UCID-30021, Rev. 1
Computer Documentation

LAWRENCE LIVERMORE LABORATORY

University of California/Livermore, California

PRELIMINARY USER'S MANUAL FOR THE STAR SYSTEM SOFTWARE

Joseph E. Requa
Harriet G. Coverston
Pierre Du Bois
Donald R. Emery
Douglas A. Kent
Paul E. Lund
Marilyn D. Richards
David F. Storch
George E. Vranesh

March 9, 1972

Prepared for U. S. Atomic Energy Commission under contract no. W-7405-Eng-48

PREFACE

This document represents the current status of the STAR software system which is being implemented at LLL for the Control Data STAR-100 Computer. As such it is subject to change without notice.

TABLE OF CONTENTS

TABLE OF CONTENTS

APPENDICIES

STAR SYSTEM PHILOSOPHY

The intent of the STAR system is to provide a means of fully utilizing the STAR computer while at the same time maintaining compatibility with the OCTOPUS network and extending the time sharing philosophy developed by and currently in use at the Laboratory.

The time sharing philosophy of the Lab differs from that outside the Lab in several important respects:

1. File Orientation. All user information in memory has corresponding storage space on rotating storage so that jobs may be entered into and removed from memory as system requirements dictate.

2. Language Independence. Communication between the system and user processes should be independent of software conventions of any language so that the user is free to utilize whatever software tools he sees fit.

3. No Terminal Language. Since it is impossible for a system of a finite size to provide all the terminal language capabilities desired by the broad class of users at the Laboratory, the system only supports an ID line to connect the user to the computer, an EXECUTE line to indicate execution of a code already existing in a file on rotating storage, a BYE line to log the user off, and a series of system status requests preceded by a Control-E character. Only the EXECUTE line causes user code to be executed.

4. Primitive Function Oriented System Calls. The system provides a series of calls which, when issued by a user program, will cause system functions to be performed for the user code. These calls provide for resource allocation, file manipulation, message handling, obtaining system information and performing input and output.

5. One Job Can Initialize and Run Another Job. This function allows the capability of implementing batch processors and message interface routines in a straightforward manner.

6. No Input/Output Limitations. For those devices or portions of devices to which the user has access, he should be able to do input and output in any manner of which the device is capable. For example, the system provides the means of creating a disk file containing absolute column binary card images. The user can read the file utilizing logical address within the file and hence is capable of processing any card deck in any format. Any sub-system such as COBOL or FORTRAN is then free to implement internal data structures required for the sub-system without system overhead and without forcing any other sub-system to be compatible with its data requirements.

With these unique features, the time sharing systems at the Lab can support a multitude of terminal languages, language processors and utilities with very little system overhead.

A set of files with global access called public files replace the normal terminal language. These files may contain routines to perform functions, through system calls, which would normally be performed by a terminal language. One or more may also contain command interpreters for a terminal language which then run other files to perform the interpreted functions. Any user who needs a specialized terminal language for his application is free to write his own. Batch processing can be implemented in a straightforward manner by simply interpreting messages obtained from a file rather than from a terminal. Once a batch processor is initiated, the user may log off the terminal and his jobs will be run to completion by the batch processor.

The end result of this approach is to give the user all of the advantages of a terminal oriented time sharing system both from the terminal and from a user code with a low system overhead. Also, since system requests are between a program and a program rather than between a man and a program, more information can be transferred in a more compact manner. Hence more powerful and complex system functions can be provided than are found in the normal time sharing system.

Naturally a number of public files are provided by the system programmers as a necessary adjunct to the system for the casual user, such as compilers, loaders, batch processors and utility routines, but they are not a part of the system, and are treated exactly the same as any user job.

Because of the virtual memory structure of STAR, and the file orientation of the system, the STAR system contains a powerful set of calls for file manipulation and mappings between files and virtual space of which existing 6600, 7600 system calls are a subset.

Terminal message handling has also been modified to give the user more flexibility. In general, the STAR system contains existing system functions as a subset with the balance of the functions provided as logical extensions to current modes of operation to make full use of STAR capabilities.

MEMORY CONCEPTS

The terms virtual memory and paging define concepts employed in the STAR hardware to facilitate multiprogramming or time sharing. The paging concept is generally present in virtual memory systems and has the effect of causing main memory, i.e. memory from which instructions may be executed, to appear larger than it really is and facilitates dynamic relocation of program segments. The STAR main memory is a core memory of 512 K (1K = 1024 words) 64-bit words. The main memory is considered to be divisable into eight blocks of 65536 words each. A block of this size is referred to as a large page. Each large page is divisable into 128 blocks of 512 words each. These are called small pages.

The virtual memory concept provides an extension of addressable space to make it appear to the programmer that he has all of main memory and all of auxiliary memory immediately available. Auxiliary memory in the STAR hardware system is provided in the form of CDC 817 disc storage units. Since the programmer does not really have all of auxiliary memory available but only apparently available, the concept is termed virtual memory. A hardware mechanism is provided to translate each program - generated virtual address to a physical core memory address according to a table whose content is controlled by the STAR software system. This table contains one entry for each page currently assigned by the software system. The table is of sufficient length to catalog the 1024 small pages possibly concurrently assigned to core memory. Each entry contains a physical page address, a user identifier - called a lock - and the virtual page address as it is known to the user. Utilizing this hardware, a page of the user's space may be loaded into any available physical core page and execution may proceed.

Since a user identifier is provided as part of a table entry, two or more users may have a page in memory with the same user virtual address but different physical addresses. Thus, the user identifier, or lock, can be recognized as a memory protection device. It allows more than one program having the same virtual address range to execute simultaneously in core memory with no address conflicts.

The CDC STAR-100 hardware system has a bit-addressable main memory. The address field allowed is 48-bits wide, allowing reference to $2^{34} - 1$ small pages or $2^{27} - 1$ large pages. This range is considerably greater than the totality of storage media provided with the hardware system. For this reason it is sometimes conceptually convenient to consider virtual memory as symbolically "named space" rather than as "virtual address space." During execution of a program, the virtual addresses it references are little more than symbolic pointers to some segment of the program which may be dynamically relocated in physical memory several times.

Until the advent of paged virtual memory hardware, the technique for handling problems too large for the core store had been to divide the program into segments and to provide a set of instructions for a loader as to when and where to replace program segments. Program segmentation using overlays had been the individual programmer's responsibility. Virtual memory and paging techniques permit the programmer to use memory as though it were entirely available to him. When a program reference a segment not in the main memory, the executive system intervenes. It takes care of locating the page containing the referenced address in the auxiliary memory and placing that page into main memory and makes the association between the virtual address referenced and the physical core address assigned to the segment. The latter is accomplished by completing the entry in the hardware address table already discussed. The lock portion of the entry is filled in from one of four such locks provided each executing program. These locks are in the form of numeric codes which are catenated to the address referenced by the program to form the virtual address which the hardware will interpret. These codes when supplied to the program are known as <u>keys</u> and are related to the program's descriptor block number.

Each program may have four such keys, one each for referencing 1) read/write space, 2) read/only space, 3) library space and 4) shared space or write/only space. The program then, has the key, the hardware address table has the lock and if the two are identical, the referenced virtual space is accessible.

The operating system provided by LLL for the STAR-100 will consider every program to be executable only in virtual space. Data files scheduled for use by such programs may be defined in either virtual space or physical space. Virtual program and virtual data files must follow certain format specifications. (see Page 3.4) Generally, each virtual disc file is prefaced with a "minus page" which is a 512 word segment containing information needed by the operating system to control execution of the program. A part of this minus page is called the "bound virtual map." It is the function of this map to relate virtual addresses to logical disc addresses. A disc file which is defined by the user as being part of his virtual space may have up to 40 virtual partitions. Each of these is represented by an entry in the "bound virtual map" which describes the virtual address associated with the beginning of that piece of virtual space, the disc sector address corresponding and the length of the particular piece of the file. A virtual code file must have all its map entries up to date prior to execution. This is not a requirement for virtual data files the code may wish to use. As a virtual data file is opened, the program may accept the definitions in the bound virtual map or may ignore them and map the file into virtual space as it sees fit. System calls are provided for these operations.

At the time a program is submitted to a loader, it must provide information regarding where in virtual address space it shall be considered to reside and whether its address space is contiguous and whether it is to be segmented into small or large pages. For each address discontinuity or access discontinuity, an entry is made in the bound virtual map by the loader relating the beginning virtual address of the space to a logical disc address and providing the continuous length of the defined space. It is through interpretation of the bound virtual map that the operating system will later understand which page is to be read from the disc file containing the loaded program into the core memory when an address interrupt occurs. The phrase "bound virtual map" can be seen to define virtual space bound absolutely to a fixed space on the disc and, hence, the virtual address is merely a symbolic reference to the disc (auxiliary memory).

Since we have a file-based system for the STAR computer, we need some disc region associated with each virtual region. This allows for complete program swap-out. We have incorporated the existing "drop file" concept into the STAR operating system. The drop file is a disc file created automatically by the system for each program as it is put into execution. The purpose of the drop file is to contain any modified pages of the program file, and modified pages of its read/only data files which have been defined to have temporary write access, and any free space which may have been attached. The drop file is considered a repository for parts of virtual space and so must have a minus page and map space. The area of a minus page reserved for such a map is known as the "drop file map."

Assume a program in execution wishes to access an area of virtual space not defined in its bound virtual map. The program may create or open a disc file and map it into the desired address space, which results in an entry being made in the bound virtual map, or if the program just wants some temporary work space, it can attach virtual space which is not defined as being associated with an existing disc file. The latter is known as "free space." The free space is mapped into the drop file map in order that these pages can have a place of residence if the operating system decides to swap the entire program to disc. The virtual address space newly defined by any of these means becomes an extension of the program's prior space and is accessible with no further effort on the part of the program. Any reference to any address in currently defined virtual space will cause system intervention to place the appropriate page into core memory.

With this paging and virtual memory scheme, a program need not ever perform any explicit I/O to or from disc storage. This construct is sometimes referred to as implicit I/O. Further, let us suppose the same executing program wishes to develop a disc file for output to some terminal device. Again the program merely creates a disc file indicating its virtual space correspondence. This causes yet another entry in the bound virtual map. The program procedes to write its output data into the defined virtual space with no explicit I/O request being required, i.e. the program fills an array. When the program is finished with the space, it may close the file causing all pages to be moved from core memory to the corresponding disc region. The act of closing the file also releases the virtual space associated with the file from the bound virtual map. That virtual space is then available for re-definition.

There is a situation where a data file might be used by a program and a decision is made by the program to modify some part of that data. If the input file is read/only, the virtual space corresponding is also considered to be read/only. In order to modify this virtual space, the program must declare the space to be "write temporary." The "write temporary" space is mapped into the drop file map as pages of it are actually written, i. e. data is stored into the space. A definition of "write temporary" virtual space is that it is read/only space which, if modified, will become part of the drop file and will exist in modified form only in the program's current execution space. When the job completes, the modified data will disappear with the drop file.

Having exposed virtual memory and paging concepts as they will be applied in the STAR-100 operating system, it should be pointed out that the system will provide a way around both concepts for non-executable files. A class of files known as "sequential data" files is provided. These files have no minus page and, hence, no virtual map. They are considered to be data files stored sequentially by continuous disc addresses. The intent of providing this class of files is to allow the programmer to do explicit I/O and manage his own buffer space in a manner somewhat analagous to current IODs on the 6600 and 7600 systems. Opening a sequential file causes an entry in the "bound sequential map" to be made. All I/O to and from the sequential file is handled by the program through its private buffers.

The STAR operating system allows for files types 1) sequential, 2) virtual data and 3) virtual code. Both virtual types must have a minus page prefixed which contains the bound virtual map of the file. The sequential file needs no minus page.

Sequential files which are being used as such must be read and written explicitly by the program. Virtual files are read and written by the operating system for the program either on a demand basis or on advice from the program before an access interrupt actually occurs. Virtual files which have write access will be updated automatically as the user modifies their virtual space. Virtual files which have read/only or execute/only access cannot be modified. Their corresponding virtual space can be modified through the mechanism provided by the "write temporary" definition which allows read/only space to be modified and become part of the drop file.

A final consideration should be that the drop file in finite. The system makes a guess at its size bu the program is free to destroy the system-created file and then creates its own drop file at the length it requires. This request must be made very early in the program before any of its pages have drifted to the drop file. Any attempt to attach free space which will result in over-subscription of drop file space will be signaled as an error and the attachment will be denied.

Several system calls exist within the structure of the STAR software system at LLL to allow the user quite a lot of freedom in defining and managing disc files and associated virtual address space. It may not be clear how these calls may be used to various ends, so this section will outline the operation of the file management calls.

CREATE

The purpose of the create call is to reserve space on a disc and to identify and define that space as specified by the user.

A. Sequential File

The specified IOC in the program minus page is filled in as required and an entry is made in the sequential map - also part of the minus page. This is sufficient information to allow the program to initiate explicit I/O to/from the disc file. Initiation implies "opening a window" onto the disc file prior to the actual read or write request. These functions will be described later. Further, the user may specify a base virtual address to be associated with the file such that it may be used in the virtual mode later.

B. Virtual File

The specified IOC is filled in as required and one entry is made in the virtual map area of the program minus page. The virtual address associated with the first word of the disc file is taken from the base virtual address field of the system call. The system assumes that the disc file represents contiguous virtual address space. If the user wishes to introduce discontinuities in the virtual address space represented by the disc file, he must first map out all or part of the initially defined space and then map in the desired virtual address space. There is a system call to accomplish this which will be discussed later. The virtual address specified in this call is examined for overlap of existing defined space, and an error is indicated if such overlap exists. The file will not have been created if such an error is extant.

All created files are given read and write access.

OPEN

The purpose of the open call is to connect a program to an already existing file so that I/O, either explicit or implicit, may be accomplished.

A. Sequential File

1. In sequential mode
The specified IOC in the program minus page is filled in as required and an entry is made in the sequential file map area of the minus page. This is sufficient to allow the program to initiate explicit I/O to/from the disc file. Before the actual I/O can begin, one must "open a window" on the file. This is done through a system call which will be discussed later.

