Paul McJones

REFERENCE MANUAL
TIME-SHARING SYSTEM

L. Peter Deutsch
Larry Durham
Butler W. Lampson
University of California, Berkeley

Document No. R-21
Revised October 22, 1968
Office of Secretary of Defense
Advanced Research Projects Agency
Washington, D. C. 20325

TABLE OF CONTENTS

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 in addition to give a number of implementation details. It also describes those features of the executive which can be invoked by a program.

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 executive, on the other hand, is concerned with the control of the directory of symbolic file names and backup storage for these files, and various miscellaneous matters. Other parts of the executive handle 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. These subjects are discussed in the executive reference manual, Document R-22.

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

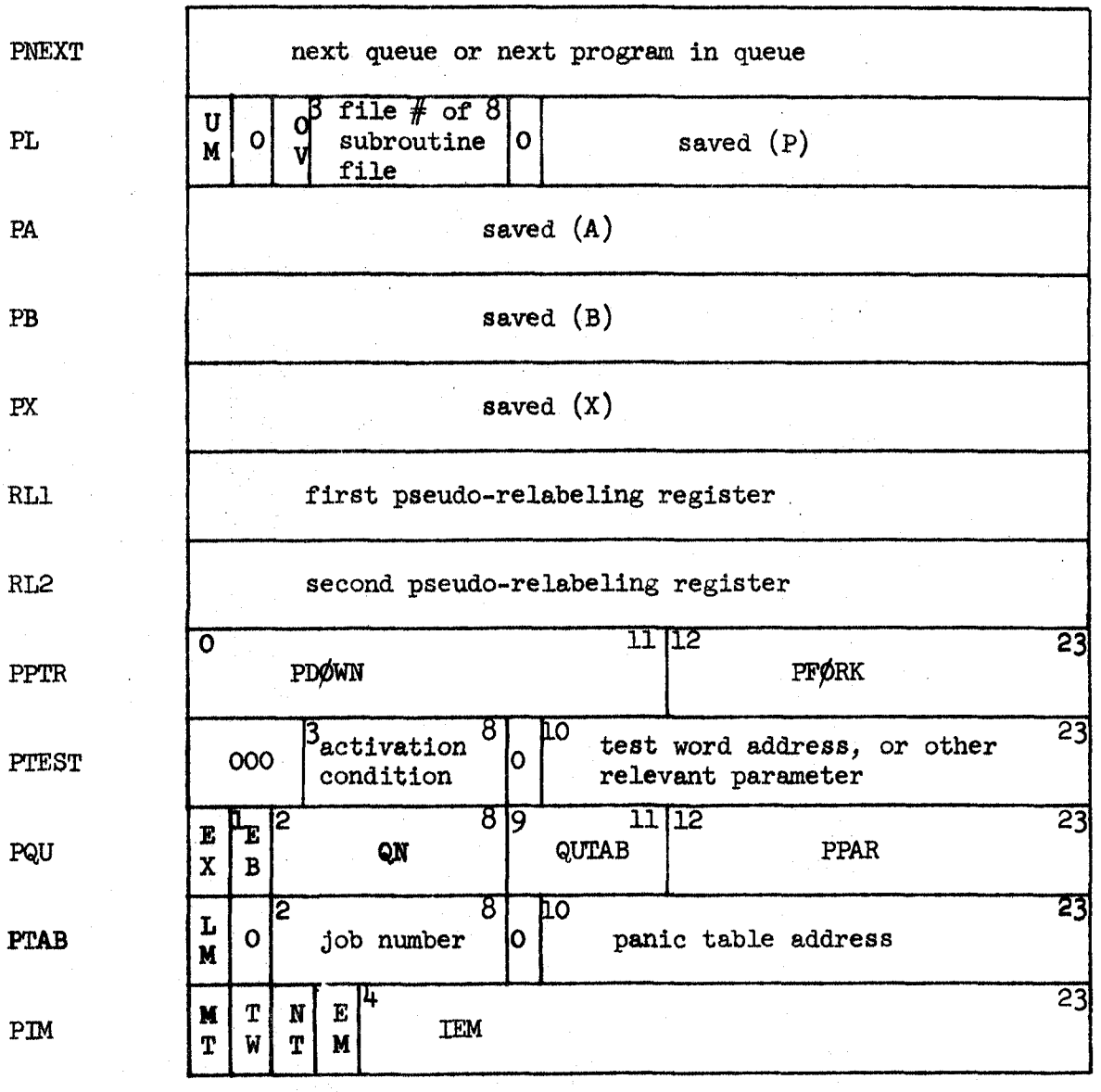
2.0 The Scheduler

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. A job table called QUR counts 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.

PAC TABLE



UM = user mode (1) or system

OV = overflow

PDØWN = PACT address of lower fork (if any)

PFØRK = 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

QN = saved queue number if on QOV

TW = waiting for termination

IEM = interrupt enabled mask

NT = non-terminable

LM = local memory

EM = destroy memory when fork is terminated

MT = add no memory

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 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. ACTR is set to -1 when a program is activated.

When a program is dismissed for I/O, TTIME is put into QUR. When the program is reactivated, TTIME is set from QUR. 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 always 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. (This word is usually something like TTYBRK for a user's teletype.) 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	executive BRS
5	BRS 109

11 Bit 1 of word is 1 (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.

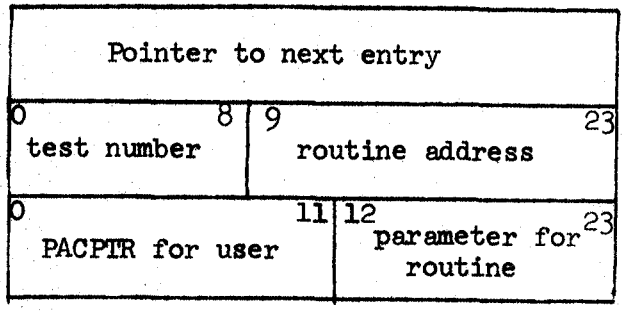
There is a permanent entry on the teletype queue for an entity called the phantom user. The activation condition for this entry is a type 4 condition which tests for two possibilities:

a) the cell PUCTR is non-zero

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

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



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

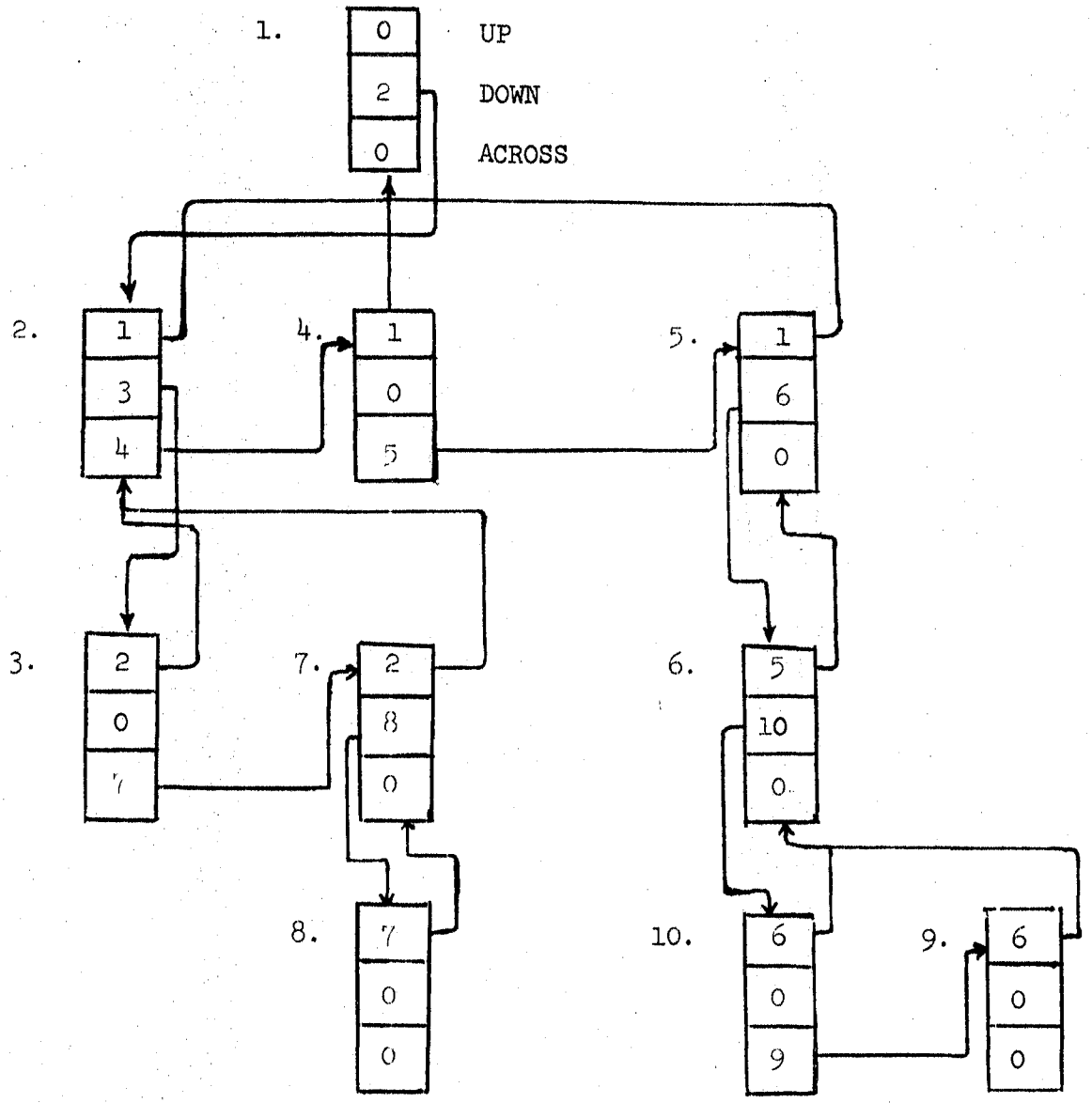
The panic table address must not be the same for two forks of the same program, or overlap a page boundary. If it is, BRS 9 is illegal. The first 7 bits of the A register have the following significance:

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.



