TIME-SHARING SYSTEM

REFERENCE MANUAL

By

Ann Hardy
David Gardner
Verne Van Vlear

TYMSHARE, INC.

Revised 7/21/67

TABLE OF CONTENTS

1.0 INTRODUCTORY

This manual describes the Berkeley Time-Sharing System as it was modified by Tymshare, Inc. The original Berkeley manual was written by Butler Lampson and this manual is a modification of that manual. 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 back-up 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, the TS block, 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. All forks have a long-time quantum and a short-time quantum. When a fork is created it is given a long-time quantum. This is a six bit number which is kept in QR. It is the number of clock cycles remaining before the fork will be dismissed on long quantum overflow. The three bit number in QUTAB points to a table of various long quantum lengths. This allows the system to vary the length of the long quantum for different forks. The length of the short time quantum is tentatively going to be the same for all forks. 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. At the same time, the fork's short-time

quantum is put into TIME. Both TIME (short quantum) and TTIME (long quantum) are decremented at every clock cycle.

When a fork is activated, it is first allowed to run for a short quantum. During this time is cannot be dismissed except by its own request.

When TIME goes negative, a word called ACTR is checked to determine whether any fork which is dismissed for I/O can be run. If not, the fork is allowed to continue. At each subsequent clock cycle the fork may be dismissed if any fork dismissed for I/O is ready to run. It may also be dismissed when the long quantum is exhausted if any other forks are waiting to run. In either case it is said to be dismissed for quantum overflow. If ACTR indicates that another fork dismissed for I/O is ready to run at the end of the short quantum, the fork is also dismissed for quantum overflow.

In order to allow an efficient implementation of this scheme ACTR is incremented by every interrupt routine which takes action allowing a fork which is waiting for I/O to run.

Since ACTR is set to -1 when the first fork from QSQ is activated, this means that the clock interrupt needs only to do:

```
SKR   TIME
BRU   *+3
SKN   ACTR
BRU   *+3     Ready to dismiss
SKR   TTIME
BRI           Return to program
```

in order to check both the conditions which may require further action. If ACTR is positive or the short quantum has not run out, it is of course ignored, in accordance with the above discussion.

When a fork is dismissed for I/O, TTIME is put into QR. When the fork is reactivated, TTIME is set from QR. TIME is reset to the full short quantum. That is, the long quantum is allowed to run down while a program computes, regardless of whether it has to wait for I/O between computations. On the other hand, a fork is always given a full short quantum.

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 the 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 Activate if TTYBRK is less than REAL.
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 200000000 = 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 or disc I/O
QIO	Programs dismissed for teletype output
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 scheduler then begins at QTI and scans through the queue structure looking for a program whose activation condition is satisfied. When one if 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 or QTI with an immediate activation condition. They, therefore, take priority over all programs dismissed for quantum overflow.

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

a) The cell PUCTR is non-zero,

b) Three 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 (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.

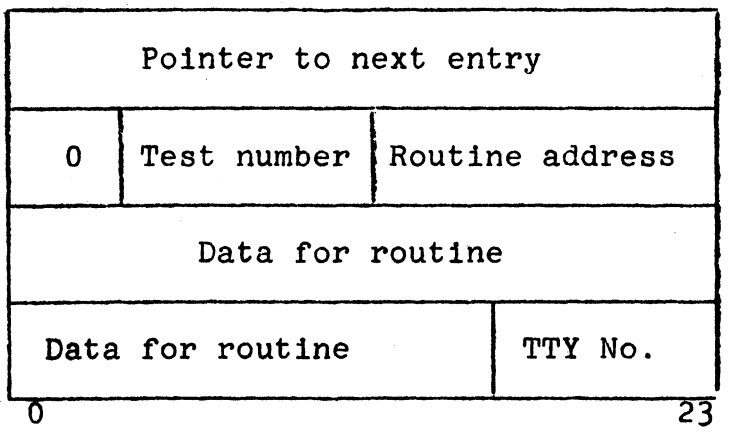
PAC TABLE - ONE PER FORK

PNEXT	Next queue or next program is queue <0=>next program >0=>next queue																						
PL	UM		0	OV	3 file # of subr. file 8							0	10 Saved (P) 23										
PA	Saved (A)																						
RL1	First pseudo-relabeling register																						
RL2	Second pseudo-relabeling register																						
PPTR	0 PDOWN 11												12 PFORK or chain for free entries 23										
PTEST	0				3 activation condition 8							0	10 test word addr., or int. #, or status of dead fork 23										
PQU	EX	EXB	0	3 QR 8								9 QUTAB 11				12 PPAR 23							
PTAB	LM	EX1	TS	Job No.								0	10 Panic Table Address 23										
PIM	MT	TP	NT	0	4 IEM 9											12	14	15 0		18 XPB		23 TO	
					1	2	3	4	5	6	7	8				9	10	11					

UM= User Mode (1)
OV= Overflow
QUTAB= addr. for short quantum
EX=System status
QR= Amount of long quantum left
IEM= Interrupt Enabled Mask
LM= Local Memory
MT= Add no memory
EXB= Exec BRS fork
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 during 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	Give fork system status if current fork has system status.
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	Give fork subsystem status if current fork has subsystem status.