2. In virtual mode
This option allows the program to use a file formatted in the sequential mode (no minus page), in the virtual mode. The specified IOC in the program minus page is

completed and an entry is made in the virtual map area of the minus page. No entry is made in the sequential map. Even though the file is sequential, explicit I/O may not be accomplished if the file is opened virtual. One may, however, open the file more than once, concurrently, in various modes. The sequential file opened in the virtual mode is considered to begin at the virtual address given in the working virtual address field of the system call. The virtual address space represented by the file is considered contiguous over the entire length of the file. After the open is complete the user may map out the just-defined space and map in the file in whatever manner he wishes. All the implicit I/O attributes which normally pertain to virtual files are applied to sequential files when being used in the virtual mode.

B. Virtual File

1. In virtual mode

The specified IOC in the program minus page is filled in and, optionally, the virtual map entry(s) is completed. The user may elect to use the map of the file as recorded with the file on disc, in which case the map entries are simply copied to the program map space. The user may, alternatively, choose to open the file and have the file map copied into the program's call buffer. In this case, only the IOC is filled in, no virtual map entries are made and the program does not have implicit access to the file. This type of open call is expected to be succeeded by a map-in call which will tell the system how to relate virtual space with the physical disc file. Any virtual map entries made are checked for overlap of existing virtual space and an error is signaled if overlap occurs. Note that the file remains open, i.e. the IOC remains, in the case of an address overlap error. This allows the program to reschedule its virtual space through the map-out and map-in calls. These will be discussed later.

2. In sequential mode
This call allows the program to have access to all of a virtual file including its minus page, but all I/O must be explicitly done through the program's I/O buffers. The specified IOC in the program minus page is filled in and one entry is made in the sequential map. The file is mapped beginning with word zero of the file minus page. Sufficient information, as a result of this call, is recorded to allow the program to initiate explicit I/O to/from the file. No implicit access is possible to any of the virtual space usually represented by the file when it's open in the sequential mode. Note that the file may be open more than once concurrently in differing modes.

MAP

The purpose of the map call is to define some virtual address region as part of the executing program's accessible space. This may be an association of virtual address space with an already open disc file or it may be an attachment of free space, that is, virtual address space not related to any existing file. Release of defined space is allowed.

A. Map-In

In order to implicitly access virtual space, the definition of that space must be in the virtual map area of the program minus page. The map-in call provides the means to do this. Up to forty discontinuous address regions may be cataloged. The user relates some virtual starting address and length with some disc address of an open file and indicates the access rights pertaining to that virtual region. The system makes the necessary entries in the virtual space map of the program. Overlaps are signaled as an error. If all forty entries of the map are full, an error is signaled and no further map-ins are allowed until some space is released via the map-out option. There is sufficient data available as a result of this call to allow the system to process page exceptions for the defined space. In the case of a free space attachment, the defined virtual space is given a part of the program drop file on which it may reside it a core-to-disc swap becomes necessary. Free space attachments, therefore, are not given an entry in the bound virtual map but are cataloged in the program drop file map. This map can hold up to 170 entries of up to 31 pages each. This allows for as many as 170 non-contiguous address spaces to be part of the drop file.

B. Map-Out

The map-out option allows for release of virtual address space. This may be a release of space associated with an open disc file or a release of free space. Virtual address space which has been mapped out is no longer accessible to the program. The corresponding disc region may be re-defined in other virtual space. The disc file itself is not closed, that is, the IOC is left intact. Mapping out free space causes the corresponding drop file map entries to be deleted and frees the disc space for re-assignment. If the disc file region represented by a virtual space has write access and is mapped out, all modified pages of that space will be written on that disc file before the map-out process is complete. If the parent file did not have write access, all modified data is lost through the map out process.

Note that the map call has no significance in dealing with sequential files.

CLOSE

The close call is provided to allow a program a means of severing its connection with a previously opened disc file. The disc file itself continues to exist.

A. Sequential File

The specified IOC and the corresponding sequential map entry(s) is erased from the program minus page. The program no longer has access to the file through that IOC.

B. Virtual File

The specified IOC and the corresponding virtual map entry(s) is erased from the program minus page. Any modified pages of a write access file are gathered from core and drum and are written back to the parent file being closed. The program no longer has access to the file or the virtual space representing it. The virtual address space associated with the closed file is no longer defined.

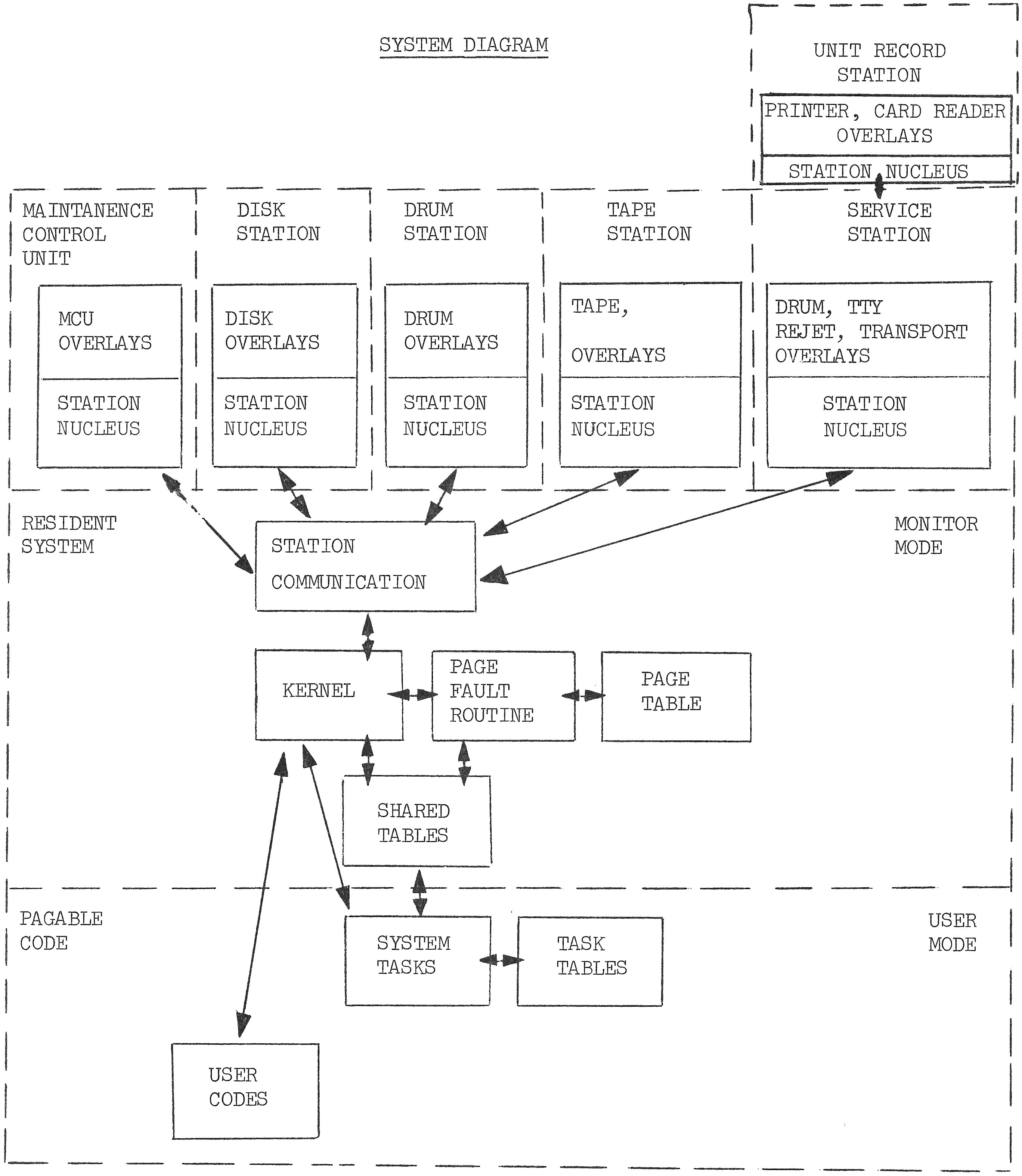
DESTROY

The prupose of the destroy call is to allow a program to terminate the existence of a disc file. The file need not be open to accomplish the destroy. If the file is open, the destroy is processed as usual and, additionally any pages of the file are erased from the core-drum system. The specified IOC, if any, and all related map entries are erased. The file and all its corresponding virtual address space cease to be defined.

SEQUENTIAL I/O

The sequential I/O call with its options allows the program to read or write an open sequential disc file. Prior to the actual I/O operation the program must "open a window" on the file. This means, simply, to associate some region of the program's defined virtual space with some region of the disc file. One might think of this call as temporarily allocating some virtual space to some area of a disc file. Having opened the window, the program may explicitly read or write the correspond disc region. The window virtual address may remain fixed and the disc region may be redefined so that the program may look out the window and "see" a different part of the file. The window may be closed by the program. This disassociates the virtual space buffer and the disc region it represented. No I/O may be requested at the virtual address of a closed window. Two windows may be open concurrently in a sequential file IOC.

SYSTEM DIAGRAM



STAR SOFTWARE STRUCTURE

The STAR operating system is divided into four parts:

1. Resident System. The resident system runs in Monitor mode, is always resident in core and references memory by absolute address.

2. Virtual System Tasks. The virtual system tasks run in user mode, are pagable and reference memory by virtual address. They may modify system tables.

3. Privileged User Tasks. The privileged user tasks have the same characteristics as virtual system tasks, except that they may not directly modify system tables. They perform tasks which require a long time compared to the virtual system tasks.

4. Peripheral System. The peripheral operating system runs in the peripheral I/O computers attached to STAR.

The resident system is divided into two parts, the KERNEL, which is responsible for time slicing and message handling and the PAGER, which is responsible for memory management and page swapping.

The time slicing portion of the KERNEL is controlled by the alternator loop. The alternator loop may be considered a circular table with each entry of the table containing a pointer to a minus page table entry, a descriptor block entry, and three sets of flag bits -- one set for KERNEL usage, one set for virtual system usage, and one set for shared usage. These bits define the status of each entry in the alternator loop. One entry in the alternator is unique in that it is shared by all virtual system tasks. Only one system task is allowed to run at a time to prevent two routines from modifying the same system table simultaneously. The system alternator slot has highest priority and is always run unless the slot is blocked for I/O, or PAGER action. If the slot is empty, the next task is selected from the job task queue and run. If the slot is blocked or the slot is empty and the system task queue is empty, then the rest of the slots are examined. This examination is controlled by two pointers, MAJOR and MINOR. Time slices are given in increments called tick times. MAJOR points to the alternator who currently is to run his tick time. If MAJOR can run, he is run, otherwise MINOR moves ahead of MAJOR to the next job which can run. Whenever MAJOR can run again, MINOR is reset to MAJOR. When MAJOR has run his tick time, MAJOR is advanced to the next slot which can run and the job which was previously MAJOR is given a new tick time. If, when MINOR is ahead of MAJOR, MINOR exhausts his tick time, he is marked as cycle-blocked. If MAJOR moves to a cycle-blocked job, that job is given a new tick time and MAJOR is advanced again. In this manner, for each circuit of the loop, each alternator gets one tick time. If MINOR moves all the way around the loop without finding a job which can run, MINOR is reset to MAJOR, all cycle-blocked jobs are given a new tick time, and the scan is tried again. If no job can run, the system monitors the station input queues for responses or requests until some action occurs which will allow a job to run. Station queues are periodically checked in the scan loop so responses and requests from stations can be processed in parallel with job executions.

User jobs, privileged user tasks and virtual system tasks communicate messages to the KERNEL by use of the exit force instruction. PAGER communicates messages by direct calls. The peripheral system communicates messages to the KERNEL by moving pointers in the station queueing structure without the use of external interrupts. The KERNEL communicates to the peripheral system by moving pointers then setting station channel flags. All communications between the various portions of the system are by messages. All of these messages either pass through the KERNEL, in which case it acts as a message switcher, or are processed directly by the KERNEL. The functions and formats of these messages make up a large portion of the balance of this document.

All access interrupts as well as certain messages dealing with core allocation are passed to the PAGER by the KERNEL. The PAGER dynamically allocates both large and small pages and performs all required implicit I/O necessary to free memory pages and obtain the pages causing access interrupts. The PAGER operates in a demand paging mode utilizing a least-recently-used page algorithm. If the page faulting rate becomes too high, causing an overload in page swapping, one or more jobs are disconnected from the alternator to alleviate the problem. If the number of pages on the paging drums becomes excessive, a virtual system task is brought up to alleviate the congestion. A degree of pre-demand paging is implemented by means of the advice message. This message can also be used to eliminate unneeded pages.

The virtual portion of the system controls the entry of users into the system, the entry of jobs into the system, the ordering of jobs by priority, and the entry of jobs in and removal of jobs from the alternator loop. In addition, it contains the system file management routines, `the explicit I/O routines and the teletype message handling routines. Virtual system task are placed in the job task queue by one of five occurrences:

1. A communication from the service station requires processing.
2. A user job requests a system service not provided by the resident system.
3. Bits are set in one or more alternator slots indicating virtual system action is required.
4. An entry in the periodic table shows that it is time to run a virtual system periodic.
5. A virtual system task requests the KERNEL to queue another virtual system task.

Virtual tasks are all of equal priority and are run on a first in/first out basis.

The privileged user tasks are run under special user numbers and are allowed to make either normal user calls or privileged system calls. They are not allowed to modify system tables except by means of calls. Privileged user tasks include:

1. TIMEDATA is a routine that runs at deadstart times and periodically thereafter. Its primary responsibility is to update system tables via Call #23.
2. TIMECARD runs periodically to move information from the accounting table to a disk file. If will also run aperiodically if the time card buffer fills before its normal time to run.
3. CARDREDR is brought up by the service station to move a card file from the service station drum to disk whenever the service station has a full card file.
4. PRINTOUT is brought up by the virtual system to move printer files from disk to the service station drum whenever the system detects a completed printer file.
5. HSPOUT is brought up to move files from disk to tape for off-line printing whenever one or a family of such files is available to the system.
6. DD80OUT is similar to HSPOUT, but processes files destined for off-line plotting on the DD80.
7. PUNCHOUT is similar to 5. and 6. but for punch card batch tape.

Communications between the STAR central system and input and output devices is done passing messages to and from stations through central memory. Each station has a table of messages which it can service and of messages it can send. Although there are a number of stations, a large portion of the peripheral software is common to all stations.

The basic peripheral system consists of two parts:

1. A resident basic system called the NUCLEUS common to all stations.
2. A set of overlays which perform tasks related to the individual stations.

Each station's software is stored on its microdrum.