Hierarchy of Processes

4. make fork local memory. New memory will be assigned to it independently of the controlling fork.
5. make fork ephemeral memory. Memory that it acquires will be released when the fork terminates.
6. set interrupt mask from seventh word of panic table.

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 tract 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. Document R-22) 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 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.

TSDA	drum address of TS block
TTNO	teletype associated with this job
ETTB	amount of CPU time used
DBA	drum blocks available
QUR	time left in long quantum

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.

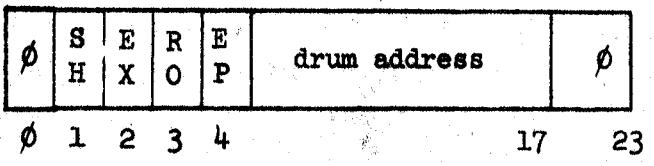
To read the interrupt mask into A, the program may execute BRS 49.

5.0 The Swapper and Memory Allocation

Pseudo-relabeling

The 940 hardware allows the user's address space to be fragmented into eight pages of 2048 words each. This means that the monitor must keep track of eight drum addresses for each process. This is done by means of eight six-bit pseudo-relabeling registers. Each of these registers is an index to a table which contains the drum address of the user's page.

This table is called the Private Memory Table (PMT) and is held in the job's TS block. Each of the 64 entries in PMT has the following format:



EX - Process must be executive to reference the page.
RO - Read only (attempt to store will generate a trap).
SH - Shared.
EP - Will be destroyed when not in any map.

During the startup for each user, the system copies the first NCMEM (currently 35) entries out of a resident table called the Shared Memory Table (SMT) into the new PMT. These entries describe memory that most processes will need, such as the monitor, the exec, and some of the subsystems. Thus, a program has a maximum of (64 - NCMEM) private pages.

When a program is run, his TS block is swapped and its pseudo-relabeling registers (in the PACT table) are used to read out the proper bytes from PMT and construct a list of drum pages that may need to be read from 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. In the course of optimizing the drum commands, the swapper may skip a sector. If this is the case, it searches through the memory tables and writes out a dirty page in that sector. The scheduler then simply hangs up until the swapping is complete. In the scan which sets up the drum read commands, the swapper collects from DHT 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.

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, any 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 and is implemented particularly for the Exec 'RECOVER FROM FILE' operation. If A is 0, BRS 120 assigns a 2K page and skips; this operation is not exec-only.

Shared Memory. The system maintains a table called the shared memory table (SMT) which describes all the memory which can be shared between jobs in the system. All the common subsystems occupy positions in SMT, and some part of SMT is copied into each job's PMT permanently. To run a subsystem, the exec must determine if the subsystem map is already in PMT (which it will be if all the bytes are below NCMEM) and, if not, arrange for the bytes to be put into PMT.

The exec makes an entry in SMT by executing BRS 68 with a byte number in A. The block addressed by the specified byte in the pseudo-relabeling registers is put into SMT and the pointer in SMT of this byte is returned. By putting an index in SMT in A and executing BRS 69, the SMT entry is copied 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 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

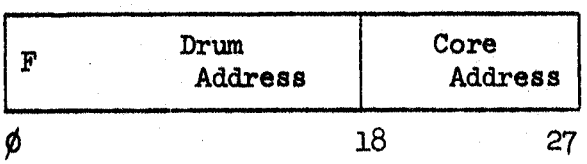
should be clear. Bit 0 of A will be reset if the block was formerly read-write or set if it was formerly read-only. If the program doing this is not an executive program, then the block must not be an executive block. Only executive programs may make a shared page read-write.

The drum is divided into 84 bands, each containing 16,000 words arranged in 8 blocks of 2K each. Up to 72 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 acquired, it is written as nearly as possible in adjacent blocks, so that it may be read in without undue drum latency time. Rotational positions are chosen by adding, mod 8, the user's job number to the PMT byte number of the new block.

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.

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.

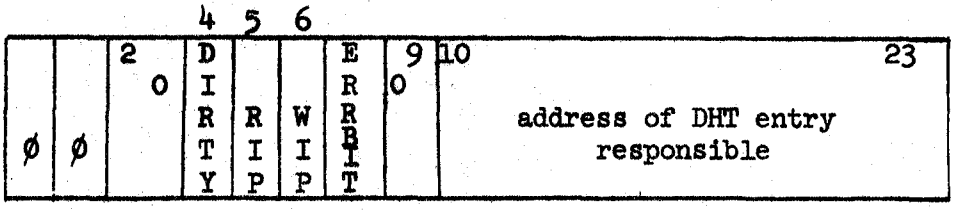
Real memory is housekept by means of several tables. The most important of these is a table (hash)-indexed by drum address which describes all those drum pages which are currently in core. The Drum Hash Table (DHT) has more entries than core pages. An entry has the following format:



F = Free entry

The core address field of DHT indexes 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. No block with non-negative RMC will be used by the swapper.

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



DIRTY - Page has been modified

RIP - drum read in progress
WIP - drum write in progress
ERRBIT - drum read error

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.

RES 140 scans through the real memory tables and for each page with a negative RMC takes one of two actions: If the page is not dirty, the PMT/SMT entry is marked as on drum and RMT is emptied. If the page is dirty, a write is started. This has the affect of forcing core and drum copies of most pages to correspond.

6.0 Miscellaneous Features

A user may dismiss his program for a specified length of real time by executing BRS 81 with the number of milliseconds for which he wishes to be dismissed in A. At the first available opportunity after this time has been exhausted, his 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 PA. The activation condition causes the current value of the clock 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 data and time, he can execute BRS 39. 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 some 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.

BRS 141 reads the panic table of the caller, and BRS 142 sets the state from a table specified by X.

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 shown in Figures 7A and 7B. No buffers are attached to the teletype unless input from or output to the teletype is taking place. As characters are output by the program, buffers are attached to the teletype. These buffers are released as soon as they are emptied by the teletype interface. On input buffers are attached to the teletype as characters are received from the teletype, and they are released as soon as the program empties them.

Input and output buffers are logically and physically independent, although they come out of the same buffer pool.

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 (TOS2 > -1) then the character to be echoed is put into the last byte of the buffer word which contains the input character. When the character is read from the buffer by the program, the echo, if any, will be generated. This mechanism, called deferred echoing, permits the user to type in while the teletype is outputting without having his input mixed with the teletype output.

There are four standard echo 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 echoes, but all characters except letters, digits, and space are break characters. Routine 2 again has the same echoes, 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 word listed as TTYTBL on page 7A.

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 or output SYSPOP) occurs. The top bit is left clear unless the activity is a rubout input. This cell is checked by the rubout processor to determine whether the rubout should reset the job to the exec. See p. 3-6.

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 pushed 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

```
LDA     =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.
BRS  138   waits until a process gets dismissed because
           the input 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 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 if the accept input bit is on.

To steal teletype output, the operation

