1.0 Introductory

The Berkeley Time-Sharing System is divided into three major parts: the monitor, the 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 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 exec, on the other hand, is concerned with the command language by which the user controls the system from his teletype, the identification of users and specification of the limits of their access to the system, the control of the directory of symbolic file names and backup storage for these files, and other miscellaneous matters.

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

2.0 The Scheduler

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

An active program is defined by its entry in the program active table (PAC table or PACT). This table contains all of the information required to specify the instantaneous state of the extended computer which the user is programming, except for that contained in the user's memory or in the system's permanent tables. The structure of a PACT entry is displayed on the following page, 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, B and X registers. 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 program 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 2 through 8 the number of the job to which this program belongs. The top of PQU contains information about the amount of time for which the program is allowed to compute before it is dismissed. Seven bits of QR count the number of clock cycles remaining before the program is dismissed, and three bits of QUTAB point to a table which specifies the length of time which the program should be allowed to run when it is activated. All times in the discussion are measured in periods of the 60-cycle computer clock.

A program is allowed to run for a fixed period of time, after which it is dismissed if any other programs are ready to run. This time is called a long quantum. It may be different for different programs. In fact, the size of the long quantum is determined by the entry in QTAB which is pointed to by the program's QUTAB bits in PACT.

When a program is activated, it is first allowed to run for a short quantum. During this time it cannot be dismissed except by its own request. The length of the short quantum is tentatively going to be the same for all users. It is put into a word called TIME; the long quantum is also put into a word called TTIME at this time. Both are decremented at every clock cycle.

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

Since ACTR is set to -1 when a program 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 program is dismissed for I/O, TTIME is put into QR. When the program 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 program is <u>always</u> given a full short quantum. If a program is dismissed for quantum overflow, it is given a new long quantum when it is reactivated.

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 program 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 program, although they may set up conditions in memory which will cause some other program 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 program 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 program 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 program is to be dismissed. The trap will occur and cause dismissal as soon as the program returns to user mode.

The PACT word called PTEST contains the activation condition for a currently inactive program. The condition for activation is contained in the 6 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 contain a time count, in the case where the activation condition is that a certain amount of time should elapse. It is also possible for the address to hold a mask 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.

7	Special: address =	0	dead
		1	running
		2	BRS 31
		3	BRS 106
		4	BRS 109
		5	executive BRS

11 Word 20000000= 0 (buffer ready)

12 Word less than zero

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 program in the system, i.e., a program which is executing instructions, or will be executing instructions after the currently pending interrupts have been processed. An active program (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 has dismissed itself with BRS 31, 106 or 109 (cf. section 5) it is said to be in limbo and is pointed to only by the PFORK, PDOWN, and PPAR of the neighboring programs in the fork structure.

2) If it has been dismissed for any other reason, it is on one of the scheduler queues. There are four queues of dismissed programs. In order, they are:

QTI	programs dismissed for teletype input/output
QIO	programs dismissed for other I/O
QSQ	programs dismissed for exceeding their short quanta
QQE	programs dismissed for exceeding their quanta.

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 is found, it is removed from the queue

structure and turned over to the swapper to be read in and run. If there are no programs which can be activated the scheduler simply continues scanning the queue structure.

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

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

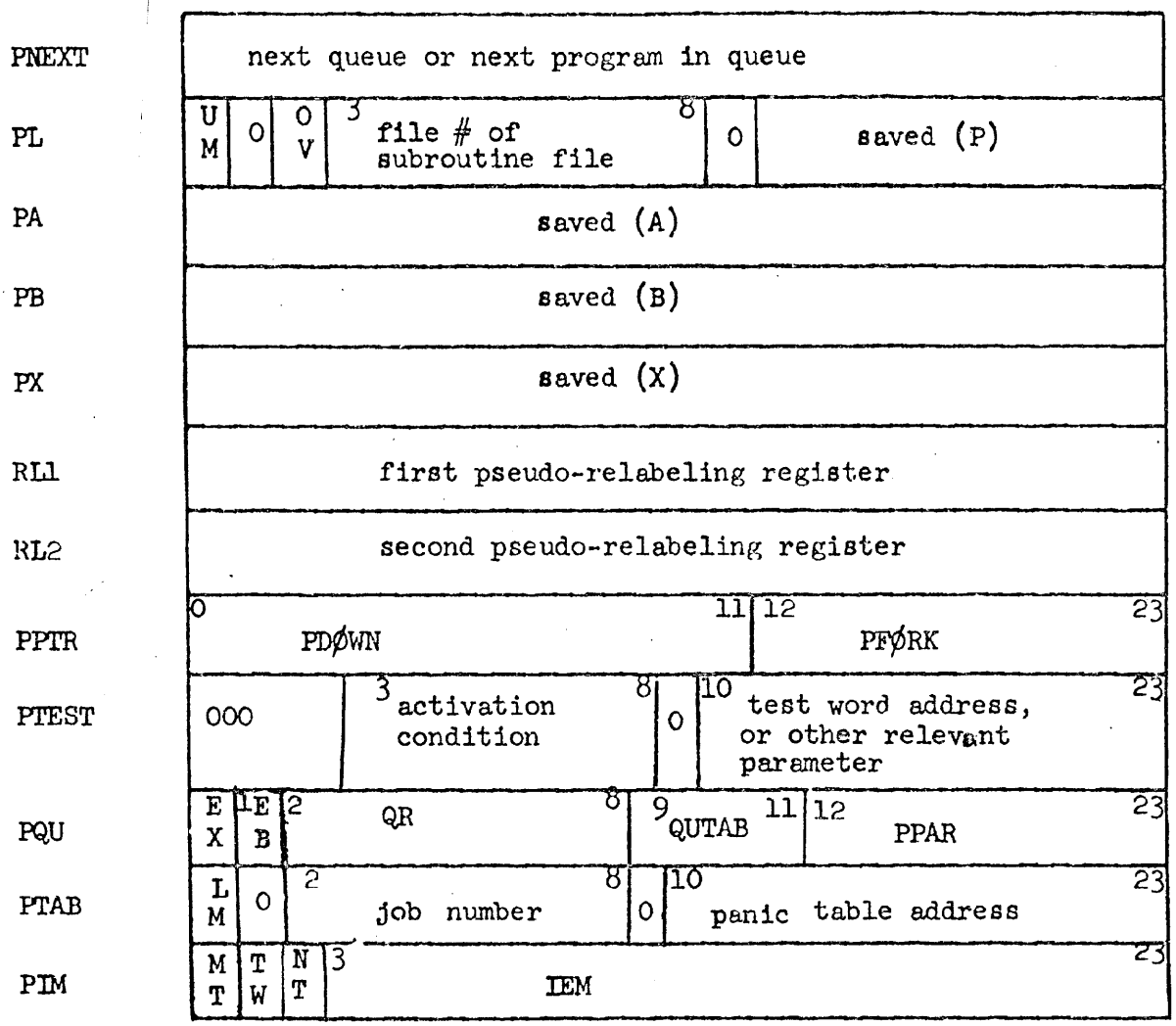
a) the cell PUCTR is non-zero

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

When the phantom user is activated by (b), it runs around the system checking that everything is functioning properly. In particular, it checks that the W-buffer has not been waiting for an interrupt for an unusual length of time, and that all teletype output is proceeding normally. Details of this procedure are described in sections 9 and 7.

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 drawn on page 2B; it is essentially a very abbreviated PAC table entry. Such an entry is made when the system has some activity which it wants to carry out more or less independently of any user PAC table entry: tests for tape ready (on rewind) and card reader ready, and processing of rubouts (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



UM = user mode (1) or system

OV = overflow

PDOWN = PACT address of lower fork (if any)

PFORK = PACT address of upper fork (if any)

PPAR = PACT address of parallel fork (ends with 0)

QUTAB = address of word in table indicating quantum lengths

EX = executive type program

EB = exec BRS

QR = amount of quantum remaining

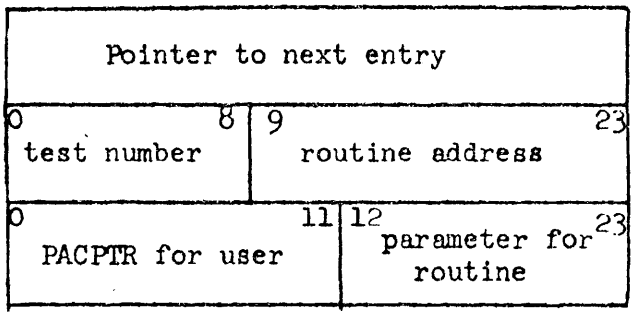
TW = waiting for termination

IEM = interrupt enabled mask

NT = non-terminable

LM = local memory

MT = add no memory



Phantom user queue entry

3.0 Forks and Jobs

3.1 Creation of Forks

A program 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 7-word table with the following format:

Program counter

A register

B register

X register

First relabeling register

Second relabeling register

Status

The status word may be:

-2 dismissed for input-output

-1 running

0 dismissed on rubout 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 forks of the same program, or overlap a page boundary. If it is, BRS 9 is illegal. The first 5 bits of the A register have the following significance:

0 make fork executive if current program is executive

1 set fork relabeling from panic table. Otherwise use current relabeling

2 propagate rubout 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.

make fork local memory. New memory will be assigned to it independently of the controlling fork.

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

The fork structure is kept track of by pointers in PACT. For each program 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 on the previous page. The arrows indicate the various pointers.

The program executing a BRS 9 continues execution after the instruction. 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 program in the following three ways:

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

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

BRS 31 causes the controlling program to be dismissed until the subsidiary fork causes a panic. When it does, the controlling program 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 into X.

BRS 32 causes a subsidiary 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 program to be dismissed until any subsidiary fork causes a panic. When it does, the controlling program 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 (cf. section 14) has been exceeded. If so, a memory panic (see below) 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 propagated 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 drum 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 condition which may cause a fork to be terminated are listed in the description of the status word above. When any of these conditions occurs, 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 program panic and may be caused by either of two conditions:

A) the rubout button on the controlling teletype is pushed. This terminates some fork with a program panic. A fork may declare that it is

the one to be terminated by executing BRS 90. In the absence of such a declaration the highest user fork is terminated. When a fork is terminated in this way its controlling fork becomes the one to be terminated. If a user fork is terminated by rubout 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 rubout has armed interrupt 1, this interrupt will occur instead of a termination. The teletype buffers will not be affected. If there is only one fork active, control goes to the location EXECP in the executive. This consideration is of no concern to the user. Executive programs can turn the rubout button off with BRS 46 and turn it back on with BRS 47. A rubout occurring in the meantime will be stacked. A second one will be ignored. A program which is running with rubout turned off is said to be non-terminable and cannot be terminated by a higher fork. BRS 26 skips if there is a rubout pending.

If two rubouts 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 rubouts can be conveniently generated with the repeat button on the teletype.

B) A BRS 10 may be executed in the lower fork. This condition can be distinguished from a panic caused by the rubout 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.

An executive program can clear the fork structure of a job by putting the job number in A and executing BRS 22. The effect is as though enough rubouts had occurred to send the job back to the top-level executive fork.

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.

If interrupt 2 is armed and the fork is executive, interrupt 2 will occur instead of an illegal instruction panic.

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 block. If interrupt 2 is armed, it will occur instead of the memory panic.

3.4 Debugging Fork Structure

Some special machinery exists in the monitor to assist in the debugging of programs with complex fork structure. The use of this machinery is restricted to executive type programs. The idea behind it is that it is possible to detach a section of a user's fork structure and leave it hanging, and later to re-attach it and continue execution.

An executive type program may give a so-called wait command, BRS 74, which sets up a special activation condition for the fork. This instruction is otherwise equivalent to BRS 31. Any program may give a freeze command, BRS 75, which causes the fork structure to be scanned upward for a program with this activation

condition. If such a program is not found, an illegal instruction trap is generated. Otherwise, all the forks below the one found are removed from the queue structure. The status of each fork is read into the panic table of the immediately higher fork as though each fork had been terminated. The fork found is restarted and it is given the PACT address of the next lower fork in A, the location of the freeze command in X, and the depth of the fork containing the freeze command in the fork structure in B. All the forks below the one being reactivated are entirely disconnected from the rest of the fork structure. The only handle on these forks is the PACT address which is returned to A.

