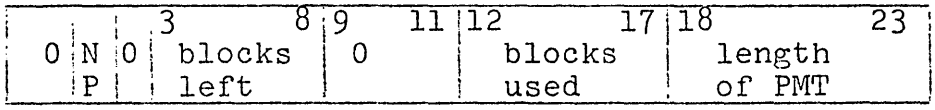
Hierarchy of Processes

JOB TABLES

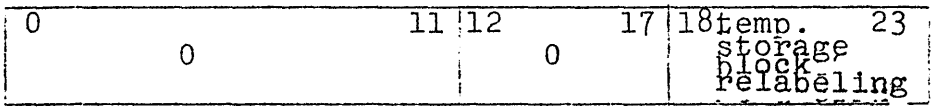
PMTP



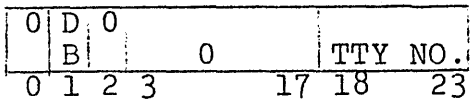
PMA



RL3



TTNO Teletype associated with this job



ETTB amount of CPU time used

NP = don't charge memory against machine size.

DB = disc busy bit for BRS BE+1,2

4.0 PROGRAM INTERRUPTS

A facility is provided in the monitor to simulate the existence of hardware interrupts. There are eleven possible interrupts; five are reserved for special purposes and six are available to the programmer for general use. A fork may arm the interrupts by executing BRS 78 with an 11-bit mask in the A register. This causes the appropriate bits in PIM to be set or cleared according to whether the corresponding bit in the mask is 1 or 0. Bit 4 of A corresponds to interrupt number1, etc. No other action is taken at this time. When an interrupt occurs (in a manner to be described) the execution of an SBRM* to location 200 plus interrupt number is simulated in the fork which armed the interrupt. Note that the program counter which is stored in this case is the location of the instruction being executed by the fork which is interrupted, not the location in the fork which causes the interrupt. The proper return from an interrupt is a BRU to the location from which the interrupt occurred. This will do the right thing in all cases including interrupts out of input-output instructions.

A fork may generate an interrupt by executing BRS 79 with the number of the desired interrupt in the A register. This number may not be one, two, three, four, or eleven. The effect is that the fork structure is scanned, starting with the forks parallel to the one causing the interrupt and proceeding to those above it in the heirarchy (i.e., to its ancestors). The first fork encountered during this scan with the appropriate interrupt mask bit set is interrupted. Execution of the program in the fork causing the interrupt continues without disturbance. If no interruptable fork is found, the interrupt instruction is treated as a NOP. Otherwise, it skips on return.

Interrupts 1 and 2 are handled in a special way. If a fork arms interrupt 1, a program panic (BRS 10 or escape key) which would normally terminate the fork which has armed interrupt 1, will instead cause interrupt 1 to occur, that is, will cause the execution of an SBRM* to location 201$_8$. This permits the programmer to control the action taken when the escape key is pushed without establishing a fork specifically for this purpose. If depressing the escape key causes an interrupt to occur rather than terminating a fork, the input buffer will not be cleared.

If a memory panic occurs in a fork which has armed interrupt 2, it will cause interrupt 2 to occur rather than terminating the fork. If an illegal instruction panic occurs in an executive fork which has armed interrupt 2, it will cause interrupt 2 to occur rather than terminating the fork.

Interrupt 3 is caused, if armed, when any lower fork terminates. Interrupt 4 is caused, if armed, when any input-output condition occurs which sets a flag bit (end of record, end of file and error conditions can do this).

Interrupt 11 is caused, if armed, if a disc error is encountered during a BRS BE+1 or BRS BE+2. These BRS's require system status. Consequently, interrupt 11 has no meaning for user or subsystem forks.

Whenever any interrupt occurs, the corresponding bit in the interrupt mask is cleared and must be reset explicitly if it is desired to keep the interrupt on. Note that there is no restriction on the number of forks which may have an interrupt on.

A fork may be interrupted after a specified period of time by issuing BRS BE+12. It takes the interrupt mask in A, the time in msec in B and the interrupt number in X. If the specified interrupt is armed when the time is up, the fork will be interrupted.

To read the interrupt mask into A, the program may execute BRS 49.

5.0 THE SWAPPER, MEMORY ALLOCATION AND RAD ORGANIZATION

Because of the necessity in various parts of the system for relabeling registers which do not change with time, the user has been denied any access to ordinary relabeling. In place, he is given access to so-called pseudo-relabeling. His pseudo-relabeling registers consist, as do the ordinary relabeling registers, of eight six-bit bytes. Each one of these bytes points, however, not to a real page of memory, but to an entry in the user's pseudo-memory table, PMT. This table may contain up to 64 words, each one specifying a certain 2K block of memory, herein referred to as a page. The first version of the system, however, will allow access to only 14 words. The possible forms of an entry in the pseudo-memory table are shown at the end of this section entitled "PMT Entries". All of the entries are more or less self-explanatory, except the second, which will be discussed in considerable detail later.

When it is necessary to activate a user, his pseudo-relabeling registers are used to read out the proper bytes from PMT and construct a list of pages which need to be read in from the RAD. When this list has been constructed, the current state of core is examined to determine whether any pages need to be written out to make room for those which must be read in. If so, a list of pages to be written out is constructed. The RAD command list is then set up with the appropriate commands to write out and read in the necessary pages. In the scan which sets up the RAD read commands, the swapper collects from PMT or SMT the actual absolute memory addresses of the page called for by the pseudo-relabeling and constructs a set of real relabeling registers which it puts in two fixed locations in the monitor (RRL1 and RRL2). It then outputs these relabeling registers to the hardware and activates the program.

Matters are slightly complicated by the existence of a system parameter called NCMEM. Pseudo-relabeling bytes with values from 1 to NCMEM-1 (0 means an unassigned page) actually refer directly to the first NCMEM-1 pages of SMT, the shared memory table and the user's own PMT is addressed beginning at NCMEM. the "common" portion of SMT is used to hold the most common subsystems.

There are two BRS's which permit the user to read and write his pseudo-relabeling. BRS 43 reads the current pseudo-relabeling registers into A and B. BRS 44 takes the contents of A and B and puts them into the current

pseudo-relabeling registers. An executive program may set the relabeling registers in arbitrary fashion by using this instruction. A user program, however, may add or delete only pages which do not have the executive bit set in PMT. This prevents the user from gaining access to executive pages whose destruction may cause damage to the system. Note that the user is doubly restricted in his access to real memory, firstly because he can only access real memory which is pointed to by his pseudo-relabeling, and secondly because he is only allowed to adjust those portions of his pseudo-relabeling which are not executive type.

The user can also set the relabeling of a fork when he creates it. See section 3. The same restrictions on manipulation of executive pages of course apply.

The system maintains a pair of relabeling registers which the executive and various subsystems think of as the user's program relabeling. For the convenience of subsystems, an executive program can read these registers with BRS 116 and set them with BRS 117.

The memory allocation algorithm is described in section 3. A user can release a page which is in his current relabeling by putting any address in that page into A and executing BRS 4. The PMT entry for the page is removed and in any other fork which has this PMT byte in its relabeling, the byte is cleared to 0.

Equivalent to BRS 4 is BRS 121, which takes a pseudo-relabeling byte in A rather than an address. An inverse operation is BRS 120, which takes a pseudo-relabeling byte in A, generates an illegal instruction trap if the corresponding PMT entry is occupied, and otherwise obtains a new page and puts it in that entry. This is an exec-only operation, of course.

A word of PMT whose first three bits are 001 contains a pointer to the shared memory table, SMT. An entry in SMT looks exactly like an unused or private entry in PMT. It refers to a page of memory which has a fixed location on the RAD and may be referred to by more than one program.

By putting an index in SMT in A and executing BRS 69, a pointer to the specified location in SMT is put into the first free byte of a user's PMT and the byte number is returned in A.

The user may declare a page read-only by executing BRS 80 with the pseudo-relabeling byte number of the page in A and with bit 0 of A set. To make a page read-write, bit 0 of A should be clear. Bit 0 of A will be reset if the page was formerly read-write or set if it was formerly read-only. If the program doing this is not an executive program, then the page must not be an executive page. Only an executive program can make a read-only PMT entry which points to SMT into a read-write entry, for obvious reasons. The significance of a read-only page to the swapper, of course, is that it need not be rewritten on the RAD when it is removed from memory.

A RAD is divided into blocks of 32K. Each user is assigned a block depending on his job number. The first page in each block is always the user's TS page. Each block of 32K consists of eight bands with two pages per band. The list of swapping commands alternates pages whenever possible to minimize swap time. A bit map is kept in the TS page which maps the user's 32K. When the user requires more memory the free page nearest the beginning of his block is taken. The first several blocks on the first RAD contain the subsystems, exec and swapable monitor pages.

It should be noted that whenever a user is reactivated, all of the memory in his current relabeling registers is brought in. The user does, however, have considerable control over precisely what memory will be brought in, because he can read and set his own relabeling registers. He may, therefore, establish a fork with a minimal amount of memory in order to speed up the swapping process if this is convenient.

To make a page executive, execute BRS 56 with the same argument as for BRS 80, make page read-only. This instruction is legal only for executive type programs.

The system keeps track of the state of real core with two tables called the real memory table (RMT) and the real memory use count table (RMC). An RMC entry is -1 if a page is not in use; otherwise it is one less than the number of reasons why it is in use. Every occurence of this page in the relabeling of a process which is running or about to be run counts as such a reason. In addition, other parts of the system can increment an RMC word to lock a page in core. No page with non-negative BRM can be released by the swapper.

The format of an RMT entry (one per real page) is:

U		2	9	10	23
S	R	0	0	address of PMT or SMT entry	
E	O			responsible	

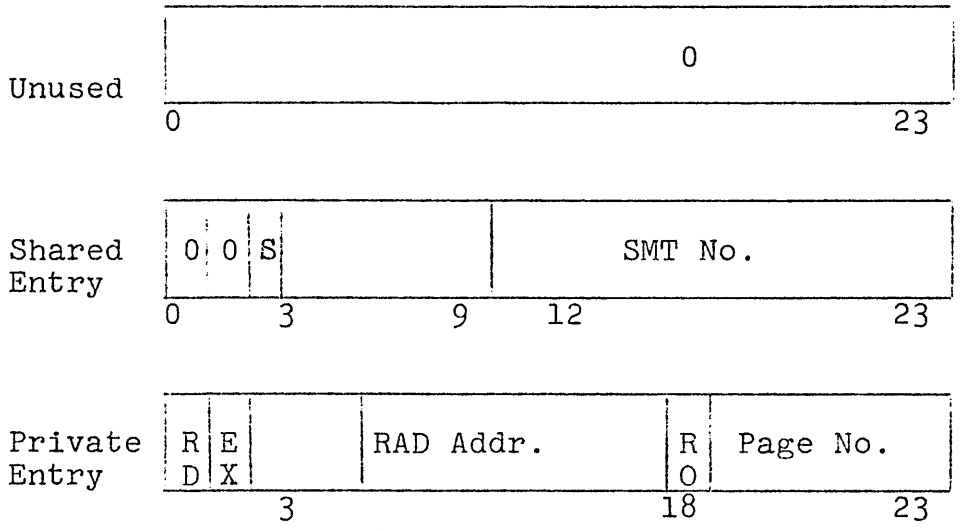
USE = in use RO = read only

There is one more table indexed by real memory, called the real memory aging table. Whenever the swapper is entered, every word in this table is shifted right one bit. All pages which show up in the real relabeling computed from the pseudo-relabeling with which the swapper was entered then have bit 1 turned on. The pages with lowest RMA are selected for swapping out; of course, their RMC entries must be negative.