The SCANNER program is the basic control mechanism of the NUCLEUS. The mechanism consists of a scanner program, scanner bits and the scantable which has one entry per scanner bit. The scanner bits are ordered by priority with the highest priority normally assigned to hardware status bits. These bits have an associated exclusive - or mask which allows a change-of-bit condition to be detected. A change in a scanner bit causes a call to the routine associated with it. If the routine is not in core, the overlay driver is entered automatically to read it in.

STAR PERIPHERAL SYSTEM

Each station has different resources and its own tasks to do. These specific tasks are all implemented in a common manner and are executed within the framework of a simple resident operating system. There is a large commonality of software between the stations. The main features of the structure are:

1. A small resident basic operating system called the Nucleus. The Nucleus provides an efficient priority interrupt mechanism and an ordered allocation of the processor to the routines which require it.

2. Modular software, written as small routines designed to run as overlays. One of these routines contains a set of library routines. The overlay implementation gives a large degree of implicit memory management.

3. Concentration of tasks into larger task processing routines. This provides, for example, on-line error handling and maintenance procedures common to all stations.

4. Grouping of station functions into different systems to minimize system tables. Any one system contains only those routines necessary to its job.

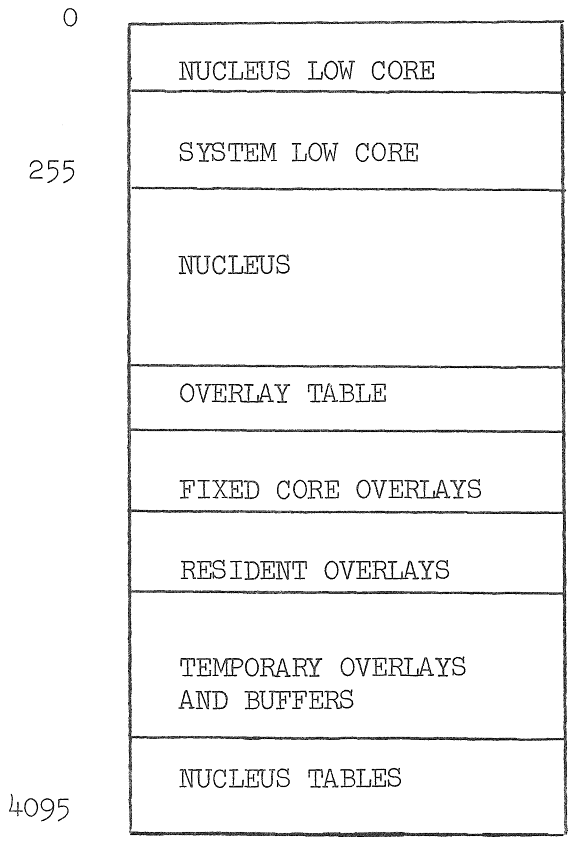
The Nucleus is a standard program used by each station. It consists of a set of simple diagnostic routines, a system dead start program, driver programs for the microdrum and keyboard/display, programs to manage the overlay mechanism and the main control and organizational program. This last is called the scanner program.

A station can contain up to nine different software systems on the microdrum. At dead start time, the Nucleus can be loaded and the proper system initiated. The initiation process consists of setting up the pointers to those global subroutines which are involved in the particular selected system and defining the conditions under which they are called as overlays.

Systems consist of any or all of six types of overlays.

1. Nucleus.
2. Low Core Overlay. The first 256 locations of core are directly addressable and are called low core. The first half of low core is assigned to the Nucleus, and the other half to the system. This second half is called the low core overlay for that system.
3. Fixed Core Overlays. These are overlays which are not relocatable and must be placed at fixed addresses in core. Up to four such overlays are allowed.
4. Resident Overlays. These overlays are resident in core.
5. Conditionally Resident Overlays. These are brought into core when needed and remain there until they are released.
6. Temporarily Resident Overlays. These are also brought into core when needed but are automatically released on exit.

The overlay space is allocated in contiguous segments of 128 bytes. Core is laid out in the following manner.



SCANNER MECHANISM

The scanner program provides a low-overhead mechanism for handling asynchronous, external events by initiating program execution in a predetermined priority. The mechanism consists of a scanner program, scanner bits, and the scan table which has one entry for each scanner bit. Each bit is related to a specific overlay routine; multiple bits may be assigned to a single overlay to provide multiple entry points. The scanner bits are segmented into 16-bit words which may actually be the 16 bits of an input channel. Such bits are for channel flags, drum busy, card ready input signal, etc. These bits have an associated exclusive OR mask which allows a change-of-bit condition to be sensed.

Scanner bits contained in core words not associated with an input channel are set by routines wishing to call other routines. Parameters are passed from routine to routine either via specified low core locations or Control Packages.

All bits in the scanner have an associated product mask used for maintenance and station configuration. The scanner program, which is entered by all routines on their exit, searches the scanner bits in priority by applying the appropriate masks. The scanner re-enters itself if no interrupt is detected; that is, no bits are changed.

A change in a scanner bit is taken to be a call to the routine associated with it, and this is entered via the start address given in the scan table. If the overlay is not resident, the overlay driver is entered automatically to read it in. The overlay driver arranges to be entered by placing its own address in the scan table entry when the overlay for that entry is not in core storage.

PAGING MECHANISM

Cases Handled

Handles following hardware interrupts:

1. Page not found - virtual bit address invisible package.
2. Write or read or execute violation.

Handles following software created functions:

3. Create N* small pages - a KERNEL request.
4. Get virtual address, V for alternator number, A (looks like case 1.) - a KERNEL request.
5. Advise for N* small pages, starting at virtual address V; or advise for 1 large page at virtual address V.

For case 2, a read or execute violation is always fatal. For write violations, a search is made of the user's bound virtual map. If the page originated from the source file or a write temporary file, or if the individual map entry has write access, then the key for the page will be changed to read/write and an entry will be made attaching that virtual space to the drop file.

A page initially receives a read-only key if the virtual space is defined in the source file; a write temporary file; a file whose IOC entry designates the file as read-only; or a file whose IOC entry designates the file as read/write but the individual map entry has read-only access.

Determine Page Definition

Cases 1, 4, and 5:

a) Check made to see if page is already on its way _into_ core.

*N $\leq$ 8, V and A are input.

b) Check made to see if page is on its way out of core. The processing of the fault is terminated if the first check is fruitful, is delayed if the second is fruitful.

c) Next, the user's map for the drop file is searched. If a hit, future key is tagged read/write and process turns to core allocator.

d) Next, user's bound virtual map is searched. If a hit, the future key is tagged read-only if the file is source, write temporary or IOC states read-only or map access bits state read-only. If the file is write only, the key is write/only. Page size is taken from map entry here as well as case c), and process goes to core allocator.

e) If c) and d) fail, then a create is assumed, and an attachment is entered for the drop file for that virtual space. Small page size is assumed here unless this was an advise with large page flag set. Fatal errors occur here if drop file already full or the drop file map is already full. The future key is tagged read/write.

Case 3: Processing goes directly to core allocator.

CORE ALLOCATION

SMALL PAGES

There exist two types of small pages; system locked and ordinary small pages. The system locked pages are the "minus" page and the "zero" page that come into being whenever a descriptor block is created. This is done for each execute line and each controllee initilization. Initially both of these pages/job will be locked down for the life of the job. Eventually the "zero" page will be unlocked whenever the job is not in the alternator loop.

The potentially long lived system locked pages will be allocated core within the large core blocks not assignable as large pages. Large blocks ∅ and _1_ (there are 8 large blocks in total ∅ through 7) will never be assigned as large pages and, hence, will always be within the "special region." If the maximum number of large pages allowable is X (X = ∅ to 6) then the "special region" will be large blocks ∅ to 7-X.

Ordinary small pages will be allocated core as follows. If a large page reserve is set, then the following steps will exclude that large block from consideration:

1. If sufficient free space is available, allocation will start within the large blocks outside of the "special region" defined above, and will proceed within the "special region" if necessary.
2. If sufficient free space is not available, the system will first look for any unlocked, non-reserved large pages belonging

to a disconnected job. If this search fails then the system will proceed as in Step 1 for whatever free space is available with the remainder of the pages (in the case of a multiple page advice) allocated as a result of writing to the paging drum the oldest, unlocked small pages in the page table.

NOTE: If the system disallows large pages altogether, then there exists no special region and all pages are "ordinary" pages and allocation starts at large block Ø.

LARGE PAGES

There exist large page limits for each job class* and for the machine as a whole. Initially the machine limit will be 5 in the daytime debug hours and 6 otherwise. I (interactive) and S (standby) classes will have zero limits. P (priority) class will be allowed the machine limit at all times and IB (interactive batch) and B (batch) classes may have the machine limit except during daytime debug hours. Perhaps IB class will be limited to 2, B class to 4, during daytime debug hours.

Procedure if the large page reserve is set:

1. If the requesting job is the reserve job and the reserved page is now unlocked, clear reserve and start necessary I/O. If the reserved page is not yet free, force user to fault again as a delaying tactic.

2. If the requesting job is not the reserve job and the requesting job has priority over the reserve job, disconnect the reserve job and reset the reserve for the requesting job and go to Step 1. If the requesting job has no priority over the reserve job, disconnect the requesting job. Priority determination is as follows:

* See Page 1.8.1 for definition of Job Classes (i.e. Wait Queues)

X = requesting job, Y = reserve job

a) if X is in P class, X has priority

b) if X is in IB class, Y has priority

c) if X is in B class and
   if Y is in P class, Y has priority
   if Y is in IB class, X has priority
   if Y is in B class, then whichever job started execution first has priority

Procedure if the large page reserve is not set:

1. If individual limit is not reached, go to 2. (NOTE: individual limit - class limit) If it is reached, swap one of the requesting job's unlocked pages. If all are locked, force user to fault again.

2. If class limit not reached, go to 3. If it is reached, the first attempt will be to swap any unlocked disconnected page within the class. The second attempt will be to swap any locked disconnected page within the class. If both of these attempts fail, then all pages within the class belong to active jobs of which there are at least 2 active jobs. Note that this cannot happen if the requesting job is in the P class. So if the requesting job is in the IB class, disconnect the requesting job as there is no way to determine priority in this case. If the requesting job is in the B class, the job to be disconnected will be the one which started execution last. If the disconnected job is not the requesting job, then repeat the first two attempts in this step (2).

3. If the machine limit is not reached, go to step 4. If it is reached, the first attempt is to swap any unlocked and disconnected page. The second attempt is to swap any locked and disconnected page. (NOTE: Any swap of a locked page results

in the large page reserve being set.) Again (as in Step 2) if the first two attempts fail, then all pages belong to active jobs and there are at least 2 active jobs. Priority determination is as follows:

a) if the requesting job is in P class, disconnect an IB job if there is one. If there are no IB class jobs disconnect the lowest priority B class job and repeat the first two attempts.

b) if the requesting job is in IB class, disconnect the requesting job.

c) if the requesting job is in B class, disconnect an IB job if one exists. If not, disconnect the lowest priority B class job (which could be the requesting job).

4. No limits are reached. The first attempt will be to allocate a free large block. This search would start at block #7 backwards to block # (8-machine limit). The second attempt will be to clear a large block containing small pages with the search starting as immediately above. The system will search first for a large block containing small pages none of which are I/O locked. If that is not possible, then the large page reserve will be set for the large block containing the least number of I/O locked small pages.

I/O Handling

The pager is set up to handle a large number of faults simultaneously in the hopes of driving three (I/O) devices concurrently. These devices are:

1. System page drum,
2. User's page drum (for core overflow),
3. Disk station.

a) The first step is to issue I/O to release core pages (if necessary). Here the core page entry for the outgoing page(s) is deleted and an entry is made for the new page(s) under a null key (unique for each user).

b) The second step is to poll the user's paging drum (unless new page is a large page or a definite create is decided before hand) for the page(s) in question, and if they exist there then the drum station will write them into core.

c) The third step is, if page is not on the drum and is not a create, issue I/O to read the disk.

d) The fourth step: after necessary I/O is completed, then the null key is replaced with the correct one, the page is unlocked, and the user is unblocked and free to execute again. Note: Advise requests do not block the user from execution.

Shared Library Pages

If a user faults or advises for a library page and a check reveals the same page is already on its way (due to another user's previous fault), the second user becomes library blocked and both (or more) users will be unblocked simultaneously when the page I/O is completed.

Library pages are read-only pages and will drift to the user drum when core overflows. When the drum overflows, they will killed there.

Multiple Page Advises

The entire virtual address range in any one request must not straddle file boundaries. However, some may exist on the drum without all of them on the drum (the remainder being on the disc). If only some exist in core then the request is abandoned, and the user will obtain the remainder of his pages via demand paging.

JOB SCHEDULING

QUEUE MANAGEMENT

There are five (5) separate wait Queues. The jobs in each Queue are waiting for CPU time only.

P Class Queue - First come, first served.
Long tick, long slot.

I Class Queue - Round robin.
Short tick, short slot.
Requeued after each slot time.

IB Class Queue - Round robin.
Long tick, long slot.
Requeued after each slot time.Inclusion in queue via system call only if in I class queue.

B Class Queue - First come, first served except if the job requests a suspend state at which time it will be put at the end of the Queue. Long tick, long slot.

S Class Queue - Round robin. Run only during idle time periods.
No tapes.

The job catagory is defined on the execute lines; file name / T X where T=time limit in minutes and X=P, I, B or S. Note that this replaces the current "bid priority" designation.

SLOT ASSIGNMENT

For P and S class job, the slot has meaning only as a maximum accounting period. This is so because any time the system decides to disconnect a P job or an S job, it is done immediately without regard to slot times.

For I, IB and B class jobs, there has to be an upper limit due to its accounting period function. But here, slot also has another function. Namely, an I, IB and B class job that is disconnected is able to finish its current slot time.

Note that slot times are used up by "charges" other than CPU time. Namely: any explicit I/O charges, implicit Disc I/O charges for page faulting (core overflow I/O charges and drum overflow I/O charges are not included), system call charges and core/drum residence charges.

PSLOT = 15 seconds, S slot = 15 seconds, IB slot = 15 seconds

BSLOT = 15 seconds, I slot = 3 seconds

TICK ASSIGNMENT

The tick assignment, would be a number 1-100 representing a percentage of the length of time it would take to get once around the alternator loop. This time length would be a constant derived from hardware & software constrain concerning interrupts.