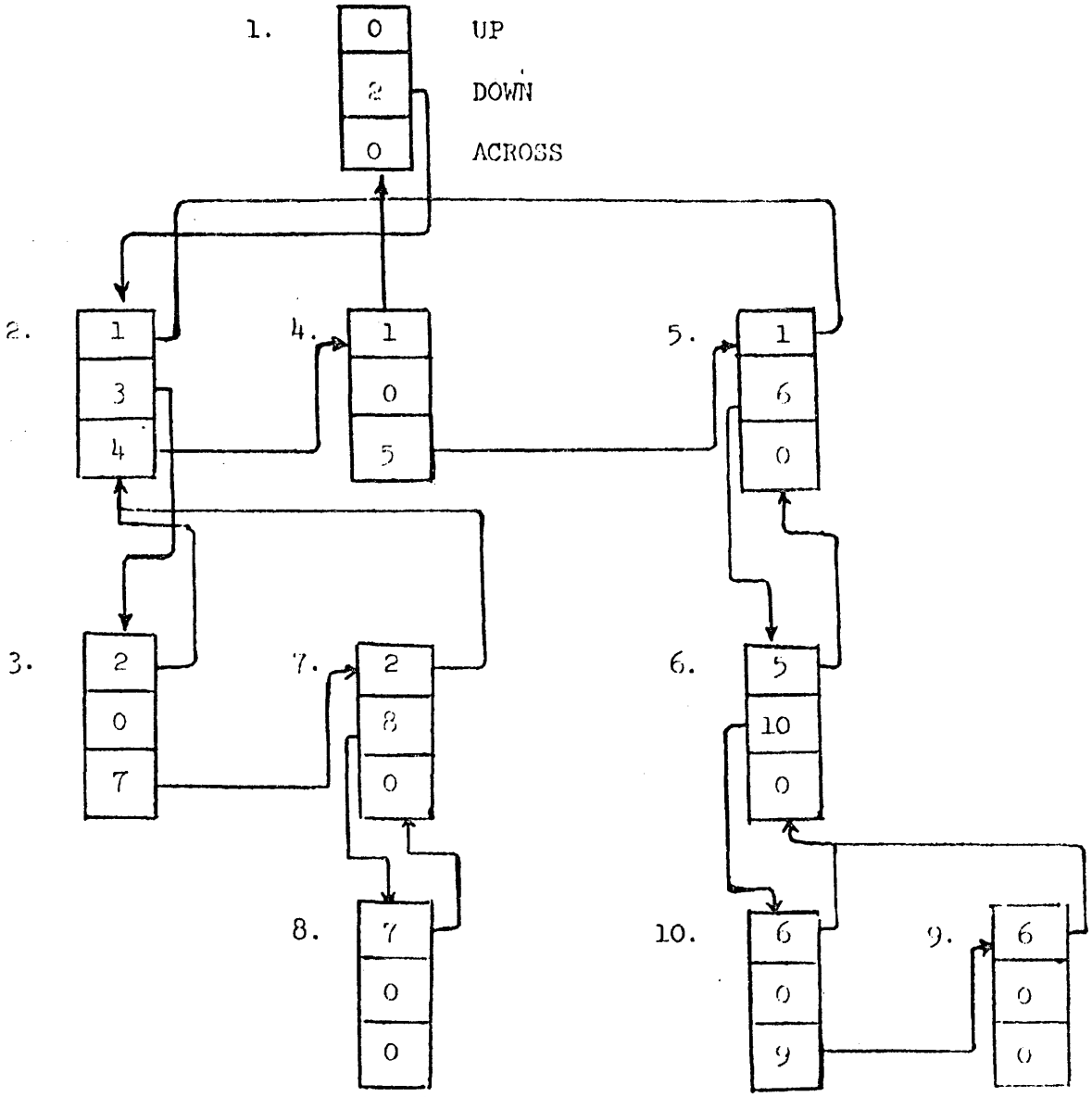
After the waiting program has been reactivated, it may proceed to generate a new fork structure. Eventually, however, it will wish to do something with the fork structure which has been frozen. It may do one of two things: the melt command, BRS 76, takes in A the PACT address which was returned by the freeze command, and re-establishes the fork structure as it was when the freeze command was given. It does not, however, activate any of the forks, since this will be done by the scheduler in the normal course of events.

If the fork which did the wait does not wish to reactivate the frozen fork structure, it may simply destroy it by putting the PACT address which it received from the freeze command into A and executing BRS 77. This causes all the PAC table entries in this structure to be returned to the free program list. No other action is taken.

A variant of freeze is provided by BRS 89, which takes a panic table address and freezes the fork structure beginning with the subsidiary fork which has the specified panic table and continuing to all its subsidiary structure. The action taken is similar to that taken in a freeze, and the PACT address of the disconnected fork structure is returned in A.

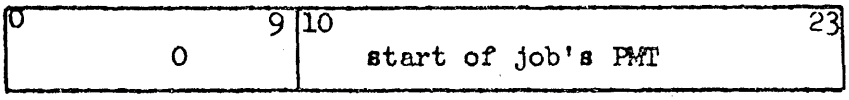
3.5 Jobs

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 shown on page 3B, and indicate more or less what it is that is specifically associated with each job.

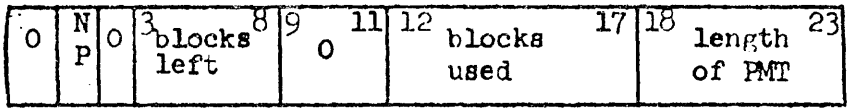


Hierarchy of Processes

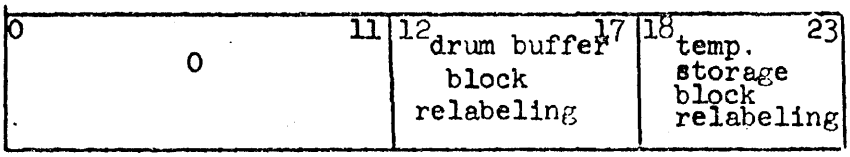
PMTP



PMA



RL3



TTNO teletype associated with this job

ETTB amount of CPU time used

DBA drum blocks available

NP= don't charge memory against machine size.

4.0 Program Interrupts

A facility is provided in the monitor to simulate the existence of hardware interrupts. There are 20 possible interrupts; four are reserved for special purposes and 16 are available to the programmer for general use. A fork may arm the interrupts by executing BRS 78 with a 20-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 number 1, 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 the interrupt number is simulated in the fork which armed the interrupt. Note that the program counter which is stored in the 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 or four. 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 hierarchy (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 rubout button) 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. This permits the programmer to control the action taken when the rubout button is pushed without establishing a fork specifically for this purpose. If pushing the rubout button causes an interrupt to occur rather than terminating a fork, the input buffer will not be cleared.

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

Interrupt 3 is caused, if armed, when any subsidiary 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).

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.

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

5.0 The Swapper and Memory Allocation

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 consists, as do the ordinary relabeling registers, of 8 six-bit bytes. Each one of these bytes points, however, not to a real block of memory, but to an entry in the user's pseudo-memory table, PMT (but see below). This table may contain up to 64 words, each one specifying a certain 2K block of memory. The first version of the system, however, will allow access to only 14 bytes. The possible forms of an entry in the pseudo- memory table are shown on page 5A. All of the entries are more or less self-explanatory, except the second, which will be discussed in considerable detail later.

When it is necessary to activate a user, his pseudo-relabeling registers are used to read out the proper bytes from PMT and construct a list of blocks which need to be read in from the drum. When this list has been constructed, the current state of core is examined to determine whether any blocks need to be written out to make room for these which must be read in. If so, a list of blocks to be written out is constructed. The drum command list is then set up with the appropriate commands to write out and read in the necessary blocks. The scheduler then passes on in an attempt to activate another program while the drum operations are being performed. If it is not successful in doing so, it simply hangs up until the swapping is complete. In the scan which sets up the drum 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. It then outputs these relabeling registers to the hardware and activates the program.

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

There are two BRS's which permit the user to read and write his pseudo-relabeling. BRS 43 reads the current pseudo-relabeling registers into A and B. BRS 44 takes the contents of A and B and puts them into the current pseudo-relabeling registers. An executive program may set the relabeling registers in arbitrary fashion by using this instruction. A user program, however, may add or delete only blocks which do not have the executive bit set in PMT. This prevents the user from gaining access to executive blocks 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 blocks 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 on page 3-2. A user can release a block which is in his current relabeling by putting any address in that block into A and executing BRS 4. The PMT entry for the block is removed and in any other fork which has this PMT byte in its relabeling, the byte is cleared to 0.

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

A word of PMT whose first two bits are 01 contains a pointer to the shared memory table, SMT. An entry in SMT looks exactly like an unused or private entry in PMT. The read-only bit is not used by the swapper, however, since the read-only status of the page is taken from the PMT word which points to it. It refers to a block of memory which has a fixed location on the drum and may be referred to by more than one program. Entries in SMT may be made by the exec or by a user.

The exec makes an entry in SMT by executing BRS 68 with a byte number in A. The block address by the specified byte in the pseudo-relabeling registers is put into SMT and the pointer in SMT of this byte is returned. If bit 0 of A is set, the byte is made read-only. 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 read-only bit in the SMT entry is propagated to the PMT entry thus created. To delete an entry in SMT, the exec may deliver its index in A and execute BRS 70. The detailed use of this machinery is discussed in section 16.

The user may declare a block read-only by executing BRS 80 with the pseudo-relabeling byte number of the block in A and with bit 0 of A set. To make a block read-write, bit 0 of A should be clear. Bit 0 of A will be reset if the block was formerly read-write or set if it was formerly read-only. If the program doing this is not an executive program, then the block must not be an executive block. 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 block to the swapper, of course, is that it need not be rewritten on the drum when it is removed from memory.

The drum is divided into 84 bands, each containing 16,000 words arranged in 8 blocks of 2K each. Up to 48 of these bands may be used by the swapper for program storage. A bit table is maintained to indicate the availability of 2K blocks in these bands. The table consists of 8 words, each containing 24 bits, one for each band. If a bit is zero, it indicates that the block is in use. If it is set , the block is available. When the user's memory is written out onto the drum, it is written as nearly as possible in adjacent blocks, so that it may be read in without undue drum latency time. This method for keeping track of available blocks facilitates optimum output of the user's program.

It should be noted that whenever a user is activated, 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.

For a user with an especially great need for rapid response, an instruction is provided to make a block permanently resident in core. Use of this instruction is restricted to users with the appropriate bit of the user directory set. To make a block permanently resident, execute BRS 55 with the pseudo-relabeling byte number of the block in A. To make the block swappable again, execute BRS 55 with 0 in A. To reserve a block of core memory for the use of BRS 55 instructions, execute BRS 54 with 0 in A. To release this block, execute BRS 54 with -1 in A.

To make a block executive, execute BRS 56 with the same argument as for BRS 80, make block 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 occurrence of the 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 block in core. BRS 55 also does this. No block with non-negative RMC can be released by the swapper.

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

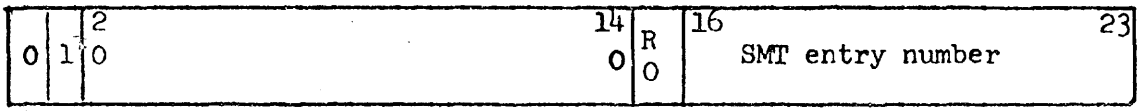
USE = in use RO = read-only

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 blocks 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 blocks with lowest RMA are selected for swapping out; of course their RMC entries must be negative.