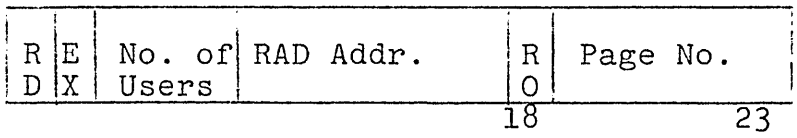
Tee swapper also contains a device called the simulated associative memory or SAM, which contains pseudo-relabeling and real relabeling for the most recently used maps. It serves to reduce the amount of time needed for map-changing when little swapping is taking place. It is cleared whenever a RAD read takes place, since this changes the contents of real memory and potentially invalidates all real relabeling registers.

Two BRS's exist for reading and writing pages at specified places on the RAD. They are of course restricted to executive programs. To read a page, put the RAD address into B and the core address in A and execute BRS 104. To write a page use BRS 105. RAD errors cause these instructions to generate illegal instruction panics.

PMT ENTRIES



SMT ENTRY



RD = On RAD
EX = Exec
S = Shared
RO = Read Only

6.0 MISCELLANEOUS FEATURES

A user may dismiss his fork for a specified length of real time by executing BRS 81 with the number of milliseconds for which he wishes to be idsmissed in A. At the first available opportunity after this time has been exhausted, his fork will be reactivated. The contents of A are lost by this BRS.

He can read the real-time clock into A and the system start-up date and time into B by executing BRS 42. The number obtained increments by one every 1/60th of a second. Its absolute magnitude is not significant. An exec fork can read the elapsed time counter for the user into A by executing BRS 88. This number is set to 0 when he enters the system and increments by 1 at every 1/60th second clock interrupt at which his fork is running.

To obtain the date and time, he can execute BRS 91. This puts string pointers into the A and B registers. The string contains in order, the month/day, hour (0-23): minute at which the instruction is executed.

A user may dismiss a fork until an interrupt occurs or the fork in question is terminated by executing BRS 109.

A fork can test whether it is executive or not by executing BRS 71. The type of executivity is returned in B. If B equals 1, the fork is subsystem. If B equals 0, the fork is user. If B equals -1, the fork is system and subsystem. If B equals -2, the fork is system. If B is negative, the BRS skips on return.

An executive fork can dismiss itself explicitly. See section 2.

There are two operations designed for so-called executive BRS's which operate in user mode with a map different from the one they are called from. BRS 111 returns from one of these BRS's, transmitting A, B and X to the calling fork as it finds them. BRS 122 simulates the addressing of memory at the location specified in A. If new memory is assigned, it is put into the relabeling to the calling fork. A memory panic can occur, in which case it appears to the calling fork that it comes from the BRS instruction.

An executive fork can cause an instruction to be executed in system mode by addressing it with EXS.

There are switches in the monitor which can be set by an exec fork with a BRS BE+13. It takes the new switch value in A and the switch number in X. It returns the old switch value in A.

An absolute location in the monitor relabeling can be read or changed by an exec fork with BRS BE+4. The absolute location is in X, the new value, if any, in A. The BRS reads if B is positive and changes the word if B is negative.

An exec fork can also force a new page to be read from the RAD with BRS BE+15. It takes an SMT pointer in A.

An exec fork can test the state of any breakpoint switch with BRS BE+7. The switch number is in X. The BRS skips if the switch is down.

An exec fork can crash the system with BRS BE+8.

7.0 TELETYPE INPUT-OUTPUT

We begin with an outline of the implementation of the teletype operations. This should serve to clarify the exact disposal of the characters which are being read and written. Every teletype has attached to it a table which is shown at the end of this section entitled "Teletype Table". Also associated with the teletype is a buffer which contains input and output characters in the following format:

0 7	8 15	16 23
input character	output character	character to echo (if any)

As characters are output by the program, they are added to the output buffer, which may be regarded as logically independent from the input buffer in spite of the fact that it resides in the same words. The characters are then output by the teletype interrupt routine as rapidly as the teletype will accept them.

These buffers are called character ring buffers (CRB's) and they are not necessarily associated with teletypes.

When a character is typed in on a teletype, it is converted to internal form and added to the input buffer unless it is escape on a controlling teletype. The treatment of escapes is discussed in section 3. The echo table address is then obtained from TTYTBL. The echo table determines what to echo and whether or not the character is a break character. The available choices of echos and break characters are discussed later in this section. If the character is a break character, and if a user's program has been dismissed for teletype input, it will be reactivated regardless of the number of words in the input buffer. In the absence of a break character, the user's program is reactivated only when the input buffer is nearly full.

If the teletype is in the process of outputting (TOS2>-1) then the character to be echoed is put into the last byte of the buffer word which contains the input character. When the character is read from the buffer by the program, the echo, if any, will be generated. This mechanism, called deferred echoing, permits the user to type in while the teletype is outputting without having his input mixed with the teletype output.

There are four standard echo tables in the system, referred to by the numbers 0, 1, 2 and 3. Zero is a table in which the echo for each character is the character itself, and all characters are break characters. Table 1 has the same echos, but all characters except letters, digits and space are break characters. Table 2 again has the same echos, but the only break characters are control characters (including carriage return and line feed) and exclamation mark. Table 3 specifies no echo for any character, and all characters are break characters. This table is useful for a program which wishes to compute the echo itself.

Normally a carriage return and line feed are both echoed if either is received from a teletype. However, only the first one received is sent to the program and if the other one is also received it is ignored. A program may, however, receive both by issuing BRS BE+11. If A is negative, both characters will be sent to the program. If A is positive, only the first character will be sent to the program.

If either line feed or carriage return is output by a program both are sent to the teletype unless the carriage is at the left margin. In this case, only a line feed is output for either a carriage return or a line feed. If a program wishes to send only one character, it should output 102B for line feed or 105B for carriage return.

To set the echo table, put the teletype number, or -1, in X and the echo table number in A and execute BRS 12. Note that BRS 12 is also used to turn on 8-level mode (see below). To read the echo table number into A, put the teletype number, or -1, in X and execute BRS 40. This operation returns the echo table number in A. If the teletype is in 8-level input mode, the sign bit of A is set and the terminal character is in A.

To input a character from the controlling teletype (the teletype on which the user of the program is entered) into location M in memory the SYSPOP

```
TCI   M   (teletype character input)
```

is used. This SYSPOP reads the character from the teletype input buffer and places it into the 8 rightmost bits of location M. The remainder of location M is cleared. The character is also placed in the A register, whose former contents are destroyed.

The contents of the other internal registers are preserved by this and all the other teletype SYSPOPS and BRS's.

To output a character from location M, the SYSPOP

```
TCO   M    (teletype character output)
```

is used. This instruction outputs a character from the rightmost eight bits of location M. In addition to the ordinary ASCII characters, all teletype output (other than 8-level) operations will accept 135 (octal) as a multiple blank character. The next character will be taken as a blank count, and that many blanks will be typed.

The TTYTIM cell in the teletype table is set to the current value of the clock whenever any teletype activity (interrupt or output SYSPOP) occurs. The top bit is left clear unless the activity is an escape input. This cell is checked by the escape processor to determine whether the escape should reset the job to the system exec. (See section 3.)

Every teletype in the system is at all times in one of two states:

a) It may be the controlling teletype of some user's program. It gets into this state when a user logs in on it. Controlling teletypes are also known as attached teletypes.

b) It may be completely free.

The status of the teletype is reflected by the contents of TTYASG. If the teletype is free, TTYASG contains 3777B. If it is a controlling teletype, TTYASG contains the PACPTR of the fork to terminate on escape.

A teletype becomes a controlling teletype when an "on" interrupt (from that line) is received by the computer. This indicates that someone has called that line. The user then has one and a-half minutes to log in before the system hangs up the line again. The system checks for carrier presence on a line before sending out any characters. To do this a system fork may issue BRS BE+3 with the line number to check in A.

The user may disconnect the line by hanging up the phone. BRS 112 is executed when an "off" interrupt is received

by the system or when a user logs out. If an "off" interrupt has been received, BRS 112 merely makes the line available again. However, if a user has logged out without hanging up the phone, BRS 112 makes the teletype the controlling teletype for another job immediately and the next user van log in without dialing the system again. BRS 112 takes the job number associated with the teletype in X. A job may terminate itself. This operation also releases all teletypes attached to the job. BRS 112 requires system status.

An exec fork can turn a line on or off by issuing BRS BE+6. It takes the line number in A and turns it on if B is negative or off if B is positive.

The user has considerable control over the state of the teletype buffers for the controlling teletype. In particular, he may execute the following BRS's. All these take the teletype number in X. Recall that -1 may be used for the controlling teletype.

```
BRS  11  clears the teletype input buffer.
BRS  29  clears the teletype output buffer.
BRS  13  skips if the teletype input buffer is
         empty.
BRS  14  waits until the teletype output buffer
         is empty, but not until the interrupt has
         been received for the last character.
```

Special provision is made for reading 8-bit codes from the teletype without sensing escape or soing the conversion from ASCII to internal which is done by TCI. To switch a teletype into this mode, execute

```
LDX  = teletype number
LDA  = terminal character + 40000000B
BRS    12
```

This will cause each 8-bit character read from the teletype to be transmitted unchanged to the user's program. The teletype can be returned to normal operation by

1) reading the terminal character specified in A, or

2) setting the echo table with BRS 12.

No echoes are generated while the teletype is in 8-level mode. Teletype output is not affected.

A parallel operation, BRS 85, is provided for 8-level output. BRS 86 returns matters to the normal state, as does any setting of the echo table.

To simulate teletype input, the operation

```
STI   =teletype number   or =-1
```

is available. STI puts the character in A into the input buffer of the specified teletype. Either the teletype number must be the controlling teletype or the fork issuing STI must be a system fork.

TELETYPE TABLE

TIS2 number of characters in input buffer

TMS4 next available space in input buffer (pointer)

TIS5 next filled space in input buffer (pointer)

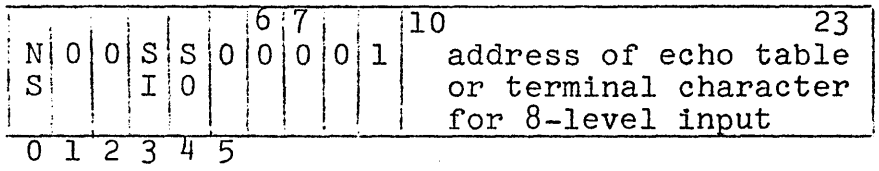
TOS2 number of characters in output buffer; -1 = inactive

TOS3 <0 = not in multiple blank mode; 400 = just saw 135 (multiple blank character); other = number of blanks

TOS4 next filled space in output buffer (pointer)

TOS5 next available space in output buffer

TTYTBL

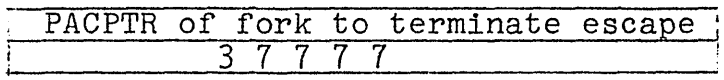


TTYFLG don't listen for input (excett escape) when 0. Set when input buffer is full.

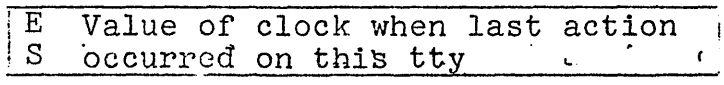
TTYBRK waiting for break character when -1

TTY Status

TTYASG



active
inactive



NS=not 8-level
SI=8-level input
SO=8-level output
ES=last action was input of escape

8.0 DISC AND BUFFER ORGANIZATION; DEVICES

8.1 File Storage on the Disc

The disc used by this system actually consists of from eight to 32 physical discs each with a movable arm. The arms have 64 positions numbered 0 to 63. Each arm position on each disc consists of 8192 words each, however, the files use the disc in groups of 256 words, thus disc addresses for file blocks are always MOD 4.

The disc is divided into two major sections, system date and file storage (see disc map at end of this section for disc layout). The organization of the system data area is discussed later in this section. The file storage area is divided into 256 word blocks which form the physical records for storage of files.