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 inself 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 fork. 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 system or subsystem 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 excuting 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 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. System status 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:

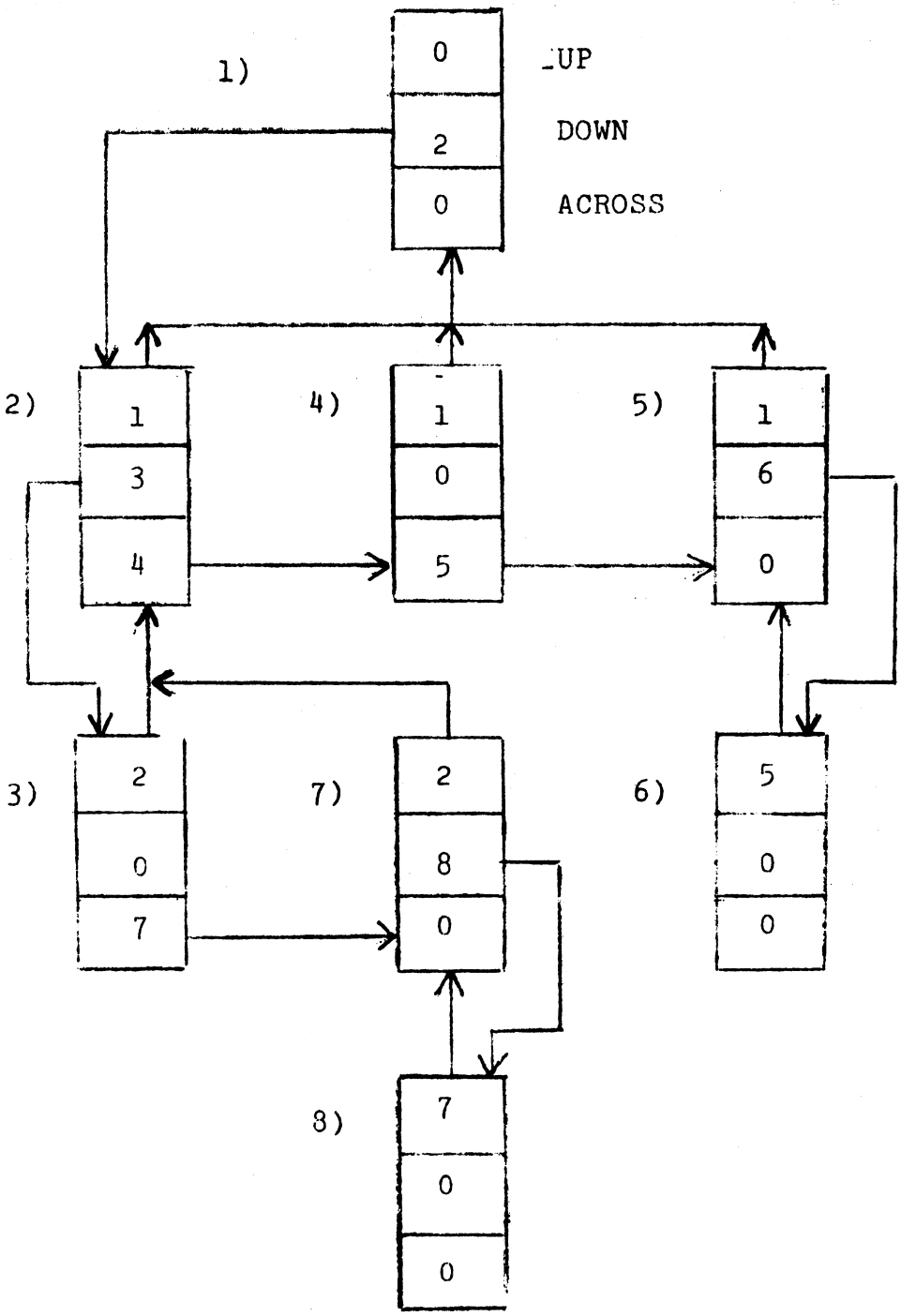
1) Machine instructions which are privileged.
2) SYSPOPs which are forbidden to the user or which have been provided with unacceptable arguments.

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.

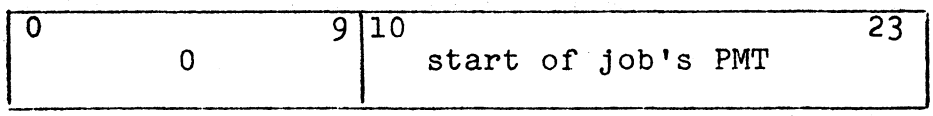
FORK STRUCTURE



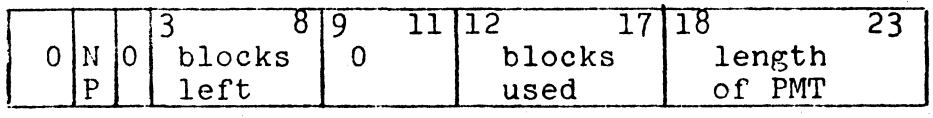
Hierarchy of Processes

JOB TABLES

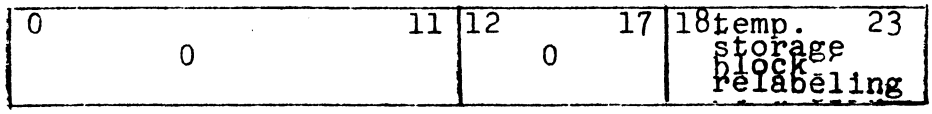
PMTP



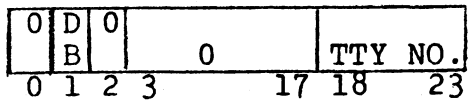
PMA



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 a system status 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 of 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 using 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 15 words. These words are numbered from 61 to 77 octal. The TS block is also in the PMT and is numbered 60. However, it is not accessible to the user. 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.

Since relabeling consists of eight six-bit bytes it is kept in two words or in the A and B registers. The relabeling may be thought of in the following way ----/----. Each dash represents one of the user's pages beginning with page 0. Therefore, addresses in the page represented by the first dash are from 0 to 3777. When a fork has memory, then the dash is replaced by the appropriate pseudo-relabeling byte as follows: 61---/----. In this case the fork has one page (page 0) in its memory. The PMT entry is number 61. If the user wants to see PMT entry 61 as page 6 instead of page 0, he can set his relabeling to ----/--61-. Relabeling can be changed with a BRS 44 or in DDT by typing the correct relabeling as follows: 0,6100;R for putting PMT entry 61 into page 6. To see program relabeling while in the exec, the STATUS command is available.