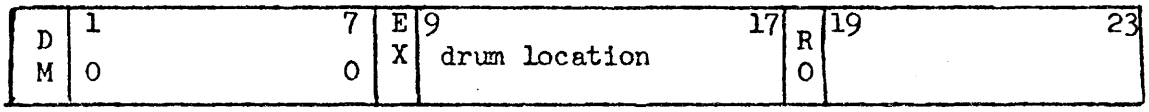
The swapper also contains a device called the simulated associative memory or SAM, which contains pseudo-relabeling and real relabeling for the most recently used maps. It serves to reduce the amount of time needed for map-changing when little swapping is taking place. It is cleared whenever a drum read 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 2K blocks at specified places on the drum. They are of course restricted to executive programs. To read a block, put the drum address into B and the core address in A and execute BRS 104. To write a block use BRS 105. Drum errors cause these instructions to generate illegal instruction panics.

(a) Unused



(b) Shared



(c) Private

DM = on drum

EX = executive

RO = read only

Possible Formats for PMT Entries

6.0 Miscellaneous Features

A user may dismiss his program for a specified length of real time by executing BRS 31 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 program will be reactivated. This feature is implemented with a special activation condition and the value of the clock at the time when a user is to be reactivated is kept in the address of PTEST. The activation condition causes the current value of the clock, modulo 2^{14}, to be compared with this value. When the clock becomes greater, it is time to reactivate the program.

He can read the real-time clock into A by executing BRS 42. The number obtained increments by one every 1/60th of a second. Its absolute magnitude is not significant. He can read the elapsed time counter in 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 program is running.

To obtain the date and time, he can execute BRS 20. This puts six 8-bit characters into AB. These characters contain, in order, the year, month, day, hour (0-23), minute and second at which the instruction is executed.

A user may dismiss his program until an interrupt occurs or the fork in question is terminated by executing BRS 109.

A program can test whether it is executive or not by executing BRS 71, which skips in the former case.

An executive program can dismiss itself explicitly. See section 2.

There are two operations designed for so-called executive BRSs, which operate in user mode with a map different from the one they are called from. BRS 111 returns from one of these BRSs, transmitting A, B and X to the calling program 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 program. A memory panic can occur, in which case it appears to the calling program that it comes from the BRS instruction.

An executive program can cause an instruction to be executed in system mode by addressing it with EXS.

7.0 Teletype Input-Output

We begin with an outline of the implementation of the teletype operations. This may 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 given on the following page. Also attached to the teletype is a buffer which contains input and output characters in the following format:

0 — 7	8 — 15	16 — 23
input character	output character	character to echo (if any)

As characters are output by the program, they are added to the output buffer, which may be regarded as logically independent from the input buffer in spite of the fact that it resides in the same words. The characters are then output by the teletype interrupt routine as rapidly as the teletype will accept them.

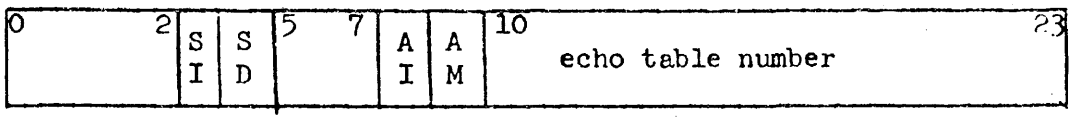
These buffers are called character ring buffers (CRBs), and they are not necessarily attached to teletypes. This question is discussed in detail in section 9.5.

When a character is typed in on a teletype, it is converted to internal form and added to the input buffer unless it is rubout on a controlling teletype. The treatment of rubouts is discussed in section 3. The echo routine address is then obtained from TTYTBL and called. It figures out what to echo and whether or not the character is a break character. The available choices of echos and break characters are listed below. 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 (TØS2 > -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 routines in the system, referred to by the numbers 0, 1, 2 and 3. 0 is a routine in which the echo for each character is the character itself, and all characters are break characters. Routine 1 has the same echos, but all characters except letters, digits and space are break characters. Routine 2 again has the same echos, but the only break characters are control characters (including carriage return and line feed). Routine 3 specifies no echo for any character, and all characters are break characters. This routine is useful for a program which wishes to compute the echo itself.

To set the echo routine, put the teletype number in X and the echo routine 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 routine number into A, put the teletype number in X and execute BRS 40. This operation returns in A the following word:



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 8 bits of location M. In addition to the ordinary ASCII characters, all teletype output 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 on output SYSPOP) occurs. The top bit is left clear unless the activity is a rubout input. This cell is checked

a) by the rubout processor to determine whether the rubout should reset the job to the exec. See p3-6

b) by the phantom user's ten-second routine to check that no output interrupt has been dropped by the teletype interface. If no activity has occurred for 2 seconds and characters are waiting to be output, the interface is awakened.

c) by the phantom user's ten-second routine to check whether any activity has taken place in the last 30 seconds. If not, a control character is output to reassure the user that the system is alive.

Every teletype in the system is at all times in one of three states:

a) It may be the controlling teletype of some user's program. It gets into this state when a user enters on it.

b) It may be attached to some user in a manner about to be described.

c) It may be completely free.

The status of the teletype is reflected by the contents of TTYASG. There are mechanisms to be described by which the user may direct output to any teletype

in the system which is willing to accept it and receive input from any teletype which is not free. If, however, he wishes to have better control over a teletype (for instance, to prevent other users from accessing it) he may attach it by executing the instructions

```
LDA     =teletype number
BRS     27
```

If the indicated teletype is free, it is attached to the user whose program executes the instruction, and the BRS will skip. Otherwise the teletype status is not affected, and the BRS does not skip. In the following discussion we will say that a teletype is attached to a user even if it is the controlling teletype.

To release an already attached teletype, execute the instructions

```
LDA     =teletype number
BRS     28
```

If the specified teletype is not already attached to the user, this is an illegal instruction and causes a panic. All attached teletypes are, of course, released when the user logs out.

A teletype becomes a controlling teletype if it is dormant and rubout is psuhed on it. It can be returned to its dormant state by BRS 112, which takes the job number of the job associated with the teletype in X. A job may terminate itself. This operation also releases all teletypes attached to the job.

The user may specify for his controlling teletype or for one which he has attached, whether or not messages from outside will be accepted, and whether or not input from outside will be accepted. The former condition is governed by the accept messages bit, the latter by the accept input bit. The accept message bit controls execution of OST instructions and the setting of teletype output links. The accept input bit controls execution of STI instructions and the setting of teletype input links.

To set these bits, the user may execute

```
LDX     =teletype number
LDA     BITS
BRS     25
```

The last bit of BITS will set the accept input bit, the next to last the accept messages bit. Setting or clearing these bits will not affect any teletype links currently active.

To do input and output to specified teletypes (rather than implicitly to a controlling teletype as in TCI and TCO) the SYSPOPs IST and OST are available. To input a character from a specified teletype, execute the instruction

```
IST     =teletype number     (input from specified teletype)
```

which brings the character into the A register. This instruction is illegal unless the teletype is attached to the user. To output a character to a specified teletype, execute the instructions

```
LDA     =character
OST     =teletype number          (output to specified teletype)
```

This instruction is illegal if the following three conditions are satisfied:

(1) the specified teletype is not attached to the user,

(2) the specified teletype does not have its accept messages bit set,

(3) the program executing an instruction is a user rather than an executive program. If these conditions are satisfied, an illegal instruction panic will be generated.

Note that attached teletypes do not have the same status as the controlling teletype for a user. In particular, pushing the rubout button on an attached teletype will have no effect.

The instruction

```
CIO     =teletype number + 1000
```

is exactly equivalent to

```
IST     =teletype number.
```

The instruction

```
CIO     =teletype number + 2000
```

is exactly equivalent to

```
OST     =teletype number.
```

This mechanism permits the user to do I/O to specified teletypes within the framework of the sequential file machinery.

The user has considerable control over the state of the teletype buffers for the teletypes attached to him. 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.
```

There is one additional piece of machinery which permits output to go to a teletype other than the controlling teletype. This machinery is implied by the top bits of TTYTBL, which specify whether any link bits are set. Associated with each teletype are two words called the absolute input link control word (LCW) and the absolute output LCW. Each of these words contains one bit for each teletype in the system. If the bit for teletype m is set in the input LCW, for teletype n every character which goes into n's input buffer will also go into m's input buffer. If the bit is set in the output LCW, every character which is output to n, including echoes, will also be output to m.

Also associated with each teletype are relative LCW's for input and output. The bits in these LCW's are set by BRS 23. Each time any relative LCW is changed, the absolute LCW's are all recomputed. The Boolean matrix formed by the absolute

input (output) LCW's is the infinite product of the matrix of the relative input (output) LCW's.

The instructions

```
LDX     =teletype number
LDA     =TABLE
LDB     CTL
BRS     23
```

will set one of the relative LCW's for the indicated teletype. TABLE is the address of a list of teletype numbers terminated with -2. The bits of CTL are interpreted as follows:

0	0=output LCW
	1=input LCW
1	0=clear all links first
	1=do not clear links first
2	0=set link bits for teletypes whose numbers are in the table
	1=clear link bits for teletypes whose numbers are in the table

From the old relative LCW and the information supplied by BRS 23 a new relative LCW is created. New absolute LCW's for all teletypes are then computed.

An output link can be set up between two teletypes only if each of the teletypes satisfies at least one of the following conditions:

a) it is the controlling teletype of the program executing BRS 23

b) it is attached to the program

c) its accept messages bit is on (destination only)

d) the fork executing the BRS is executive.

An input link can be set up only if the same conditions are satisfied for the accept input bit.

To clear all links, input and output, to or from a teletype, execute

```
LDX     =teletype number
BRS     24
```

Special provision is made for reading 8-bit codes from the teletype without sensing rubout or doing the conversion from ASCII to internal which is done by TCI. To switch a teletype into this mode, execute

```
LDX     =teletype number
LDA     =terminal character + 40000000B
BRS     12
```

This will cause each 8-bit character read from the teletype to be transmitted unchanged to the user's program. The teletype can be returned to normal operation by

(1) reading the terminal character specified in A, or

(2) setting the echo table with BRS 12.

No echoes are generated while the teletype is in 8-level mode. Teletype output is not affected.

A parallel operation, BRS 85, is provided for 8-level output. BRS 86 returns matters to the normal state, as does any setting of the echo table.

To simulate teletype input, the operation