```
STO     =teletype number
```

takes a character from the teletype's output buffer and returns it in A. STO is legal only if the accept input bit is on.

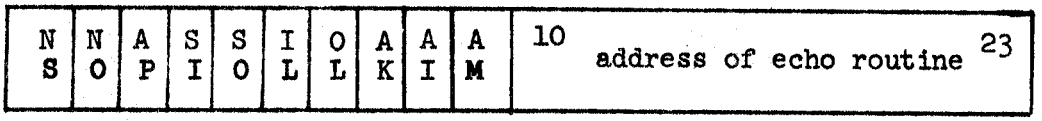
To disable output from a buffer to a teletype, execute BRS 139 with the teletype number in X. If bit Ø of A is 1, the NO bit will be set; otherwise, the NO bit will be cleared.

TELETYPE SYSTEM POINTERS

TTYOB Pointer to next available buffer in buffer pool

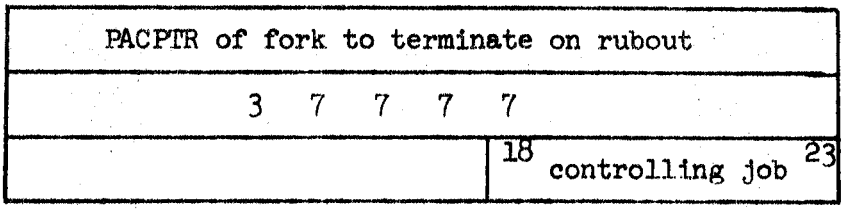
TTYOBC Count of available buffers in buffer pool

TTYTBL



TTYBRK Waiting for break character when -1

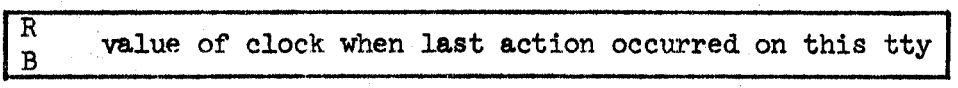
TTYASG



ROLCW Relative output link control word

RILCW Relative input link control word

TTYTIM



TTYDEV device (normally physical teletype) using this buffer.

NS = not linked or 8-level

AM = accept message

IL = input linked

OL = output linked

NO = don't output to TTY interface

AK = accept input links

AI = accept input

SI = 8-level input

SO = 8-level output

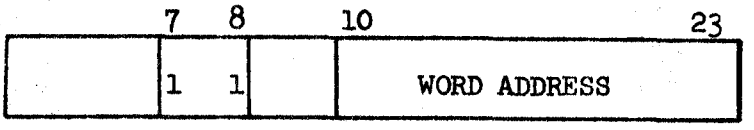
RB = last action was input of rubout

AP = accept output links

TELETYPE TABLE

TIS2 number of characters in input buffer

TIS4 next available space in input buffer (pointer)



bits 7 and 8:

0	1	byte 1
1	0	byte 2
1	1	byte 3

TIS5 next filled space in input buffer (pointer with same format as TIS4)

TIS6 deferred echo byte count. Input characters are echoed when the NO bit is set in TTYTBL and this count goes negative. It is decremented when a character is taken out of the input buffer.

TIBB1 word pointer to the oldest acquired input buffer: =0 no buffer attached

TIBB2 word pointer to the last acquired input buffer

TIBLC count of input buffer that can be acquired

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 same as TIS4)

TOS5 next available space in output buffer (pointer same as TIS4)

TOS6 < 0 = not terminated during output to links; $\geq$ 0 = next link that output has to be sent to.

TOBB1 word pointer to oldest acquired output buffer; = 0 no buffer attached

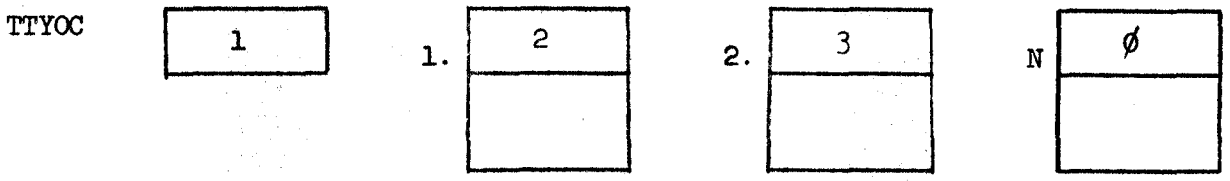
TOBB2 word pointer to last acquired output buffer

TOBLC count of output buffers that can be acquired

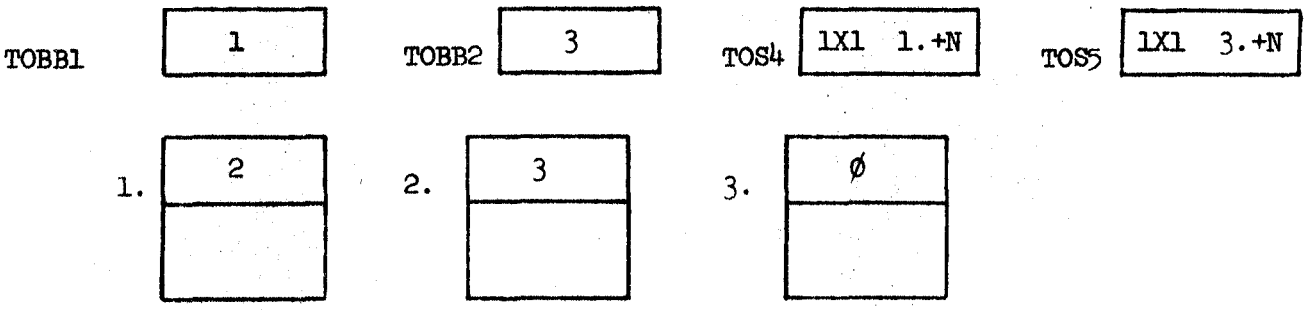
TTYPN contains physical teletype number associated with this buffer, or 4B7 if no physical teletype attached

TTYLN contains logical teletype buffer associated with this physical teletype or zero (ø) if no buffer is attached.

TELETYPE BUFFERS



BUFFER POOL POINTERS



1X1= byte count in word

N = word displacement in buffer.

PROGRAM OUTPUT BUFFER POINTERS

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 area. Only the first 141 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 of 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 three buffers. An additional block may be assigned with room for five more buffers if more than three 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 on input or end of the buffer on output or written in 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.

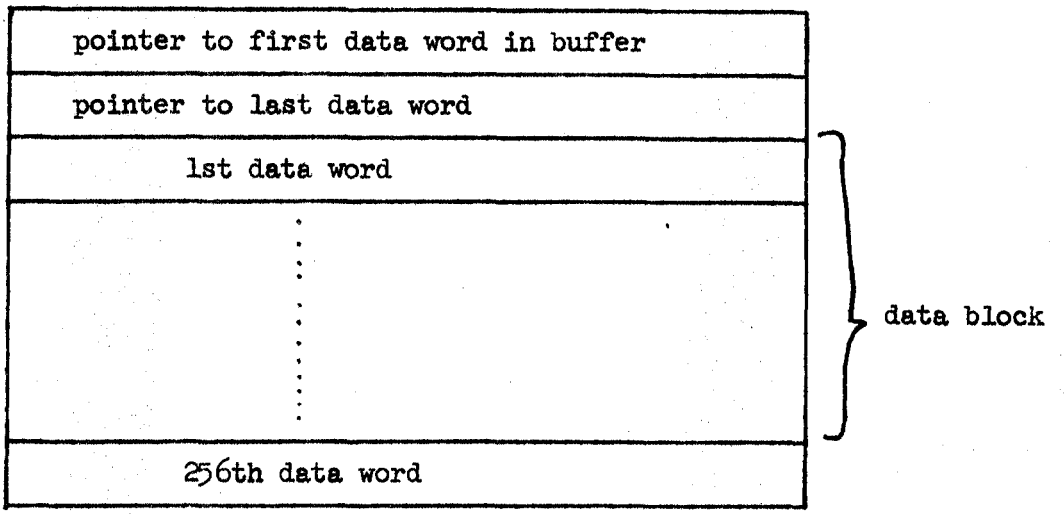
Page 8B shows the tables indexed by device number. Note the multiplicity of bits which specify the characteristics of the device. Some of these call for comment. A device may be common (shared by users, who must not access it simultaneously; e.g., tape or cards) or not common (e.g., drum); this characteristic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

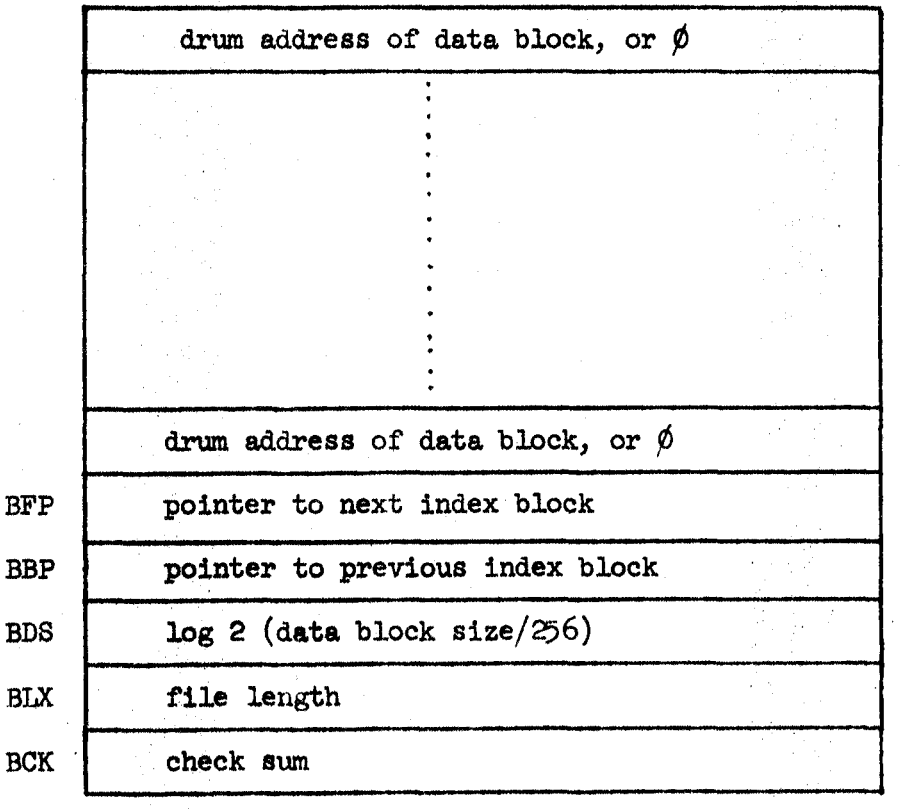
- the opening routine, which is responsible for the operation necessary to attach it to a file.
- the GPW routine, which performs character and word I/O.
- the BIO routine, which performs block I/O.

Minor parameters are:

- 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.

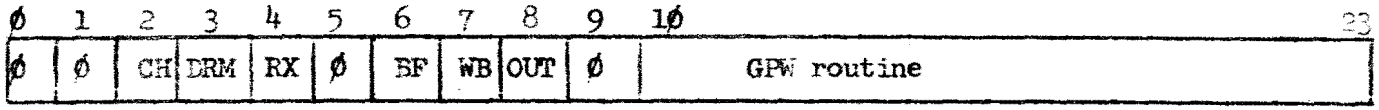


a) layout of a file buffer



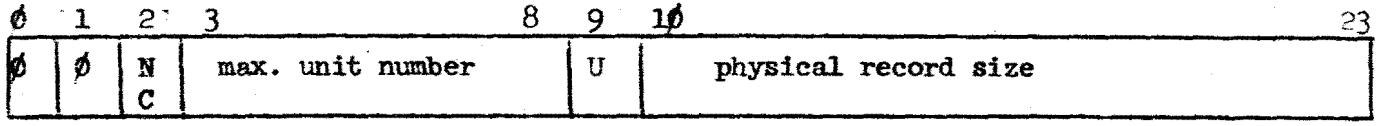
b) format of an index block

DEV word or character I/O routine



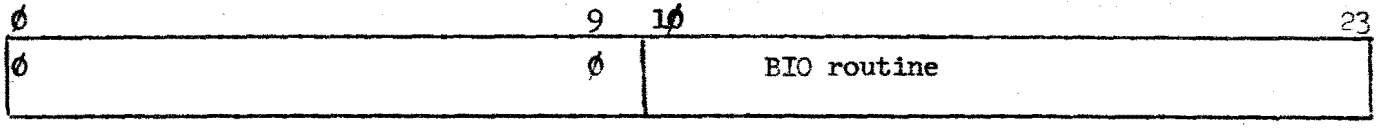
CH char oriented RX random access WB W buffer
DRM drum BF requires buffer OUT output

BUFS
buffer size

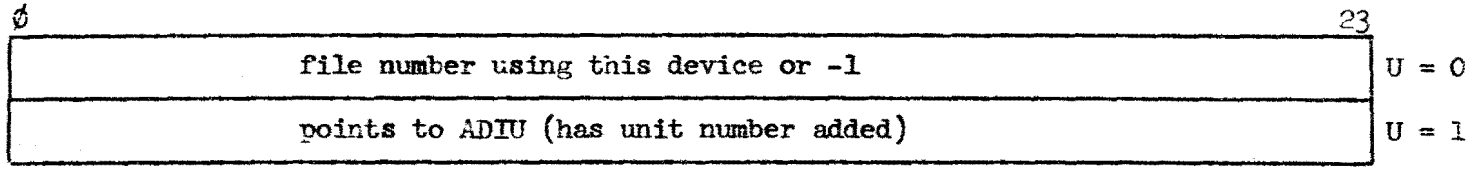


U check unit number NC not common (i.e. don't set DIU)

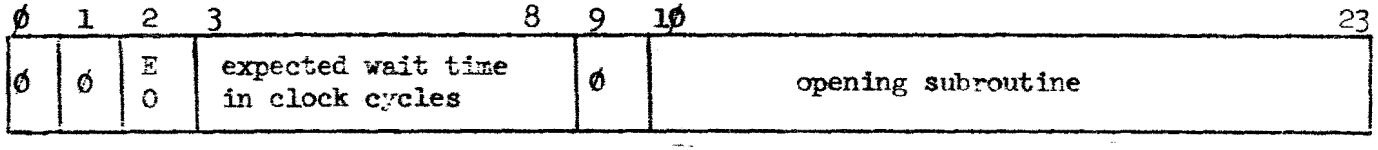
BDEV
block I/O routine



DIU
device in use



OPNDEV
opening routine



EO exec only allowed to open

DEVICE INDEXED TABLES

9.0 Sequential Files

9.1 Sequential Drum Files

1.94 includes a major revision to the drum file system. Basically, a file appears as an address space with an arbitrary upper bound called its "length." The maximum possible length is around four million words (22 bits). The file is internally paged into an arbitrary number of data blocks whose size may be any power of two larger than 256 words.

The following discussion will be of interest to those who need to make efficient use of large, randomly accessed files:

1. The data blocks are kept track of by the use of index blocks chained together. Each index block can describe about 140 data blocks. Random access will be slow if there are several index blocks to chain through. A large data block size will minimize this source of inefficiency.
2. Block transfers are implemented in the most efficient manner, with as much as possible of the data requested transferred directly into the user's memory. The slop on either end is buffered in 256-word blocks. Programs cognizant of the structure of their files can avoid all buffering. All word operations use the 256-word buffer.

The user's access to an open file is housekept by means of a position pointer to the file. This pointer may be moved explicitly by the user or implicitly by any of the I/O operations. The I/O operations always leave the pointer pointing at the word following the last transfer. CIO is a confusion factor.

When a file is opened, any of three types of access to that file may be given to the user: read, write, or position. Position access is intended to implement append-only files. You may not move the pointer or perform random operations unless you have position access to the file.

file opened as	access given
sequential input	r p
sequential output	w
random read-only	r p
random read-write	r w p

The mechanisms for setting the length, data block size, or for moving the pointer are described under BRS 143-144.

BRS 66 deletes the contents of an open drum file. In the case of a sequential output file, it sets its length to zero. You must have write access to the file to use BRS 66.

BRS 67 takes a file number in A and deletes all trace of the file. Use of this BRS is limited to the EXEC.

BRS 143, 144 have been implemented as general read and set status of a thingy. The calling sequence is:

A: table address or data (depends on BØ)
If a table address, A is incremented to point to one past the last word transferred.

X: thingy number

B: decodes as follows
bit Ø: Ø if A has data, 1 if A points to a table
bits 1-11: "type" of thingy
bits 12-23: "attribute"

"type" is 1 for a drum file, 2 for a job. No other thingies have been implemented as yet.

For drum files, the attribute field specifies the following:

1. (Position) One may read or set the sequential I/O pointer. The bottom 22 bits are the current word pointer, the top two the character offset. (ØØ means a word boundary.) The offset must be Ø if the pointer is set.
2. (Length) Reads or sets the length of a file.
3. (Sequential I/O mode) The sequential I/O operations (CIO, WIO,BIO) are interpreted as input if the sequential mode is Ø, output if the mode is 4B7.

4. (Capabilities) Opening a file may give read/write/position access to the file. These bits may be read in bits 21-23. If you try to set them, they are ANDed with the existing capabilities.
5. (Data Block Size) Returns n where 2^{n+8} is the number of words in a data block. May be set only if the file length is ∅.
6. (User words) Each file in the system has five arbitrary words associated with it. Anyone may read them but only the exec may set them.
7. ("structure") A file may have voids in it. If you are interested, you can find out where these are. BRS 143 returns the number of words from the current sequential pointer to the next transition. It also moves the pointer. If you are crossing a void part of the file with this operation, the sign bit of the number is turned on. Setting this state with BRS 144 is interpreted by releasing this many words beyond the pointer. This also moves the pointer.
8. (count data) Can only be read. Sets the pointer to the beginning of the file.
9. (copy the index block) Can only be read. Gives a copy of the entire index block.

For jobs (type = 2) BRS 143 and 144 interpret the attribute field thusly:

1. (files) Returns a bit word telling which files are open. This word may not be set.
2. (PMT) Reads the private part of PMT.

The Exec opens a sequential drum file 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 input if already open for output or for output if already open at all.

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 only to executive programs, user programs should use BRS 20 instead.

To close all his open files an executive program may execute BRS 8. The corresponding operation for normal user programs is BRS 17.

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. It is not possible to have a file open for both input and output at the same time: this may be circumvented by using random files.

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. Note that an end-of-record condition only occurs for type files and is of concern only to the Exec. 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 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 file at one-half the maximum drum transfer rate.

The flag bits (0 and 7) of the file number are set by the system whenever end of file is encountered and cleared on any input-output operation in which this condition does not occur. 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. Bit 6 of the file number may be set on an error condition. Whenever any flag bit is set as a result of an input-output operation in a fork, interrupt 4 will occur in that fork if it is armed.

The 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 Document R-22, or via BRS 63 (vide infra).

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 bits.

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 initial settings of the r, w, and p capabilities, and the sequential I/O mode flag, should be given in B.

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.

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
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 also accepts the following three character mnemonics instead of the device numbers. Either the name or the number goes in X for the call.

1	PTI	paper tape input
2	PTO	paper tape output (not available)
3	CDI	card input (not available)*
4	MTI	mag tape input*
5	MTO	mag tape output*
6	PDI	PDP5 input
7	PDO	PDP5 output
8	FSI	drum input*
9	FSO	drum output*
10	FIL	drum input and output*
11	LPO	line printer out (not available)*
12	MDI	direct mag tape input*
13	MDO	direct mag tape output*
14	CSI	controlling teletype input
15	CSO	controlling teletype output
16	TTI	specified teletype input
17	TTO	specified teletype output
18	NON	nothing
19	IOS	subroutine file
20	SNP	snooper counters (Berkeley only)

* requires executive status

BRS 1 is inverted by BRS 110, which takes a file number in A and returns the corresponding device name 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 only available to executive-type programs.