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. In this system NCMEM is equal to 60. 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 the 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.

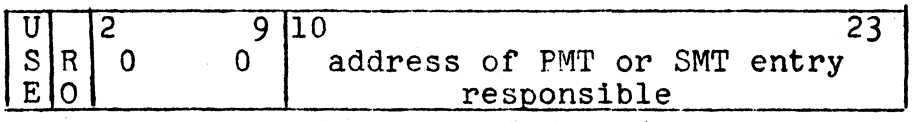
A RAD is divided into blocks of 32K. Each user is assigned a block depending on his job number. The first page in each block is always the user's TS page. Each block of 32K consists of eight bands with two pages per band. The list of swapping commands alternates pages whenever possible to minimize swap 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 RMC can be released by the swapper.

The format of an RMT entry (one per real page) is:



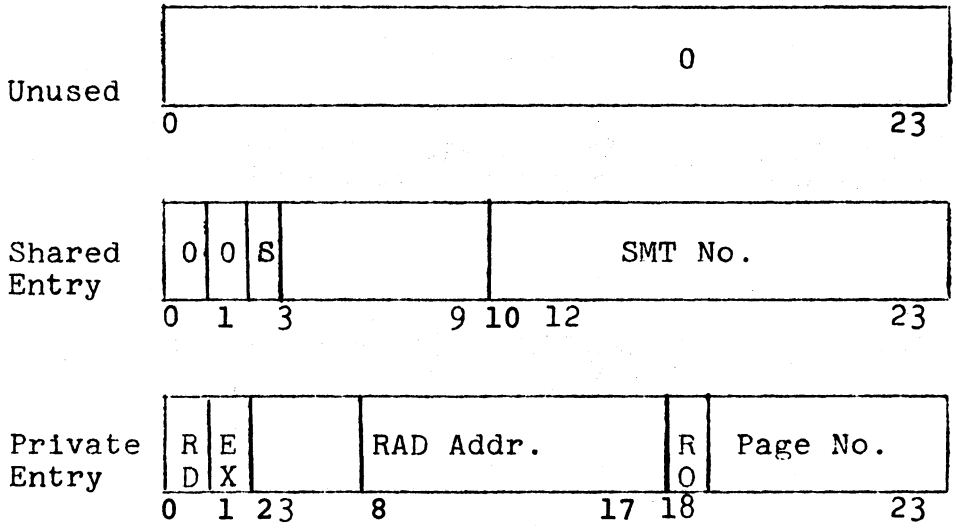
USE = in use RO = read only

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.

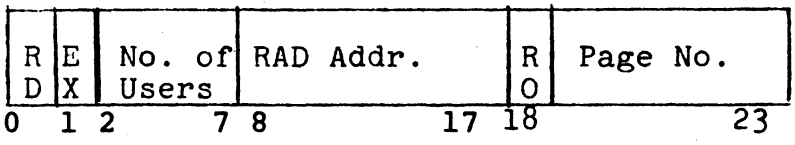
The swapper also contains a device called the simulated associative memory or SAM, which contains pseudo-relabeling and real relabeling for the most recently used maps. It serves to reduce the amount of time needed for map-changing when little swapping is taking place. It is cleared whenever a RAD read 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 dismissed 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 of 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 set 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. The characters in the output and echo buffers are in true ASCII instead of trimmed ASCII.

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 echoes 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 may 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 requires 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 and 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,
         and 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 doing the conversion from ASCII to internal (trimmed ASCII) 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. If a fork is using 8-level output, it should be sure all characters have been sent out (BRS 14) before

releasing control to a higher level fork which can reset 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

TIS4 next available space in input buffer (pointer) used by interrupt routine.

TIS5 next filled space in input buffer (pointer)

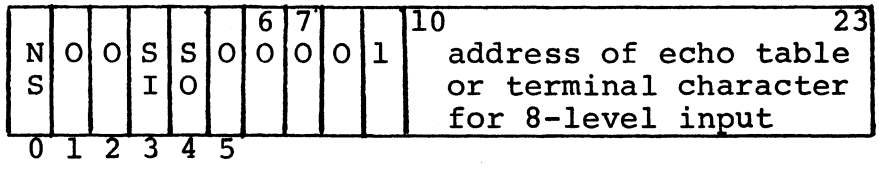
TOS2 number of characters in output buffer; -1 = inactive

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) used by interrupt routine

TOS5 next available space in output buffer

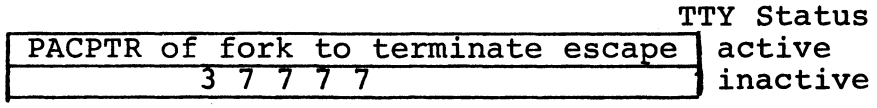
TTYTBL



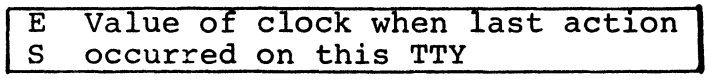
TTYFLG don't listen for input (except escape) when 0. Set when input buffer is full.

TTYBRK waiting for break character when 37777777B. Fork is activated when TTYBRK <real

TTYASG



TTYTIM



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 82 physical discs each with a moveable arm. The arms have 64 positions numbered 0 to 63. Each arm position on each disc consists of 8192 words. 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 data 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. Two of the 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 (MOD 4) for all the physical blocks used to hold information for the file. In addition, one word contains a checksum for the index block and one word contains the user number.