```
STI     =teletype number
```

is available. STI puts the character in A into the input buffer of the specified teletype. It is legal only if the accept input bit is on.

TELETYPE TABLE

TIS2 — number of characters in input buffer

TIS4 — next available space in input buffer (pointer)

TIS5 — next filled space in input buffer (pointer)

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

TOS3 — < 0 = not in multiple blank mode; 400 = just saw 135 (multiple blank character); other = number of blanks

TOS4 — next filled space in output buffer (pointer)

TOS5 — next available space in output buffer

TTYTBL

NS	0	0	SI	SO	IL	OL	0	AI	AM	address of echo routine (10–23)

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

TTYBRK — waiting for break character when -1

TTYASG

	TTY Status
PACPTR of fork to terminate on rubout	active
3 7 7 7 7	inactive
controlling job (18–23)	attached

ROLCW — relative output link control word

RILCW — relative input link control word

TTYTIM

RB	value of clock when last action occurred on this tty

TTYDEV — device (normally physical teletype) using this buffer.

NS = not linked or 8-level
AM = accept message
IL = input linked
OL = output linked
AI = accept input
SI = 8-level input
SO = 8-level output
RB = last action was input of rubout

8.0 Drum and Buffer Organization; Devices

8.1 File Storage on the Drum

The drum is divided into two major sections, program swapping and file storage. The organization of the program swapping area is discussed in section 5. 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 144 words of the index block are used, however, for data storage. A couple of additional words are used to chain the index blocks for any particular file, both forward and backward. The index blocks for a file contain the addresses for all the physical blocks used to hold information for the file.

Available storage in the file area of the drum is kept track of with a bit table similar to the table used to keep track of program swapping storage. Since there are sixty-four 256-word blocks around the circumference of the drum and a maximum of 72 drum bands (out of the 84 available) may be used for file storage, a 192-word bit table which contains 3 words or 72 bits for each row of physical blocks suffices. If a bit in this table is set, it indicates that the corresponding block on the drum is in use. Again, as with program swapping storage, the organization of this table makes it easy to optimize the writing of files. This is done by putting consecutive physical blocks in the file in alternating rows on the drum. The intervening row between each two physical blocks provides the time for the channel to fetch a new command and the heads to switch. The result of this organization is that information may be transferred from a file on the drum into core at one-half core memory speed if conditions are right.

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 on page 8A. 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 not used only by drum files, but is present in all cases.

Each job has associated with it a temporary storage block, which is always the first entry in the job's PMT. This block is used to hold information about the user and for the system's temporary storage. It also has room for 3 buffers. An additional block may be assigned with room for 5 more buffers if more than 3 files are open at one time. The pseudo-relabeling for the extra buffer block and the TS block 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.

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 its dormant state, both words point to the first data 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 system has a device number. The numbers applicable to specific devices are given in section 9; here the various tables indexed by device number are described. 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 (section 4.3) which is addressed by the file number.

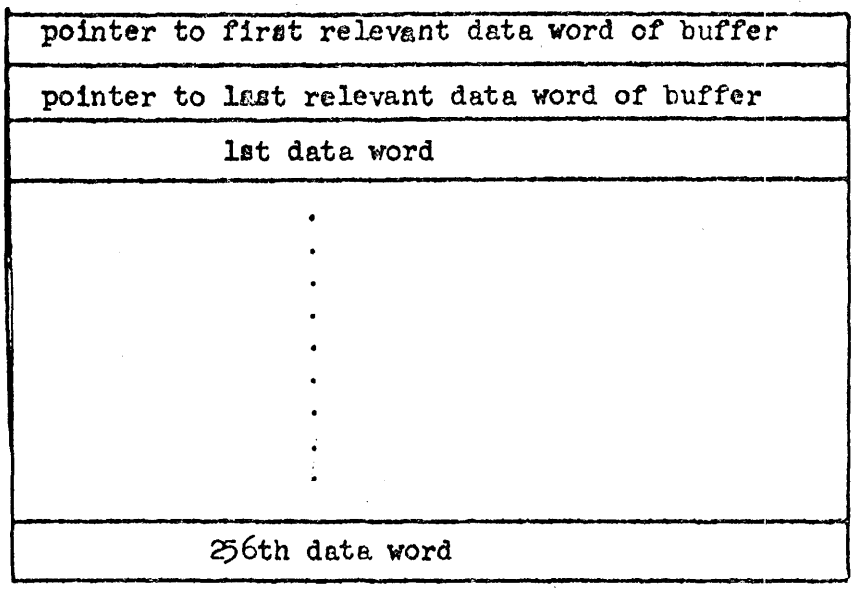
Page 8B shows the tables indexed by device number. Note the multiplicity of bits which specify the characteristics of the device. Some of these call for comment. A device may be common (shared by users, who must not access it simultaneously; e.g. tape or cards) or not common (e.g. drum); this characteristic is defined by NC. It may have units; e.g. there may be multiple magtapes. 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

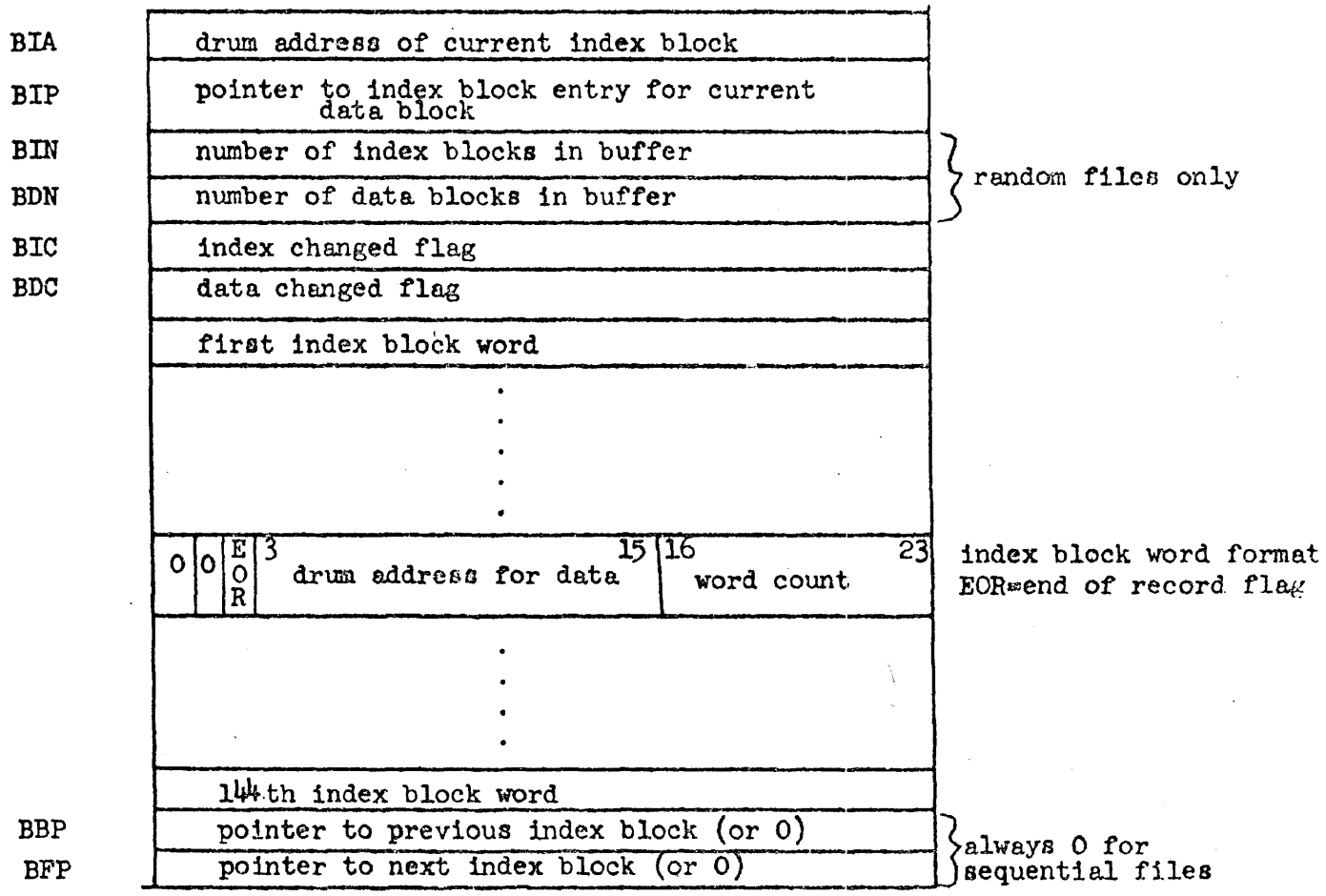
- the opening routine, which is responsible for the operation necessary to attach it to a file
- the GPW routine, which performs character and word I/O
- the BIO routine, which performs block I/O

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.

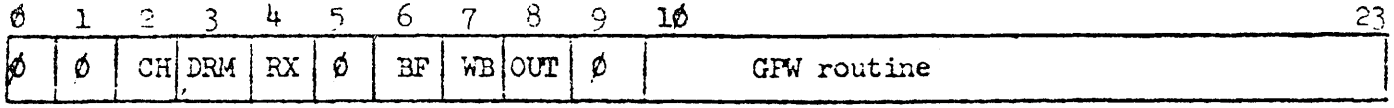


(a) Layout of a file buffer



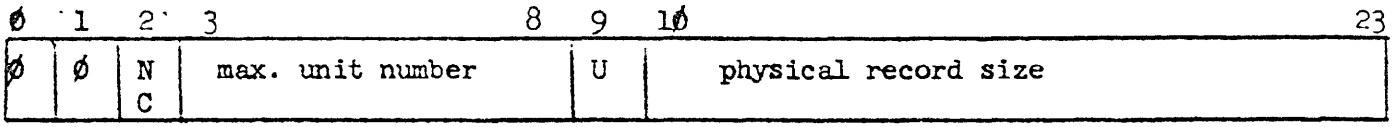
(b) Layout of index block buffer and associated pointers for a drum file.

DEV word or character I/O routine



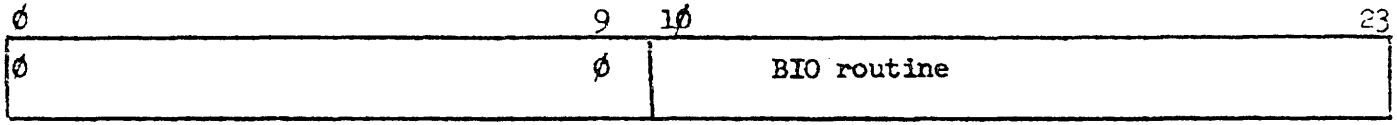
CH char oriented RX random access WB W buffer
DRM drum BF requires buffer OUT output

BUFS
buffer size

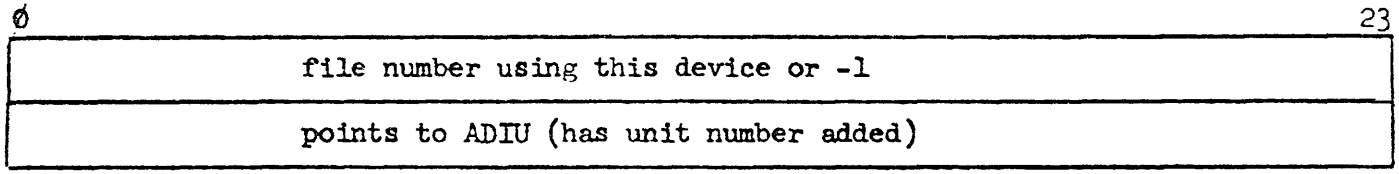


U check unit number NC not common (i.e. don't set DIU)

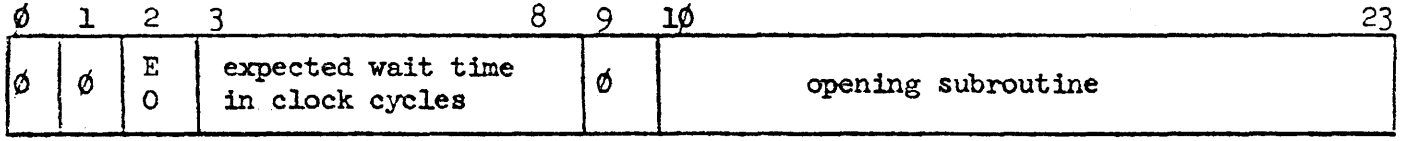
BDEV
block I/O routine



DIU
device in use



OPNDEV
opening routine



EO exec only allowed to open

9.0 Sequential Files

9.1 Sequential Drum Files

There are two basically different kinds of files which the user may write on the drum: sequential and random. A sequential file has a structure very similar to that of an ordinary magtape file. It consists of a sequence of logical records of arbitrary length and number. Drum 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 drum file may be opened by the following sequence of instructions:

```
LDX     =device number, 8 (input) or 9 (output)
LDA     =unit number, 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 executive type programs. User programs may access drum files only through the executive file handling machinery. BRS 1 can also be used to open other kinds of files. The device and unit numbers are used to determine the physical location of the file. See section 9.2.

If BRS 1 fails to skip, it returns in the A register an indication of the reason:

-2 too many files open -- no file control blocks or no buffers available.

-1 device already in use. For the drum, produced by an attempt to open a file for output twice

0 no drum space left. This inhibits opening of output files only.

See section 9.2 for other error conditions.

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 severs his connection with the file. 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 SYSPOPs handles input and output indifferently, since the file must be specified as an input or an output file when it is opened. If the user desires to read and write on the same file at the same time, he should open it twice, once as an input file and once as an output file.

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 7 in the file number (these are called flag bits) and return a 134 or 137 character, respectively. If interrupt 4 is armed, it will occur. The end of record condition occurs on the next input operation after the last character of the record

has 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.