Every file has one or more index blocks which contain pointers to the data blocks for the file. An index block is a 256 word block, as are all other physical blocks in the file storage area. Only the first 128 words of the index block are used. A couple of additional words are used to chain the index blocks for any particular file, both forward and backward. The index blocks for a file contain the addresses for all the physical blocks used to hold information for the file.

Available storage in the file area of the disc is kept track of with a bit table. If a bit in this table is set, it indicates that the corresponding block on the disc is free. The bit map is set every time the system is brought up to agree with the files in the file directories. To set the bit map, BRS BE+4 is used. It requires an index block pointer (MOD4) in A. When all files have been checked, the BRS is called with a -1 in A, the new overflow pointer in B, and the accounting area address in X.

8.2 File Buffers

Every open file in the system with the exception of purely character-oriented files such as the teletype has a file buffer associated with it. The form of this buffer is shown at the end of this section entitled "Buffers".

Part (a) of this figure shows the buffer proper, and part (b) shows the index block buffer and pointers associated with it. Part (b) is used only by disc files, and is present in all cases.

The temporary storage page which is associated with each job is always the first entry in the job's PMT. This page is used to hold information about the user and for the systen's temporary storage for that user. It also has room for three buffers. The pseudo-relabeling for the TS page is held in a table called RL3 which is indexed by job number, and is put into the monitor map whenever any fork belonging to that job is run. The TS page is always relabeled into page 7.

Note that the amount of buffer space actually used is a function of the device attached to the file. In all cases, the two pointer words at the head of the buffer indicate the location of the data. The first word points to the beginning of the relevant data and is incremented as data are read from an input buffer. The second word points to the end of the data and is incremented as data are written into an output buffer. When the buffer is in a dormant state, both words point to the first word of the buffer. Whenever any physical I/O operation is completed, the first pointer contains the address of this word.

8.3 Devices

Every different kind of input-output device attached to the syttem has a device number. The numbers assigned to specific devices are given in section 9. The various tables indexed by device number are described here. The entries in these tables addressed by a specific device number together with the unit number (if any) and the buffer address, completely define the file. All this information is kept in the file control block (see section 9). which is addressed by the file number.

The tables indexed by device number are shown at the end of this section entitled "Device Tables". Note the multiplicity of bits which specify the characteristics of the device. A device may be common (shared by users, who must not access it simultaneously; e.g., tape or cards) or not common

(e.g. disc); this characteristic is defined by NC. It may have units; e.g., there may be multiple mag tapes. The U bit specifies this. The DIU word indicates thich file is currently monopolizing the device; in the case of a device with multiple units, DIU points to a table called ADIU which contains one word for each unit.

The major parameters of a device are:

the opening routine, which is responsible for the operation necessary to attach it to a file,

the GPW routine, which performs character and word I/O,

the BIO routine, which performs block I/O.

The minor parameters are:

maximum legal unit number,

physical record size (determining the proper setting of buffer pointers and interlace control words for the channel),

the expected time for an operation; the swapper uses this number to decide whether it is worthwhile to swap the user out while it is taking place.

8.4 System Data is Kept on the Outer Arm Positions of the Disc

Arm positions 62 and 63 contain systems which are loaded by a special routine which is kept on paper tape. This routine dumps the first 32K of core on discs 0 and 1, then reads a new system into the first 16K of core. The disc from which the new system is read is determined by console switch settings.

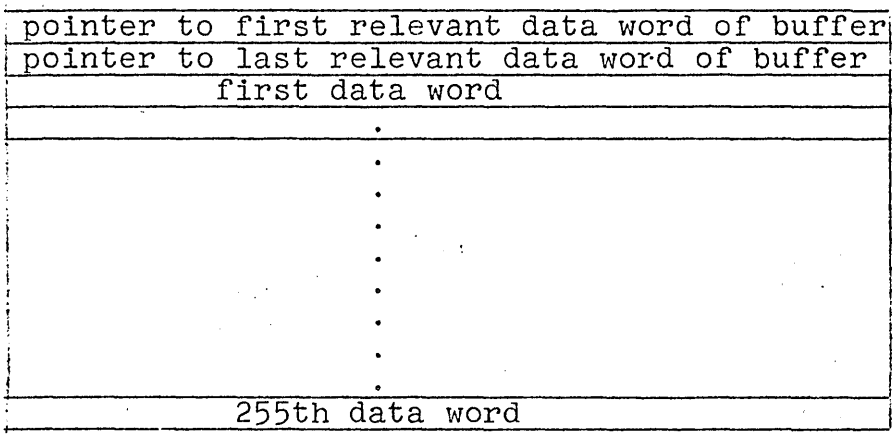
Arm positions 0 and 1 contain the file directories, accounting information, and mailbox data. These are explained in the TSS Executive Reference Manual.

There are four BRS's available to system level forks to read and write the system data on the disc. These are BRS BE+1, BRS BE+2, BRS BE+9 and BRS BE+10. They require the core address in A and the

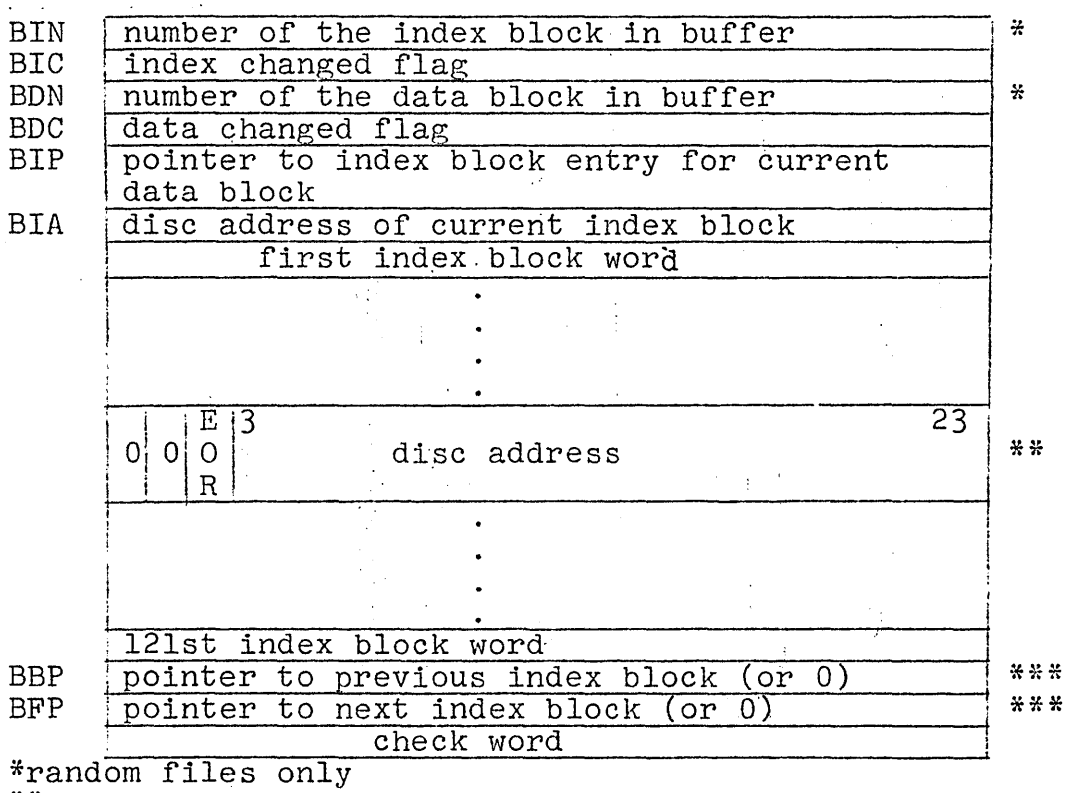
disc address in B. In addition BRS BE+1 and BRS BE+2 take the word count in X. BRS BE+9 and BRS BE+10 always read or write a page (2K) from or to the disc.

BUFFERS

(a) Layout of a File Buffer



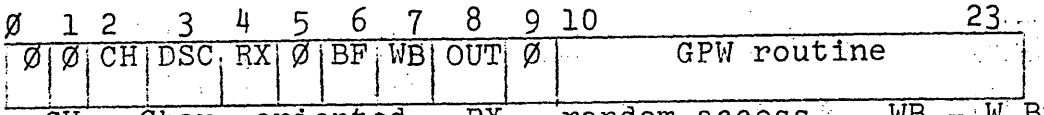
(b) Layout of Index Block Buffer and Associated Pointers for a Disc File



*random files only
**index block word format. EOR=end of record flag.
***always 0 for sequential files

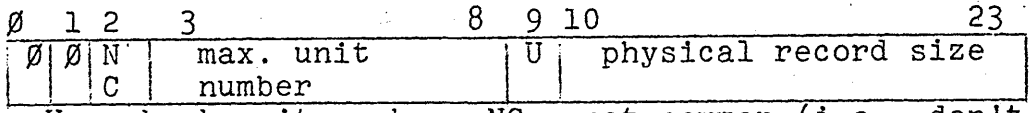
DEVICE TABLES

DEV word or character I/O routine



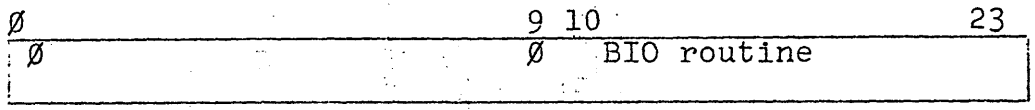
CH - Char. oriented RX - random access WB - W Buffer
DSC - Disc BF - requires buffer OUT - output

BUFS buffer size

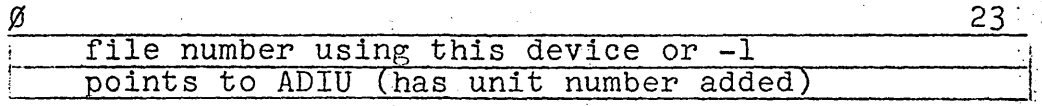


U - check unit number NC - not common (i.e., don't set DIU)