There are two parts to the disc area reserved for files. The area at the outer edges of the discs is kept for less frequently changed files. Files are moved to this area by a special non-timesharing routine. The center area is for active files and is controlled by 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 (MOD 4) 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. At this time the BRS BE+4 turns on the accounting.

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 system'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 the monitor's 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 which 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 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.

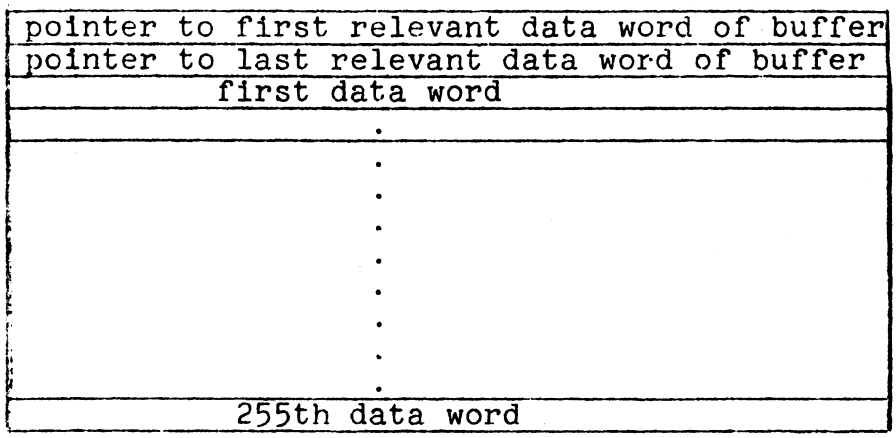
Arm positions 0 and 1 contain the file directories, accounting information, and mailbox data. These are explained in the TSS Executive Reference Manual.

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

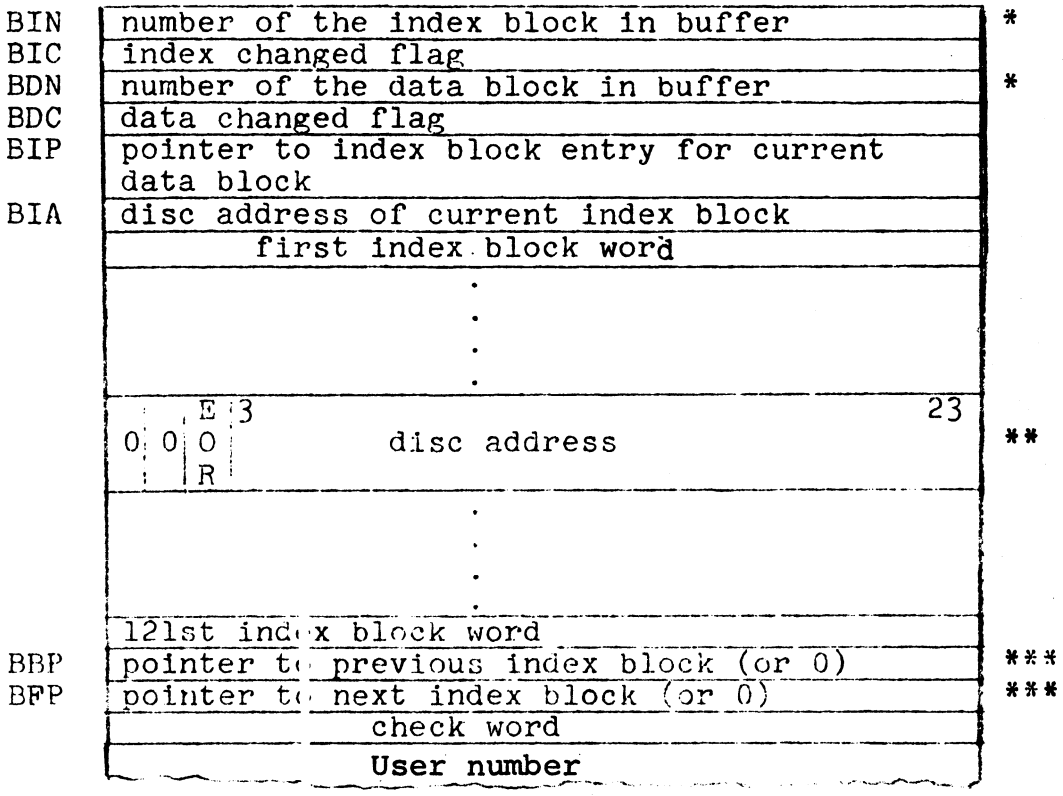
disc address in B. In addition BRS BE+1 and BRS BE+2 take the word count in X. BRS BE+9 and BRS BE+10 always read or write a page (2K) from or to the disc.

BUFFERS

(a) Layout of a File Buffer



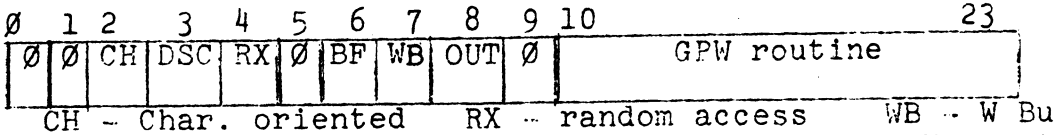
(b) Layout of Index Block Buffer and Associated Pointers for a Disc File



* random files only
**index block word format. EOR=end of record flag.
***always 0 for sequential files