To input a word to the A register or output it from the A register,

```
WIO     =file number
```

is executed. An end of record condition returns a word of three 134 characters as well as setting the flag bit, and an end of file returns a word of three 137 characters. If the condition occurs when a partially filled-out word is present, the word is filled out with one of these characters.

Mixing word and character operations will lead to peculiarities and is not recommended.

To input a block of words to memory or output them from memory, the instructions

```
LDX     =first word address
LDA     =number of words
BIO     =file number
```

should be executed. The contents of A, B and X will be destroyed. The A register at the end of the operation contains the first memory location not read into or out of.

If the operation causes any of the flag bits to be set, it is terminated at that point and the instruction fails to skip. If the operation is completed successfully, it does skip. Note that a BIO cannot set both the EOR and the EOF bits.

BIO is implemented with considerable efficiency and is capable of reading a properly written file (or writing any file) at one-half the maximum drum transfer rate.

The flag bits (0 and 7 or 8) of the file number are set by the system whenever end-of-record or end-of-file 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 7) of the file number may be set. An output operation never sets either one of these bits. Bit 6 of the file number may be set on an error condition (which can occur only on devices other than the drum). Whenever any flag bit is set as a result of an input-output operation in a fork, interrupt 4 will occur in that fork if it is armed.

The CTRL SYSPOP provides various control functions for sequential drum files. To use this operation, execute the instructions

```
LDA     =control number
(LDB    =record count, if required)
CTRL    =file number
```

The available control numbers are

1	write end of record on output or skip the remaining part of the logical record on input. This control does not take a record count.
2	Backspace (B) records
3	Forward space (B) records.
4	Delete (B) records (legal on output only).
5	Space to end of file and backspace (B) records.
6	Space to beginning of file and forward space (B) records.
7	Insert logical record (legal on output file only). This control does not require record count.
8	Write end of file (output only).

The user may delete all the information in a drum file by executing the instructions

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

He may also eliminate the file entirely by giving an executive command described in section 14.

The index block for a sequential drum file contains one word for each physical record in the file. This word contains the address on the drum of the physical record in the bottom 12 bits. The top bit is set if the physical record is the last record of a logical record. The intervening bits indicate the number of data words in the physical record. A sequential file may have only one index block, or a maximum of 146 X 256 = 37376 words of data.

Putting the file number of a sequential file in A and executing BRS 113 will cause the file to be rewound, scanned to find the total number of data words, and rewound again. 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 drum 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.

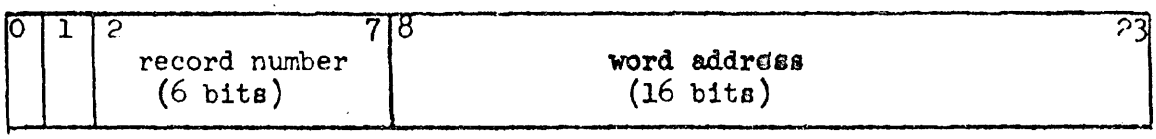
```
BRS     67
```

returns the drum block with address in A to available storage. To read an index block into core

```
BRS     87
```

may be used. It takes the address of the block in A and in X the first word in core into which the block is to be read.

A single word of a sequential file may be directly addressed by specifying the logical record number and word number within the logical record. All the operations legal for random files (see section 10) can also be used for sequential files with this convention. The format of the address is



9.2 Other Sequential Files

In addition to drum sequential files, the user has some other kinds of sequential files available to him. These are all opened with the same BRS 1, except for the device number. Available device numbers are

Paper tape input	1
Paper tape output	2
Card input	3
Magtape input	4
Magtape output	5
PDP-5 link input	6
PDP-5 link output	7

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 (see p. 9-8).

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 drum 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 record
3 forward space file
4 backspace file

5	write three inches blank tape
6	rewind
7	write end of file

These controls may be executed only by executive type programs. I/O operations to the magtape may, of course, be executed by user programs if they have the correct file number.

An executive program may arrogate 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:

1) All non-trailing strings of more than 2 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 characters 155 and 152 (carriage return and line feed) at the end of each card.

The result of all this machinery is that the string of characters obtained by reading in a card deck may be output without change to a teletype and will result in a correct listing of the deck.

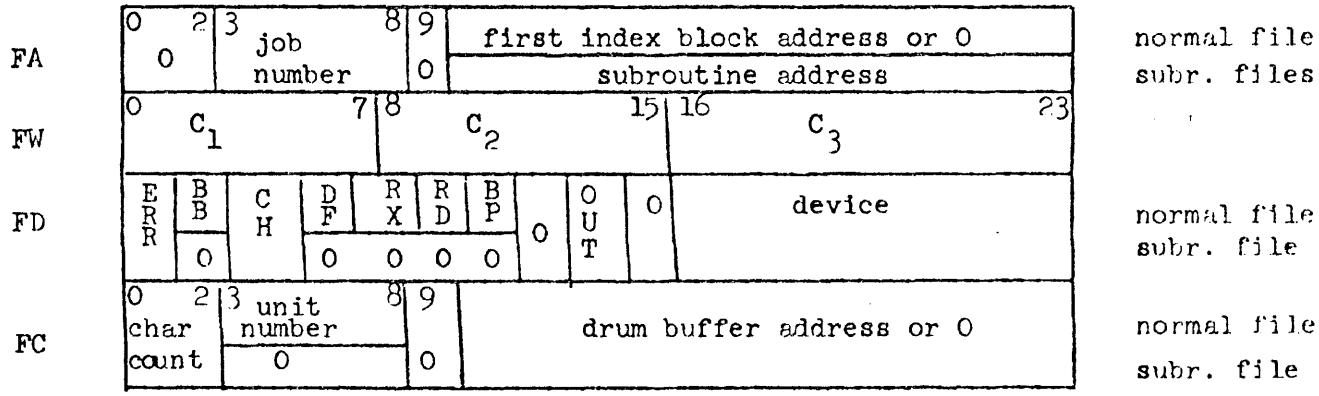
Whenever a card reader error (feed check or validity check) occurs, the program is dismissed until the reader becomes not ready.

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

The phantom user's 10 second routine checks to see whether a W-buffer interrupt has been pending for more than 10 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:



Cn = word being packed or unpacked

char count = -1 to 2

CH = character oriented

ØUT = output

BB = buffer busy

DF = drum file

Drum files only:

RX = random access

RD = read only

BP = buffer in use and protected

ERR = error

9.4 Permanently Open Files

There are a few built-in sequential files with fixed file numbers:

0	controlling teletype input
1	controlling teletype output
2	nothing (discard all output)
1000+n	input from teletype n
2000+n	output to teletype n

These files need not be opened and cannot be closed.

9.5 Character Buffers

Section 7 describes the format of a teletype buffer. Such an object is capable of dealing with any character-oriented device, not merely with a teletype. For this reason the character ring buffers are not directly indexed by the physical number of the teletypes to which they are attached. Instead, a table indexed by physical teletype number is used to obtain the buffer number. It is possible for other devices to obtain buffers; the mechanism for doing this is not spelled out in detail at the moment.

10.0 Random Drum Files

A random drum file is very similar in physical structure on the drum to a sequential drum 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 256 and a location of the word within the block which is the remainder of this division. Further division by 144 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. Drum 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 POPs work exactly like DWI and DWO except that they take the drum 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 SYSPOPs should contain the drum 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 drum file space is very similar to the mechanism for allocation of core memory. Whenever the user addresses a section of a random drum 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 drum memory, execute the instructions

```
LDA     =number of words to be zeroed
LDB     =initial word to be zeroed
LDX     =file number
BRS     59
```

The specified section of the file is cleared to zero. Physical blocks which are entirely zero will be released. A more drastic clearing operation may be obtained with BRS 66, which deletes the entire information content of the file.

11.0 Subroutine Files

In addition to the above-mentioned machinery for performing input-output through physical files, a facility is provided in the system for making a subroutine call appear to be an input-output request. This facility makes it possible to write a program which does input-output from a file and later to cause further processing to be performed before the actual input-output is done, simply by changing the file from a physical to a subroutine file. A subroutine file is opened by executing the instructions

```
LDX     parameter word
BRS     1
```

This instruction never skips. The opcode field of the parameter word indicates the characteristics of the file. It may be one of the following combinations:

11000000	Character input subroutine
11100000	Character output subroutine
01000000	Word input subroutine
01100000	Word output subroutine

I/O to the file may be done with CIO or WIO, regardless of whether it is a word or a character oriented subroutine. The system will take care of the necessary packing and unpacking of characters. BIO is also acceptable.

The opening of a subroutine file does nothing except to create a file control block and return a file number in the A register. When an I/O operation on the file is performed, the subroutine will be called. This is done by simulating an SBRM to the location given in the word following the BRS 1 which opened the file. The contents of the B and X registers are transmitted from the I/O SYSPOP to the subroutine unchanged. The contents

of the A register may be changed by the packing and unpacking operations necessary to convert from character-oriented to word-oriented operations or vice versa. The I/O subroutine may do an arbitrary amount of computation any may call on any number of other I/O devices or other I/O subroutines. A subroutine file should not call itself recursively.

When the subroutine is ready to return, it should execute BRS 41. This operation replaces the SBRR which would normally be used to return from a subroutine call. The contents of B and X when the BRS 41 is executed are transmitted unchanged back to the calling program. The contents of A may be altered by packing and unpacking operations. A subroutine file is closed with BRS 2 like any other file.

In order to implement BRS 41, it is necessary to keep track of which I/O subroutine is open. This information is kept in 6 bits of the PAC table. The contents of these 6 bits is transferred into the opcode field of the return address when an I/O subroutine is called, and is recovered from there when the BRS 41 is executed.

12.0 The Exec. Treatment of Files

Because of the possible conflicts which may arise when several users are simultaneously trying to access the same peripheral device, such devices cannot be handled directly by users at the level offered by BRS 1 -- which is available only to programs with executive status. At the user level, storage devices can only be referenced in an indirect manner, by writing or reading a "file".

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

The file types are:

1. Core Image - The information in this originates from specified segments of core memory.
2. Binary - The information has the form of an assembled, but unloaded program.
3. Symbolic - The information is of a form which can be readily listed on some printing device.

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

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

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

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

PAPER TAPE
TELETYPE

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

NOTHING

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

A commoner source for symbolic files is the output from some subsystems, notably the text editor, QED and publication preparer, AUTO-SEC.

Type 2, "binary" files may originate from paper tape, but, more commonly, arise as the output from the machine-language assembly subsystem, Arpas.

Until the actual process of output from the subsystem occurs, identification

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

12.1 File Naming

The names which the user is free to invent and assign to files are of two types:

1. unquoted names
2. scratch names

Scratch names differ from unquoted names in that they and the files associated with them are lost when the user leaves the system, using the LOGOUT command; they are otherwise treated identically.

An unquoted name is an arbitrary string of characters not beginning with ' or /. A scratch name is of the form / < arbitrary character string >.

As unquoted names we have -

ABC
PROGRAM 1
124

while as scratch names we have -

/ABC
/421/

Any unquoted or scratch file name may be quoted by surrounding it with single quote marks. Thus 'ABC' and '/001/' are quoted file names. The quoted name refers to exactly the same file as the unquoted one; it differs only in the way it is recognized by the Exec.

When reference is made to an unquoted or scratch file name, the Exec will anticipate the user and consider the name to be fully delivered as

soon as it has received sufficient characters to distinguish the name from all others currently defined by the user. This means that a new name can never be introduced in its unquoted form. A quoted name, on the other hand, is always accepted in its entirety from the user. The initial and terminal quotes are then removed and the name compared with the directory of names currently defined by the user. If it matches one of them, it is taken to refer to that file, just as though it had been presented in unquoted form. If it is new, however, it will normally give rise to an error message unless it appears in one of the following contexts:

a) In the DEFINE NAME command (cf. section 5.5)

b) As an output file name, in which case a new file with the specified name will be created to hold the output.