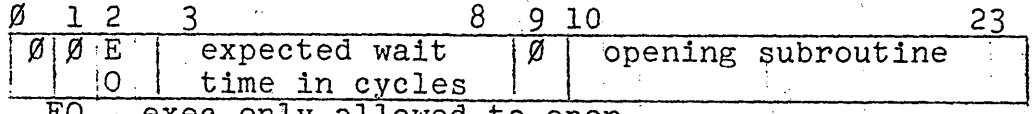
BDEV Block I/O routine



DIU device in user

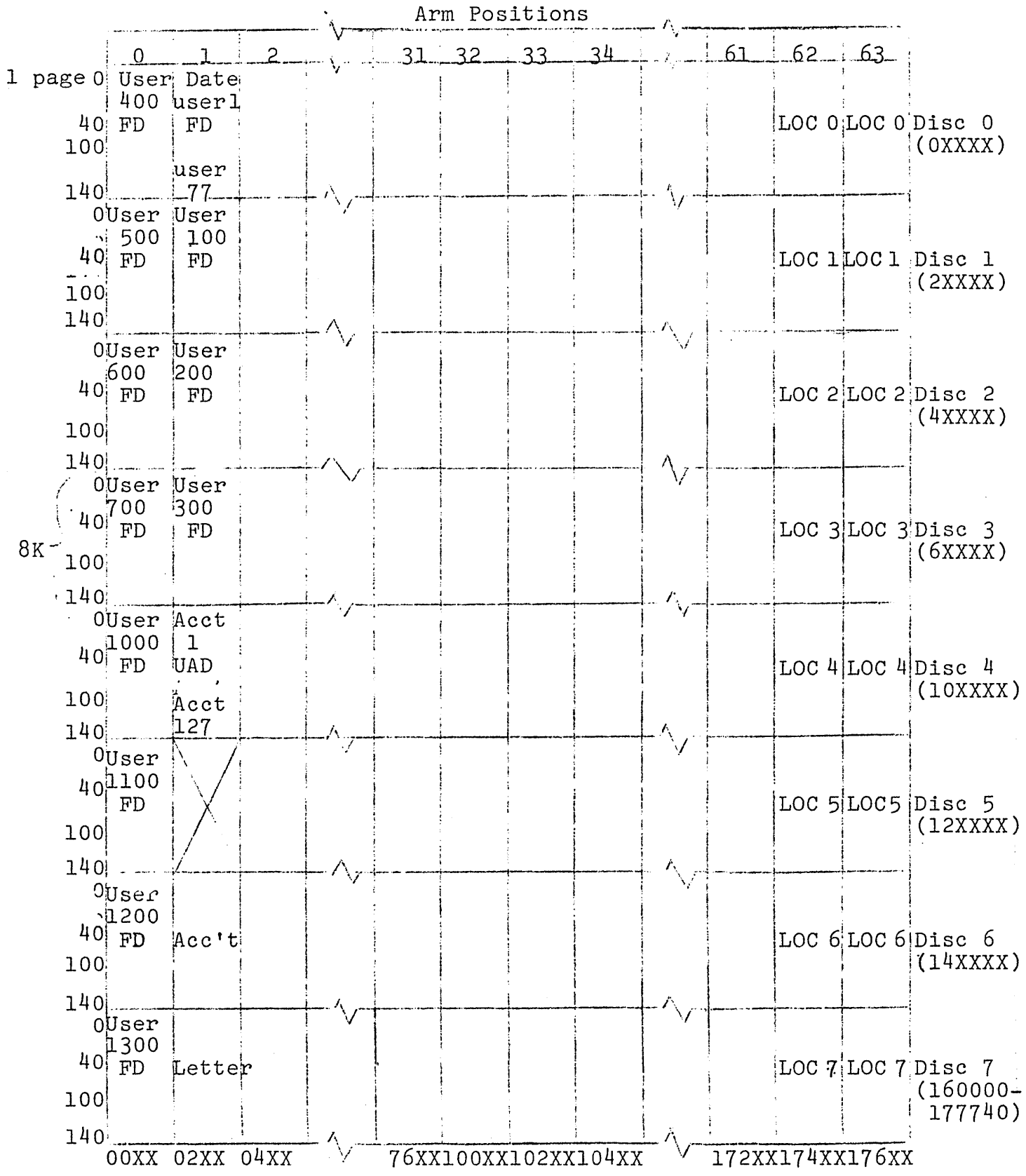


OPNDEV opening routine



EO - exec only allowed to open

DISC MAP


Arm Positions
0 1 2 31 32 33 34 61 62 63
1 page
0 User 400 FD
40
100
140
Date user1 FD
user 77
LOC 0 LOC 0 Disc 0 (0XXXX)
0 User 500 FD
40
100
140
User 100 FD
LOC 1 LOC 1 Disc 1 (2XXXX)
0 User 600 FD
40
100
140
User 200 FD
LOC 2 LOC 2 Disc 2 (4XXXX)
0 User 700 FD
40
100
140
User 300 FD
LOC 3 LOC 3 Disc 3 (6XXXX)
8K
0 User 1000 FD
40
100
140
Acct 1 UAD
Acct 127
LOC 4 LOC 4 Disc 4 (10XXXX)
0 User 1100 FD
40
100
140
LOC 5 LOC5 Disc 5 (12XXXX)
0 User 1200 FD
40
100
140
Acc't
LOC 6 LOC 6 Disc 6 (14XXXX)
0 User 1300 FD
40
100
140
Letter
LOC 7 LOC 7 Disc 7 (160000-177740)
00XX 02XX 04XX 76XX100XX102XX104XX 172XX174XX176XX

9.0 SEQUENTIAL FILES

9.1 Sequential Disc Files

There are two basically different kinds of files which the user may write on the disc, sequential and random. A sequential file has a structure very similar to that of an ordinary mag-tape file. It consists of a sequence of logical records of arbitrary length and number. Disc sequential files are, however, considerably more flexible than corresponding files on tape, because logical records may be inserted and deleted in arbitrary positions and increased or decreased in length. Furthermore, the file may be instantaneously positioned to any specified logical record.

A sequential disc file may be opened by the following sequence of instructions:

```
LDX   =device number, 8 (input) or 9 (output)
LDA   Address of first index block
BRS   1
```

If the file is opened successfully, the BRS skips; otherwise it returns without skipping. Use of this BRS is restricted to users with system status. User programs may access disc files only through the executive file handling machinery. BRS 1 can also be used to open other kinds of files (see section 9.2).

If BRS 1 fails to skip, it returns in the A register an indication of the reason:

-2 too many files open -- no file control blocks or no buffers available.

-1 device already in use. For the disc, produced by an attempt to open a file for output twice.

0 No disc space left. This inhibits opening of output files only.

BRS 1 returns in the A register a file number for the file. This file number is the handle which the user has on the file. He may use it to close the file when he is done with it by putting it in the A register and executing BRS 2. This releases the file for other uses. BRS 2 is available to both user and executive programs.

To close all his open files the user may execute BRS 8.

If the sign bit of A is set when the BRS 1 is executed, the file is made read-only. This means that it cannot be switched from input to output. If this bit is not set, then the instructions:

```
LDA   =file number
LDB   =1
BRS   82
```

will change the file to an output file regardless of its initial character. The instructions:

```
LDA   =file number
LDB   =0
BRS   82
```

are always legal and make the file an input file regardless of its initial character.

Three kinds of input-output may be done with sequential files. Each of these is specified by one SYSPOP. Each of these SYSPOP's handles input and output indifferently, since the file must be specified as an input or an output file when it is opened. A file that is open for output cannot be opened again for either input or output and a file that is open for input cannot be opened for output. However, a file may be opened for input any number of times.

To input a single character to the A register or output it from the A register, the instruction:

```
CIO   =file number
```

is executed. On input an end of record or end of file condition will set bits 0 and 8 or bits 0 and 7 in the file number (these are called flag bits) and return a 134_8 or 137_8 character, respectively. In interrupt 4 is armed, it will occur. The end of record condition occurs on the next input operation after the last character had been input. The end of file condition occurs on the next input operation after the end of record, which signals the last record of the file. The user may generate an end of record while writing a file by using the control operation to be described. An error condition sets bits 0 and 6 in the file number.

To input a word to the A register or output it from the A register,

```
WIO   =file number
```

is executed. An end of file condition returns a word of three 137_8 characters.

Mixing word and character operations will lead to peculiarities and is not recommended.

To input a block of words to memory or output them from memory, the instructions:

```
LDX   =first word address
LDA   =number of words
BIO   =file number
```

should be executed. The contents of A, B and X will be destroyed. The A register at the end of the operation contains the first memory location not read into or out of.

If the operation causes any of the flag bits to be set, it is terminated at that point and the instruction fails to skip. If the operation is completed successfully, it does skip. Note that a BIO cannot set both the EOR and the EOF bits.

BIO is implemented with considerable efficiency.

The flag bits of the file number are set by the system whenever end-of-record (0 and 8) or end-of-file (0 and 7) is encountered and cleared on any input-output operation in which neither of these conditions occurs. Bit 0 is set on any unusual condition. In the case of a BIO the A register at the end of the operation indicates the first memory location not read into or out of. For any input operation, the end of record bit (bit 8) of the file number may be set. An output operation never sets either one of these bits. Bits 0 and 6 of the file number may be set on an error condition. Whenever any flag bit is set as a result of an input-output operation in a fork, interrupt 4 will occur in that fork if it is armed.

The CTRL SYSPOP provides various control functions for sequential disc files. To use this operation execute the instructions:

```
LDA    =control number
LDB    =count, (if required)
CTRL   =file number
```

The available control numbers are:

1 write end of record on output or skip the remaining part of the logical record on input. This control does not take a record count.

2 backspace (B) physical tape blocks.

3 forward space (B) physical tape blocks.

4 delete (B) tape blocks (legal on output only).

5 space to end of file and backspace (B) physical tape blocks.

6 space to beginning of file and forward space (B) physical tape blocks.

7 insert logical record (legal on output file only). This control does not require record count.

8 write end of file (output only).

A program may delete all the information in a disc file by executing the instructions:

```
LDA   =file number
BRS   66
```

The index block for a sequential disc file contains one word for each physical record in the file. This word contains the address on the disc of the physical record in the bottom 21 bits. Bit 2 is set if the physical record is the last record of a logical record. A sequential file may have only one index block, or a maximum of 121 X 255 = 30855 words of data.

Putting the file number of a sequential file in A and executing BRS 113 will cause the file to be scanned to find the total number of data words. The number of data words is added to X. This also

works for random files.

Three operations are available to executive programs only. They are intended for use by the system in dealing with file names and executive commands.

A new disc file with a new index block can be c created by BRS 1 with an index block number of 0 in A. The file number is returned in A as usual and the index block number in X. The read-only bit may be set (bit 0 of A) as usual.

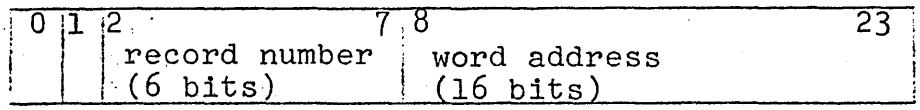
```
BRS   67
```

returns the index block with address in A to available storage. An exec fork may read an index block into core with

```
BRS 87.
```

It takes the address of the block in A and in X the first word in core into which the block is to be read.

A single word of a sequential file may be directly addressed by specifying the logical record number and word number within the logical record. All the operations legal for random files (see section 10) can also be used for sequential files with this convention. The format of the address is



9.2 Other Sequential Files

In addition to disc sequential files, the user has some other kinds of sequential files available to him. These are all opened with the same BRS 1:

```
LDX   =device number
LDA   =unit number
BRS   1
```

Available device numbers are:

paper tape input	1
paper tape output	2
magtape input	4
magtape output	5
card punch Hollerith	6
card punch binary	7
line printer output	11
card input Hollerith	12
card input binary	13

The device number is put into X. The unit number, if any, is put into A. The file number for the resulting open file is returned in A. If BRS 1 fails it returns an error condition in A as described in section 9. Three error conditions apply to magtape only:

0	Tape not ready
1	Tape file protected (output only)
2	Tape reserved

BRS 1 is inverted by BRS 110, which takes a file number in A and returns the corresponding device number in X and unit number in A.

These files may also be closed and read or written in the same manner as sequential disc files. The magtape is not available to the user as a physical device.

CTRL =1 (end of record)

Is available for physical sequentail files 3 and 5 (paper tape and magtape output). Several other controls are also available for magtape files only. These are:

2	backspace block
3	forward space file
4	backspace file
5	write three inches blank tape
6	rewind
7	write end of file
8	erase long gap

These controls may be executed only by executive type programs. I/O operations to the magtape may, of course, be executed by user programs if they have the correct file number.

An executive program may allocate a tape unit to itself by putting the unit number in A and executing BRS 118, which skips if the tape is not already attached to some other job. BRS 119 releases a tape so attached.

It is possible for magtape and card reader files to set the error bit in the file number. The first I/O instruction after an error condition will read the first word of the next record -- the remainder of the record causing the error is ignored. The magtape routines take the usual corrective procedures when they see hardware error flags, and signal errors to the program only as a last resort.

In order to make the card reader look more like other files in the system, the following transformations are made by the system on card input:

L) All non-trailing strings of more than two blanks are converted to a 135 character followed by a character giving the number of blanks. The teletype output routines will decode this sequence correctly.

2) Trailing blanks on the card are not transmitted to the program.

3) The card is not regarded as a logical record. However, the system generates the character 155 (carriage return) at the end of each card.

The result of all this machinery is that the string of characters obtained by reading in a card deck may be output without change to a teletype and will result in a correct listing of the deck.

Whenever a card reader error (feed check or validity check) occurs, the program is dismissed until the reader becomes not ready.