```
LDA     =1 (end of record)
CTRL    =file number
```

is available for physical sequential file 5 (magtape output). Several other controls are also available for maptape 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
8	write 15 inches blank tape

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.

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

Direct tape I/O package. A mechanism for accessing arbitrarily formatted mag tape is available. The appropriate operations are:

BRS 1	open
BRS 2	(or 17 or 20) close
BIO	block input/output
CTRL	control

BIO is used in the normal way, with a word count in A and core address in X. BIO will not give you more data than specified by A. In no case may the block requested cross a page boundary. On input, BIO will skip if the word count presented is exactly right; otherwise, it will not skip, and will leave the number of words actually transferred in A and the next core address in X. The flag bits (EOR and EOF) in the user's file number are set as with the normal BIO for tapes.

In addition to controls 3-8 for tape in the CTRL operation, CTRL 9 has been implemented to allow the user to set the mode for the tape. This operation takes a $\emptyset$ or 1 in B21 for setting the tape in odd or even parity. (TSS tapes use odd parity.) B22 and B23 contain the "frame count," a mysterious feature of the W-buffer. Use one less than the number of 6-bit characters per word to be shipped. On reading the characters are stored right-justified in memory. On writing they are taken out left-justified. The word count for transfers covers the numbers of words in core actually used. When the tape is opened, the mode is set to odd parity, four characters per word.

Snooper Counters. The Berkeley system has a collection of hardware counters which monitor external signals. These may be opened as a file with BRS 1. In addition, two operations are provided.