PTICK = 5∅*#P NOTE: #P=∅ or 1

STICK = 100/#S If no other class in alternator
= 1 otherwise

The TICK computation for I, IB and B incorporates the following base:

BASE = 100 - PTICK *#P IF ONLY ONE OF THE CLASSES I,
IB or B IS PRESENT IN THE ALTERNATOR

BASE = (100 - PTICK * # P) / 2 IF TWO OF THE CLASSES
I, IB or B EXIST IN THE ALTERNATOR

BASE = (100 - PTICK * # P) / 3 IF ALL THREE CLASSES
I, IB AND B EXIST IN THE ALTERNATOR

THEREFORE:

ITICK = MIN (BASE / # I , 2)
IBTICK = MIN (BASE / #IB , 10)
BTICK = MIN (BASE / # B , 10)

ANY TICK WILL BE A MINIMUM OF 1.

ALTERNATOR MANAGEMENT

The alternator loop needs to be "managed" only if core is full with active job pages and/or the alternator slots are full. The "management" consists of a decision as to which job gets disconnected.

Given the following rules:

Rule 1. Only one P class in alternator.

Rule 2. If an S class exists in the alternator, then either, all jobs in the alternator are S class or, only two jobs are in the alternator, one S and one non-S.

Rule 3. The one P class can keep <u>all</u> others out.

Rule 4. B class will be guaranteed an alternator slot if more than one I class exists.

Rule 5. IB class will be guaranteed an alternator slot if more than one I class and more than one B class exists.

Decisions made when a job wants entry into the alternatòr loop, when all entries are taken or core is full.

S class - does not get into the loop. Other classes:

1. If any S class job's exist in the loop, they are the first to be disconnected.

2. If more than one IB class exists, IB is the next to be disconnected.

2a. If only one IB class exists and an IB wants in, the existing IB is disconnected.

3. If more than one B class exists, a B class is the next to be disconnected.

3a. If only one B class exists and a B wants in, the existing B class will be disconnected if it arrived in the B class queue <u>after</u> the job desiring entry.

4. If more than one I class exists, an I class is the next to be disconnected.

4a. If only one I class exists, and an I class wants in, the existing I class will be disconnected.

5. If none of the above conditions are met then if an IB class exists, IB is disconnected; secondly if a B class exists, B is disconnected. If these two conditions fail, then if the job wanting in is <u>not</u> P class, it does not get in. If it is a P class job, then the one existing I class is disconnected.

TIME USAGE AND CHARGING

(A) General Flow of Time Usage vs. Charge for STAR

When a job is initialized, the time limit is supplied by the user via TTY execution line or via "initialize controllee" system call. This time limit is converted to microseconds and stored in the descriptor block in variable -TL-. In the user table block is a variable -MONEY-, which contains the amount of time available to this job in microsecond units.

Each time a job is entered into an alternator slot, a variable -HORA-, in the user's minus page, is cleared and the variable -SLOT-, also in the minus page, is computed by the subroutine -SLOTAC-. An initial value of 1/4 minutes is compared with -TL-. If the job is in the interactive Queue, the initial value is 1/2 ∅ minute. If -TL- is less than the initial value, the value of -TL- is used. This value multiplied by the user's priority (.1 if standby job, unity otherwise) is compared with -MONEY-. If -MONEY- is less, then a value is chosen, which when multiplied by priority = -MONEY-. This value is stored in the variable -SLOT-. (Microsecond units.)

During execution, various time usages are collected into the variable -HORA-. When -HORA- $\geq$ -SLOT-, the user's -MONEY- variable and -TL- variable are decremented (as described in Section C). A new value for -SLOT- is computed as above and -HORA- is cleared. -MONEY- and -TL- are also decremented whenever a job is disconnected from its alternator slot. Also, at the time of the decrementing (in subroutine -BANKAC-), time card entries are made in the subroutine -ACCTG-.

HORA - Rightmost 32 bits, word 18_{10} in minus page.

SLOT - Rightmost 32 bits, word 17_{10} in minus page.

TL - Rightmost 48 bits, word 4 in descriptor block.

MONEY - Rightmost 40 bits, word 8 in user table block.

(B) Money and TL Decrementing

Subroutine -BANKAC- decrements -MONEY- (Bank Account) and -TL-. See Section A for frequency. Prior to decrementing MONEY and TL, each temporary sum (multiplied by a weight factor) is added to its respective accumulated sum (except TPHOOK).

TCPUC → CPUCHG, TMEMC → MEMCHG, TEXIO → EXIO, TIMIO → IMO, TSYSC → SYSCG, TREMIO → REMIO. These accumulated sums are available to the user via a system call. These sums are cleared only at initial execution (i.e., new TTY execute line, or new "INITIALIZE CONTROLLEE" call). TL is decremented by the sum of these temporaries *Weight factor, i.e., TL = TL - ((TCPUC*CPFACT) + (TMEMC*MEFACT) + (TEXIO*EXFACT) + (TIMIO*IMFACT) + (TSYSC*SYFACT) + TREMIO*REFACT)). MONEY is decremented by the same sum as -TL-, but multiplied by priority first (.1 for standby job, unity for others).

(C) Time Card Entries

This is the table of time usage. It is periodically written to a disc file for later editing. Maximum period is one hour. Period also occurs whenever a buffer fills (256 words) or a bank update occurs. This table contains 2 buffers, each 256 words in length. The first 5 words of each buffer and the last word of each buffer is fixed.

Word 1	Wyman clock when buffer started
" 2	Internal microsecond clock when buffer started
" 3	System downtime
" 4	Wyman clock when buffer written to disc
" 5	Internal microsecond clock when buffer written
Word 256	if = zero, this buffer in use if = #222222222222, this buffer to be written to disc
Words 6-255	may contain the following 3 different types of time usage records. A zero entry where the first word of a record should begin indicates the end of the records in the given buffer.

(C) Cont'd.

Type Ø - Regular Entry (7 words)

Regular Entry

E

Word 1	T 2	TPHOOK 6	USERNO 24	DEPT 32
2	DISACC 16	DSECT 16	DSECMIN 32	
3	DRACCD 16	ACCTNO 48		
4	TPWDS 32	TPACC 8	DRACC 24	
5	JOB 8	SYSCHG 24	CPUTIME 32	
6	CDPGMILL 34	CDPAGES 30		
7	DRACCO 16	FILTP 12	TPFUNCT 12	DISCSEC 24

TTY Entry

1	T 2	6	USERNO 24	DEPT 32
2	16	TMESS 16	TWORDS 32	
3	TERMS 16	ACCTNO 48		

Disc Purge & File Process Entry

1	T 2	6	USERNO 24	DEPT 32
2	DISACC* 16	DSECT 16	DSECMIN 32	
3	TPWDS* 32	TPACC* 8	ACTDEG 24	

(C) Cont'd.

ACTDEG	=	Alpha portion of account number (3 ASCII char's.)
ACCTNO	=	Account number (6 - 8 bit ASCII characters)
CDPAGES	=	Core/drum storage # pages in CDPGMILL
CDPGMILL	=	" " " , # pages * Milliseconds
CPUTIME	=	User execution time in microseconds
DEPT	=	Division code (4 - 8 bit ASCII characters)
DISCACC	=	# of disc accesses
DISCSEC	=	# of sectors in disc I/O
DRACC	=	# drum accesses for page fault & explicit I/O
DRACCO	=	" " " when user causes core overflow
DRACCD	=	" " " due to drum overflow
DSECMIN	=	Disc storage, # of sectors * minutes
DSECT	=	Disc storage, # of sectors in DSCEMIN
FILTP	=	# data bursts to mass store, TMDS
JOB	=	Job class 1: priority 2: interactive 3: batch 4: standby
SYSCHG	=	System call CPU time in milliseconds
T	=	Flag for which type of entry = ∅, 1, 2
TERMS	=	TTY hookup time in seconds
TMESS	=	# of messages to/from TTY
TWORDS	=	# of words to/from TTY
TPACC	=	# of tape accesses on read/write
TPFUNCT	=	" " " " for function requests
TPHOOK	=	Tape drive hookup time in minutes
TPWDS	=	# words in tape I/O read/write
USERNO	=	User number in binary

NOTE: For T = 2, * items are recorded if userno = 999999 and indicates system resources to process users DD80, PUNCH & HSP files. ACTDEG & DEPT are those of the original user unless userno 999999 is destroying its own file (a rare occurrence).

MINUS PAGE WORDS FOR TIME USAGE

Temporary Collection Over Slot Time

146	TPWDS 16	DISCACC 12	DISCSEC 12	CDPGMILL (TMEMC) 24	
147	SYSCHG (TSYSC) 16	TPHOOL 8	TPACC 8	8	CPUTIME (TCPUC) 24
148	DRACC 16	TREMIO 16	TIMIO 16	TEXIO 16	
149	DRACCO 16	LASTUP 48			
150	DRACCD 16	FILTP 8	TPUNCT 8	8	CDPAGES 24

The following fields are entered into a time usage entry each "slot" time: TPACC, DISCACC, DISCSEC, FILTP, TPFUNCT, TPHOOK, TPWDS, CPUTIME, CDPGMILL, SYSCH, DRACC, DRACCO, DRACCD and CDPAGES,

The following fields are entered (i.e. summed) into the execution collection of usage charges: CPUTIME, CDPGMILL, SYSCHG, TREMIO, TIMIO AND TEXIO.

Execution Collection of Charge of Time Usage

Word 140	PGFLT 16	CPUCHG 48
141	DRFLT 16	MEMCHG 48
142	EXIO 32	REMIO
143	IMIO 32	SYSCHG 32

where: CPUCHG = CPUTIME * CPFACT

MEMCHG = CDPGMILL * MEFACT

EXIO = TEXIO * EXFACT

STAR SYSTEM TERMINAL INTERFACE

STAR ID Line Sequence

The ID line for STAR has the form

IDT NNNNNN L AAAAAA PL CCCCCC

Where: NNNNNN is the six digit employee number of the user loggin in.

T is the letter designator for STAR

L is an alphabetic suffix A-D under which the user wishes to operate. (Each user may have up to four jobs active, one under each suffix).

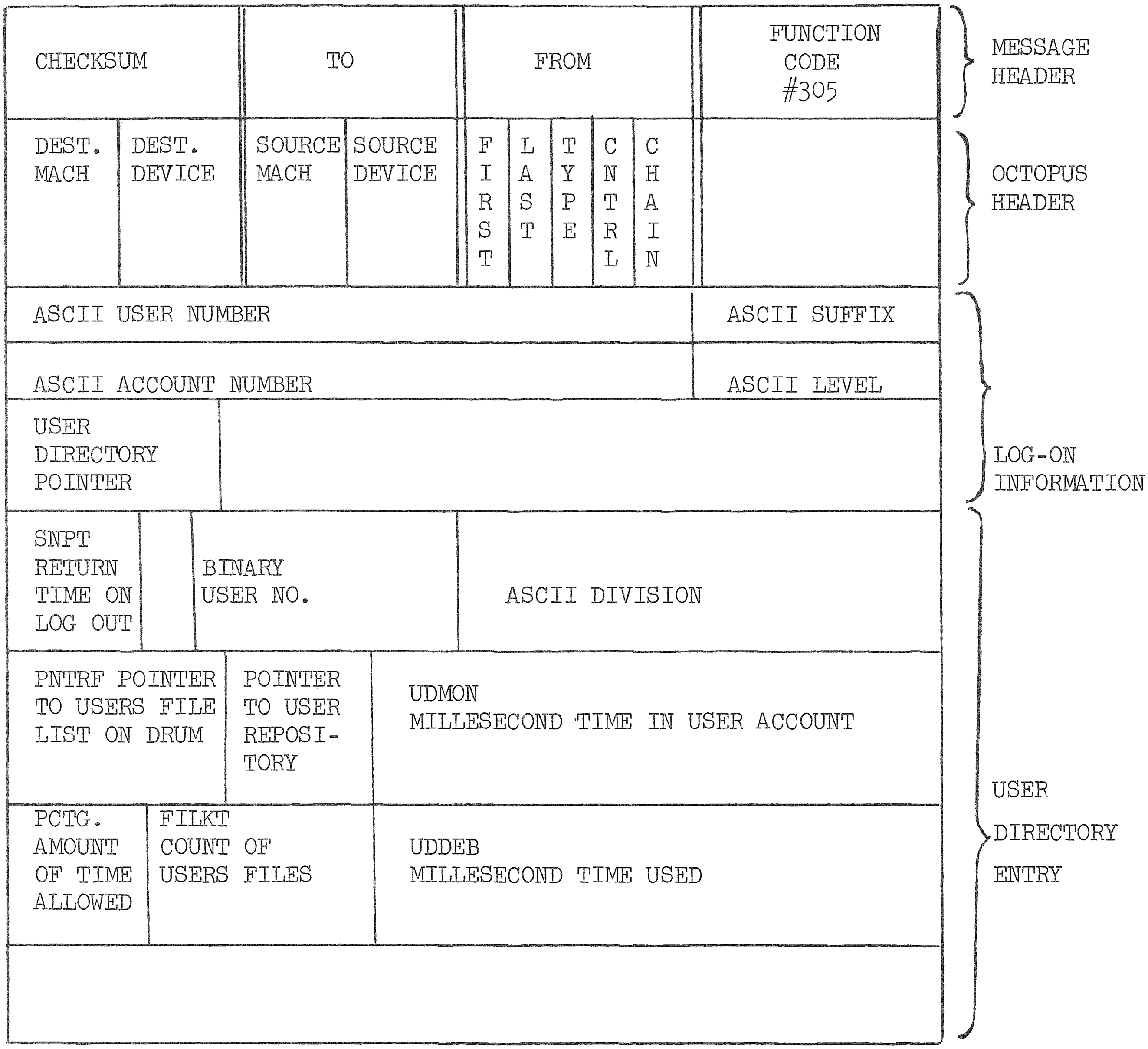
AAAAAA is a billing number consisting of three alphabetic characters followed by three numeric characters.

PL is a letter designator (optional) for protection level if CCCCCC is used. Valid letters are P, A, S, K

CCCCCC is an optional six alphabetic character combination (not echoed to teletype) used for classified access.

The teletype line is assembled by the PDP-8 to which the teletype is connected, prefaced by an Octopus network header of 48 bits and routined to the STAR service station.

The service station verifies the parameters in the ID line to determine if it is valid. If it is, the service station sends a message to central consisting of a one-word header with function code #305 followed by a message consisting of the Octopus header left justified in the first 64-bit word, a three-word block of log-on information in the second, third, and fourth words, and a four-word user dictionary entry in the fifth through eighth words. The message has the following format.