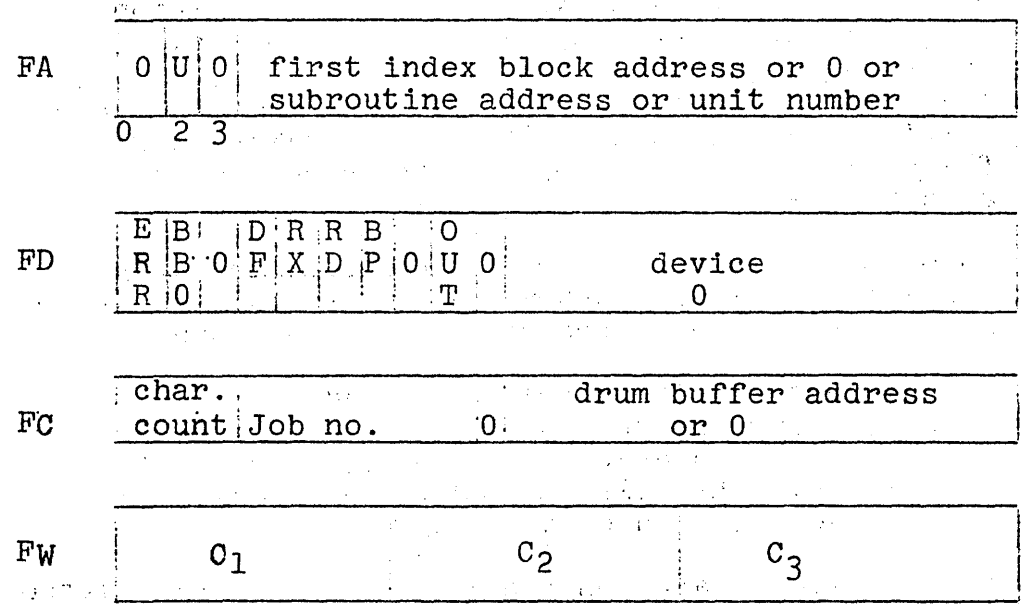
The EOF light is sensed as an end of file at all times.

The phantom user's ten second routine checks to see whether a W-buffer interrupt has been pending for more than ten seconds. If so it takes dtastic and ill-defined action to clear the W-buffer. BRS 114 also takes this drastic action; it can be used if a program is aware that the W-buffer is malfunctioning.

9.3 File Control Blocks

Every open file in the system has associated with it a file control block. This block consists of four words in the following format:

FILE CONTROL BLOCK 1.85 - TYMSHARE



C_n = word being packed or unpacked
Char. count = -1 to 2
CH = character oriented
OUT = output
BB = buffer busy
DF = disc file
*RX = random access
*RD = read only
BP = buffer in use and protected
ERR = error
U = unused

*Disc files only

9.4 Permanently Open Files

There are a few built-in sequential files with fixed file numbers:

0	controlling teletype input
1	controlling teletype output
2	nothing (discard all output)
1000+n	input from teletype n
2000+n	output to teletype n

These files need not be opened and cannot be closed.

9.5 Character Buffers

Section 7 describes the format of a teletype buffer. These buffers are capable of dealing with any character-oriented device, not merely with a teletype. For this reason the character ring buffers are not directly indexed by the physical number of the teletypes to which they are associated. Instead, a table indexed by physical teletype number is used to obtain the buffer number. It is possible for other devices to obtain buffers; the mechanism for doing this is not spelled out in detail at the moment.

10.0 RANDOM DISC FILES

A random disc file is very similar in physical structure on the disc to a sequential disc file. The only major difference is that there are no logical records and that the bits in the index block which keep track of logical record structure are always 0. Furthermore, the non-zero words of the index block are not necessarily compact. The reason for this is that information is extracted from or written into a random file by addressing the specific word or block of words which is desired. From the address which the user supplies, the system extracts a physical block number by dividing by 255 and a location of the word within the block which is the remainder of this division. Further division by 121 yields the appropriate index block. A random file may have any number of index blocks.

A random file may be opened by using BRS 1 with a device number 10. No distinction is made between input and output to a random drum file. A random file may also be closed by BRS 2, like any sequential file. However, CIO, WIO and BIO are not used for input-output to random files.

Instead, the following operations are available:

To read a word from a random file, execute the instructions:

```
LDB   =address
DWI   =file number
```

The word is returned in A.

To write a word on a random file, put the word in A and execute the instructions:

```
LDB   =address
DWO   =file number
```

Block input-output to random files is also possible. To input a block, execute the instructions:

```
LDX   =first word address
LDA   =number of words
LDB   =first address in file
DBI   =file number
```

To output a block of words to a random file, execute the instruction:

```
DBO   =file number
```

with the same parameters in the central registers. These block input-output operations are done directly to and from the user's memory, as is BIO. Disc buffers are not involved and the operation can go very quickly.

If the sign bit of A was set when BRS 1 was executed to open the file, then output to it is not allowed and the file is said to have been made read-only. This is a natural extension of the treatment of read-only sequential files.

It is possible to define a random file which has been previously opened as the secondary memory file. To do this, execute the instructions:

```
LDA   =file number
BRS   58
```

The specified file remains the secondary file until another secondary memory file is defined or until the file is closed. To access information in the secondary memory, two SYSPOPS are provided. These POP's work exactly like DWI and DWO except that they take the disc address from memory instead of requiring it to be in B. To read a word of secondary memory into the A register, the instruction:

```
LAS   address
```

should be executed. To store a word from A into the secondary memory, the instruction:

```
SAS   address
```

should be executed. The word addressed by either one of these SYSPOP's should contain the disc address which is to be referenced. This word may also have the index bit set, in which case the contents of the index register will be added to the contents of the word to form the effective address which is actually used to perform the input-output operation.

The mechanism for acquiring and releasing random disc file space is very similar to the mechanism for allocation of core memory. Whenever the user addresses a

section of a random disc file which he has not previously used, the necessary blocks are created and cleared to 0. Note that the user should avoid unnecessarily large random drum addresses, since they may result in the creation of an unnecessary number of index blocks.
To release random disc memory, execute the instructions:

```
LDA   =number of words to be zeroed
LDB   =initial word to be zeroed
LDX   -file number
BRS   59
```

The specified section of the file is cleared to zero. Physical blocks which are entirely zero will be released. A more drastic clearing operation may be obtained with BRS 66, which deletes the entire information content of the file.

11.0 SUBROUTINE FILES

In addition to the above-mentioned machinery for performing input-output through physical files, a facility is provided in the system for making a subroutine call appear to be an input-output request. This facility makes it possible to write a program which does input-output from a file and later to cause further processing to be performed before the actual input-output is done, simply by changing the file from a physical to a subroutine file. A subroutine file is opened by executing the instructions:

```
LDX   parameter word
BRS   1
```

This instruction never skips. The opcode field of the parameter word indicates the characteristics of the file. It may be one of the following combinations:

110 00000(octal)	Character input subroutine
111 00000(octal)	Character output subroutine
010 00000(ocatl)	Word input subroutine
011 00000(octal)	Word output subroutine

I/O to the file may be done with CIO or WIO, regardless of whether it is a word or a character oriented subroutine. The system will take care of the necessary packing and unpacking of characters. BIO is also acceptable.

The opening of a subroutine file does nothing except to create a file control block and return a file number in the A register. When an I/O operation on the file is performed, the subroutine will be called. This is done by simulating an SBRM to the location given in the word following the BRS 1 which opened the file. The contents of the B and X registers are transmitted from the I/O SYSPOP to the subroutine unchanged. The contents of the A register may be changed by the packing and unpacking operations necessary to convert from character-oriented to word-oriented operations or vice versa. The I/O subroutine may do an arbitrary amount of computation and may call on any number of other I/O devices or other I/O subroutines. A subroutine file should not call itself recursively.

When the subroutine is ready to return, it should execute BRS 41. This operation replaces the SBRR which would normally be used to return from a subroutine call. The contents of B and X when the BRS 41 is executed

are transmitted unchanged back to the calling program. The contents of A may be altered by packing and unpacking operations. A subroutine file is closed with BRS 2 like any other file.

In order to implement BRS 41, it is necessary to keep track of which I/O subroutine is open. This information is kept in 6 bits of the PAC table. The contents of these 6 bits is transferred into the opcode field of the return address when an I/O subroutine is called, and is recovered from there when the BRS 41 is executed.

12.0 EXEC TREATMENT OF FILES

The user's only access to files is through the system executive. The executive provides a connection between a symbolic name for a file, which is created by the user, and the file number which the user must have in order to execute input-output operations. This connection is established through the file directory. Supplementary to this function is the need to prevent the user from destroying other people's files.

We begin with a description of the file naming system as it appears to the user, and continue with a description of the executive tables which implement the various features.

A user may give his files arbitrary names containing any characters other than ' or /. The names of new disc files must be surrounded by /, and the names of new tapes files must be surrounded by '. When a file is created it's name must be enclosed within one or the other of these characters.

When a user types a file name not enclosed within slashes, or quotes he need only type enough characters of the name to determine it uniquely. If the user starts an output file name with a quote or slash, he must type the entire name. If it is an output file name and not already in his file directory, a new file will be created. In any other context, a name not in the file directory is in error.

When an output file name is being typed, the system, after determining the name, will type out either OLD FILE or NEW FILE and await a confirmation that the name has been given correctly. If the user types either of the characters, line feed or carriage return, the name will be regarded as correct. Any other character will be regarded as an indication that the name was incorrect. This machinery is intended to make it more difficult for the user to destroy old files or create new ones inadvertently.

When a new slashed output file name is given to the system, a new entry in the file directory and a new index block on the disc are created for it. If the name is being given to an executive command, it will be assumed that the file is a sequential one.

It is possible for the user to reference files belonging to users other than himself if the file name contains a control character or an @. He does this by preceding the file name with the account number and user name enclosed in parentheses. Thus, to get at file /@PROGRAM/ belonging to user JONES, he might type:

```
(A1;JONES)   /@PROGRAM/
```

Jones may control the extent to which other users can access his files. For another user to reference a file, the name must contain at least one control character or an @.

Files in a Public File Directory may be accessed by typing the file name in quotes:

```
"PROGRAM".
```

The previous paragraphs have described the behavior of the system's file naming logic when it is recognizing names typed in on a teletype. The BRS's which recognize file names are capable, however, of accepting them in many other ways. Essentially, they accept a string pointer to the portion of the name already known (which may be null) and file numbers for the input file to be used in obtaining the rest of the name and the output file on which the name should be completed. In most cases the first or the second of these items will be irrelevant.

A program may open a disc file and obtain a file number by executing BRS 15 and BRS 16 (input) or BRS 18 and 19 (output). BRS 15 and BRS 18 expect to get the file name from the teletppe. If the name is known to the rrogram, they may be replaced by BRS 48. These BRS's are used as follows:

```
LDA   =file number
BRS   15 (or BRS 18)
EXCEPTION RETURN
NORMAL RETURN
```

The normal return leaves a file directory pointer in A, and BRS 18 leaves the character typed after OLD FILE/ NEW FILE in B. If no character was read, B contains a -1. The X register is modified.

```
LDA    =file directory pointer
LDX    =file type (BRS 19 only)
BRS    16 (or BRS 19)
EXCEPTION RETURN
NORMAL RETURN
```

The normal re urn leaves a file number in A, and BRS 16 leaves the file type in B. X is modified.

There are four standard file types:

1 File written by executive save command (sequential)
2 General binary file (sequential)
3 Symbolic file (sequential)
4 Dump file (sequential)

BRS 48 or 60 may be substituted for BRS 15 or 18. BRS 48 is used if the name is in the file directory and BRS 60 will create a new name if necessary.

```
LDP    =string pointers(1)
BRS    48 or 60
EXCEPTION RETURN
NORMAL RETURN
```

The string pointers point to the file name to be looked up in the file directory. The normal return leaves a file directory pointer in A. All other registers are modified. If the file name cannot be located in the file directory, the BRS 48 takes the exception return, while the BRS 60 will attempt to place the new name in the file directory; if it is unable to do so because the file directory is full, it will take the exception return.