```
CTRL   fn
```

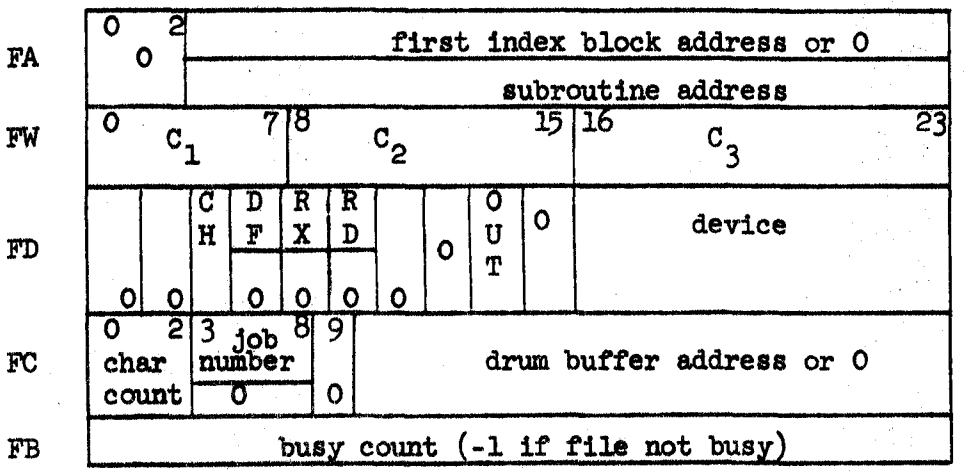
requires a 1 in A, and 1/32 of the number of machine cycles to be monitored in B.

```
BIO    fn
```

reads in the counters.

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

OUT = output

DF = drum file

Drum files only:

RX = random access

RD = read only

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 cannot be opened and need not be closed.

10.0 Random Drum Files

A random drum file is identical in physical structure on the drum to a sequential drum file. The only major difference is that 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 the data block size 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 file may have any number of index blocks.

A random file may be opened by using BRS 1 with a device number of 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, and CIO, WIO, and BIO may be used for input-output to random files. The sequential I/O mode (input or output) is controlled with BRS 143 and 144.

The following additional 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.

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, use BRS 144.

10.1 Direct Drum Access

An even more efficient method of accessing information on the drum is provided by an interface which allows the user to acquire 2K pages on the drum and read or write on them directly. This space is assigned from the swapping area on the drum and referred to directly by its drum address; a bit table private to the user is used for validity checking.

To acquire a 2K page, execute

```
BRS     126
```

with the desired angular position on the drum of the page to be assigned in the bottom bits of A. If no more space is available, BRS 126 returns without skipping. Otherwise, BRS 126 skips and returns, in A, the drum address of a 2K page as a word address (i.e., with the bottom 11 bits zero). A page may be released by putting this address in A and performing

```
BRS     127.
```

To release all pages acquired in this manner, execute

```
CLA
BRS     7.
```

This is done automatically by the RESET command in the executive, as well as by RECOVER and by a call for a new subsystem. It should be noted that DUMP does not preserve pages acquired by BRS 126.

To read or write on a page acquired with BRS 126, use

```
LDA     =core address
LDB     =drum address
LDX     =word count
{BRS    124   to read
 BRS    125   to write}
```

These BRS's preserve all the central registers and normally skip. A no-skip return indicates an uncorrectable transmission error.

The following restrictions are checked by the monitor and will result in an illegal instruction trap if violated:

1) The drum address must be a multiple of 256 (decimal) and lie within some page assigned to the user via BRS 126. (The latter restriction does not apply to executive programs.)
2) The transfer must not cross a 2K page boundary either in core or on the drum.
3) It is illegal to attempt to read into a read-only page with BRS 124 (this produces a memory trap if violated).

11.0 Subroutine Files

An 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 + subroutine address
BRS     1
```

This instruction skips or returns an error code, as for sequential files. 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 address field of the X register given to 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 six bits of the PAC table. The contents of these six bits is transferred into the opcode field of the return address when an I/O subroutine is called, and is recovered from there when the BRS 41 is executed.

The user should be warned that a subroutine file should not be used by a program in a different address space from the subroutine itself. In particular, subroutine files may not be given to the BRSs which involve acccess to named files (described in the next section).

12.0 File Naming System

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 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 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 four types, numbered 1 to 4. This type number is carried along with the information content and may be checked whenever the file is referenced.

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.

Symbolic information may come directly from paper tape or teletype. These devices may be referenced as type 3 files by using the name of the corresponding physical medium, viz. -

PAPER TAPE

TELETYPE

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

NOTHING

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

A commoner source for symbolic files is the output from some subsystems, notably the text editor, QED.

Type 2, binary files normally arise as the output from the machine-language assembler ARPAS.

Until the actual process of output from the subsystem occurs, identification of the information is handled by the 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. Permanent names
2. Scratch names

Scratch names differ from permanent 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.

A permanent name is an arbitrary string of characters not beginning with / or :. A scratch name is an arbitrary string of characters beginning with / or :. To those users who have drum file privileges, a / identifies a drum file, : a disk file.

As permanent names we have -

ABC

PROGRAM 1

124

while as scratch names we have -

/ABC

:421/

Any permanent 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. Control A (backspace) is legal on any name being typed to the file system unless command recognition is taking place.