MESSAGE HEADER

CHECKSUM	TO	FROM	FUNCTION CODE #305

OCTOPUS HEADER

DEST. MACH	DEST. DEVICE	SOURCE MACH	SOURCE DEVICE	FIRST	LAST	TYPE	CNTRL	CHAIN	

LOG-ON INFORMATION

ASCII USER NUMBER	ASCII SUFFIX
ASCII ACCOUNT NUMBER	ASCII LEVEL

USER DIRECTORY POINTER	

USER DIRECTORY ENTRY

SNPT RETURN TIME ON LOG OUT		BINARY USER NO.	ASCII DIVISION

PNTRF POINTER TO USERS FILE LIST ON DRUM	POINTER TO USER REPOSI-TORY	UDMON MILLESECOND TIME IN USER ACCOUNT
PCTG. AMOUNT OF TIME ALLOWED	FILKT COUNT OF USERS FILES	UDDEB MILLESECOND TIME USED

The kernel picks up the message from the service station, moves it to a free slot in the teletype message buffer (TTYS), queues the teletype message processor if it is not already queued and returns a response to the service station.

The teletype message processor recognizes the message as a log-on message by virtue of the "first" bit being set in the Octopus header word. A user table entry is assigned and filled in if no entry for this user already exists. An entry will already exist if the user had previously logged out leaving a job active in the system. If no user table entry already existed and the user had files catalogued in the inactive file index on the service station drum, the inactive entries are read and placed in the active file index in central memory. At this point, the log-on sequence is complete.

STAR Execute Line

The STAR system expects an execute line whenever a user is logged on under a suffix and no program is active under that suffix. An execute line is a message to the system to start a program under that suffix. An execute line has the form.

FILENAME$_\wedge$MESSAGE $_\wedge/_\wedge$ T$_\wedge$X

where: FILENAME is the name of a virtual code file containing the program to be run.

∧ is the blank character (space bar).

MESSAGE is an initial message for the problem and may contain any character string which does not have the string $_\wedge/_\wedge$ as a substring.

T is a decimal (possibly integer) number specifying the number of minutes of real time the program is allowed to run.

X Is The Job Class

P	Priority
B	Batch
I	Interactive
S	Standby

The character string ∧/∧T∧X may be ommitted, in which case T = 1, X = I will be assumed by the system.

The teletype line is assembled by the PDP-8 to which the teletype is connected, prefaced by an Octopus network header of 48-bits and routed to the STAR service station. The service station sends a message to central consisting of a one-word header with function code #305, the Octopus header left adjusted in the second word and the execute line starting in the third word.

The KERNEL picks up the message, moves it to a free slot in the teletype message buffer, queues the teletype message processor if it is not already queued, and sends a response to the service station.

The teletype message processor recognizes this as an execute line by noting that no DB number is filled in the user table PROG slot for this suffix and the message is not a break and does not begin with a (CTRL-D) or (CTRL-E). It then verifies that the FILENAME exists and is a virtual code file. If this check fails, an error message is sent to the teletype, otherwise the message processor:

1. Assigns a free DB.
2. Sets system action in progress for the assigned DB in DBSTAT
3. Assigns keys for the DB.
4. Sets PROG in the user table for the appropriate suffix.
5. Sets STATE = #30 for this DB.
6. Sets FCNTLR in the DB if the message is not void.
7. Fills in an entry in the XEQBUF table.

Since the message processor may have more than one execute line to process when it is run, there may be multiple entries made in XEQBUF.

If more than eight execute lines are waiting, the first eight are processed and an error message sent on the remainder. The message processor then calls subroutine CRSDF in the file management routines. For each execute line the file manager will:

1. Fill in MP3 in the DB.
2. Read the minus page into the appropriate MPT entry.
3. Examine IOC (17) for an existing drop file and verify certain items in that IOC.
4. Examine virtual map entries for any pertaining to IOC (16), the source file IOC and verify certain items in the map entries.
5. Create an automatic drop file if none exists and fill in IOC (17).
6. Fill in IOC (16) with source file information.
7. Update the XEQBUF table and return to the execute message processor.
8. The execute message processor will examine XEQBUF. It will send out an appropriate error message for those lines in error and release their DB's, keys, and zero their PROG's. If any execute lines are OK, it will set their DB states to #11 and call subroutine QUEUER. The processed entries will be removed from XEQBUF. QUEUER will then reorder the queue of jobs to be run.

STAR BYE Line

The BYE line enters the central system in the same manner as ID and EXECUTE lines. The message itself consists of the teletype character generated by simultaneously striking the control key and the D key (denoted (CTRL-D)). The purpose of this line is to sever the connection between the terminal and the user table entry. Any active jobs remain active. Note that logging off of the terminal may affect the behavior of the problem program, since some of the message calls to the system give different results, depending on the presence or absence of a terminal connection.

STAR Break Line

The BREAK key is used to terminate a job. If no problem program is running under the logged in suffix, the message "NO PP" is sent to teletype. However, if PROG is set, the system searches down the controllee chain until finding a problem program in a RUNNING OR WAIT ALT state and terminates this problem program and all (if any) of its controllees. If no problem program is in the above states, the entire chain is terminated. The following message is sent to the next higher level problem program controller:

```
#0D414C4C20444F4E    "CR ALL DONE CR LF ETB B"

#450D0A1742000000
```

or if the terminated problem program was attached to teletype, the message "BREAK" is sent to teletype.

STAR Message Line

Whenever a problem program is active under the suffix to which a teletype is connected and the system receives a message from that teletype whose first character is not a (CTRL-E) character, the message is assumed to be a message to the problem program (or possibly to one of the problem programs in the controllee chain for that suffix designated by a message control call). The message is removed from the teletype input buffer, placed in a system buffer and pointed to by an entry in the appropriate descriptor block. The message is then obtained by the program connected to that descriptor block by a GET MESSAGE from controller or a GET SYMBOLS from controller call. If a message is waiting for a problem program and a second message is typed before the first message is asked for by the problem program, the second message replaces the first and the first is lost.

System Inquiry Messages

There exists a class of messages, each preceded by a (CTRL-E) character which are considered to be messages to the system and which may be sent whether or not a problem program is active under the suffix

System Inquiry Messages Cont'd.

under which the teletype is logged in. These include:

1. (CTRL-E) S get program state.
2. (CTRL-E) T get time and date.
3. (CTRL-E) ? get date, time, state, bank.
4. (CTRL-E) GXX get XX minutes from the repository to which this user belongs.
5. (CTRL-E) I teletype interrupt.
6. (CTRL-E) U list time used by this user today.
7. (CTRL-E) OP MESSAGE send "message" to the operator's teletype.
8. (CTRL-E) SU list active suffixes.
9. (CTRL-E) BB list bank account.
10. (CTRL-E) BP list time in repository to which user is connected.
11. (CTRL-E) PR list number of jobs in I Class waiting to be connected to an alternator slot.

Program State Mnemonics

Mnemonic	
RUNNING	Program is in the alternator loop.
WAIT ALT	Program is waiting for an alternator slot.
WAIT TPE	Program is waiting on tape assignment.
WRT CNTR	Program is waiting for controller to get on disk.
WRT CNTE	Program is waiting for controllee to get on disk.
RCV CNTR	Program is waiting to get a message from controller.
RCV CNTE	Program is waiting to get a message from controllee.
RCV PDP	Program is waiting to get a message from the PDP-6.
SND CNTR	Program is waiting to send a message to controller.
SND CNTE	Program is waiting to send a message to controllee.
SND PDP	Program is waiting to send a message to the PDP-6.
SND OPR	Program is waiting to send a message to the operator.
SND TTY	Program is waiting to send a message to the teletype.
DUMPING	Program is in a state of being dumped to disk.
FINISH	Program is finished. Clean-up is in progress.
SUSPEND	Program is suspended.

ERROR MESSAGES DETECTED BY EXECUTE LINE PROCESSOR

1. NO FILE - File name does not appear in User's private file index or the Public file index.

2. NON-EXECUTABLE FILE - File is not a virtual code file.

3. NO TL - After$_\wedge$/$_\wedge$ illegal or no time limit specified.

4. BAD CLASS - Class is not P, I, B, or S.

5. NO TIME IN BANK - User has no MONEY in his bank account.

6. NOT ENOUGH TIME FOR JOB - TL* CHARGE (unity for P, I, and B, .1 for S) is greater than MONEY.

7. SYSTEM TABLES FULL. TRY AGAIN - No room for job.

8. SYSTEM DROP FILE CREATE ERROR - System cannot create drop file.

9. DISK TROUBLE

10. SOURCE OR DROP FILE ANOMALY

11. DROP FILE TOO SMALL

STAR FILE MANAGEMENT

A requirement exists for a catalogue of all files currently being stored by the system. This catalogue is called the file index. Another requirement is a map of allocated disk space. This map is called the disk map.

Because of the large number (up to 21000) of files to be catalogued for each 817 disk file, it would be inefficient to attempt to keep the full catalogue of files in central memory; hence the file index is divided into an active file index in central memory and an inactive file index on the service station drum. A file index entry requires 8 words allowing 64 files to be catalogued in a small page. Because most users have less than 64 files, and for retrieval from the inactive file index we wish to block entries by the user, and, furthermore, the service station drum is alterable in quarter pages, the inactive file index is allocated in quarter page blocks cataloging 16 files. For users with more than 16 files, inactive index blocks can be chained together. A bit map is used to record full and free blocks in the inactive file index. The inactive file index contains the entries of all users who do not currently have an entry in the user table.

The active file index will be a resident system table in the initial systems. For later systems, it will be made pagable. Entries in the inactive file index have the same format as in the inactive file index, but the use of the table will differ from that of the inactive file index. When an inactive user logs on, his user directory entry is sent to central as part of the log-on message. The directory contains a pointer to the users first block in the inactive file index. The pointer is null if the user has no files. The file system reads the blocks containing the user's files from the service station drum. For each file, a hash address in the active file index is generated using a concatenation of the file name and user number. The hash address provides a starting point for a quadratic search through the active file index to find a vacant entry for the file. This technique produces a very even fill of the active index. As the user's files are entered in the active index, they are chained together and a pointer to the head of the chain is placed in the user table to facilitate searches of the user's private files.

For later systems, the technique for entering files in the active index will change, since it is desirable to produce a dense fill of the index with each user's private file set close together to reduce page faulting by the file system.

The disk map will require one page per unit. (Each 817 disk contains two units.) This map is unaware of discrete files. It merely maps assigned space on disk. Voids are defined by the disparity between two consecutive entries when comparing the first word address plus length of the first entry with the first word address of the second entry. Thus, it is possible, in theory, to completely fill a disk with a number of files and have only one entry in the disk map.

When a user logs on, he is inhibited from executing any jobs until his files have been entered in the active file index and his inactive file index blocks have been released. This is because when a user sends an execute line, his private file index is searched before the public file index in order to allow him to utilize files with the same name as public files.

If a user logs off with no jobs active, his files are returned to the inactive file index. Note that since his inactive index blocks were released at log on, new blocks must be assigned when his files are returned to the inactive index, and his user directory entry must be updated with a new inactive index pointer.

If the user has a job active at log-off time, his files remain in the active index. When the job completes a time is set in his user table entry. At some time ΔT later, if the user has not logged back on, he is considered to be inactive, and his files are returned to the inactive index. His user table is not, however, released since it contains a message pointer to the last message to be sent to the terminal so that the user may determine what happened to his job when he next logs on.

Because of the limited storage available on disk, private files which have not been referenced for a fixed period of time are purged from the system. Purging of files for an active user is the responsibility of the file management system. Purging of files in the inactive index is done by the service station.

It is also the responsibility of the file system to process all user calls pertaining to files.

THE STAR DROP FILES

Since the STAR system requires that each page in the memory drum system have some disk correspondence for its current image, cases often occur when a page with a read-only disk image is modified or a free page is assigned, and hence disk space is required which has not previously been specified by the user. In order to handle this situation, whenever an execution is started, the system automatically provides a file called a drop file into which these pages may be mapped. The actual disk correspondence is kept in the drop file map portion of the minus page and the minus page as well as page zero are two pages which are entered in the drop file.

In this manner, every executing process consists of at least two files, the file whose name appeared on the execute line termed the source file and the system created file termed the drop file. More files may be associated with the process through explicit user action.

The default options for the drop file are a name created from the source file name and some random hash characters with a length equal to the source file length. The user has two ways of controlling system action on drop files. He may specify a drop file length to be associated with a source file utilizing the close call, or he can explicitly create a drop file with the desired name and length utilizing a create call within the source file. In the second case, the create call must occur before any pages are written to the drop file.

The user should be aware that the drop file map is constructed essentially on a page-by-page basis, and the drop file map is of finite size. Attempting to add a page to a drop file map which is full is a fatal error. Users desiring large blocks of virtual space not represented in some file should create a virtual file and map it into the desired space to avoid this difficulty (via Map-In Call).

STAR DISK FILES

Ownership Categories

Public - Public files are accessible by the entire body of users. The files have execute-only protection, i.e., they may not be read or written by a user -- only executed. These files are expected to be general purpose programs which augment the capability of the STAR operating system.

Shared Private - Shared private files are those which are accessible by some subset of the body of users. Typically, a file in this category can be accessed by any member of the subset according to the access rights given by the originating user. There may be subsets bounded by Laboratory divisional codes, by security pools or some other boundary. These are, as yet, undefined subsets as are the rules for manipulating the files, but the skeletal structure for implementing them exists.

Private - Private files may be accessed exclusively by the originating user. He is free to manipulate the content, access rights, security level, external access, lifetime, etc., as he wishes. The operating system will maintain the right to terminate a file based on its quiescent lifetime. This will be done to allow a reasonable amount of the disk store to be available at all times.

Management Categories

Private - Management of private disk files is left entirely to the originating user. The operating system will protect these files from access by any other user. The sole exclusion is the system's right to purge based on lifetime, as mentioned above.

Scratch - Scratch files may be created only by a user program. They will exist only for the duration of the activity of the originating program. When the program terminates normally, all scratch files will be destroyed. If the system terminates the program because of a fatal error, or because the BREAK message was received, or if the program terminates with a request to save its drop file, the scratch files will be saved as read/write files. If the system call to close a file is issued on a scratch file, it will be destroyed. Scratch files will have read/write access.

Output - Output files may be created only by a user program. They will exist only for the duration of the activity of the originating program. When the program terminates normally, all output files will be given to the system privileged user for processing. Output files must have legitimate names for the devices for which they are destined. These names will follow existing Laboratory tradition.