(1) A string pointer is a character address found by multiplying the word address by three and adding 0, 1 or 2. The string pointer in A points to the character before the beginning of the file name. The pointer in B points to the last character of the name.

ARPAS assembles string pointers as follows for string pointers P1 and P2:

```
P1   DATA   (R) Z-1
P2   DATA   (R) Z+2
Z    ASC    '/T/'
```

It is possible for a user to rename his files by typing:

```
RENAME   /PROGRAM/   as ROUTINE
```

The rename logic protects the user against creating file names that conflict with existing file names or with the file type.

The file directory consists of an SPS hash table together with a table of equal length, called the description table (DBT), which has a three-word entry corresponding to each three-word entry in the hash table. In addition, there is a string storage area for storing file names and a few words of miscellaneous information. The parameters of a file directory are shown at the end of this section entitled "File Directory Arrangement" and the format of a single hash table entry and matching DBT entry is also shown at the end of this section entitled "Hash Table Entry". Executive commands for examining the file directory and setting various bits are described in section 13. In addition, a number of BRS's are provided which permit the user's program to affect the contents of the file directory.

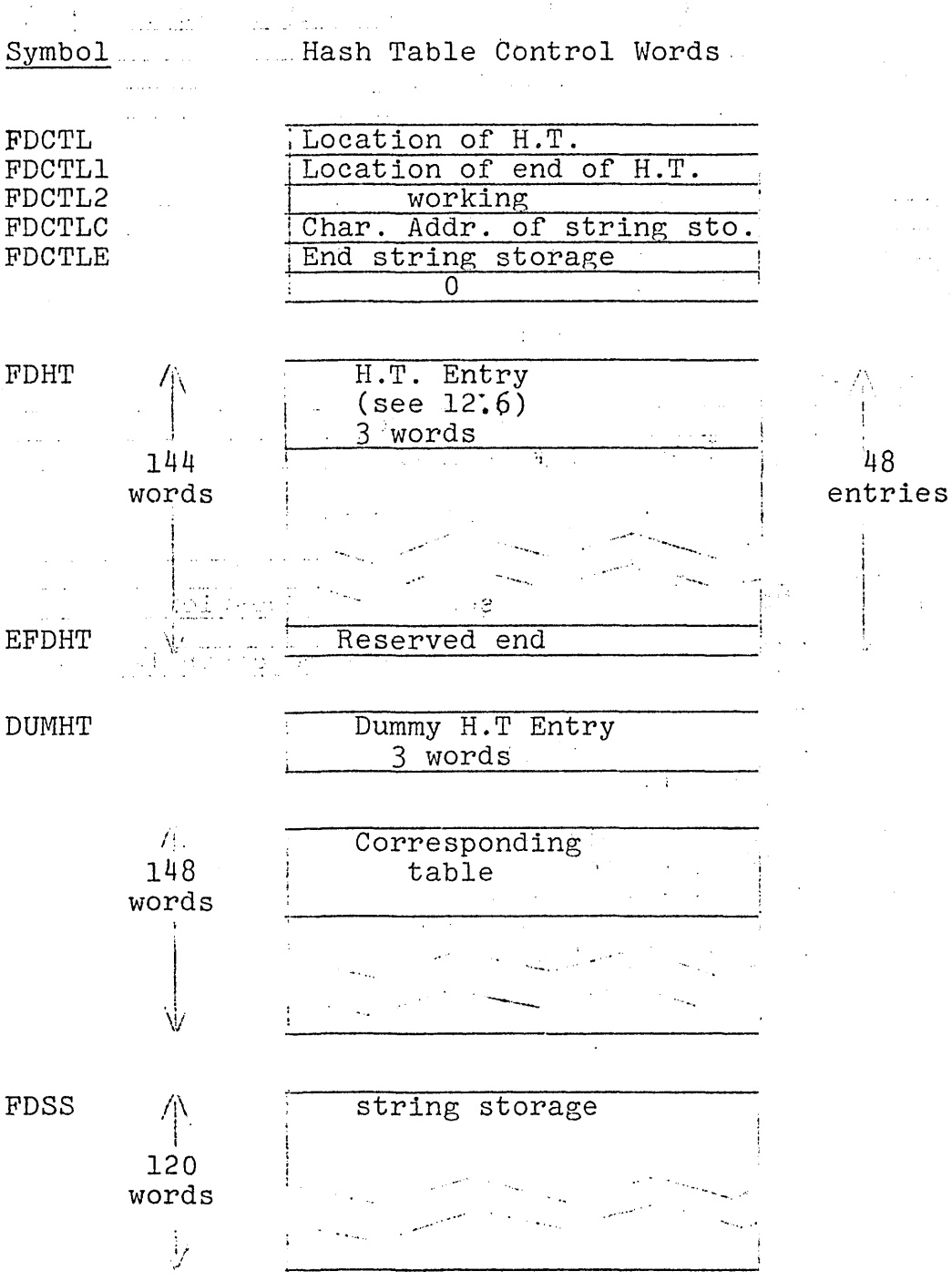
The creation date of file is set to the current date each time it is opened as an output file. The field "No. of Accesses" is incremented each time the file is opened for input or output.

There are five file names built into the system. They are:

```
PAPER TAPE
CARDS
PRINTER
TELETYPE
NOTHING
```

These names may be used at any time and have the obvious significance. If the device referred to is not available because it is attached to some other user, a suitable error message will be generated. Paper tape or card output files opened by giving this name to the executive will have the type of the file punched as the first word (or card). Similarly, paper tape or card input files opened by giving this name to the executive will read the first word from the paper tape or the first card and deliver it as the type.

FILE DIRECTORY ARRANGEMENT



HASH TABLE ENTRY

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23	
0 FT	Pointer
LTP	To File Name
HTP 0	FS

Physical Device

1 0	Pointer to
0	File Name (In "PFDSS" table)
1 2 3 4	- 0 - DN

Disc File

2 FT	Pointer
0	To file Name
Index Block Pointer	

Corresponding Table Entry

C Change in file size			FL
0 Account No.	No. of accesses	Creation date	
		Month	Day
CB FT	LTP	Future controls	

FT = File Type
LTB= Low Order Tape Position
HTP= High Order Tape Position
FS = Tape File Size
FL = File Length for Disc Files
C = Change in File Length
CB = File Control Bits - 0=Tape File
2=Disc File
F = End of Entry Flag (1)

13.0 EXECUTIVE COMMANDS RELATED TO FILES

When a user "LOGSIN" to the system, his complete file directory is read in from the disc and placed in the file directory hash table along with the name of the physical devices. The "LOGIN" procedure is described in the "Tymshare Reference Manual".

The following executive commands related to the users file directory are available:

a. FILES
b. WRITE FD
c. DF
d. FD FOR
e. DELETE FILE
f. RENAME

Commands (e) and (f) are completely described in the "Tymshare Reference Manual", but simply, DELETE FILE is used to delete a file from the directory, and RENAME is used to change the name of a file in the directory.

Commands (a) and (d) are also described in the "Tymshare Reference Manual" as used by the normal users; FILES causes the complete directory to be typed while FD/FOR types only a single entry. Byt executive class users who are able to set system status (see the TSS Executive Reference Manual") will receive the following special output:

P,DT,S name

KEY	TAPE FILES	DISC FILES
P =	Tape position (octal)	0
D =	Blank	2
T =	File type (1 through 4)	File type (1 through 4) See Section 12
S =	File size	Index Block Pointer

A colon typed after either of the above commands, will cause the length (in numbers of words) of a disc file to be typed out; the format is then as follows where L is the length:

P,DT,S,L Name

Another feature of the system status typeout is that any control characters in the file name will be typed out in two characters with the first character, the ampersand "&". For example, if the name of the file was /(bell)PROGRAM/, it would type out as follows:

```
0,23,12640 /&GPROGRAM/
```

The command "DF" can only be used by users with a special system status since it can create new file names while bypassing all system protrection. The complete file parameters must be typed as follows:

```
DF   file name   AS   P,DT,S
```

where the key to the parameters is the same as described above.

The command "WRITE FD" causes the current file directory (as it appears in the file directory hash table) to be written on the disc. See the appendix for a description of the disc format.

14.0 EXECUTIVE COMMANDS

The commands which are accepted by the executive are described in detail in the TSS Executive Reference Manual.

15.0 SUBSYSTEMS

The time-sharing system software is organized into a monitor, a system executive, and a number of sub-systems which perform specialized functions. Each of these sub-systems is called by giving its name to the executive as a command. The result of this operation is to bring the subsystem off the RAD and to transfer to its starting point. The system will thereafter remember the subsystem which is in use and will accept the CONTINUE command as an instruction to re-enter the subsystem without any initialization. Thus, for example, the command:

```
-DDT
```

would call the debugging subsystem. The line:

```
-CONTINUE
DDT
```

would re-enter DDT without initializing. Most of the subsystems are permanently present in the shared memory table, and may be called on by a user program.

Subsystems presently available in the time-sharing system are:

ARPAS:	A symbolic macro assembler
DDT:	The debugging system
QED:	The symbolic text dditor
FTC:	FORTRAN II compiler
FOS:	The FORTRAN II loader and operating system
FORTRAN:	The CCS FORTRAN IV system
CAL:	Conversational algebraic language
BASIC:	Conversational algebraic language

16.0 MISCELLANEOUS EXECUTIVE FEATURES

The executive provides a number of BRS's which are services for the user. The BRS's all declare a fork to execute. This group of BRS's are run in user mode and are called class 3 BRS's in the Monitor.

To get the date and time into a string, the operations

```
LDP   PTR
BRS   91
```

may be executed. The current date and time are appended to the string provided in A and B and the resulting string is returned. The characters appended have the form:

```
mm/dd   hh:mm
```

Hours are counted from 0 to 23.

All other system executive BRS's have been described in previous sections.

17.0 MISCELLANEOUS MONITOR BRS'S

The monitor provides a number of BRS's which are services for the user. Many of these are incorporated in the string processing system or in the floating point package and are described in the next two sections. These are called class 2 BRS's in the Monitor.

To put an integer to any radix the instructions:

```
LDB   =radix
LDX   =file
BRS   38
```

may be executed. The number, which may be preceded by a plus or minus sign, is returned in the A register and the non-numeric character which terminated the number in the B register. The number is computed by multiplying the number obtained at each stage by the radix and adding the new digit. It is, therefore, unlikely that the right thing will happen if the number of digits is too large.

To output a number to arbitrary radix the instructions:

```
LDB   =radix
LDX   =file
LDA   number
BRS   36
```

may be executed. The number will be output as an unsigned 24 bit integer. If the radix is less than 2, an error will be indicated.

18.0 STRING PROCESSING SYSTEM

A resident part of the system is a package of string handling routines. These are discussed in detail in their own manual, document 30.10.20 and will only be listed here.

GCI	Get character and increment
WCI	Write character onto string
WCH	Write character onto string storage
SKSE	Skip on string equal
SKSG	Skip on string greater
GCD	Get character and decrement
WCD	Write character and decrement
BRS 5	Look up string in hash table
BRS 6	Insert string in hash table (must be preceded by BRS 5)
BRS 33	Input string
BRS 34	Output string given word address
BRS 35	Output string given string pointer
BRS 37	General command lookup

SPS includes symbol table lookup facilities, and a string storage garbage collector is available as a library subroutine. Strings are composed of 8 bit characters packed 3 per word and are addressed by 2 word string pointers. Two SYSPOP's which are formally part of SPS but which are useful in floating point operations and in general programming are:

LDP	Load pointer
STP	Store pointer

These are double word operations which load A and B from the effective address and the next location or store A and B into the effective address and the next location, respectively.