For example, let 'XYZ' be the name of an existing file and /123 be a new unattached file name. Then

COPY 'XYZ' TO '/123'.

has the effect of creating a new scratch file, called /123, having the same information content as XYZ. If /123 is, however, already attached to some existing file, then the information content of that file is replaced by that of XYZ.

In summary, it will be seen that the Exec's file name recognition apparatus works in two ways, depending essentially on whether the name is quoted or not. Quoted names must always be given in entirety; the Exec waits for the terminating quote before attempting to recognize the name. Unquoted names are anticipated; the Exec recognizes or rejects them as soon as it can, insisting that they match some name

already in the user's directory of file names. Note that the BEGINNER, NOVICE and EXPERT commands apply to file name recognition (see section 5.7 of Document R-22.

12.2 Accessing Other User's Files, Special Groups

The naming system described is adequate to reference all the files belonging to the current user, in whose name the Exec was entered. However, to refer to files belonging to another user, it is possible to augment the file name by that user's name together with, optionally, a special accessing code called the "Group" name.

To do this the basic file name must be prefixed by one of:

(< user name >)

or (< user name >, < group name >)

Thus for example:

(JONES)'FILE1'

or (JONES,GROUP1)'FILE1'

When such a string as the last is collected from a teletype by BRS 15, 16 or an Exec. command the characters ",GROUP1" are not echoed to the teletype so that the secrecy of the special group name is preserved. The access that any other user may have to each of Jones' files is in the hands of Jones himself. Jones may declare that a member of the public at large who tries to access his 'FILE1' using (JONES)'FILE1' has entire (read-write) access, read-only access, or no access at all. It is also open to Jones to define independently a greater degree of accessibility to a user who quotes the group name.

Special groups can be created by BRS 61 and the command SET MODES FOR FILE (5.5 of R-22) or deleted by BRS 62 and the same command.

BRS 61 - Define Special Group

Takes a string pointer in AB.

The string is an arbitrary string of characters and is taken to define a new special-group name. The BRS associates with it a number, n, in the range $1 \leq n \leq 15$, which it skip returns in A. A file may then be pl·ced in that special group by setting this number in the appropriate bits of the file mode word (see BRS 48).

A user may have up to 15 currently defined, distinct special groups; an attempt to define more results in a no skip return with A=0. An attempt to define an already existing special group name also results in a no skip return, but with the group number in A.

BRS 62 - Delete Special Group

Takes a special group number in A.

The associated special group name is deleted and the number made available for reassignment to a new name. All files belonging to the special group are released from it. If no name is attached to the number the BRS has no effect.

12.3 Pseudonyms

By means of the command USE NAME it is possible for a user to insert in his file directory a pseudonym, that is, a name which, instead of being a tag for a real file, is a tag for another name possibly including a user name and group name. If he later uses the pseudonym, the action taken is exactly the same as if he had typed the entire name for which the pseudonym stands.

12.4 Doing I/O to Files, File Numbers

The file name is an unwieldy and inconvenient handle for the I/O routines to use in transferring data. These routines instead reference the file by a compact, 1-word file number which is more closely related to the file's whereabouts. Thus system subroutines are provided to assign to a given file name some temporary file number.

The user may find it useful to remember that the system subroutines which perform information transfers to and from sequential files are the same for input as for output. The distinction is carried by the file number with which they are used -- whose character is in turn determined by whether it was returned by BRS 15 (input) or BRS 16 (output). Hence a program which was designed to output information can, without ill-effect, be delivered an input file number. The effect will be to lose the characters which the program would be trying to output, while taking in characters in their place -- these too, due to the nature of the program, will in general be ignored and lost.

Names are recognized and a file number provided, if required, by the system subroutines BRS 15, 16; they may be deleted by BRS 63. The preceding description of the manner in which file names are recognized largely assumes that they are being typed in on a teletype. They may, however, be presented to the BRS's as a ready-made string of characters in core. Entry parameters for the BRSs include a string pointer to a string in core together with an input-file number. The character string may be null or an initial part of a file name or an entire file name. In the first two cases sufficient characters are appended from the input file to ensure recognition or rejection of the name.

[A Remark on "Random" files on Tape

Random and sequential files may be stored and accessed with equal

facility on "random" storage devices, such as the drum and disk. On the other hand sequential devices, such as magnetic or paper tape, cannot be conveniently or efficiently accessed in the manner of random files and are restricted to holding only sequential files. However, the command 'COPY FILE' will allow a user to copy information from an existing random file, say on the drum, to a sequential medium, such as magnetic tape. The file created is, of necessity, sequential but has a special format which does not allow it a sensible interpretation as a sequential file but permits the original random format to be restored when it is copied back to a random device. Such a "random" file on a sequential medium will result in the return of the apparently paradoxical information, 1-0 in bits 0,1 of X when the file is opened by BRS 15, 16. Before accessing information in such a file the user should copy it (using the Exec. command or BRS 92) to a non-sequential medium.]

BRS 15 - Open named file for input:

Takes in A a control word

in B the address of a string pointer

in X a dual file number.

The function of this BRS is to recognize an existing file name, optionally, open the file for input and return a file number for use with subsequent data-input commands.

Designation of the File

The string addressed by B must be the complete or incomplete name of a predefined file. If the name is incomplete, characters will be appended from the input file whose number is given in the least significant 12 bits of X -- until sufficient characters are available to determine uniquely a file name

(or no such name). If the file name is unquoted so that prerecognition occurs, the "tail" of the name is echoed back to the output file whose number is given in the most significant 12 bits of X.

If B=0 on entry a null string is assumed and characters collected from the input file are not transmitted to the caller's memory. If bit 0 of B is set, the string delivered is considered null -- its position being defined by the first word of the string pointer. Unless B=0 on entry the completed or, in the case of non-recognition, partially completed file name will be transmitted to the caller's memory. If a pseudonym was delivered, it will be replaced by the string for which it stands.

Unless the file name was complete on entry (i.e. no characters need be taken from the input file), a terminating character must be delivered to confirm or abort the file name. Confirming characters are those with an internal code representation 0 to 16_8, also semicolon, tab, line feed and carriage return; the aborting character is ?. All other characters cause ? to be output and are otherwise ignored.

Action:

This is dependent on options which are specified by bits 1 and 2 of A on entry. These are:

bit 1, if set, suppresses opening the file (no file number is returned)

bit 2, if set, suppresses the need for a terminating character; when these bits are not set, the action is as follows:

If the name is recognized and a valid terminating character is received, the file is opened for input. There is a skip return with

in A, a file number

in B, the terminating character

in X, is a composite word comprising --

in bits 6 to 23, the file length
in bits 3 to 5, the file type
bit 0 is set if the file is random
bit 1 is set if the file is not stored on a sequential medium.

Error Conditions

All error conditions are followed by a no-skip return with an indicator in X; A and B are undisturbed.

$-5 \leq X \leq -1$ shows that the file could not be opened. The possible reasons correspond one-one with those associated with a no-skip return from BRS 1 with $-2 \leq A \leq 2$ (see pp. 9-1, 9-7).

X=1 This exit occurs if the name given is not a predefined name in the specified user's file directory.

X=2 indicates that the file name was aborted by delivering ? as a terminating character.

X=0 Any such error is accompanied by one of the following 'error messages' being sent to the command output file (normally the teletype).

```
?
ILLEGAL USE OF PSEUDONYM
-NOT PUBLIC
```

BRS 16 - Open named file for output

Takes in A a control word

in B the address of a string pointer

in X a dual file number.

This BRS is provided to read an existing or new file name and, optionally, open the file for output and return a file number for use with subsequent data-output commands.

Designation of the File

The file name is obtained from B and X in exactly the manner of BRS 15 (q.v.) except that if the name is enclosed between quotes and is not delivered in association with some other user's name, then it may be new.

Action

This is again dependent on the control word in A, on entry.

Bit 0, according as it is 0 or 1, specifies that the file to be created is sequential or random.

Bit 1 is normally zero, to indicate that the specified file should be opened and a file number returned in A. If the user does not wish to open the file this bit should be set.

Bit 2 if set suppresses the need for a terminating character. It also suppresses output of the message OLD FILE or NEW FILE, which is normally produced after identification of a quoted file name.

Bits 3 to 5 =t, indicate the file type.

The type of a new file is always set to be t.

The type of an old file is changed to t unless t=0, when the old file type is retained. An attempt to open the teletype as anything but a type 3 file is an error.

Bits 6 to 23 = S, significant only for tape files.

S is taken to be the number of words of information about to be written. If a new tape file is specified, a space of 3/2 S words is reserved after the current last file on tape. For an old tape file, S is compared with the amount of tape space currently reserved for the file. If it is greater, an error message - TOO SHORT is produced, followed by a no-skip exit; the file is not opened.

The normal return from the BRS is with a skip, the same parameters being returned in A, B and X as for BRS 15 viz.

in A a file number (if opened)

in B the terminating character (if delivered)

in X a composite word comprising the file length, type and logical structure (random or sequential) -- see BRS 15.

Error Conditions

All error conditions are followed by a no-skip return with an indicator in X; A and B are undisturbed.

$-5 \leq X \leq -1$ shows that the specified file could not be opened. The possible reasons correspond one-one with those associated with a no-skip return from BRS 1 with $-2 \leq A \leq 2$ (see pp. 9-1, 9-7).

X=0 This exit follows the printing of one of the following error messages on the command output file.

NOT PUBLIC
READ ONLY
WRONG TYPE
FILE TOO SHORT
FILE DIRECTORY FULL

X=1 if the file name is new and either unquoted or is delivered in association with the name of another user.

X=2 if the abort terminator (?) is delivered.

Notes:

1) Although new tape files for the ordinary user will be created on the standard user's tape, some users can specify the tape on which a new file is to be created. For such users a message

TAPE SYS. NO. =

is printed and a decimal number must here be delivered through the command-input medium.

2) If the file name is quoted and not built in, one of the messages OLD FILE or NEW FILE is sent to the command output medium. As described above, this message may be suppressed by setting bit 2 of A on entry.

3) An attempt to change the logical structure of an old file (from random to sequential or vice versa) will elicit a message to notify the user before the name terminator is delivered.

BRS 63 - Delete name from file directory

Takes in B a string pointer

in X a dual file number

The entry parameters are used to designate a name in the file directory in the manner of BRS 15. The name is removed from the directory subject to the following conditions:

A tape file or built-in file cannot be deleted in this way. The BRS will in this case allow the user to delete all its names except the last.

When a pseudonym is delivered to the BRS the pseudonym itself is lost.

When the last name of a scratch file is deleted, the file's contents is also lost.

A successful deletion is followed by a skip return.

A no-skip return indicates that the attempt to delete failed. The contents of X will indicate the reason for failure as follows:

X=-3,-2,-1 correspond to no-skip returns from BRS 1 with A=-2,-1,0 respectively. Such an exit results only from an attempt to delete a drum file.

X=0 indicates an attempt to delete the last name of a tape or built-in file.

X=1 if the name is not in the file directory.

12.5 Opening Scratch Files

Scratch files are all kept on the drum. They differ from ordinary files in that they disappear completely when the user who created them logs out. A fixed amount of drum space is available to each user for scratch files, which he may allocate as he sees fit. If ever he attempts to exceed the allocation a message will be given.

A scratch file may be created by BRS 16 or any of the commands which create a new file, by delivering to them a new scratch name (see 12.1). Alternatively, for a scratch file with a name of the form /ddd/ where d is any decimal digit, the elaborate string delivery and recognition procedure of BRS's 15, 16, 63 can be bypassed by using BRS's 18, 19, 65 respectively. Instead of a string pointer and dual file number, these three BRSs take, for file identification, an integer in X. The decimal equivalent of this number as a string of three digits enclosed between slashes is then used as a file name to refer to the file in the conventional way.

BRS 18

Takes in A a code word

in X an integer

This provides an alternative way of referencing and opening for input, scratch files whose names are decimal integers.