When reference is made to an unquoted 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 (c.f. Doc. R-22, 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 the exec command

```
@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 R-22,Section 5.7).

12.2 Accessing Other Users' 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 or 16, 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 supplies the group name.

Special groups can be created by BRS 61 and the command SET MODES FOR FILE (R-22, Section 5.5) 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 placed 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 (most commonly teletype). 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 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.]

12.5 Opening Input Files

BRS 15 - Open named file for input:

Takes in A a control word
in B the address of a string pointer, or ∅
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:

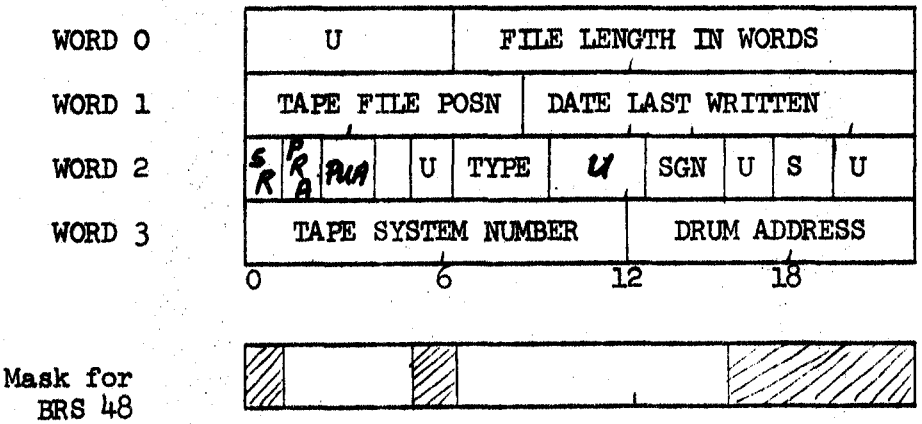
Bit 1, if set, suppresses opening the file (no file number is returned)

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 --

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. <u>Device</u>

WORD 0 = 0
WORD 1 [9 to 11] = no. of tape unit
WORD 1 [12 to 17] = device no (O/P)
WORD 1 [18 to 23] = device no. (I/P)

b. Permanent file no. WORD 0 ≠ 0

WORD 1 [6 to 11] = file no. (O/P)
WORD 1 [18 to 23] = file no. (I/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

Description Block Format

FILE DIRECTORY DESCRIPTION

(A) PREAMBLE AND STORAGE ARRANGEMENTS

0	FLTH	ZRO	File directory length
1	CFTA	ZRO	Address of compressed file input table (CFIT)
2	SGUS	ZRO	(Bits set to indicate special group numbers in use)
3	FDIX	SRO	Drum index block address for this file directory
4	FUNO	ZRO	User number
5	BSS	ZRO	Address of beginning of description block storage
6	HTL	ZRO	Beginning of hash table (BRS 5,6 table)
7	EHTL	ZRO	End of hash table
10	FDSS	ZRO	Character address of beginning of string storage (WCH table)
12	EFDSS	ZRO	End of string storage
13		ZRO	Garbage collection option

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, CFTA)

USER DIRECTORY DESCRIPTION

(A) PREAMBLE 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	BUDB	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) TYPICAL HASH TABLE ENTRY

0	STRING POINTER TO
1	USER NAME
2	USER NUMBER

(C) TYPICAL DESCRIPTION BLOCK ENTRY

0	HTA	ZRO	Address of hash table entry
1	FDL	ZRO	File directory drum address
2	DA	ZRO	Maximum drum block allowance
3	AW	ZRO	Access word
4	PW	ZRO	Password hash code
5	CTW	ZRO	CPU time word (60ths of a second)
6	LTW	ZRO	LOGIN time word (seconds)

ACCESS BITS ARE

0	WAIT	BRS 37 mode
1	IDIOT	
2	PRFFLG	permanent file flag
3	XMOK	exec mode OK
4	NTFFLG	new tape file flag
5	UTTFLG	new files to user tape
6	OPTFLG	operator flag

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
-NO GROUP NAME ATTACHED
-WRONG GROUP NAME

When the requested file exists on magnetic tape it is possible to receive about 20 different error messages, most of which are self explanatory. The position check message, "(PC: n)" means only that the tape has reset its position after becoming "lost" and should be of no concern.

12.6 Opening Output Files

BRS 16 - Open named file for output

Takes in A a control word
In B the address of a string pointer, or ∅
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 instructions.

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 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 (in addition to the possible messages given for BRS 15):

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.

12.7 Miscellaneous File Operations

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 built-in file cannot be deleted. The BRS will, however, 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 file is deleted, the file's contents are 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 built-in file.
X=1	if the name is not in the file directory.

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. 12A) are skip-returned in A, B and X respectively.

BRS 48 - Set file modes

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. 12A). 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.

12.8 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 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 is 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.

12.9 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 Section 14 of this manual. 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.

The format of a single hash table entry with attached file description block is sketched on page 12 A.

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 and 48.

12.10 Miscellaneous Services

BRS 92 - Copy file to file

By means of this BRS information can be copied from one file to another. The entry parameters consist of an input file number, an output file number and some bits to determine the nature of the files. If the information transfer is successful, there is a skip-return; if unsuccessful, a no-skip return, possibly preceded by a message.

On entry, the contents of A are taken to refer to the input file as follows:

bits 15 to 23	give the input file number
bits 3 to 5	give the file type
bit 1	is 0 for a sequential device (tape, teletype) or 1 for a random device (drum, disk)
bit 0	is 0 for a sequential file, 1 for a random file

The contents of B refer to the output files. Only bits 0, 1, and 15 to 23 are significant and have a similar interpretation to the corresponding bits of A. The necessary information for setting bits 0 and 1 correctly is returned by BRS's 15, 16 as bits ∅, 1 of X.

The copy will be successfully terminated when any of the following terminators is read from the input file.

1) Input file sequential
   a) An EOT (144_8) character, if and only if the input file is a teletype.
   b) An EOF (137_8) character for other type 3 files.
   c) 2 consecutive termwords (27657537_8) for all other sequential files.

2) Input file random

a) 1 termword if the file is stored on a sequential device.

b) Otherwise the copy terminates when the end of the index-block chain is reached.

The return after a successful copy is with a skip.

Errors

Errors may be

a) Calling BRS 92 with inadmissible parameters.

b) Unusual conditions detected during a data transfer.

Errors of type (a) may be any one of the following:

Attempt to copy a sequential file to a random file.

Attempt to copy a "random" file on tape to a sequential file.

Attempt to copy a non-symbolic file to teletype.

Attempt to copy directly from magnetic tape to a teletype or vice-versa.

They are all followed by a no-skip return.

Errors of type (b) are all signalled by a message, which is sent to the command output medium. The messages may be any of:

-END OF TAPE

UNTIMELY EOF IN INPUT

UNTIMELY EOR IN INPUT

RANDOM FILE TOO BIG, TRANSFER TERMINATED AT ADDRESS <n>

FAILED TO READ INDEX BLOCK

INPUT ERROR

OUTPUT ERROR

All but the last two are followed by a no-skip return. In the case of the last two the transfer continues from the point at which the transfer error was detected until the entire file is copied.

BRS 93 - Make a "save" file

Takes in A the address of a core-bounds list

in B the address of a 2-word map

in X a sequential output file number

This BRS may be used to preserve the contents of specified ranges of core (in the map given by B) to the output file given by X; note that this file must be sequential.

The core bounds list addressed by A, is a contiguous list of positive numbers terminated by any negative number. The first entry of the list is taken as a "starting address" - and is the address to which a transfer of control will be made when the data preserved by this BRS is read back into core by the GO TO command. Subsequent entries in the list are taken in pairs--each pair defining a range of memory from which information is to be saved. The two addresses in each pair may be given in either order. All addresses are taken with the map whose core address is given in B--if B is zero, it will be assumed that the user's current program memory is to be saved.

If the information is successfully transferred to the file, there is a skip return. Any failure in the data transfer results in an immediate no-skip return.

The formats of the core bounds list and the resultant save file are:

Format of Core Bounds List

Starting Address
m_1
n_1
m_2
n_2 ⋮ m_k
n_k
negative number

Format of Save File

$l_1 = \min(m_1, n_1)$
$u_1 = \max(m_1, n_1)$
Starting Address
⋮   data from   core range l_1 to u_1
l_2
u_2
data from core l_2 to u_2 ⋮   data from core l_k to u_k
term word
term word

BRS 94 - Restore a save file to core

Entry A,B = relabeling
X = file no. of sequential save (type 1) file

The save file, which should have the format described in BRS 93, is transferred to the memory given by the map in A,B. If the transfer is successful, there is a skip return with the starting address (see BRS 93) in A and the file number in X. An unsuccessful data transfer results in a no-skip return.

BRS 131/132 - (open tape for input/output) [privileged]

Given in: A = the desired tape position (0<A<256)
B = the user number of the file owner (BRS 132 only)
X = the tape system number or the tape unit number with bit 0 set.

Return No Skip: A = error flag (-2<A<18)
All errors result in a typed message

Skip: A = file number
B = user number of file owner
X = tape unit number

The tape can be open for input (BRS 131) without executivity being set. In fact, BRS 131 can be executed by users with operator privileges even if they do not have executive privileges. If the desired tape is not logically mounted, it can still be accessed by loading the unit number in X and setting X_0. This will cause the tape status vector to record the tape system and reel numbers. No new files can be created with BRS 132. A more complete description can be found in M-17.

13.0 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. If no digits are typed, the sign bit of B is set.

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 unless the sign bit of B is set, in which case it will be signed. If the magnitude of 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 23.

BRS 39 returns the date and time in AB as six 8-bit bytes giving year, month, day, hour, minute, second, respectively.

BRS 123 Read teletype and user number

Entry X = -1 or teletype number

Exit A = user number or 0.
B = job number or 0.
X = teletype number

BRS 97 Find user's teletype

Entry A = -1 or teletype number
X = user number

Exit No skip: user not entered; A,B,X undefined
Skip:
A = teletype number
X = user number

This BRS may be used to find on which teletypes a user is entered. A search is made to see if the user whose user number is given in X is entered on any teletype with a number higher than that given in A. If no such teletype is found, the BRS does not skip on return. Otherwise, there is a skip return with the next higher such teletype number in A.

BRS 104 Find user number from user name

Entry B = string pointer address
X = dual file number

Exit No skip: A,B,X undefined, illegal user name
Skip: A = X = user number

BRS 104 uses B and X to collect a user name in the same manner as BRS 15. If an illegal name is typed, there is a no-skip return from the BRS. The characters typed are appended to the string (if any) given on entry. Otherwise, there is a skip return with the required user number in both A and X.

BRS 105 Find user name from user number

Entry A,B = string pointer
X = user number

This BRS reverses the action of BRS 104. If the user number is valid, it appends to the string addressed by A,B the users name corresponding to the given number and returns with a skip.

If the user number is not valid the BRS does not skip and A,B,X are unchanged.

BRS 100 Read subsystem relabeling

Requires in B the address of a string pointer to the subsystem name
in X a dual file number or -1.

Returns skipping with the subsystem relabeling in A,B and the starting address in X.

14.0 String Processing System

The string processing system (SPS) consists of eight SYSPOPs and six BRSs. SPS strings are stored three 8-bit characters per word. Strings are addressed by two-word pointers. The first word contains the character address of the character before the first character of the string. The second word contains the character address of the last character of the string. The character address of a character is obtained by multiplying by 3 the address of the word containing it and adding 0, 1 or 2 depending on its position in the word. All string pointers contain character addresses. The character pointers used by GCI, GCD, WCI and WCH must have the first 8 bits cleared.

The following SYSPOPs are independent of the hash table mechanism which is described later. Any of them may be indexed or indirectly addressed (as may most other SYSPOPs).

14.1 String Pointer Load and Store Operations

LDP ADDR loads the A and B registers with the contents of ADDR and ADDR+1. X is undisturbed. STP ADDR stores the contents of the A and B registers in locations ADDR and ADDR+1. A, B, and X are undisturbed.

14.2 String Read and Write Operations

GCI ADDR tries to load the A register with the first character of the string addressed by the pointer pair in ADDR and ADDR+1. If the string is null or empty (i.e., if the contents of ADDR is greater than or equal to the contents of ADDR+1), then nothing is done and the next instruction in sequence is executed. If the string is not null, its first character is loaded into A right-justified and the contents of ADDR are incremented by 1, so that the string pointer now points to the string with the first character deleted. The top 16 bits of A are cleared, and the next instruction in sequence is skipped. Unless a copy of

the original pointer is saved, the contents of the string are effectively destroyed by GCI. For example, the code:

```
        GCI     STRING
        BRU     ØUT
        BRM     PRØCESS
        BRU     * - 3
ØUT ...
```

will call the subroutine PRØCESS with each character of the string addressed by STRING and go to ØUT after the last character is processed. To save the contents of STRING, the following commands could have been executed first:

```
        LDP     STRING
        STP     SAVE
        etc.
```

The X register is not disturbed by GCI. The B register is destroyed. Timing: 43 cycles. GCD is in every way similar to GCI except that the character is taken from the end of the specified string and the second string pointer is decremented. WCI ADDR writes the character in the last 8 bits of A on the end of the string addressed by ADDR. The contents of ADDR+1 are incremented by 1. A and X are not changed. B is destroyed.

To use a WCI in constructing a string, it is necessary to start with a null string. Suppose the string is to be put into a buffer called LINE and defined by