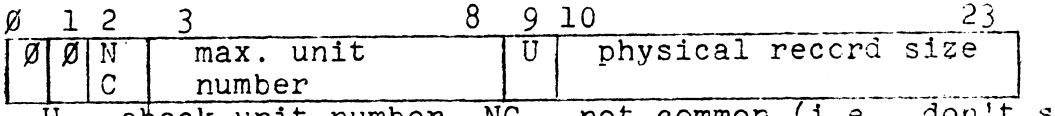
DEVICE TABLES

DEV word or character I/O routine



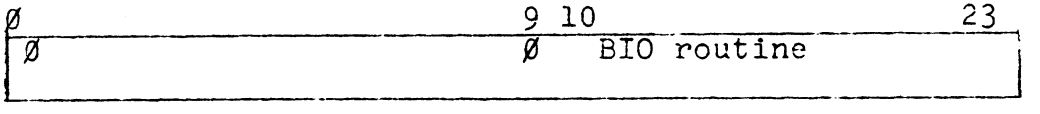
CH - Char. oriented RX - random access WB - W Buffer
DSC - Disc BF - requires buffer CUT - output

BUFS buffer size

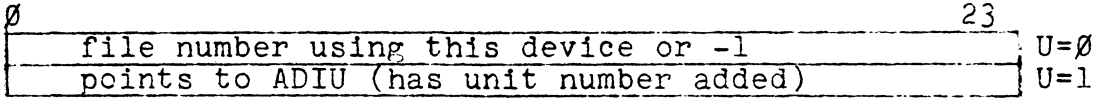


U - check unit number NC - not common (i.e. don't set DIU)