The number in X is transformed into its equivalent string of three decimal digits enclosed between slashes, 5 characters in all, (a number which exceeds 999 is taken to designate the string /999/). This string should be a predefined name in the caller's file directory. The subsequent action of this BRS is to open the file for input in exactly the manner of BRS 15, i.e. dependent on bits 1 and 2 of A; the return conditions are the same as for BRS 15.

BRS 19

Takes in A a code word

in X an integer

By means of this BRS a scratch file with a decimal-integer name can be opened for output. As for BRS 18, the number in X is first transformed to a string of three decimal digits enclosed between slashes. The name is then treated as a possibly new name for a scratch file, belonging to the caller, in exactly the manner of BRS 16. Bits 0 to 5 of A also have the same significance as for BRS 16.

BRS 65

Takes in X an integer

The integer is converted into a string of three decimal digits, as in BRS 18, 19. The action thereafter is exactly as for BRS 63, successful deletion being indicated by a skip return.

12.6 Format of the File Directory, some Implementation Details

File names, group names and pseudonyms are contained in a hash structure of the type described in the SPS Manual (Document R-17). The first two words of each hash table entry are the conventional string pointers to the file name. The third word (the string "value") is a pointer to a 4-word "description block". In these four words is held all the information necessary to characterize the name, whether it be the name of a drum file, tape file, special group, pseudonym, etc. Notice that several entries in the hash table may point to a single description block; the associated names are then synonyms for the same object, which can be referenced by any one of them.

The command DEFINE NAME creates a new name to point to an existing description block; conversely DELETE NAME detaches the name from its description block, the description block itself is lost only if this was the only name pointing to it.

Certain parameters associated with each file directory are listed on page 12A. The format of a single hash table entry with attached file description block is sketched on page 12B.

Executive commands and BRSs are available for interrogating and changing parts of the user's file directory. The commands FILE DIRECTORY and SET MODES FOR FILE are described in the manual for the TSS Executive (Document R-22). The corresponding BRSs are BRS 60, 48.

BRS 60 - Interrogate file Description Block

Takes in B the address of a string pointer

in X a dual file number

The entry data are used, in the manner of BRS 15, to determine a file. The first three words of the description block for that file (see p. 12B)

are skip-returned in A, B and X respectively.

BRS 48 - Set File Mode

Takes in A a file mode word

in B a string pointer address

in X a dual file number.

B and X are used, in the manner of BRS 15, to determine a file name. BRS 48 will then use the information in A to set or change the special group membership, type and accessibility of the specified file (which must belong to the caller).

All of these characteristics are determined by bits 1 to 4, and 6 to 16 of the third, "mode", word of the description block associated with the file (see p. 12B). BRS 48 directly replaces these bits by the corresponding bits of A after checking A for consistency and existence of the specified special group.

A successful mode change is denoted by a skip return, failure by a return without skipping.

FILE DIRECTORY DESCRIPTION

(A) PREFIX AND STORAGE ARRANGEMENTS

0	FLENGT	ZRO	File directory length
1	CFTADD	ZRO	Address of end of file directory
2			Unused
3	SGNUSD	ZRO	(Bits set to indicate special group numbers in use)
4			Unused
5	STRN	ZRO	System tape number
6	PTRN	ZRO	Private tape reel number
7	FIDLC	ZRO	Drum index block address for this file directory; user number in opcode bits
8	TAPPAM	ZRO	System tape parameters: number of first tape file (bits 0-11), number of tape files allowed (bits 12-23)
9	BSS	ZRO	Address of beginning of description block storage
10	HTL	ZRO	Beginning of hash table (BRS 5,6 table)
11	EHTL	ZRO	End of hash table
12		ZRO	BRS 5,6 link
13	FDSS	ZRO	Character address of beginning of string storage (WCH table)
14	EFDSS	ZRO	End of string storage
15		ZRO	Exit to garbage collector

The remaining parts of the file directory appear in the following order:

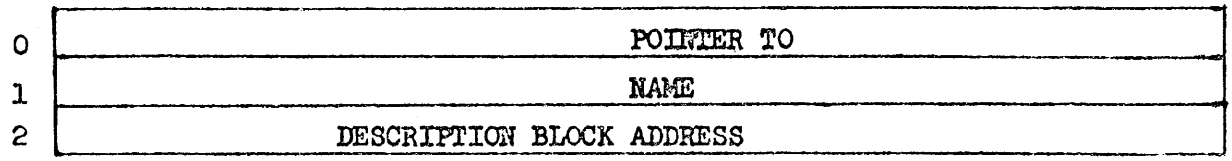
Hash table (HTL, EHTL)

String storage (EHTL, BSS)

File description block storage (BSS, CFTADD)

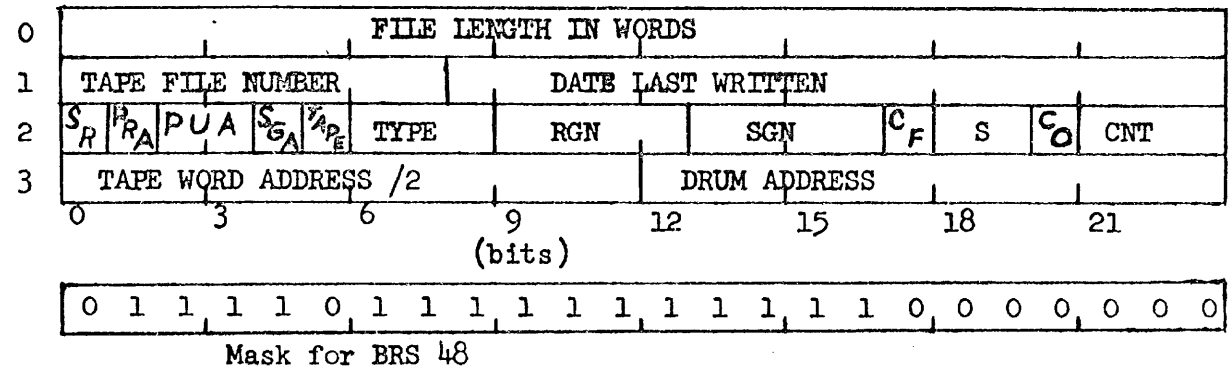
FILE DIRECTORY DESCRIPTION

(B) TYPICAL HASH TABLE ENTRY



The name may be a file name, special group name, read-in group name, or a pseudonym.

(C) TYPICAL DESCRIPTION BLOCK ENTRY



SR = sequential or random 1 = random

PRA = private accessibility 1 = read only

PUA = public accessibility 0 = denied to public
1 = public read only
2 = public read and write

SGA = special group accessibility 0 = read and write
1 = read only

RGN = read-in group number 0 = none

SGN = special group number 0 = none

CF = change flag

S = status 0 = file permanently on drum
1 = file on drum
2 = file on system tape
3 = file on private tape

TAPE 0 file came from system tape
1 file came from private tape

CO = called by owner

CNT = number of people who have called the file and not logged out.

POSSIBLE FORMATS FOR A FILE DIRECTORY ENTRY

(A) THREE-WORD HASH TABLE ENTRY

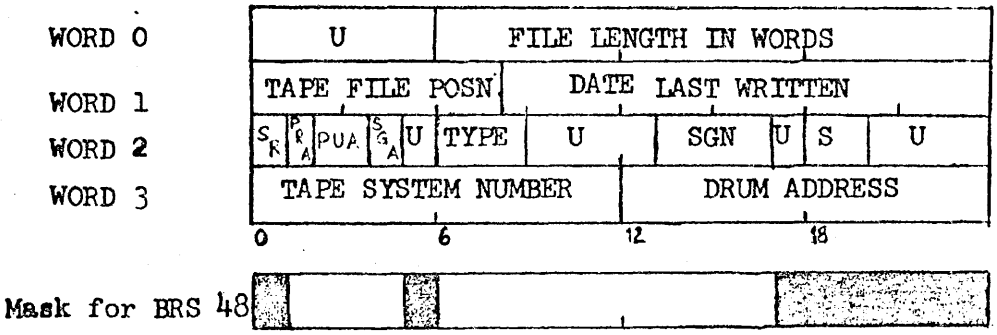
WORDS 0,1 [Bits 8 to 23] - String pointer to name in F.D. string storage

WORD 2 - Pointer to 4-word description block
or -1 (unattached name)

Unattached names may be created by commands which are not completed. If not deleted explicitly, they will be so automatically, when next encountered in a command or BRS.

(B) FOUR-WORD DESCRIPTION BLOCK

1. TAPE or PERMANENT FILES



SR = sequential or random 1 = random

PRA = private accessibility 1 = read only

PUA = public accessibility 0 = denied to public
1 = public read only
2 = public read and write

SGA = special group accessibility 0 = read and write
1 = read only

SGN = special group number 0 = none

S = status 0 = file permanently on drum
1 = file on drum
2 = file on system tape
3 = file on private tape

U = unused

2. SCRATCH FILES

WORD 1 = -1

WORD 0 = 0, WORDS 2,3 as for TAPE FILES

3. BUILT-IN FILES

WORD 3 = -2 ; WORD 2 = 0

a. Device WORD 0 = 0
WORD 1 [9 to 11] = no. of tape unit
WORD 1 [12 to 17] = device no. (0-P)
WORD 1 [18 to 23] = device no. (1-P)

b. Permanent file no. WORD 0 ≠ 0
WORD 1 [6 to 11] = file no. (0-P)
WORD 1 [18 to 23] = file no. (1-P)

4. SPECIAL GROUPS

WORD 2 = -1

WORD 0 = 0 WORD 1 = creation date WORD 3 [20 to 23] = group no.

5. PSEUDONYMS

WORD 3 = -1

WORDS 0,1 = string pointer to real string WORD 2 = 0

FORMAT FOR USER DIRECTORY ENTRY

(A) PREFIX AND STORAGE ARRANGEMENTS

0	BUHT	ZRO	Beginning of hash table (BRS 5,6 table)
1	EUHT	ZRO	End of hash table
2		ZRO	BRS 5,6 link
3	BUDSS	ZRO	Character address of beginning of string storage (WCH table)
4	EUDSS	ZRO	End of string storage
5		ZRO	Garbage collection option
6	BUDBT	ZRO	Address of beginning of description block table
7	LUDB	ZRO	Length of each user description block

The remainder of the directory appears in the following order:

Hash table (BUHT, EUHT)

String storage (EUHT, BUDBT)

User description blocks (BUDBT, end of directory)

(B) THREE-WORD HASH TABLE ENTRY

WORDS 0,1 [Bits 8 to 23] - String pointer to user name in U.D. string storage

WORD 2 [15 to 23] - User number

(C) SEVEN-WORD DESCRIPTION BLOCK ENTRY

0	HTA	ZRO	Address of hash table entry (backward pointer)
1	FDL	ZRO	Drum address of file directory
2	DA	ZRO	Maximum drum block allowance
3	AW	ZRO	Access word (determines user's status)
4	PW	ZRO	Password hash code
5	CTW	ZRO	Total computation time (in real-time-clock cycles)
6	LTW	ZRO	Total time logged in (in seconds)

13.0 Executive Commands Related to Files

Executive commands related to files are described in detail in Document R-22, the TSS Reference Manual.

14.0 Executive Commands

The commands which are accepted by the executive are described in detail in Document R-22, the TSS Reference Manual.

15.0 Subsystems

The time sharing system is organized into a central executive, which performs a minimum number of functions, and a considerable number of subsystems which perform more specialized functions. Each of these subsystems is called by giving its name to the executive as a command. The result of this operation is to bring the subsystem off the drum with the shared memory logic described in Section 16 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 line

@DDT.

would call the debugging system. The line

@CONTINUE DDT.

would re-enter DDT without initializing. Most of the subsystems are permanently present in the shared memory table which is discussed in Section 16, and may therefore 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 compiler
FOS:	The Fortran loader and operating system
ALC:	The Algol compiler
AOS:	The Algol loader and operating system
CAL:	Conversational algebraic language
LISP:	The list processing language

SNOBOL: A string processing language

TRAC: Another string processing language

IMP: An integrated machine language programming system

QAS: A question answering system

All subsystems and the executive itself will accept the command HELP, which will call in the question answering subsystem and the appropriate data base which will be able to answer questions about the system involved. The QAS subsystem is used to prepare data bases for commands.