```
        LINE    BSS     20
```

The instructions

```
        LDA     =LINE
        MUL     =3
        LSH      23
        STA     PTR
        STA     PTR+1
```

will make PTR a pointer to a null string beginning (and ending) with the first character (not the Oth) in LINE. To start with the Oth character a SUB=1 could be inserted after the LSH. LINE can now be filled, say from the teletype by

```
        CIO     = 0
        WCI     PTR
        BRU     * - 2
```

WCD is the same except that it writes the character on the front of the string and decrements the first pointer.

WCH takes a character in A and a table address in the operand field. The table comprises three words:

```
ZRO     CLB
ZRO     CUB
 OP     ADDR
```

WCH tries to write a character into the area defined by the character addresses CLB, CUB. Provided that CUB>CLB, WCH will write the character in A into character position CLB+1 and increment CLB. If CLB≥CUB the character is not written and control is transferred to the third word of the table with A and X undisturbed and the address of the offending WCH in B. This can be an error trap or an exit to a routine which allocates more memory, by garbage collection or otherwise, for further WCH's.

14.3 String Compare Operations

SKSE ADDR skips if the string addressed by the pointer in AB is identical with the string addressed by ADDR. If the strings are of different lengths or have different contents, SKSE does not skip. This instruction is essentially identical to SKE, except that it acts on strings rather than numbers. A, B, X are not disturbed by SKSE.

SKSG ADDR skips if the contents of the string addressed by AB is greater than the contents of the string addressed by ADDR and ADDR+1. Comparison is made character by character, and terminates with the first unequal characters; the numerical, internal code representation of characters is used to determine inequality. If the strings are equal for the entire length of the shorter one, the longer one is indicated as the greater. A, B and X are not disturbed by SKSG.

14.4 String Input/Output

BRS 33 accepts a string pointer address in A, a file number in X and a "terminal character" in B. It collects characters from the file and appends them to the string until the terminal character is seen; this is not added to the string. It then returns the updated string pointer in AB; the string

pointer in core is also updated. If bit 0 of A is set on entry the string is taken as null with the second pointer equal to the first.

BRS 34 accepts a file number in X, a word address in A and a count in B. It outputs B consecutive characters starting with the first character of the specified word. If B=-1 on entry characters are output until / is encountered and the character $ is interpreted as carriage return, line feed.

BRS 35 accepts a file number in X and a string pointer in AB. It outputs the string to the file.

14.5 Hash Table Lookup Instructions

The hash table is a structure for minimizing the effort required to perform certain scan-and-compare operations when the operands are strings.

A hash table is a contiguous set of 3-word "augmented string pointers." The addresses of the first and last-plus-one locations of the hash table we shall denote by HT, EHT respectively. Each augmented string pointer occupies three consecutive locations of the hash table. Bits 8 to 23 of each of the first two locations hold the actual string pointer; bits 0 to 7 of these two words, as well as the entire third word (the so-called string "value") may hold arbitrary information. Note, however, that bits 0 to 7 of the string pointer words must be zero if used with GCI or WCI.

There are three BRSs to perform operations on a hash table: they are BRS 5, BRS 6, BRS 37. BRS 6 is used to introduce new strings into the table. BRS 5 and BRS 37 each perform a scan of the hash table for a string to match a given string.

Before using BRS 5 and BRS 6 to insert string pointers into an initially empty hash table, the hash table area must be cleared to zeros.

BRS 5 takes a string pointer in A, B, a table address in X. The table comprises 3 words:

```
ZRO     HT
ZRO     EHT
ZRO     0
```

The first two define the hash table bounds, the third is used for communication with BRS 6.

BRS 5 searches the hash table for a string to match the given one. If successful it returns in B the address of the hash table string pointer (the string "index")--and in A the string "value"; it skips on return. If the search is unsuccessful, BRS 5 returns with A, B unchanged and the address of the next free table entry in word 3 of the table (this will be -1 if the table is full). X is not disturbed.

BRS 6 takes a string pointer in A, B and a table address in X. The table is as for BRS 5. This operation inserts the string pointer into the hash table at the point determined by the last BRS 5 which failed (i.e., at the location specified by the third word of the table). If this word is -1, there is an illegal instruction trap. BRS 6 is intended for use only in inserting into the hash table a string pointer for which BRS 5 failed to find a match and should not be used except after a failing BRS 5. Furthermore, string pointers should not be placed in the hash table except with BRS 6 (otherwise the scanning algorithm used in BRS 5 will not work). Note that BRS 6 does not physically move the characters to which (AB) points.

On exit, BRS 6 returns in B the address of the first word of the new hash table entry and in A, the "value" word of the entry; X is not disturbed. To delete a hash table entry, put -1 (not 0) in the first word.

BRS 37 takes a dual file number in A, a string pointer address in B and, in X, the address of two words containing table bounds HT, EHT. A dual file number is a single word holding an output file number in the first 12 bits and an input file number in the second. If the output file number is zero, the user's teletype will be used. The table has the same form as a hash table, but the string pointers may be put into it in arbitrary order; it is not necessary to use BRS 5 and BRS 6.

The behavior of BRS 37 depends on the command recognition mode currently set in the exec (see R-22, Section 5.5). If the

mode is BEGINNER, the hash table is scanned for a string to match exactly the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended to it until either an exact match is obtained or a match becomes impossible. The exit is described below.

If the mode is NOVICE, the hash table is scanned for a string to match the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended until the string is long enough either to determine a unique hash table string, with a matching initial part, or for no match to be possible. In the former case, if the hash table string now contains three or less as-yet-unmatched characters, more characters are taken from input until an exact match is obtained or no match is possible; if the hash table string contains four or more as-yet-unmatched characters these unmatched characters are sent to the output file. If the input file is the teletype, BRS 37 waits until all the characters have been output, and the input file buffer is cleared before exit.

If the mode is EXPERT the hash table is scanned for a string to match the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended until the string is long enough either to determine a unique hash table string, with a matching initial part, or for no match to be possible. In the former case the remaining characters of the hash table string are sent to the output file.

Exits are as follows:

The no-match condition causes a no-skip exit with a string pointer in AB to the string so far collected; X is undisturbed. If a match is found there is a skip exit with the address of the matching table entry in A and the string value in B, X is undisturbed.

The following subroutine illustrates a use of the hash table facility. A string is input from the teletype and appended to WCH string storage until a carriage return is encountered;

it is assumed that string storage does not overflow in the process. The hash table is then searched for the string; if it is not already there it is inserted. In any case, an exit is made with the value of the string in A and the address of the string pointer in B. On entry X contains the address of the table for BRS 5, 6. CTL is the address of a table for WCH.