19.0 FLOATING POINT

Floating point arithmetic and input-output operations have been incorporated into the 940 system through the use of programmed operators. This allows the user to perform useful arithmetic and I/O operations in a single instruction. A brief summary of the most commonly used arithmetic and I/O POPS is outlined herein.

The floating point numbers referenced in this section are normalized double word values. The first word is a sign bit followed by the high order 23 bits of the mantissa bits followed by a 9 bit exponent field which, like the mantissa, is always represented in two's compliment form.

Unless otherwise specified, the POP's do not make a skip return.

Floating Point Load/Store Instructions

NAME: LDP
FUNCTION: Load Pointer
CALLING SEQUENCE: LDP MEMORY

DESCRIPTION: Loads A, B with MEMORY, MEMORY+1. LDP is a single instruction that is equivalent to:

```
LDA   MEMORY
LDB   MEMORY+1
```

NAME: STP
FUNCTION: Store Pointer
CALLING SEQUENCE: STP MEMORY

DESCRIPTION: Replaces MEMORY, MEMORY+1 with the contents of A,B. STP MEMORY is a single instruction that is equivalent to:

```
STA   MEMORY, STB   MEMORY+1
```

Double Word Floating Point Arithmetic

NAME: FAD
FUNCTION: Floating Add
CALLING SEQUENCE: FAD MEMORY

DESCRIPTION: The floating point value at MEMORY, MEMORY+1 is added to the floating point value in A, B. The sum replaces the value in A, B. Memory is unaffected.

NAME: FSB
FUNCTION: Floating Subtract
CALLING SEQUENCE: FSB MEMORY

DESCRIPTION: The floating point value at MEMORY, MEMORY+1 is subtracted from the floating point value in A,B. The difference replaces the value in A,B. Memory is unaffected.

NAME: FNA
FUNCTION: Floating Negate
CALLING SEQUENCE: BRS 21

DESCRIPTION: The floating point value in A,B is negated. The result is left in A,B.

NAME: FMP
FUNCTION: Floating Multiply
CALLING SEQUENCE: FMP MEMORY

DESCRIPTION: The floating point value at MEMORY, MEMORY+1 is multiplied by the floating point value in A,B. The product replaces the value in A,B. Memory is unaffected.

NAME: FDV
FUNCTION: Floating Divide
CALLING SEQUENCE: FDV MEMORY

DESCRIPTION: The floating point value in A,B is divided by the floating point value at MEMORY, MEMORY+1. The quotient replaces the dividend in A,B. Memory is unaffected. Division by zero causes an overflow.

NAME: FIX
FUNCTION: Conversion from Floating Point to Fixed Point
CALLING SEQUENCE: BRS 50

DESCRIPTION: The floating point value in A,B is converted to fixed point. A is replaced by the integer part of the original value; the fractional part is left adjusted in B. If the integer is too large, the most significant bits are lost.

NAME: FLOAT
FUNCTION: Conversion from Fixed Point to Floating Point
CALLING SEQUENCE: BRS 51

DESCRIPTION: The integer in A is floated. The floating point result in left in A,B.

The remaining floating point SYSPOP's and BRS's use a format word in register X which contain the following information.

Format Word

BITS	FIELD NAME	SIGNIFICANCE
0-2	T	Format types: 0 - Octal 1 - Integer 2 - E format with the number right justified in the specified field on output. 3 - F format with the number right justified in the specified field on output. 4 - J format with the number left justified in the specified field on output. 5 - F format with the number left justified in the specified field on output. 6 - Double precision format. Same as 2 on input. On output same as 2 except a D will be output for the exponent if bit 16 is 1. 7 - Free form (output left justified).
3-8	D	Number of digits following the decimal point.
9-14	W	Total field width. In J format this is the number of digits before the decimal point.
15	O	Overflow action. If the field width is too small on output and this bit is 1, the first character of the output field will be a star and characters to the right will be lost. If this bit is zero and overflow occurs, characters on the right will be lost.
16	E	If this bit is 1 E format of output will be used to represent the exponent. If this bit is 0 the @ symbol will be output. Either the E or @ is always acceptable on input.

18	If this bit is 0 on input the symbol @ will be treated as a legal exponent identifier; i.e., 1.0@+2 will be legal input. If this bit is 1 the symbol @ will be treated as an illegal character. This bit has no effect on output.
19	If this bit is 0, illegal characters in the input string will be ignored. The error flag will be set when one is read. If this bit is 1 and an illegal character is read the scan will be terminated, the error flag will be set and the string pointer will be set to the character read. The conversion will take place on the characters read to that point. This bit has no effect on output.
20	If this bit is zero, the input string $\pm N \pm M$ is legal. N is treated as the mantissa and M is the exponent of a floating, real number. If this bit is 1, the second occurrance of a sign will be treated as an illegal character. This bit has no effect on output.
21	Must be zero.
22	Must be zero.
23	If a 1, the double precision accumulator will be used for numeric input-output. Significance is extended to 18+ digits. Applies to all format types.

Operating Characteristics:

On input the D field is overridden by the presence of a decimal point. If a decimal point and/or E are present, any form of the number is acceptable to any input format. It is only in the absence of these characters that the format specifications determine the interpretation of the field. Illegal characters appearing anywhere in the field may be ignored depending on bit 19 of the format word. Blanks will be converted to zero.

The maximum allowable number of input digits is twelve. If more than twelve digits are input the most significant twelve will be used. If twelve digits are used

care must be taken as overflow can occur during the conversion process. Insignificant leading or trailing zeroes will be ignored.

The maximum allowable integer on input is $\pm 2^{38}-1$ or $\pm 274,877,906,943$. Floating point numbers must like in the range:

$$9.9999999999E-78 \leq |number| \leq 5.7896044625E+76$$

Free form output will be output using an F17 if the exponent lies in the range $-1 \leq$ exponent ≤ 10 (X=10-number of digits to left of decimal point). If the number is outside this range an E17.11 will be used. Free form output always assumes a floating point number. If an integer is input it will be normalized prior to conversion.

For the E format on output, the E (@ if bit 16 of the format word is 0) is always followed by a + or - sign. On all output the sign of the number is printed only if it is negative.

Error Conditions:

If an error is detected during the conversion process a positive integer indicating the error type will be returned in the index register.

Errors detected are as follows:

X=0 No error was detected.

X=1 Number of decimal digits after the decimal point exceeds 12 for single precision and 18 for extended precision on formatted input. Twelve and 18 used respectively.

X=2 Field too short for E format on output. Overflow action will be taken depending on the value of bit 15 of the format word.

X=3 Input number exceeds the maximum allowable bounds.

X=4 Field too short for F or I format on output. Overflow action will be taken depending on the value of bit 15 of the format word.

X=5 An E format was specified for input but the input string does not contain an "E" or ".". The number will be converted using an equivalent F format.

X=6 An illegal character was encountered in the input scan. Character is ignored.

String Conversion

NAME: SIC
FUNCTION: String to Internal Conversion
CALLING SEQUENCE:
```
LDX   FORMAT
SIC   POINTER
BRU   INTEGER
BRU   FLOATING
```

DESCRIPTION: FORMAT describes the type of conversion to be done (see the CCS Implementation Manual for the FORMAT word specifications). The string of input characters starts at the character following the character pointed to by the character address in POINTER. The character address in POINTER+1 points to the last character of the input string.

NAME: ISC
FUNCTION: Internal to String Conversion
CALLING SEQUENCE:
```
LDP   VALUE
LDX   FORMAT
ISC   POINTER
```

DESCRIPTION: FORMAT describes the type of conversion to be done. (See the CCS Implementation Manual for the FORMAT word specifications). POINTER+1 contains the character address of the character immediately preceding the position where the first character of output is to go. POINTER+1 is incremented by one for each character of output added to the character string. VALUE is the double word floating point value to be converted.

NAME: FFI
FUNCTION: Formatted Input
CALLING SEQUENCE:
```
LDX   FORMAT
BRS   52
```

DESCRIPTION: Characters are read from a file and converted to internal form. Either a floating point value is left in A,B or an integer is left in A. A skip return is generated if a floating point value is read and the input mode is free format.

NAME: FFO
FUNCTION: Formatted Output
CALLING SEQUENCE:
```
LDP   VALUE
LDX   FORMAT
BRS   53
```

DESCRIPTION: The floating point value in A,B or the integer in A is output to the file specified in FORMAT.

20.0 INDEX OF BRS'S AND SYSTEM OPERATORS

20.1 BRS's

*19 Open Output File Located in File Directory
Pg. 12.3

*20 Close a Tape File - (Not included)

21 Floating Point Negate
Pg. 19.2

23 Link/Unlink Specified TTY - (not included)

24 Unlink All TTY's - (not included)

25 Set Teletype to Accept/Refuse Links - (not included)

26 Skip if Escape Waiting
Pg. 2.5

27 Attach TTY to Calling Program - (not included)

28 Release Attached TTY - (not included)

29 Clear the Output Buffer
Pg. 7.4

30 Read Status of a Lower Fork
Pg. 3.2

31 Wait for Specific Fork to Cause a Panic
Pgs. 2.4, 3.3

32 Terminates a Specified Lower Fork
Pg. 3.3

33 Read String
Pg. 18.1

34 Output Message
Pg. 18.1

35 Output String
Pg. 18.1

36 Output Number to Specified Radix
Pg. 17.1

37 General String Look Up
Pg. 18.1

38 Input Number to Specified Radix
Pg. 17.1

40 Read Echo Table
Pg. 7.2

41 Return from I/O Subroutine
Pgs. 11.1, 11.2

42 Read Real-Time Clock
Pg. 6.1

43 Read Pseudo-Relabeling
Pg. 5.1

44 Set Pseudo-Relabeling
Pg. 3.2, 5.1

45 Dismiss on Quantum Overflow
Pg. 2.3

46 Turn Escape Off
Pg. 3.5

47 Turn Escape On
Pg. 3.5

*48 Look Up Input/Output File Name
Pgs. 12.2, 12.3

49 Read Interrupts Armed
Pg. 4.2

50 Conversion from Floating Point to Fixed Point
Pg. 19.2

51 Conversion from Fixed Point to Floating Point
Pg. 19.2

52 Formatted Floating Point Input
Pg. 19.6

53 Formatted Floating Point Output
Pg. 19.7

56 Make Page System
Pg. 5.3

57 Guarantee 16ms Computing
Pg. 2.3

58 Define File as Random
Pg. 10.2

59 Release Words from Random File
Pg. 10.3

*60 Look Up I/O File Name and Insert in File Directory if not Found
Pg. 12.3

66 Delete DSU File Data
Pgs. 9.4, 10.3

67 Delete DSU File Index Block
Pg. 9.5

68 Make Pseudo-Page Shareable - (not included)

69 Get SMT Block to PMT
Pg. 5.2

71 Read Executivity
Pg. 6.1

72 System Dismissal
Pg. 2.4

73 Terminates a Specified Number of Lower Forks
Pg. 3.6

78 Arm/Disarm Software Interrupts
Pg. 4.1

79 Cause Specified Software Interrupts
Pg. 4.1

80 Make Page Read Only
Pg. 5.3

81 Dismiss for Specified Amount of Time
Pg. 6.1

82 Switch Sequential File Type
Pg. 9.2

85 Set Special TTY Output
Pg. 7.5

86 Clear Special TTY Output
Pg. 7.5

87 Read DSU File Index Block
Pg. 9.5

88 Read Execution Time
Pg. 6.1

90 Declare a Fork for Escape
Pgs, 3.1, 3.5

91 Read Date and Time into a String
Pgs. 6.1, 16.1

*95 Dump Program and Status on File - (not included

*96 Recover Program and Status from File - (not included)