Write Temporary - A write temporary file will be treated as having read-only access. However, pages from such a file may be modified in core by a program. When this happens, the modified image will be catalogued as part of the program drop file, and subsequent reference to that page address space will cause the modified page to be accessed. Of course, the space may be mapped out of the drop file in order to reference the source image again.

Drop - The drop file is that disc space set aside for dumping the altered pages of an executing program. The file is created by the operating system automatically as part of the sequence for starting a new program. It is created at the length of the source file or a particular length may be specified in the File Index entry at which a program's drop file is to be created. A program may also create its own drop file which causes the automatic drop file to be destroyed. This may be done only if no pages have been written to the existing drop file. The drop file will be preserved for any abnormal termination and may be preserved or destroyed, at the option of the program, upon normal completion.

Type Categories

Sequential Data - A sequential file is, by definition, a data file. It may not be executed. It is not associated with the executing program's virtual space. Any I/O to or from the file is done by the program explicitly by means of a windowing technique.

Virtual Data - A virtual data file is not assumed to have a suitable minus page for execution, though it may have. It must be mapped into the executing program's virtual space and any reference to the defined space is treated as an access interrupt, and the I/O to retrieve the data from disc is accomplished by the operating system. It may not be executed.

Virtual Code - This is an executable program file. It is presumed to have a suitable minus page for execution, i.e., an invisible package and virtual maps to define the physical disk-virtual space correspondence.

Access Categories

Write - A file having write access may be written into by a problem program or the operating system. In the case of a virtual data file, this means that modified pages will be returned in place to the original file.

Read - An attempt to write explicitly into a read-only file will produce an I/O error. An attempt to modify a page from a read-only virtual file will produce a fatal error. There is, however, a means for mapping in portions of read-only virtual files and giving those portions write access. The pages in these map-ins will be treated like those of a Write Temporary type file.

Execute - Any attempt to read or write an execute-only file will be denied. Typically these files will be public, utility code files. Only the system or a privileged user will be able to update them.

STAR MINUS PAGE

Every virtual file to be used within the STAR OPERATING SYSTEM must have a minus page. This rule applies to virtual code and virtual data files. Files which are known to be sequential will not have this requirement.

For the virtual code file, it will be the responsibility of a program loader to prefix a minus page to the body of the code and fill in the required virtual maps. For files which are virtual data, the creating user must manufacture the minus page and maps. The virtual map definitions contained in the disc image of virtual files may be modified dynamically through system calls.

In the instance of an executing problem program, the operating system uses the minus page to store such information as the executing IP (Invisible Package), interrupt IP, time-slicing data, input/output connections to disc files and tape drives, maps of the program's defined virtual space, time charging data, page fault statistics, etc.

Following is a diagram of a system minus page, and the definitions of the information contained in it. In some cases it has been necessary to expand some entries. The expansions and their definitions follow on succeeding pages.

STAR PROGRAM MINUS PAGE FIELD DEFINITIONS

INVISIBLE PACKAGE #1	Executing package.
INVISIBLE PACKAGE #2	Interrupt package.
Program restart temporaries	Operating system use.
TICK	CPU time slice each turn in alternator.
TICKL	Amount of tick left.
SLOT	When HORA=SLOT, charging done.
PINC	P-counter increment for call.
HORA	Accumulated charges since last accounting.
USER IOC	User I/O connector for disk, tape I/O.
SOURCE IOC	System I/O connector for program source file.
DROP IOC	System I/O connector for program drop file.
Words 136_{10} - 138_{10}	Self-explanatory. Counts are entries.
ERROR NO.	A code number defining some fatal error.
ERROR ADDRESS	Location where error occurred.
CPUCHG	CPU sec. Sum over entire execution.
MEMCHG	Core/drum residence msec. Running sum.
SYSCHG	System call msec. Sum over entire execution.
IMIO	Implicit I/O msec. Running sum.

STAR MINUS PAGE, Continued

Field Definitions

EXIO	Explicit I/O msec. Running sum.
REMIO	Remote I/O msec. Running sum.
PGFLT	Count page faults over entire execution.
DRFLT	Count drum hits on page fault. Running sum.
TCPUC	CPU sec. Temporary sum for each slot.
TMEMC	Core/drum residence msec. Sum for each SLOT.
TEXIO	Explicit I/O msec. for each SLOT.
TIMIO	Implicit I/O msec. for each SLOT.
TSYSC	System call msec. for each SLOT.
TPHOOK	Tape drive hook-up minutes for each SLOT.
TREMIO	Remote I/O msec. for each SLOT.
LASTUP	Last time memory charge accounted for.
TPACC	# of tape accessess for read/write.
DISCACC	# of disc accessess.
DISCSEC	Disc I/O, # of sectors.
FILTP	# of data bursts, mass store & TMDS.
TPFUNCT	# of tape functions.
TPWDS	# of tape words read/written.
DRACC	# of drum accesses due to page fault. or explicit I/O of user.
DRACCO	# of drum accesses caused by user. due core overflow on page fault.
DRACCD	# of drum accessess due to drum overflow.
CDPAGES	# of pages in TMEMC computation.
INTERRUPT ADDRESSES	For I/O and TTY interrrupt, see p. 2.46.1
BOUND SEQUENTIAL MAPS	
BOUND VIRTUAL MAPS	Expanded on following pages.
DROP MAPS	

STAR PROGRAM MINUS PAGE

(Loc$_{10}$)	0		63	(Loc$_{16}$)
0 -15	INVISIBLE PACKAGE (IP) #1			0 - F
16	Program Restart Temporaries			10
17	TICK 8	TICKL 24	SLOT 32	11
18	PINC 8	Monitor Restart Address 24	HORA 32	12
19 -31	Program Restart Temporaries			13 - 1F
32 -47	INVISIBLE PACKAGE #2			20 - 2F
48 -63				30 - 3F
64 -127	USER IOC # 0 - IOC # F (4 wds/entry)			40 - 7F
128-131	SYSTEM IOC #10 (Source File)			80 - 83
132-135	SYSTEM IOC #11 (Drop File)			84 - 87
136	Free 32	Bound Seq. Count	Pointer to first Seq. map entry 24	88
137	Free 32	Bound Virtual Count	Pointer to 1st bound virtual map 24	89
138	Free 32	Drop Count 8	Pointer to 1st drop map entry 24	8A
139	Error No. 16	Virtual Bit Address of Error 48		8B
140-150	Expanded on 3.4.1Ø - Time Usage Entries			8C - 96
151	Explicit I/O Information			97 - 9A
152-159	INTERRUPT ADDRESSES			98 - 9F
160-175	Bound Sequential Files Map	1 Word/entry 16 entries		A0 - AF
176-255	Bound Virtual Files Map	2 Word/entry 40 entries		B0 - FF
256-511	Drop File Map	1 Full Word 1 half Word	170 entries	100-1FF

EXPLICIT I/O AND INTERRUPT INFORMATION IN THE MINUS PAGE

151	IOUT1 8	IOUT2 8	IOUT3 8	IOUT4 8	IOUT5 8	IOUT6 8	LGPG 8	SMPG 8
152 - 157	INTNO 8	ISTCK 8	IOREQ 8	ALFWD 42				
158	INTNO 8	ISTCK 8	A 1	5	Controller interrupt P counter (word v.a.) 42			
159	INTNO	ISTCK	A 1	5	Controllee interrupt P counter (word v.a.) 42			

IOUTi A bit set means I/O is out. Each IOUT contains 8 bits for the 8 possible BETA requests. There are 6 IOUT fields for the 6 possible #100 calls.

LGPG Number of large pages with I/O outstanding.

SMPG Number of small pages with I/O outstanding.

ALFWD ALPHA word pointer for I/O request. Note the interrupt address (if one) is in the ALPHA (2) word.

IOREQ Contains the I/O BETA request which is being processed or last processed.

ISTCK A bit set means an interrupt is stacked waiting for previous I/O to finish. ISTCK contains 8-bits for the 8 possible I/O BETA requests (MW 152-157) or 1-bit for the controller (MW 158) or controllee (MW 159)

INTNO Nonzero means the PP is currently in an interrupt routine. N is set to the Beta request number if I/O (152-157) or problem DB no. if MW 158-159.

A A bit set which means only messages preceded by a CTRL-E i will interrupt.

STAR PROGRAM MINUS PAGE IOC DEFINITION

An IOC (Input/Output Connector) is a four word block used by the operating system to establish a link between the program and an I/O device.

Each program may have up to 16_{10} (0-15_{10}) such links. For the purposes of the IOC, each logical disk file to which the program connects itself is considered a separate device. The operating system assigns two extra IOC blocks, in user minus page space, for itself. These are IOC (16_{10}) for the program source file and IOC (17_{10}) for the program drop file. The drop file is automatically created through IOC (17_{10}) with the start of each new program. The name of the file is contrived from the first four characters of the source file name and four hash characters. The problem program may create its drop file via system call, if no page has been written to the automatic drop file. The drop IOC is overwritten with creation of the new drop file.

IOC formats and field definitions follow:

STAR MINUS PAGE IOC FORMATS

BOUND SEQUENTIAL FILE IOC

1	File Name (ASCII) 64									
2	Type 8	Sequential Map Pointer 16	No. of Seq. Map Entries 8	LENGTH (Sectors) 16	Logical	Disc	Unit 4	ACCESS 4	Free	OWN 2
3	Page Virtual Address Window 1 32			Size of Window 1 16	Free 16					
4	Page Virtual Address Window 2 32			Size of Window 2 16	Free 16					

BOUND VIRTUAL FILE IOC

1	File Name (ASCII) 64					
2	Type 8	Free 54	OWN 2			
3	Free 64					
* 4	Physical Disc Sector Address of First Page of This space 18	LENGTH (Sectors) 16	IOC Pointer 5	Logical Disc Unit 3	Control 6	Logical Disc Sector Address of First Page of This Space 16
	0					63

*Note that word 4 of a virtual file IOC is normally zero. It is filled in only for the program drop file IOC (17_{10}). Since no entry is made in the Bound Virtual Map for the drop file, the 4th word of its IOC is made to duplicate the second word of a bound Virtual Map entry.

STAR MINUS PAGE IOC FORMATS

TAPE IOC

<table>
<tr><td>1</td><td colspan="2">24</td><td colspan="3">Tape Name (ASCII) 40</td></tr>
<tr><td>2</td><td>Type 8</td><td colspan="2">Free 40</td><td>Log. Unit</td><td>Free 12</td></tr>
<tr><td>3</td><td colspan="5">Unused</td></tr>
<tr><td>4</td><td colspan="5">Unused</td></tr>
<tr><td></td><td colspan="3">0</td><td colspan="2">63</td></tr>
</table>

ACCESS	4 =	Execute Access
	2 =	Read Access
	1 =	Write Access
OWN	0 =	Private file
	1 =	Public file
	2 =	Political file

Expansion of TYPE field:

Principal Type 3	Mode 2	Lockout 3

PRINCIPAL TYPE	0 =	Private disc file
	1 =	Scratch disc file
	2 =	Output disc file
	3 =	Write temporary disc file
	4 =	Tape
* LOCKOUT	1 =	Execute lockout
	2 =	Write lockout
	4 =	Read lockout

*Partially implemented. Ignore until we have shared files.

MODE	0 =	Sequential
	1 =	Virtual disc file data
	2 =	Virtual disc file code

STAR MINUS PAGE FILE MAPS

Bound Sequential Files

Bound sequential files will consist of one single contiguous disc segment. One sequential map entry per sequential IOC is allowed. The IOC and its map entry will be positionally related in their respective areas through IOC number.

BOUND SEQUENTIAL MAP ENTRY FORMAT

Physical Disc Sector Address of Page Zero 18	LENGTH (Sectors) 16	⋈ 1 5	⋈ 2 3	⋈ 3 6	Logical Disc Sector Address of Page Zero 16

0 63

⋈1 - IOC Pointer
⋈2 - Logical Unit
⋈3 - Control

Bound Virtual Files

Bound virtual files may consist of discontinuous address space. Up to 40_{10} virtual space segments can be simultaneously mapped. The various segments may belong to one IOC or each segment may have its own IOC. Each segment points to the IOC currently using it. In the following diagram, the symbol ≠ will indicate those fields for which the program loader will be responsible. The operating system will make the logical-physical connections at execution time.

BOUND VIRTUAL MAP ENTRY FORMAT

<table>
<tr><td>▨ 17</td><td colspan="4">Virtual Page Address 32</td><td>Free 15</td></tr>
<tr><td>Physical Disc Sector Address of First Page of This Space 18</td><td>LENGTH (Sectors) 16</td><td>⋈ 1 5</td><td>⋈ 2 3</td><td>⋈ 3 6</td><td>Logical Disc Sector Address of First Page of This Space 16</td></tr>
</table>

⋈ 1 - IOC Pointer
⋈ 2 - Logical Unit
⋈ 3 - Control

Control Field Expansion:

C_1	C_2	C_3	C_4	C_5	C_6
42	43	44	45	46	47

Control Field Expansion, continued:

C_6	= 0	Small pages (512 words)	
	= 1	Large pages (65536 words)	
C_5	= 1	Read access	Note that the various segments of a virtual file may have differing access rights.
C_4	= 1	Write access	
C_3	= 1	Undefined	
C_2	= 1	Kill Pages In Core/Drum System Upon Map Out	
C_1		Preload Pages	

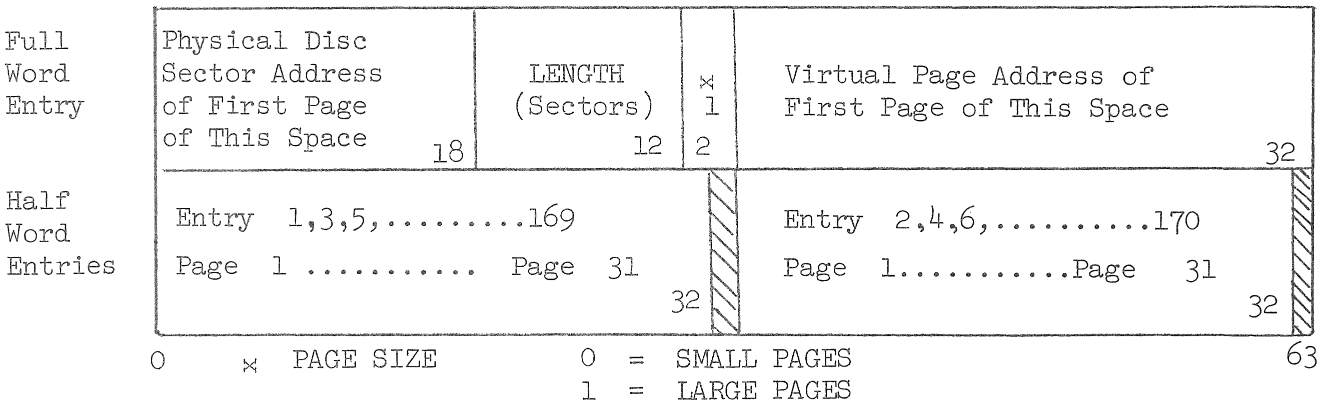
The bound virtual map will be divided such that all word 1 entries are in the first half of the map space and all word 2 entries are in corresponding positions in the second half. Entries will be sorted by ascending virtual address. Blank entries will be squeezed out.