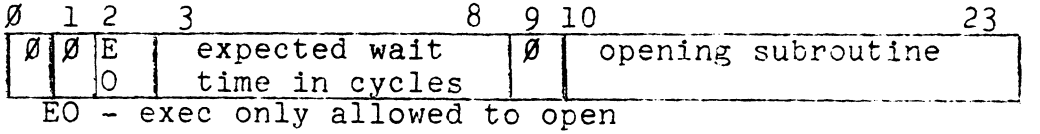
BDEV Block I/O routine



DIU device in user

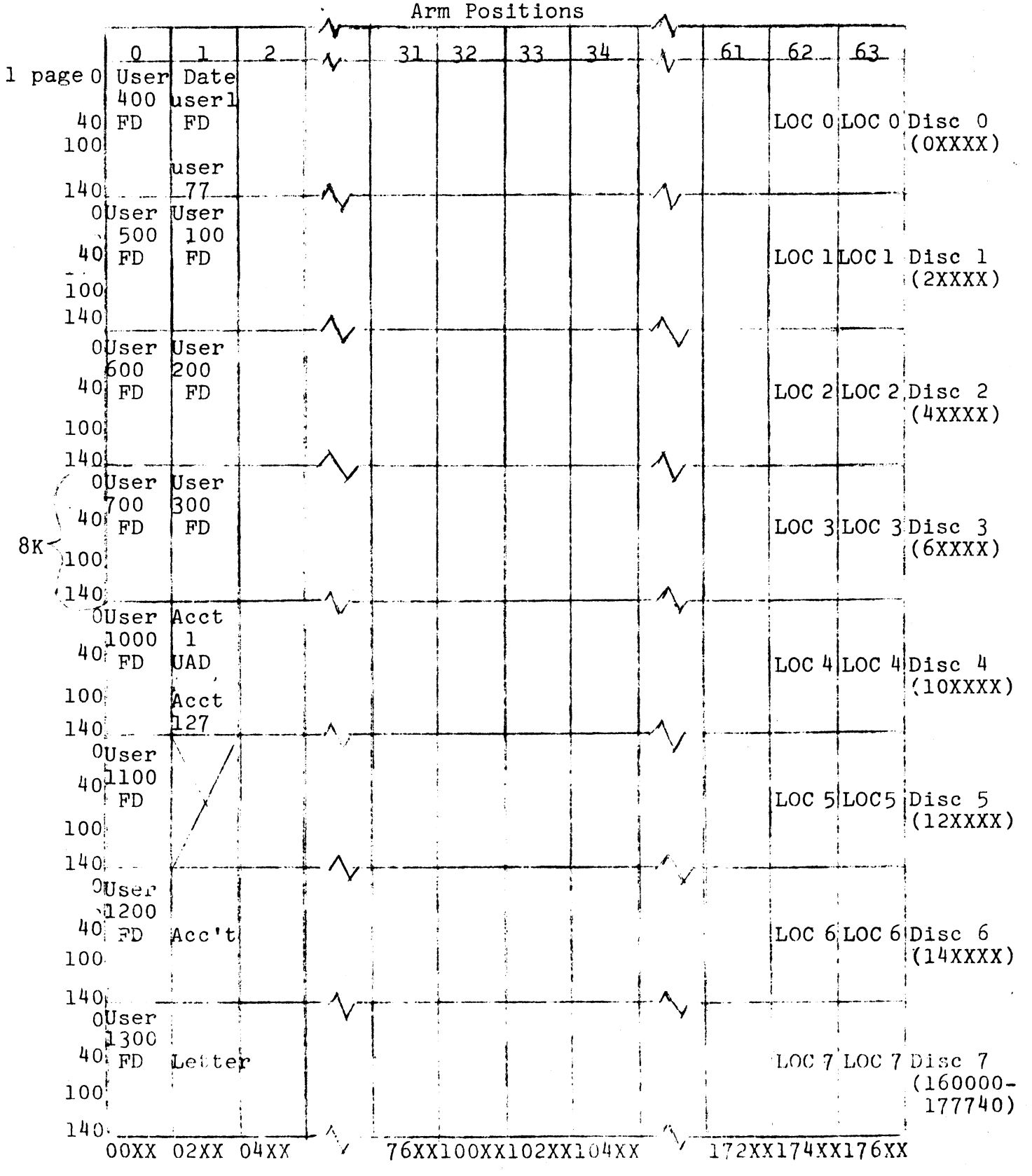


OPNDEV opening routine



EO - exec only allowed to open

DISC MAP


Arm Positions
0
1
2
31
32
33
34
61
62
63
1 page
0
40
100
140
User 400 FD
Date user1 FD
user 77
LOC 0
LOC 0
Disc 0 (0XXXX)
User 500 FD
User 100 FD
LOC 1
LOC 1
Disc 1 (2XXXX)
User 600 FD
User 200 FD
LOC 2
LOC 2
Disc 2 (4XXXX)
8K
User 700 FD
User 300 FD
LOC 3
LOC 3
Disc 3 (6XXXX)
User 1000 FD
Acct 1 UAD
Acct 127
LOC 4
LOC 4
Disc 4 (10XXXX)
User 1100 FD
LOC 5
LOC5
Disc 5 (12XXXX)
User 1200 FD
Acc't
LOC 6
LOC 6
Disc 6 (14XXXX)
User 1300 FD
Letter
LOC 7
LOC 7
Disc 7 (160000-177740)
00XX
02XX
04XX
76XX
100XX
102XX
104XX
172XX
174XX
176XX

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's 15, 16, 18 and 19). 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:

-3 Bit map not set.

-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.

-4 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's and WIO's can be mixed.

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.

A program may delete all the information in a disc file that is open as an output file by executing the instructions:

```
LDA  =file number
BRS  66
```

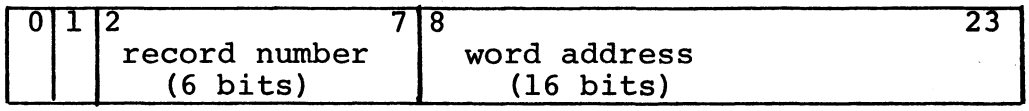
The index block for a sequential disc file contains one word for each physical record in the file. This word contains the address on the disc of the physical record in the bottom 21 bits. Bit 2 is set if the physical record is the last record of a logical record. A sequential file may have only one index block, or a maximum of 124 X 255 = 31620 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 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.

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 9.1. 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 **sequential** files 2 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, **and if the user has peripheral status.**

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_8 (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.

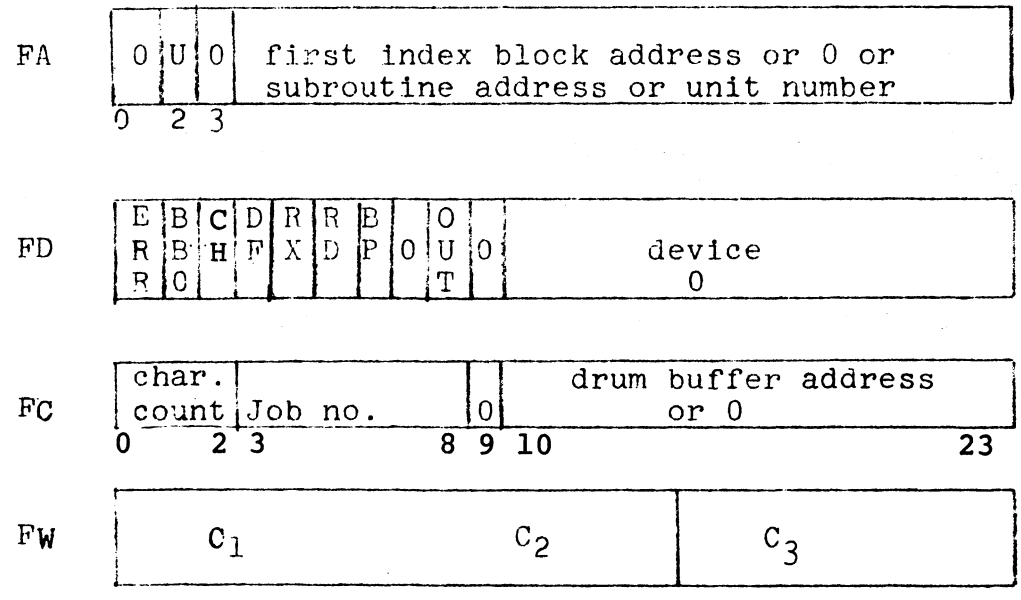
The EOF light is sensed as an end of file at all times.

The phanton user's three second routine checks to see whether a W-buffer interrupt has been pending for more than ten seconds. If so it takes drastic 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.

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 124 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 always 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(octal)	Word input subroutine
011 00000(octal)	Word output subroutine

The address field of the parameter word contains the subroutine address. 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 parameter word. 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 six bits of the PAC table. The contents of these six 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 teletype. If the name is known to the program, 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.