```
INPUT   ZRO     INPL
        LDA     CTL     remember beginning of string
        STA     TEMP
LOOP    TCI     CHAR
        SKE     =155B   terminator?
        BRU     WRITE
        LDA     TEMP    yes
        LDB     CTL
        BRS     5
        BRS     6
        SBRR    INPUT
WRITE   WCH     CTL
        BRU     LOOP
```

15.0 Floating Point Instructions

This section describes the floating point operations which are available in the system. SYSPOPs are provided to do floating addition, subtraction, multiplication and division and to convert under format control between the internal floating point representation and an external representation as a string of digits, decimal points and E (for exponent). BRS's exist which perform input-output and conversion automatically without involving the user with the external string representation. All these operations preserve the X register, except input routines which return the terminal character in X. Most destroy AB by leaving a result there.

15.1 Floating Point Representation

A floating point number is held internally as two 24-bit machine words. The format is

The number is always normalized: i.e., the most significant bit of the mantissa differs from the sign bit. All floating point operations expect normalized operands and produce normalized results. Both mantissa and exponent are treated as two complement numbers. The two words of the floating point number appear in the AB register or in memory in the order indicated.

A floating point number is represented externally as a string of characters. This string has the following form:

[±] [string of digits] [.[string of digits]] [E[±]string of digits]

The brackets indicate optional constituents. At least one digit must appear. Imbedded blanks are not allowed. The E indicates that the preceding number is to be multiplied by the power of 10 specified after the E. In general a floating point number being input may take any form which matches the template above. On output the form produced will be determined by the format specified.

15.2 Floating Point Arithmetic

There are four SYSPOPs to perform floating point arithmetic. Each of these takes one operand from AB and the other from M and M+1 where M is the effective address of the instruction. The result is left in AB in normalized form. If its magnitude is greater than 5.7896044E+76, the overflow indicator will be turned on and this value will be returned. The overflow indicator is not cleared by any of these instructions. In this respect the floating point POPs behave exactly like the integer machine instructions: a sequence of operations can be performed before the overflow bit is tested. The bit will be on if any operation caused an overflow. If the result is less than 0.803616E-77, it will be set to 0. No indication will be given.

The four operations are:

FAD	Floating add
FSB	Floating subtract
FMP	Floating multiply
FDV	Floating divide

An attempt to divide by 0 will produce an overflow.

Two SYSPOPs are provided for loading and storing double words. The words involved need not be floating point numbers, of course.

Load pointer: LDP M puts the contents of M and M+1 into AB.

Store pointer: STP M puts the contents of AB into M and M+1.

Three BRS's provide for unary operations involving floating point numbers.

Floating negate: BRS 21 returns in AB the negative of the floating point number supplied in AB.

Fix: BRS 50 converts into a double precision fixed point number. The integer part appears in A. The fraction part appears, left justified, in B. If the integer is too large, the most significant bits will be lost. The integer part is the next smaller integer. I.e., IP(-1.2)=-2.

Float. BRS 51 converts the integer in A to a normalized floating point number in AB.

15.3 Input/Output Formats and Conventions

Every I/O operation allows the user to specify a format in the X register. Format specifications are based on Fortran conventions, and are specified as follows:

Bit of X	Field Name	Significance
0-2	T	Format types: 1 integer. 2 E format with the number right justified within the specified field. 3 F format with the number right justified within the specified field. 4 E format with the number left justified within the specified field. 5 F format with the number left justified within the specified field.
3-8	D	Number of digits following the decimal point.
9-14	W	Total field width If the field width is 0, the I/O will be done in free format.
15	O	Overflow action.
16-23	N	I/O file number. 0 always refers to the teletype. ISC and SIC ignore this field.

Examples:

F6.3	30306000
E17.9	21121000
I5	10005000

On input only the W and N fields are significant. Note that exactly W characters will be read on input (unless W=0). Leading blanks and any trailing characters are ignored. Free format input will accept as many characters as it can in constructing a number which fits the external representation described above.

The input operations always return a floating point result. They skip unless overflow occurs, in which case they return the largest possible number and do not skip. Any number of digits may be provided: the first 11 digits after any leading zeros will be the ones used.

On output the W field should be made large enough to accommodate sign, decimal point, E, sign of exponent and exponent (if the format type requires any of these elements) as well as

the digits of the number itself. See the discussion of error conditions below. The sign is printed only if the number is negative.

There are two ways to output an integer: (1) integer format, or (2) F format with 0 in the D field. The former requires that the number in the A register be in integer form; the latter expects a floating point number in AB.

Free format output generates between 11 and 16 characters. If the magnitude of the number is between 1E0 and 1E9, ten digits are output with the decimal point properly placed. Otherwise, exponential format is used; in particular, the format E15.9. For example, the following numbers might be generated by the free format output.

5.379605400 -145362.5967 5.789604462E+76

15.4 Input/Output Operations

Two SYSPOPs are available to convert between the internal binary representation of a floating point number and its external decimal representation as a string of characters. The string is stored and addressed according to the standard system conventions.

String to internal conversion (SIC): Characters are read from the string pointer addressed by the POP under control of the format in X. The internal representation of the number is returned in AB. The first character after the number is returned in X if free format was specified.

Internal to string conversion (ISC): The number in AB is converted according to the format in X and the resulting external representation is written onto the end of the string addressed by the POP. The string pointer is updated.

Two BRS's are available to do input/output and conversion at the same time:

Floating input: BRS 52. Input takes place according to the format word in X. The operation of this BRS is identical to that of SIC.

Floating output: BRS 53. Output takes place according to the format word in X. The operation of this BRS is identical to that of ISC.

15.5 Output Error Conditions

There are four possible error conditions. When one of these conditions occurs the following action is taken:

a) Interrupt 5 is generated

b) An error code is put into location 200B

c) The indicated corrective action is taken and execution continues.

Code	Condition	Action
1	T field is not 1,2,3,4 or 5	Assume 2(E format)
2	Exponent field is too small	Discard characters from the left or take overflow action.
3	Integer exceeds 8388607 in magnitude	Use 8388607
4	Field for F-conversion too small	Discard characters from the left or take overflow action.

If either of error types 2 or 4 occurs and bit 0 in the format word is set, then the output field will be filled with *'s.

BRS TABLE

NAME	NUMBER	FUNCTION	
MONOPN	1	Open file	9-4, 9-8
MONCLS	2	Close file	9-4
	3		
MPT	4	Release memory	5-3
SSCH	5	SPS search	14-5
SSIN	6	SPS insert	14-5
DCLR	7	Release all space acquired via BRS 126	10-3
IOH	8	Close all files	9-4
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	12-6
EXGOFN	16	Symbolic output file name	12-8
UABORT	17	Close all files	9-2
EXSIFN	18	Scratch input file	12-12
EXSOFN	19	Scratch output file	12-12
CFILE	20	Close file	9-2
FNA	21	Floating negate	15-2
LNKS	23	Link TTY	7-6
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	7-3
RSTT	28	Release TTY	7-3
CØB	29	Clear output buffer	7-5

NAME	NUMBER	FUNCTION	
FKRD	30	Read fork	3-3
FKWT	31	Wait for fork	3-3
FKTM	32	Terminate fork	3-3
GETSTR	33	Collect string	14-3
ØUTMSG	34	Output message	14-4
ØUTSTR	35	Output string	14-4
ØUTNUM	36	Output number	13-1
GSLØØK	37	General string lookup	14-5
GETNUM	38	Read number	13-1
RMDY	39	Read date and time	6-1
RDET	40	Read echo table	7-2
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 (exec)	3-6
SRØUT	47	Turn rubout on (exec)	3-6
SETFDC	48	Set fd control word	12-11
SRIR	49	Read interrupts armed	4-2
FFIX	50	Fix	15-2
FFLT	51	Float	15-2
FFI	52	Formatted floating input	15-4
FFØ	53	Formatted floating output	15-5
	54		
MRSB	55	Make or release resident block	
MBEX	56	Make block executive	5-4
CQØ	57	Guarantee 16ms computing	2-3
SSMF	58	Define secondary memory	10-2
	59	Read PMT	
RFDC	60	Read file directory entry	12-11
SGDEF	61	Define special group	12-4
SGDEL	62	Delete special group	12-5

NAME	NUMBER	FUNCTION	
EXDEL	63	Delete named file	12-10
	64		
EXSFDL	65	Delete scratch file	12-13
DFDL	66	Delete drum file (contents only)	9-4
	67		
EBSM	68	Enter block in SMT (exec only)	5-3
GBSM	69	Get SMT block to PMT (exec only)	5-3
	70		
SKXEC	71	Skip if executive	6-1
EXDMS	72	Exec dismissal (exec only)	2-5
*EPPAN	73	Economy panic	3-4
	74		
	75		
	76		
	77		
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
	82	Sys go	
	83		
	84		
SET8P	85	Set special teletype output	7-7
CLR8P	86	Clear special teletype output	7-7
RTEX	88	Read execution time	6-1
	89		
DFR	90	Declare fork for rubout	3-5
EXRTIM	91	Time to string	13-1
ECCOPY	92	Copy	12-14
ECSAVE	93	Save	12-15
ECPLAC	94	Place	12-17
ECDUMP	95	Dump	
ECRECV	96	Recover	

NAME	NUMBER	FUNCTION	
ECFNDU	97	Find user	13-2
	98		
	99		
KBlØØ	100	Read subsystem relabeling	13-3
	101		
	102		
	103		
EXCNUN	104	Convert name to user number	13-2
EXCUNN	105	Convert user number to name	13-2
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	9-5
BRSRET	111	Return from exec BRS (exec only)	6-2
TSOFF	112	Turn off teletype station (exec only)	7-4
	113		
MTDI	114	Disconnect W-buffer (exec only)	9-6
CKDOPN	115	Skip if no drum files open	
RURL	116	Read user relabeling	5-3
SURL	117	Set user relabeling	5-3
TGET	118	Reserve tape unit (exec only)	9-6
TREL	119	Release tape unit (exec only)	9-6
APMTE	120	Assign PMT entry (exec only)	5-3
DPMTE	121	Release specified PMT entry	5-3
MPAN	122	Simulate memory panic (exec only)	6-2
RTUN	123	Read teletype and user number	13-2
RDRM	124	Read 2K block	10-3
WDRM	125	Write 2K block	10-3
DGET	126	Assign 2K page	10-3
DREL	127	Release 2K page	10-3
	128		

NAME	NUMBER	FUNCTION	
	129		
RDBA	130	Read drum assignment	
PTR	131	Position tape to read file	12-17
PTW	132	Position tape to write file (exec only)	12-17
	133		
SKUEX	134	Skip if caller executive	
	135		
	136		
	137		
WIR	138	Wait for input request	7-5
NTOS	139	Suppress or allow output	7-7
MFLSH	140	Force drum/core correspondence	5-5
RSCP	141	Read status of caller	6-2
SSCP	142	Set status of caller	6-2
RDSS	143	Read status	9-1
STST	144	Set status	9-1

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 1
CTRL	172	Input-output control	9-4, 9-6
SBRR	171	System branch and return	
SBRM	170	System subroutine call	
STP	167	Store pointer	14-1
LDP	166	Load pointer	14-1
GCI	165	Get character and increment	14-1
WCH	164	Write character	14-3
SKSE	163	Skip on string equal	14-3
SKSG	162	Skip on string greater	14-3
CIO	161	Character input-output	9-2
WIO	160	Word input-output	9-3
WCI	157	Write character and increment	14-2
FAD	156	Floating add	15-2
FSB	155	Floating subtract	15-2
FMP	154	Floating multiply	15-2
FDV	153	Floating divide	15-2
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
LAS	146	Load from secondary memory	10-3
DWØ	145	Drum word output	10-1
DWI	144	Drum word input	10-1
DBO	143	Drum block output	10-2
DBI	142	Drum block input	10-2
ISC	141	Internal to string conversion (floating output)	15-4
SIC	140	String to internal conversion (floating input)	15-4

GCD	137	Get character and decrement	14-2
STI	136	Simulate teletype input	7-7
WCD	135	Write character and decrement	14-2
STO	134	Steal teletype output	7-7
BPT	135	Breakpoint (BRS 10)	