104 Read a Page (2048 words) from RAD
Pg. 5.4

105 Write a Page (2048 words) to RAD
Pg. 5.4

106 Wait for any Fork to Terminate
Pgs. 2.4, 3.3

107 Read Status of all Lower Forks
Pg. 3.3

108 Terminate All Lower Forks
Pg. 3.3

109 Dismiss Calling Fork
Pgs. 2.4, 6.1

110 Read Device and Unit
Pg. 9.6

111 Return from Exec BRS (Exec Only)
Pg. 6.1

112 Turn Off Teletype Station (Exec Only)
Pgs. 7.3, 7.4

113 Compute File Size of a Disc File
Pg. 9.4

114 Turn Off Run-Away Magnetic Tape
Pg. 9.7

116 Read User Relabeling
Pg. 5.2

117 Set User Relabeling
Pg. 5.2

118 Allocate Magnetic Tape Unit
Pg. 9.7

119 De-Allocate Magnetic Tape Unit
Pg. 9.7

120 Acquire a New Page
Pg. 5.2

121 Release Specified Page from PMT
Pg. 5.2

122 Simulate Memory Panic
Pg. 6.1

BE+1 Read DSU
Pgs. 4.2, 8.3, 8.4

BE+2 Write DSU
Pgs. 4.2, 8.3, 8.4

BE+3 Test for Carrier Present
Pg. 7.3

BE+4 Read/Write One Word in the Monitor
Pgs. 6.2, 8.1

BE+5 Set Disc Bit Map - (not included)

BE+6 Turn a Teletype Line On or Off
Pg. 7.4

BE+7 Test a Breakpoint Switch
Pg. 6.2

BE+8 To Crash the System for Error Diagnostic
Pg. 6.2

BE+9 Read DSU Page
Pg. 8.3, 8.4

BE+10 Write DSU Page
Pgs. 8.3, 8.4

BE+11 Ignore Line Feed or Carriage Return When Followed by Carriage Return or Line Feed Respectively
Pg. 7.2

BE+12 Arm Timing Interrupt
Pg. 4.2

BE+13 Sets System Exec Switches in SYMS
Pg. 6.2

BE+14 Input String with Edit - (not included)

BE+15 Read Page from RAD
Pg. 6.2

20.2 System Operators

BIO Block Input/Output
Pgs. 9.3, 10.1, 11.1

CIO Character Input/Output
Pgs. 9.2, 10.1, 11.1

CIT Character Input and Test - (not included)

CTRL Input/Output Control
9.3, 9.4, 9.6

DWI Read a Word from a Random File
Pg. 10.1

DWO Write a Word from a Random File
Pg. 10.1

DBI Read a Block from a Random File
10.1

DBO Write a Block from a Random File
Pg. 10.2

EXS Execute Instruction in System Mode
Pg. 6.2

FAD Floating Point Addition
Pg. 19.1

FDV Floating Point Division
Pg. 19.1

FMP Floating Point Multiplication
Pg. 19.2

FSB Floating Point Subtract
Pg. 19.2

GCD Get Character from End of String and Decrement End Pointer
Pg. 18.1

GCI Get Character from Beginning of String and Increment Beginning Pointer
Pg. 18.1

ISC Internal to String Conversion
Pg. 19.6

IST Input from Specific TTY - (not included)

LAS Read a Word from Secondary Memory
Pg. 10.2

LDP Load String Pointer
Pgs. 18.1, 19.1

OST Output to Specific TTY - (not included)

SAS Store a Word into Secondary Memory
Pg. 10.2

SKSE Skip if String Equal
Pg. 18.1

SKSG Skip if String Greater
Pg. 18.1

SIC String to Internal Conversion
19.6

STI Simulate TTY Input - (not included)

STP Store String Pointer
Pg. 18.1, 19.1

TCI Teletype Character Input - (not included)

TCO Teletype Character Output
Pg. 7.3

WCD Put Character on Beginning of String and Decrement Beginning Pointer
Pg. 18.1

WCH Write Character to Memory by Table
Pg. 18.1

WCI Put Character on End of String and Increment End Pointer
Pg. 18.1

WIO Word Input/Output
Pgs. 9.3, 10.1, 11.1

Those BRS's marked with an asterisk are executive BRS's and all others are monitor BRS's.

APPENDIX A

GENERAL DESCRIPTION OF THE COMBINED FILE DIRECTORY

1. A user may have one or two file directory blocks on the disc; the second block is an overflow block. Each block consists of 128 words containing a variable number of file directory entries. Each entry is in the format pictured in (d).

2. If the first word of the block is zero, the block considered to be empty. The last entry is followed by a -1 or -2 word where the -2 indicates that there are additional entries in the overflow block.

3. The last four words of the file directory block contain the following information:

Last Word	Valid on-time for this user (1 bit per hour of the day).
Last Word -1	Accumulated computer time used.
Last Word -2	Accumulated real time used.
Last word -3	Overflow block pointer.

4. In the case of an overflow block, the last three words are zero, and the overflow block pointer is a backward pointer to the first file directory block.

FILE DIRECTORY FORMAT ON DISC

1 Entry (Disc File)

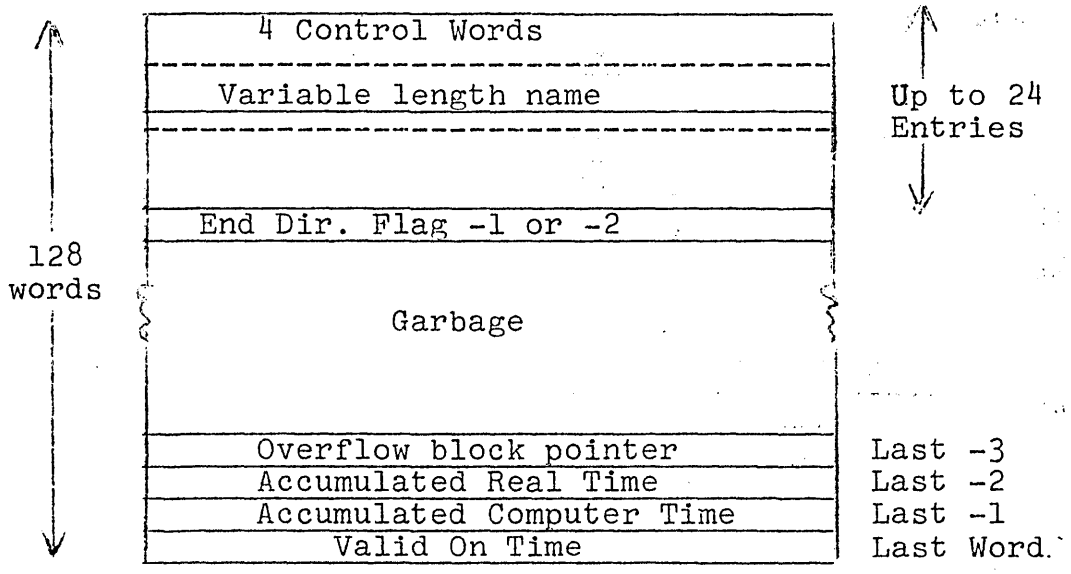
	0	1 – 8	9 – 14	15 – 23
0	0	Account No.	No. of Accesses	Creation Date
		11	12	
1	C	Change if File Size	File Length (FL)	
	2	3 – 6	11	12
2	CB	FT	LTP	Future Controls
3	Index Block Pointer			
	1 – 7	8	9 – 15	16 – 17
4	D Char. of Name	0		0
	1 – 7	8 – 15		16
N	F Char. of Name	Char. or 136 (fill)		Char. or 136 (fill)

FT = File Type
LTP = Low Order Tape position
HTP = High Order Tape position
FS = Tape File Size
FL = File Length for disc Files
C = Change in file length (file length no longer valid)
CB = File Control Bits, 0=Tape file
2=Disc file
F = End of Entry Flag (1)

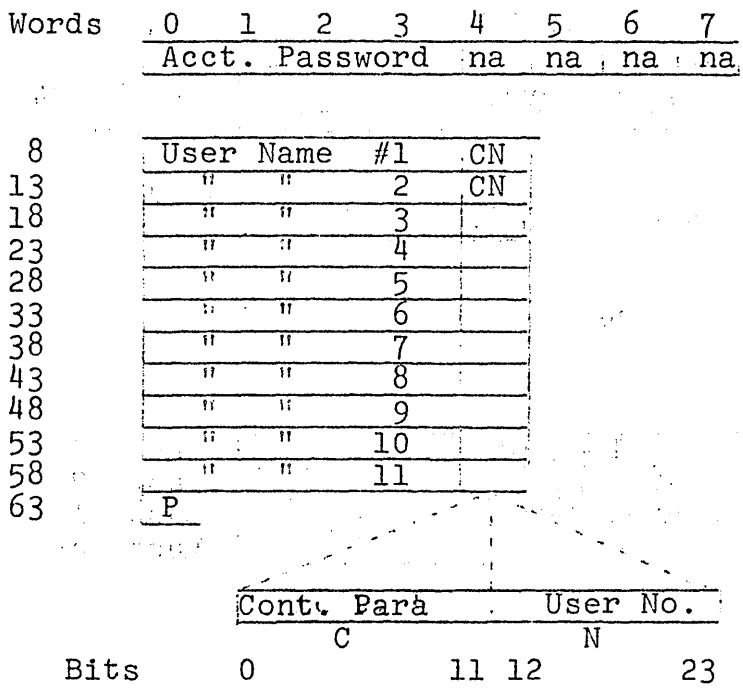
If <u>Tape File</u>, word #3 =

	0 – 5	6 – 8	9 – 23
3	HTP	0	FS

FILE DIRECTORY BLOCK



USER ACCOUNT DIRECTORY ON DISC



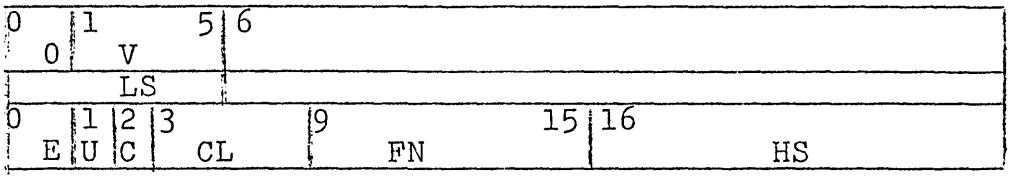
NOTES: "P" is reserved for an overflow pointer and not presently used. "na", not assigned.

The control parameter bits are assigned as follows:

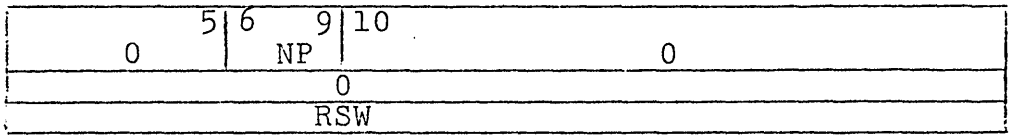
BIT	USE
0	System Status
1	Control of physical devices
2	Operator Status
3	Subsystem Status
4,5	Not assigned
6-11	Subsystem classes

SUBSYSTEM TABLE

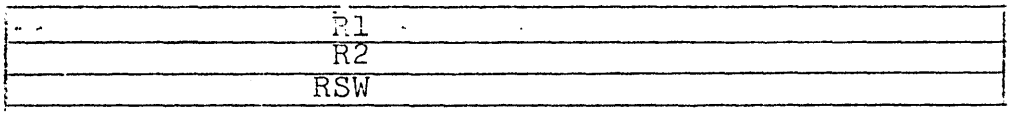
Hash Table Entry



Corresponding Table (Not Common Subsystem)



Corresponding Table (Common Subsystem)



V = Subsystem Verify Number
LS = Low-order Starting Address
E = Propogate Exec Status
U = Co-exist with Users Memory (cannot if on)
C = Common Subsystem
CL = Class (must agree with user's control parameters)
FN = File Number (location on RAD for non-common Subsystem)
HS = High-order Starting Address
NP = Number of pages for non-common subsystem
R1 = First-half SMT relabeling (4 bytes)
R2 = Second-half SMT relabeling (4 bytes)
RSW= Relabeling Status Word (8 bytes)