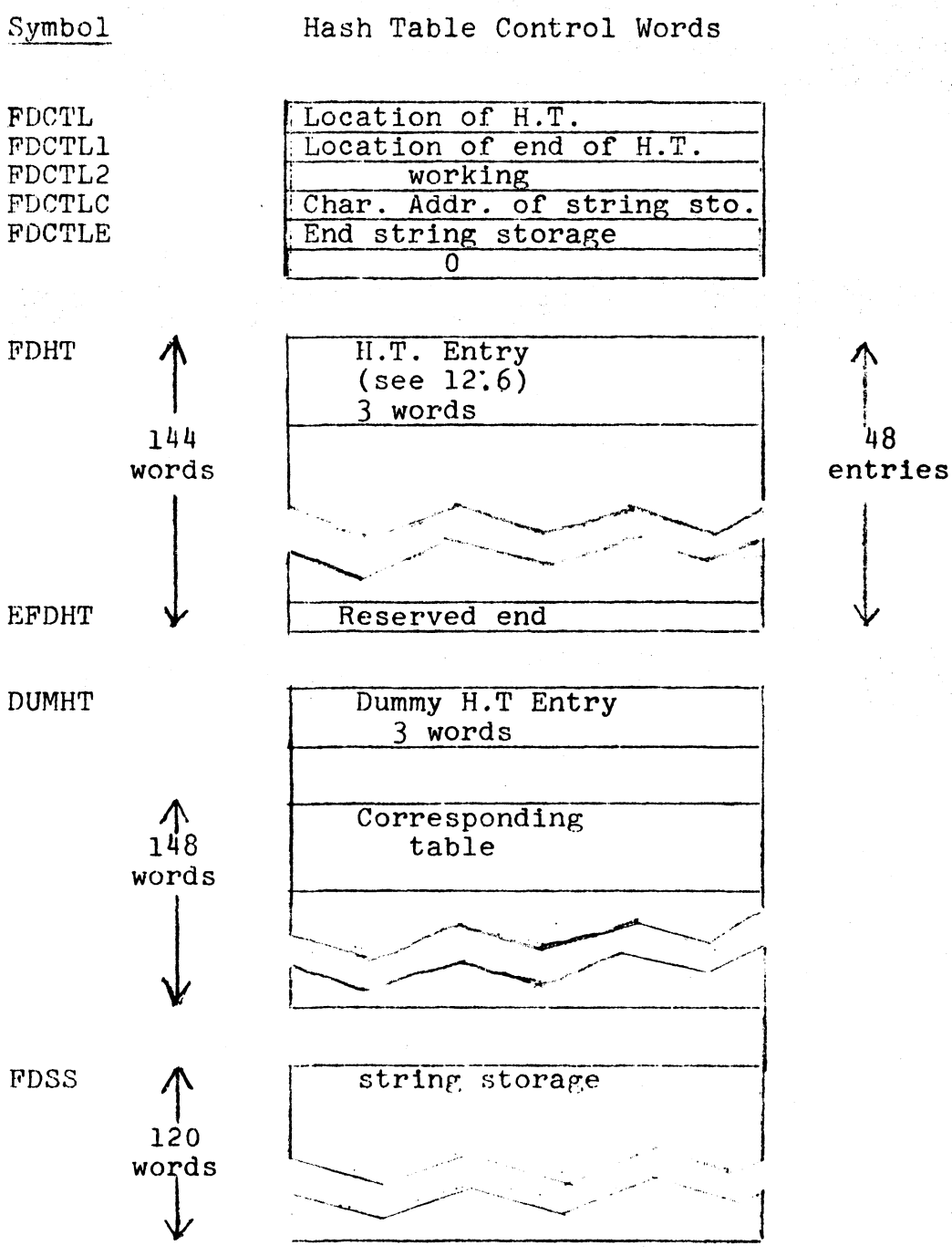
The creation date of file is set to the current date each time it is opened as an output file. The field "No. of Accesses" is incremented each time the file is opened for input or output.

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

Tape File

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 **tape file**

Physical Device **- never on disc**

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
LTP= Low Order Tape Position
HTP= High Order Tape Position
FS = Tape File Size **- True Tape size - 32K**
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. But 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 protection. 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 editor
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 ±N±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. 10.7

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

BE+16 MFSYS - Make Fork System

BE+17 CKBUF - Check for Free Buffer

BE+18 EXBRS - Get Exec Subroutines

BE+19 NOP 10 - Creation Date and Access Count

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.

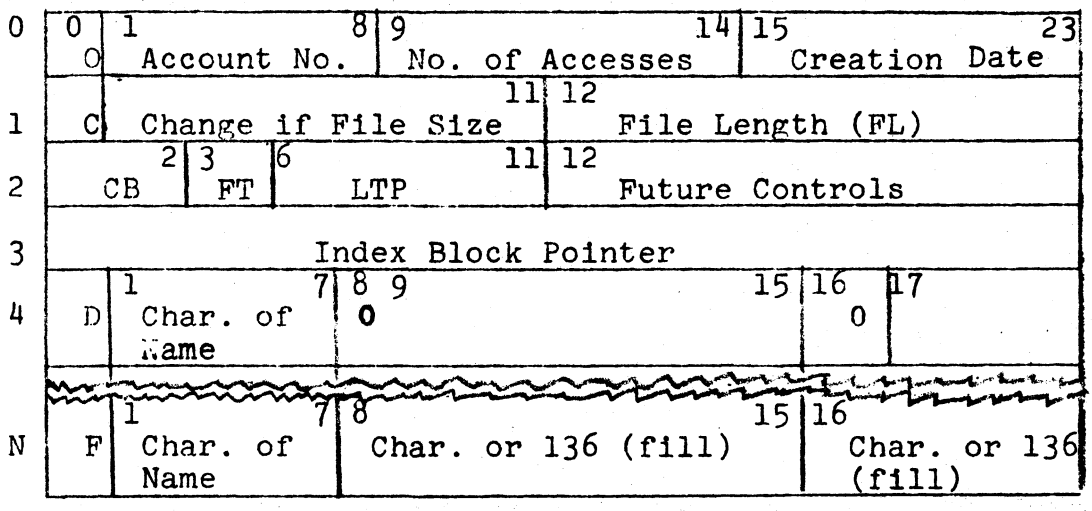
3. The last four words of the file directory block contain the following information:

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.