STAR MINUS PAGE FILE MAPS

Drop File (Free Space) Map:

This space maps the disc drop file indicated in IOC (17). It will contain entries for the program minus page and program virtual page zero, initially. As the program executes, any free space (virtual space not defined in the bound virtual map) which the program attached will be mapped here. Also, any modified pages of ▿write temporary▿ files will be entered in this map, along with any modified source file-program-pages. Up to 170_{10} such entries can be made with up to 31_{10} pages in each entry.

DROP FILE MAP ENTRY FORMAT


Full Word Entry
Physical Disc Sector Address of First Page of This Space
18
LENGTH (Sectors)
12
×
1
2
Virtual Page Address of First Page of This Space
32
Half Word Entries
Entry 1,3,5,.........169
Page 1 .... Page 31
32
Entry 2,4,6,..........170
Page 1...........Page 31
32
0
× PAGE SIZE
0 = SMALL PAGES
1 = LARGE PAGES
63

HALF WORD BIT MAP: The entire drop map is deivided into 170 full word entries followed by 170 1/2 word entries. Each bit corresponds to each of 31 pages in a full word entry. Bit off means page undefined or exists in core/drum system. Bit on indicates page has been written to disc once.

MINUS PAGE WORDS FOR TIME USAGE

Temporary Collection Over Slot Time

146	TPWDS 16	DISCACC 12	DISCSEC 12	CDPGMILL (TMEMC) 24	
147	SYSCHG 16	TPHOOK 8	TPACC 8	▨ 8	CPUTIME (TCPUC) 24
148	DRACC 16	TREMIO TI 16	TIMIO 16	TEXIO 16	
149	DRACCO 16	LASTUP 48			
150	DRACCD 16	FILTP 8	TPFUNCT 8	▨ 8	CDPAGES 24

The following fields are entered into a time usage entry each "SLOT" time: TPACC, DISCACC, DISCSEC, FILTP, TPFUNCT, TPHOOK, TPNDS, CPUTIME, CDPGMILL, SYSCNG, DRACC, DRACCO, DRACCD, and CDPAGES.

The following fields are entered (i.e. summed) into the execution collection of usage charges: CPUTIME, CDPGMILL, SYSCHG, TREMIO, TIMIO, and TEXIO.

EXECUTION COLLECTION OF CHARGE OF TIME USAGE

140	PGFLT 16	CPUCHG 48
141	DRFLT 16	MEMCHG 48
142	EXIO 32	REMIO 32
143	IMIO 32	SYSCHG 32

WHERE:

CPUCHG = CPUTIME * CPFACT
MEMCHG = CDPGMIL * MEFACT
EXIO = TEXIO * EXFACT
IMIO = TIMIO * IMFACT
REMIO = TREMIO * REFACT
SYSCHG = SYSCHG * SYFACT
PGFLT = # page faults total
DRFLT = # page faults found on drum

NOTE: The multiplying factors above are percentages from 0-100 which may be changed dynamically.

MINUS PAGE WORDS FOR TIME USAGE, cont'd.

TIMIO Implicit I/O charge.
Rate 50 millisec/access+ 1 millisec/sector

a) For DISC WRITES initiated due to closing a file, mapping out a file, advise out a page(s) and job termination.

b) For DISC READS initiated due to demand page faults or advise in page(s).

TEXIO Explicit I/O charge.
Rate 50 millisec/access+ 1 millisec/sector for all disc I/O explicitly stated by user.
Rate 8 millisec/access+ 1 millisec/sector for all tape I/O read and writes.
Rate 10 millisec/tape function

This is to include processing of printer, HSP and DD80 files by special system routines.

TREMIO Remote I/O charge
Rate some fee/data burst to mass store and TMDS
Rate disc I/O rates to process remote printer files plus some charge for printer use (?)

MEMCHG Contains that portion of the core/drum storage usage accumulated during users occupation of an alternator slot. That portion of CDPGMILL attributable to users INACTIVE state will be recorded in time usage record (along with active state data), but NOT be charged against bank account.

BANK ACCOUNT DECREMENT - only the sums of those fields recorded in minus words 140-143. Every job will have these sums weighted by 1,except for standby jobs whose weighted 'priority' will be small, perhaps 10%. Note that this does away with the current 'priority' weighting scheme.

OUTPUT FILES FOR USER 1

These are files which the user wishes to have processed by a system user for output to the line printer, high speed printer tape, DD80 plot tape or punch tape. The user must issue the Give File system call (#08) to give the file to user number 999999 or must have created them as type output. Constraints on the name of an output file exist and are defined below:

1. Files scheduled for line printer output must have names beginning with the letter p or P. The names need not be a full word in length.

2. Files destined for the off-line printer (high speed tape) must have names beginning with the letter h or H. The names need not be a full word in length. A provision for saving some number of high speed files for consecutive processing exists. This is the "family" concept currently available on other Laboratory systems.

   The second character must be a numeric sequence number in the range ∅ - #F. The name must be eight characters (full word). This "family" of files will be held in the output file chain until the family file name with the second character being x or X arrives, or until the family ages to some limit, at which time the entire family is output. Files for high speed tape which are not recognized as members of a "family" will be processed at once.

3. Files destined for the DD80 plot tape must have names beginning with the letter d or D. All such names must be eight characters in length. The family concept as in 2 above will be effective.

4. Files destined for the card punch tape must have names beginning with the letter b or B. All such names must be eight characters in length. No family grouping is possible. Each file is processed when received.

The output processing programs run on demand by the Give File call. The appropriate processors for printer files, punch files and non-family high speed printer files are initiated immediately after the Give File call. Members of families of files are stacked and no processor runs until an end-of-family name is recognized, at which time the appropriate processor is initiated.

Files processed for the line printer are handled by a program executing under user number ØØØØØ1, but the user gives the file to user 999999. The system recognizes the situation and switches the file to the user ØØØØØ1 chain.

File Ownership

Each private disc file cataloged in the system is recognized as belonging to some user number, some division code, and some account designator. When a file is given from one user to another, the user number and division code change, but the owner account stays fixed until the recipient user references the given file the first time. Then the account designator in force at the first reference to the file replaces the former account. An entry is made in the system accounting table at this point, indicating the total time of ownership of the file under the originating account designator. The account designator used for ownership liability is the alpha portion of the usual Laboratory effort-account number.

File Activity

A file is considered active if some program, active in the system, has the file open, i.e., at least one of the program's IOC entries point to the file.

A file may be destroyed by a program if the file activity counter (a count of the number of IOC's currently pointing to the file) is zero or one. If it is one, the active IOC must be in the minus page of the program requesting the file destroy.

A file may be given to another user only if its activity counter is zero. This means, in the jargon, that the file must be closed.

A program is allowed to open the same disc file in as many of his IOC's as he wishes. Each open call will result in the file activity counter being incremented.

When a program is terminally dumped to disc, any active IOC's are examined, and if they point to a disc file, the appropriate activity counter is decremented.

Finally, for statistical purposes, a reference counter is maintained in each file index entry which is a running sum of IOC's which have been connected with the file.

STAR RECORD STRUCTURED FILES

SCOPE

This document pertains primarily to files from the card reader and files destined for various printing devices. The format of these files will be discussed along with the control characters and their applications.

COMPRESSED ASCII

Explanation - this term refers to the form lines that are stored within a record structured file. These lines can be either a card image or print line and are comprised of 8-bit ASCII.

> ASCII - this term refers to an 8-bit ASCII. There are 256 possible characters within this character set. See table at end of writeup.
>
> Line - a line can be any length. Current printers are, however, limited to 12∅ characters. All lines are ended with the control character "US."
>
> Blanks - all blank fields of larger than 2 characters are compressed. The control character "ESC" followed by the number of blanks in the field denotes the blank field. An ASCII "∅" is always added to the blank count; the reason for this is to remove the blank count out of the range of the control characters.

Control characters. A detailed explanation.

ASCII	HEX	DEC	OCT	USE...
NUL	ØØ	ØØ	ØØ	Padding - used to round out to a desired boundary.
EOT	Ø4	Ø4	Ø4	Physical end of media - last character in file. Only passing, pointers and ID may follow.
FF	ØC	12	14	Top of form - appears in print files. The following line will be at the top of the next page.
SO	ØE	14	16	Mode change - the next whole word is the start of a binary field.
ESC	1B	27	33	Compressed blanks - the following 8-bit character denotes count of blanks plus ASCII Ø.
FS	1C	28	34	File separator - equivalent to LRL end-of-file.
GS	1D	29	35	Group Separator
RS	1E	3Ø	36	Record separator
US	1F	31	37	End of line - this character is at the end of every ASCII line.
Ø	3Ø	48	6Ø	ASCII zero - added to blank count.

BINARY

Explanation - this term refers to binary card images contained in a file. The card is in two parts. Control and content. Only the content is put into the file. The control uses the first 48 bits of the card.

Control bits. A detailed explanation.

Bits ØØ-Ø7 Byte count. Number of 8-bit bytes starting in column 5.

Bits ∅8-11 Denote a binary card. This field = ∅1∅1. If EOF card, then this field = 1111.

Bits 12-23 Sequence number. 1st card = ∅∅∅∅∅∅∅∅∅∅∅1.

Bits 24-47 Checksum. 24 bit arithmetic sum of the 8-bit data bytes.

Content. The card may contain up to 114 data bytes. The binary information in the file may be thought of as absolute column binary.

PRINT FILES

Explanation - a print file is a file prepared within the STAR for some external hardcopy device. It is comprised of compressed ASCII. The last four 64-bit words are the usual ASCII ID information. The last ASCII character within the file, exclusive of ID, must be an "EOT."

CARD FILES

Absolute column binary - specified by an "A" in column 3∅ of the ID card. This format treats the card as a binary bit string. Each card puts 15 64-bit words into the file.

Mixed mode - specified by a blank in column 3∅ of the ID card. A mixed mode file consists of compressed ASCII and binary in any mix.

Format of a mixed mode file. In order to determine mode changes within the file, a pointer field is used. The pointer field is at the end of the file between the last data and the ID information. The logical address of this field may be found by looking at the 5th word from the end of the file. Please note that mode changes must start at 64-bit boundaries. (See 3.6.5 for example of mixed file.)

POINTER WORD FORMAT

LOCATION OF NEXT POINTER WORD		MODE	LOGICAL ADDRESS

Bits ØØ-15 Location of next pointer word. This number added to address of pointer field points to next pointer word.

Bits 16-31 Unused.

Bits 32-39 Mode of block. 01/02/03/04/FF = compressed ACSII/ binary/record separator/group separator/last pointer word

Bits 40-63 Logical address of specified block.

<u>Example of a mixed mode deck</u>

Card 1:	C
Card 2:	C^^^^^3^BCD^CARDS.
Card 3:	ABC
Card 4:	BINARY CARD-FULL CARD-114 BYTES
Card 5:	BINARY CARD-86 BYTES
Card 6:	*^^^^^DATA

WORD								8 BITS	
∅	C	US	C	ESC	35_{16}	3	□	B	COMPRESSED ASCII
1	C	D	□	C	A	R	D	S	
2	•	US	A	B	C	US	SO	NUL	
3	BYTE 1 BIN CD 1	•	•	•	•	•	•	•	BINARY
⋮									
11	•	BYTE 114 BIN CD 1	BYTE 1 BIN CD 2	•	•	•	•	•	
⋮									
1B	•	•	•	•	•	•	•	BYTE 36 BIN CD 2	
1C	*	ESC	35_{16}	D	A	T	A	US	COMPRESSED ASCII
1D	EOT	NUL	NUL	NUL	NUL	NUL	NUL	NUL	
1E	∅ ∅ ∅ 1				∅1	∅ ∅ ∅ ∅ ∅ ∅			POINTER FIELD
2∅	∅ ∅ ∅ 2				∅2	∅ ∅ ∅ ∅ ∅ 3			
21	∅ ∅ ∅ 3				∅1	∅ ∅ ∅ ∅ 1 C			
22	∅ ∅ ∅ ∅				FF	∅ ∅ ∅ ∅ ∅ ∅			
23						∅ ∅ ∅ ∅ 1 E			LOGICAL ADDRESS OF POINTER FIELD
24 25 26 27	32 CHARACTER ASCII ID.								

Record separator and group separator cards have 7-8 - 9 and 6-7-9 punches respectively in column 1. Record and group separator cards will cause map entries with modes 3 and 4 respectively. The contents of the card will be considered to be an ASCII record and will be placed in the file accordingly.

STAR Pool Files

The STAR operating system will offer the pool file concept for sharing files in a manner somewhat similar to the 7600 implementation. This involves the appointment of some subset of users as pool "bosses." A pool boss will be allowed to have a list of private files as usual, but, in addition, will be allowed to have a second list of files which will be considered pool files. A pool file is one which is under the direct control of the pool boss in matters of integrity and disposition. The pool file may be accessed by other users if they are currently connected to the pool. The non-pool boss user may not alter the content of the pool file on disc. That is to say, he may have only read/execute access to a pool file. The rule applies regardless of the file's access type. Of course, the write-temporary definition is available. Only the pool boss may write into a pool file.

Each user in the system may have a list of pool bosses to which he may attach himself. In order to use a pool file list, the user must first attach himself to the pool list. This is accomplished by a type-in on the user's terminal:

CONTROL-E POOL POOLNAME

where POOLNAME is the name of the file pool to which the user wishes access. Pool names may be up to eight alphanumeric characters. A user may attach to as many as four independent file pools at the same time. A pool boss is automatically attached to his pool file list, if any. The pool boss may also be eligible to attach to other pools. The specification of pool bosses and pool members is handled through Computation Department administration and can be periodically updated.