16.0 Shared Memory Logic

The monitor mechanisms for controlling memory configuration and for putting memory blocks into the shared memory table have been described in Section 5. This section is concerned with the means by which a program can attach names to certain blocks of memory and obtain blocks which have been named by other users.

There is a global table called the named memory table (NMT). Its construction is very similar to that of the file directory but it points to chunks of shared memory rather than to files. The format of an NMT entry is indicated on the following page.

An NMT entry is referenced exactly like a file directory, i.e., with

NAME

or

(password) NAME

using

BRS 100

BRS 100 takes the same arguments as BRS 15 in B and X. It takes no arguments in A. If it is successful, it returns in AB the relabelling registers defined by the NMT entry and skips; otherwise it fails to skip. Whether the password is required is determined by the author of the entry when he inserts it into the table. The starting address associated with the entry is returned in X.

To make an entry in NMT, use the following instructions:

```
LDA     =NPTR
LDB     =PPTR
LDX     =RLW
BRS     99
```

NPTR is the address of the name of the entry. If there is another entry with the same name belonging to the same user, it is removed. If there is another entry with the same name belonging to another user, the BRS 99 fails and returns without skipping. PPTR is the address of a string which serves as the password for this entry. This string is encoded into 24 bits in some uniform fashion, and the same encoding is used to determine whether a password supplied with BRS 100 is correct. If the B register is 0, no password will be required for the entry. RLW is the address of the first of two relabelling registers. The word following the second relabelling register is taken as the starting address for the entry. The PMT entries indicated by these registers will be transferred to SMT and the appropriate specification will be constructed in NMT. Whenever this entry is extracted from NMT by a BRS 100, each byte of the relabelling registers returned will point to exactly the same memory block as the one addressed by the corresponding byte in the relabelling registers delivered to BRS 99. The actual bits in the byte may, of course, be different, since the pointing is done indirect through a given user's PMT.

An entry may be deleted from NMT with BRS 101, which takes the same arguments as BRS 100. Only the owner may delete an entry. All entries belonging to a given owner are deleted when he logs out. When an NMT entry is deleted, or when any PMT entry which points to SMT is deleted, a check is made to see whether there remain any entries in any user's PMT or in NMT which point to the SMT entry involved. If not, the entry is deleted from SMT.

Subsystems are permanent entries in NMT. Any user may therefore call on a subsystem simply by executing BRS 99 with a subsystem name. The result

will be, of course, to deliver the subsystem relabelling to him. He may then do whatever he wants with it. Since subsystems are always read only, there is no way for him to damage the subsystem.

Another kind of shared memory is provided in connection with the input-output machinery. It is possible for a user to create a subsystem file, a file with type 5. Such a file is in a sense equivalent to an NMT entry. It contains the contents of a number of blocks of memory together with the same information contained in an NMT specification. A subsystem file may also contain any number of starting addresses. Such a file may be created with BRS 102, which takes the arguments of BRS 16 and also takes in the following three words:

1) The address of the relabelling words
2) The address of a starting address table
3) The length of the starting address table

To read from such a file, execute BRS 103 with the arguments of BRS 15. The effect is to convert the contents of the file into an NMT entry and to set up the first starting address in the starting address table as the NMT starting address. The two words following the BRS 103 specify the location at which the starting address will be copied and the number of words on the table which can be accepted. The relabelling associated with the NMT entry is returned in A and B as it is by BRS 100.

This mechanism provides a way for a user to create a file which he and other users can regard as a subsystem. Such a pseudo-subsystem can be common and can have a number of starting addresses. In fact, it has all of the characteristics of a built-in subsystem except that it is not quite so convenient to get at. The executive GO TO command will, however, accept a subsystem file as well as a save file and will do the reasonable thing in the former case. The named memory logic is not implemented at the moment.

17.0 Miscellaneous Executive Features

The executive 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.

To input 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.

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 AB and the resulting string is returned. The characters

appended have the form:

mm/dd/yy hhmm:ss

Hours are counted from 0 to 24.

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 30.10.70, 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
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 SYSPOPs 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 are built into the system. The user may therefore think of 930 as a machine which has both in its hardware. The floating point system is described in its own manual, document 30.10.40. A brief summary of the available operations is included here.

LDP	Load pointer (two-word load)
STP	Store pointer (two-word store)
FAD	Floating add
FSB	Floating subtract
FMP	Floating multiply
FDV	Floating divide
BRS 21	FNA or floating negate
BRS 50	Fix
BRS 51	Float
SIC	String to internal conversion
ISC	Internal to string conversion
BRS 52	Formatted floating input
BRS 53	Formatted floating output

The floating point format is similar to the SDS standard: two words are used for each number and the exponent is housed in the bottom 9 bits of the second word. Conversions between this internal binary format and a string of decimal digits, decimal points and exponents can be carried out with ISC and SIC. These may be regarded as conversion rather than input-output operations. To perform the input-output and conversion simultaneously BRS's 52 and 53 are available.

BRS TABLE

NAME	NUMBER	FUNCTION	
Asterisk indicates that the BRS is not implemented on 9/1/66			
MONOPN	1	Open file	9-1, 9-5
MONCLS	2	Close file	9-1
	3		
MPT	4	Release memory	5-3
SSCH	5	SPS search	18-1
SSIN	6	SPS insert	18-1
	7		
IOH	8	Close all files	9-1
FKST	9	Open fork	3-1
PPAN	10	Programmed panic	3-6
CIB	11	Clear input buffer	7-5
CET	12	Declare echo table	7-2
SKI	13	Skip if input buffer empty	7-5
DOB	14	Wait for output buffer empty	7-5
EXGIFN	15	Symbolic input file name	12-3
EXGOFN	16	Symbolic output file name	12-3
EXSIFN	17	Scratch input file	12-8
EXSOFN	18	Scratch output file	12-8
EXSFDL	19	Delete scratch file	12-8
RMDY	20	Read month, day, year	6-1
FNA	21	Floating negate	19-1
TPPAN	22	Send user back to exec	3-7
LNKS	23	Link TTY	7-7
LNKC	24	Unlink	7-7
MSGS	25	Set AM and AI bits	7-4
SKROUT	26	Skip if rubout waiting (exec)	3-6
ASTT	27	Attach TTY	7-4
RSTT	28	Release TTY	7-4

NAME	NUMBER	FUNCTION	
CØB	29	Clear output buffer	7-6
FKRD	30	Read fork	3-3
FKWT	31	Wait for fork	3-3
FKTM	32	Terminate fork	3-3
GETSTR	33	Collect string	18-1
ØUTMSG	34	Output message	18-1
ØUTSTR	35	Output string	18-1
ØUTNUM	36	Output number	17-1
GSLØØK	37	General string lookup	18-1
GETNUM	38	Read number	17-1
	39		
RDET	40	Read echo table	7-3
IØRET	41	Return from I/O subroutine	11-2
RREAL	42	Read clock	6-1
RDRL	43	Read relabeling	5-2
STRL	44	Set relabeling	5-2
SQØ	45	Dismiss on quantum overflow	2-3
NRØUT	46	Turn rubout off	3-6
SRØUT	47	Turn rubout on	3-6
SETFDC	48	Set fd control word	12-7
SRIR	49	Read interrupts armed	4-2
FFIX	50	Fix	19-1
FFLT	51	Float	19-1
FFI	52	Formatted floating input	19-1
FFØ	53	Formatted floating output	19-1
RRSB	54	Reserve resident block	5-5
MRSB	55	Make or release resident block	5-4
MBEX	56	Make block executive	5-5
CQØ	57	Guarantee 16ms computing	2-3

NAME	NUMBER	FUNCTION	
SSMF	58	Define secondary memory	10-2
CBRF	59	Clear block in random drum file	10-3
RFDC	60	Read file directory entry	12-6
SGDEF	61	Define special group	12-7
SGDEL	62	Delete special group	12-7
RGDEF	63	Define read-in group	12-2
RGDEL	64	Delete read-in group	12-7
	65		
DFDL	66	Delete drum file (contents only)	9-5
DFER	67	Delete specified file block (exec only)	9-5
EBSM	68	Enter block in SMT	5-3
GBSM	69	Get SMT block to PMT	5-3
	70		
SKXEC	71	Skip if executive	6-1
EXDMS	72	Exec dismissal	2-5
EPPAN	73	Economy panic	3-4
*FSWT	74	Wait	3-5
*FSFZ	75	Freeze	3-5
*FSMT	76	Melt	3-5
*FSTM	77	Destroy	3-6
SAIR	78	Arm interrupts	4-1
SIIR	79	Cause interrupt	4-1
MBRO	80	Make block RO	5-3
WREAL	81	Dismiss for specified time	6-1
SWSF	82	Switch sequential file to input or output	9-2
*HELPS	83	Call Help system	
*HELPM	84	Call Help maintenance	
SET8P	85	Set special teletype output	7-8

NAME	NUMBER	FUNCTION	
CLR8P	86	Clear special teletype output	7-8
DFRX	87	Read drum block	9-5
RTEX	88	Read execution time	6-1
FSCF	89	Cause freeze	3-6
DFR	90	Declare fork for rubout	3-5
EXRTIM	91	Time to string	17-1
ECCOPY	92	Copy	14
ECSAVE	93	Save	14
ECPLAC	94	Place	14
ECDUMP	95	Dump	14
ECRECV	96	Recover	14
ECFNDU	97	Find user	14
ECCSLT	98	Consult with user	14
*ENSM	99	Make NMT entry	16-1
*GNSM	100	Read NMT entry	16-1
*DNSM	101	Remove NMT entry	16-2
*ESSF	102	Make subsystem-type file	16-3
*GSSF	103	Get subsystem-type file	16-3
RSYB	104	Read 2K block	5-5
WSYB	105	Write 2K block	5-5
FKWA	106	Wait for any fork to terminate	3-3
FKRA	107	Read all fork statuses	3-3
FKTA	108	Terminate all forks	3-3
DMS	109	Dismiss	5-1
RDU	110	Read device and unit	9-6
BRSRET	111	Return from exec BRS (exec only)	6-2
TSOFF	112	Turn off teletype station (exec only)	7-4

NAME	NUMBER	FUNCTION	
DFCD	113	Count data in drum file	9-5
MTDI	114	Disconnect W-buffer (exec only)	9-7
	115		
RURL	116	Read user relabeling	5-3
SURL	117	Set use relabeling	5-3
TGET	118	Lock up tape unit	9-8
TREL	119	Unlock tape unit	9-8
APMTE	120	Assign PMT entry	5-3
DPMTE	121	Release specified PMT entry	5-3
MPAN	122	Simulate memory panic (exec only)	6-2

SYSTEM PROGRAMMED OPERATORS

BIO	176	Block input-output	9-3
TCO	175	Teletype character output	7-2
TCI	174	Teletype character input	7-2
BRS	173	Branch to system	Appendix 2
CTRL	172	Input-output control	9-4, 9-6
SBRR	171	System branch and return	
SBRM	170	System subroutine call	
STP	167	Store pointer	18-1,19-1
LDP	166	Load pointer	18-1,19-1
GCI	165	Get character and increment	18-1
WCH	164	Write character	18-1
SKSE	163	Skip on string equal	18-1
SKSG	162	Skip on string greater	18-1
CIO	161	Character input-output	9-2
WIO	160	Word input-output	9-3
WCI	157	Write character and increment	18-1
FAD	156	Floating add	19-1
FSB	155	Floating subtract	19-1
FMP	154	Floating multiply	19-1
FDV	153	Floating divide	19-1
EXS	152	Execute instruction in system mode	6-2
ØST	151	Output to specified teletype	7-4
IST	150	Input from specified teletype	7-4
SAS	147	Store in secondary memory	10-3