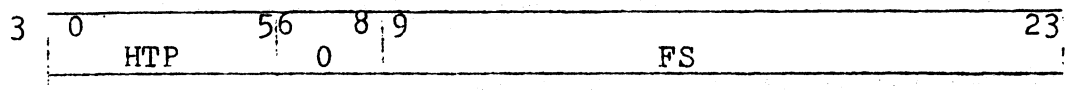
FILE DIRECTORY FORMAT ON DISC

1 Entry (Disc File)

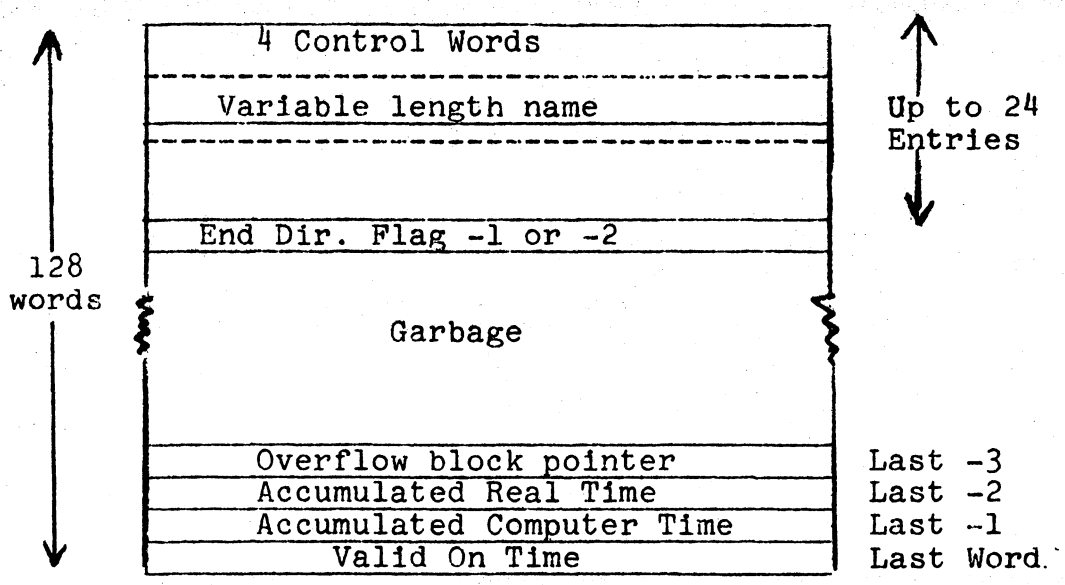


FT = File Type
LTP = Low Order Tape position
HTP = High Order Tape position
FS = Tape File Size
FL = File Length for disc Files
C = Change in file length (file length no longer valid)
CB = File Control Bits, 0=Tape file
2=Disc file
F = End of Entry Flag (1)

If Tape File, word #3 =

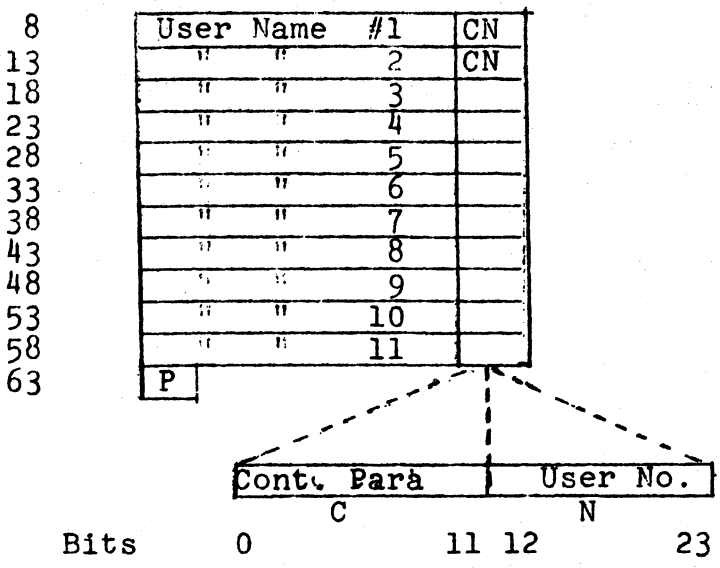


FILE DIRECTORY BLOCK



USER ACCOUNT DIRECTORY ON DISC

Words	0	1	2	3	4	5	6	7
	Acct.		Password		na	na	na	na



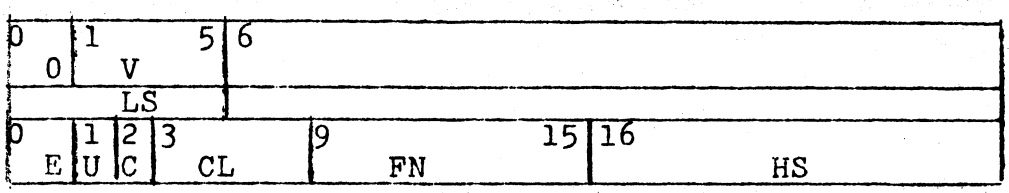
NOTES: "P" is reserved for an overflow pointer and not presently used. "na", not assigned.

The control parameter bits are assigned as follows:

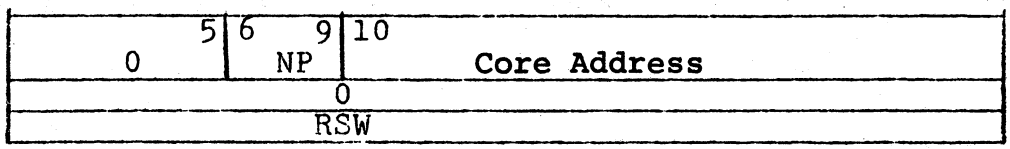
BIT	USE
0	System Status
1	Control of physical devices
2	Operator Status
3	Subsystem Status
4,5	Not assigned
6-11	Subsystem classes

SUBSYSTEM TABLE

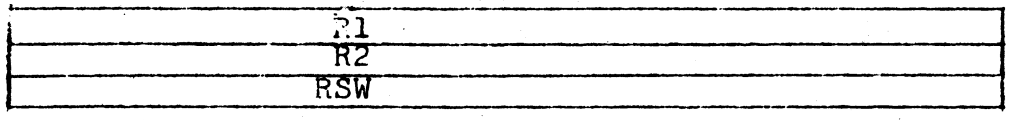
Hash Table Entry



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)