When a user, who is attached to file pools, references a file, a search is made 1) of his private file list; 2) of his pool file list(s) in the order in which he attached to them, and 3) of the public file list. Note that a pool boss actually has two independent file lists under his control and, as such, can be in control of two files of the same name and user number. However, in such a case, he couldn't reference the file in the pool list since the private list is searched first. Clearly, the pool boss may be boss of only one file pool.

If a user, who is eligible to use some pool, wishes to place one of his private files in the pool list, he may do so through an option in the GIVE Files System call. The user need not be attached to the pool at the time.

To break an established connection with a pool the users type in:

CONTROL-E POOL -<u>POOLNAME</u>

where the minus sign preceding the <u>POOLNAME</u> indicates release from the pool.

The user may list the names of the pools in the order in which he is attached to them by typing:

CONTROL-E LP

The list will be printed at the user terminal. If the user logs off the computer and leaves no program active, the connection to all pools is severed.

This implementation allows for shared files on a need-to-know basis rigorously controlled by laboratory administration.

USER CALL MESSAGE FORMAT

All user calls, whether or not they are for the KERNEL, are issued by an exit force to the KERNEL. Immediately following the exit force instruction in the instruction stream is a 32-bit instruction with its upper 16 bits #ØØEE or a 64-bit instruction with its upper 16 bits #ØØFF. In the first case the low order 8-bits of the instruction contains a full word register designator and the designated register contains the virtual bit address of the first full word of the message. In the second case the low order 48-bits of the instruction contain the virtual bit address of the first full word of the message.

The message itself consists of Alpha and Beta portions. The Alpha portion has the same general form for all calls. The Beta portion has a format dependent on the individual service required. See the section on individual function codes for Beta format specifications.

Good call returns are back to the instruction following the #ØØEE or #ØØFF instruction which may be another #ØØEE or #ØØFF instruction if chained calls are desired.

Alpha and Beta words must occur on full word boundaries and may not exist in the user's page Ø. They must exist in virtual space with read write access and they may not cross large page boundaries.

THE FORMAT OF THE ALPHA PORTION OF THE MESSAGE IS:

<table>
<tr><td>R 16</td><td>L 16</td><td>M 8</td><td>C 8</td><td>FRE 8</td><td>F 8</td><td>ALPHA (1)</td></tr>
<tr><td>N 16</td><td colspan="5">E EA 48</td><td>ALPHA (2)</td></tr>
<tr><td>BL 16</td><td colspan="5">BA</td><td>ALPHA (3)
(Optional)</td></tr>
</table>

R Response code filled in by the system when the call completes. A zero value indicates good completion. See P. 4.1.3 for non-zero values common to all system calls and for meaning of other non-zero values, see writeups of individual calls.

L Is the length of the BETA Buffer in full words if L =#FFFF. For this case, the BETA words are assumed to immediately follow ALPHA + 1 and word ALPHA + 2 does not exist. If L =#FFFF, word ALPHA + 2 exists and contains the BETA descriptor.

M Option

C Control option

FRE Reserved for future use.

F Function code specifying what function is to be performed to satisfy this call.

N Sub-function code whose usage is dependent on the primary function code.

EEA Is the virtual address to which control will be sent for R ≠Ø. EEA =Ø will be considered fatal error.

.BL If L=#FFFF, ALPHA + 2 must exist. .BL is then the length of the BETA buffer in full words.

BA If L=#FFFF, .BA is the virtual bit address of the first full word of the BETA buffer. Note that in this case, the ALPHA and BETA areas need not be contiguous.

ERROR RESPONSES COMMON TO ALL SYSTEM CALLS

#212 No ALPHA Pointer (fatal)

#213 ALPHA out of bounds, that is, ALPHA bit address is greater than 2^{47}-1. (fatal)

#215 UEEA = Ø (fatal)

#270 ALPHA read only (fatal) or BETA read only (non-fatal)

SYSTEM CALLS

This is a current list of system calls which will be available in initial operating system.

Generally, function codes in the range #Ø - #F involve disc or tape access, #10 - #1F manage message traffic, #2Ø - #2F are miscellany, #50 - #52 are explicit I/O functions.

#Code	Function
01	Create disc files or tapes
02	Destroy disc files or tapes
03	Open disc files
04	Map-in, Map-out virtual address space
05	Close disc files
06	Terminate this program with or without dump to disc
07	Advise for page pre-load, free space or page release
08	Give disc files to another user
09	List private or public file index
0A	Cutback length of a disc file
0B	Change a disc file name
0C	Give tape access to controllee
13	List controllee chain
14	Send a message to controller
15	Send a message to controllee
16	Get message or symbols from controller
17	Get message or symbols from controllee
18	Message control
19	Write all controllee pages to disc
1A	Send a message to operator's console
1B	Initialize or disconnect a controllee
1C	Problem Program Interrupt
23	User Directory Modification
24	Miscellaneous, e.g., modify T1, Pr, get PP info, etc.
25	Suspend PP for a time period
50	I/O Call
51	Release interrupt
52	Give up on I/O

SYSTEM CALL Ø1 - CREATE

The CREATE call will be issued by a problem program to reserve, and attach itself to, a hitherto undefined I/O medium.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		

BETA (1)	NAME (ASCII)≠ 64							
BETA (2)	IOC 8*	DEVICE 8	TYPE 8*	LOK 8	ACS 8*	MODE 8*	CLASS 8*	UNIT 8
BETA (3)								SS 8
BETA (4)	LENGTH 16 ≠	BASE VIRTUAL ADDRESS 48						

* set by system for device = 5 or 6
≠ not required for device = 5 or 6 but may be used

R — Response Code
Ø = Normal
1 = Error (see SS)
#211 = Error (N=Ø)

L =#FFFF — Left 16 bits of word ALPHA (3) equal length of remote BETA buffer. Right 48 bits of word ALPHA (3) equal location of remote BETA buffer.

≠#FFFF — BETA buffer immediately follows word ALPHA (2) and contains L words.

C — not used.

F — Function code = Ø1 for CREATE

N — Number of creates in this call (16 max.)

NAME — ASCII Device Name - not to exceed 8 characters for disc or 5 characters for tape.

IOC — IOC number for connection (Ø-15).

SYSTEM CALL Ø1 - CREATE cont'd.

DEVICE		Ø =	Disc File (private, read/write access)
		1 =	Scratch file (Disc file may be destroyed by system upon completion of job).
		2 =	Output file (Disc file may be processed for output upon completion of job).
		3 =	Write - temporary disc file.
		4 =	Tape drive
		5 =	Disc drop file (Destroy any existing file with name same as BETA (1)).
		6 =	Disc drop file (Notify, i.e., set SS and take error return, if a file with name same as BETA (1) exists).
TYPE	=	Ø	Sequential (to File Index).
		1	Virtual Data
		2	Virtual Code
		For Device	= 4
		7	Seven Track tape
		9	Nine Track tape
LOK	=	1	Execute lockout protection (to IOC and File Index)
		2	Write lockout protection
		4	Read lockout protection
		Ø	No protection
ACS	=	1	Write access (to File Index and IOC)
		2	Read access
		4	Execute access
MODE	=	Ø	Sequential (to IOC)
		1	Virtual Data
		2	Virtual Code
CLASS			Security access code (to File Index)
UNIT			Logical disc unit on which file is to exist. System

<u>SYSTEM CALL Ø1 - CREATE cont'd.</u>

		returns actual unit used. (To File Index and IOC).
SS		Error Code
	= Ø	Normal completion

SYSTEM CALL Ø1 - CREATE cont'd.

= #1	File already exists
= #2	No disc space
= #3	Illegal device number
= #4	Parameter or format error
= #5	Operator - initiated tape error
= #6	IOC in use
= #7	File index full
= #8	STANDBY job trying to create tape
= #9	-unused-
= #A	SS field was preset
= #B	Can't destroy existing drop file
= #C	Drop file name already exists
= #D	(For device = 5,6) Some pages have been written to existing drop file. Won't create a new one.
= #E	(For device = 5,6) Drop file length won't hold space already in drop file map.
= #15	Virtual map in minus page is full
= #17	Virtual address overlap in virtual map

LENGTH — Number of 512-word sectors (small pages) this file (to File Index & IOC).

BASE VIRTUAL ADDRESS — Is the virtual address corresponding to the first physical word of this file. (to File Index & IOC).

Note that BETA + 2 is not needed for tape create. However, for multiple creates in a single call, all three BETA words must be provided for each request.

Purpose and Operation:

The CREATE call is generally issued by a problem program to attach itself to a logical tape drive, which is assigned only to the one program. Also, the program may reserve disc space under a logical name which will be assigned to the catalog of files under the user ID number associated with the calling program. Additionally, for tape or disc, an IOC is filled in attaching the program to the CREATED device to allow I/O from/to that device.

SYSTEM CALL Ø1 - CREATE cont'd.

For a tape create, the system tape assignment table is examined for possible existence of the tape. If it is not in the table, a message is sent to the operator TTY requesting the tape. The program P-counter is set to re-issue the call and the program is dumped to disc. When the operator assigns the tape, the program is reactivated, issues the tape create call and now gets connected to the tape.

Disc file creates are immediate.

The operating system also uses the routine to CREATE disc to effect creation of an automatic drop file for new execute lines. The file is created with a recognizable file name and at the length of the source file or a pre-specified length. The program minus page is loaded to a system table and virtual maps verified.

The using program may create a new drop file to override the automatic one, if no pages have been written to the existing drop file.

Drop files are created with Read and Write Access, and at the access level at which the program is operating.

SYSTEM CALL Ø2 DESTROY

This call will be issued by a problem program to sever its connection with a tape drive and release the drive for re-assignment, or to sever its connection with a disc file and release the disc space for re-assignment.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		

BETA (1)	NAME (ASCII) 64			
BETA (2)	IOC 8	DEVICE 8	FREE 40	SS 8

R — Response Code
= Ø Normal
= 1 Error (See SS)
= #211 Error (N=Ø)

L = #FFFF — Left 16 bits of word ALPHA (3) equal length of remote BETA buffer. Right 48 bits of word ALPHA (3) equal location of remote BETA buffer.

≠ #FFFF — BETA immediately follows word ALPHA (2) and contains L words.

C — Not used.

F — Function code = Ø2 for DESTROY.

N — Number of requests in this call (16 max.)

NAME — ASCII device name as assigned by user in either CREATE or CHANGE call-not to exceed 8 characters for disc files, or 5 characters for tapes.

IOC — For tape destroy, calling program specifies IOC. For disc file destroy, this field will not be examined, but will be filled in by the system with the inclusive OR of all IOC numbers found to contain the file connection of the file being destroyed.

SYSTEM CALL Ø2 DESTROY cont'd.

DEVICE		Ø = disc file 4 = tape drive
SS		Error code
	=0	Normal completion
	=1	File name does not exist
	=2	Tape name mismatch between BETA (1) and IOC
	=3	Some other active problem program has the file open
	=4	Format or parameter error
	=5	Tried to destroy source or drop file
	=6	Illegal device number

Purpose and Operation

The DESTROY call is generally issued by a problem program to release a storage device for re-assignment by the system. These are, currently, tape drives and disc files.

In the case of a tape drive, the system expects an IOC defining the connection to exist. The system will erase its tables of tape-PP correspondence and will erase the IOC in the PP minus page.

Disc files need not have IOC's representing a connection. The fact of existence in the File Index is sufficient. A file need not be opened in order to destroy it. If it is open, however, it may be open only in one IOC and that must be an IOC of the calling program. The system will erase the IOC, erase the memory maps in the minus page corresponding to that IOC and erase representative entries from the core page table and drum page table. Hence, all virtual space connected with the destroyed file is available for re-definition.

If the disc file was classified at a level greater than PARD, it will be overwritten with disc pattern.

SYSTEM CALL Ø3 - OPEN

The open call will be issued by problem program to attach itself to an already existing disc file.

WORD:

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS (48)		

BETA (1)	NAME (ASCII) 64							
BETA (2)	IOC 8	MAPIN 8	TYPE 8	LOK 8	ACS 8	MODE 8	CLS 8	UNIT 8
BETA (3)				OPT 8	OWN 8	ST 8	W 8	SS 8
BETA (4)	LENGTH/SECTOR 16			WORKING VIRTUAL ADDRESS (BIT) 48				
BETA (5)	BUFFER/LENGTH (WORD) 16			BUFFER VIRTUAL ADDRESS (BIT) 48				

R — Response code = Ø Normal
= 1 Error (see SS)
= #211 Error (N=Ø)
= #214 Error (BETA Bounds)

L = #FFFF — Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ #FFFF — Beta buffer immediately follows word ALPHA (2) and contains L words.

C — not used.

F — Function code = Ø3 for OPEN.

N — Number of opens in this call (16 max.)

NAME — ASCII file name (8 characters max.)

IOC — IOC number for connection (Ø-15).

SYSTEM CALL Ø3 - OPEN cont'd.

MAPIN* =Ø use virtual address information in file index and as specified in minus page existing with file. IOC and map filled in by system.

≠ Ø give existing maps to calling program in buffer specified in word BETA (4) ... note that BETA (4) is not otherwise required. IOC filled in by system.
No map entries are generated.

* Not examined for sequential file being opened virtually.

OWN =Ø Private file
=1 Public file filled by system
=2 Political file

TYPE Expansion of TYPE field:

C_1	D	C_2	T
(1)	(3)	(1)	(3)

$C_1 = Ø$ open as specified by call but don't change file index type.

$C_1 \neq Ø$ open as specified by call and change file index type to that shown in T field.

D how system is to consider this file (copied to PRINCIPAL TYPE field in IOC):

D = Ø open as normal disc file

1 open as a "scratch" file (system may destroy at ALL DONE)

2 open as "output" file (system may process at ALL DONE)

3 open as "write temporary" (any modified page will be sent to drop file. Source not updated.)

$C_2 = Ø$ open as specified by call but don't change file index access.

SYSTEM CALL Ø3 - OPEN cont'd.

C_2 = 1 — open as specified by call and change file index access and lockout to that given in field ACS and LOK.

T = Ø — sequential (no minus page assumed)

= 1 — Virtual data (minus page assumed)

= 2 — Virtual code (minus page assumed)

T Field, if not specified, will be filled in by system.

LOK — Lockout protection. If file being opened is PUBLIC, this information will be taken from the FILE INDEX.

= Ø — Get from FILE INDEX
= 1 — Execute Lockout
= 2 — Write Lockout
= 4 — Read Lockout
} (to IOC)

ACS — ACCESS rights. If file being opened is PUBLIC, this information will be then taken from the FILE INDEX.

= Ø — Get from FILE INDEX. (RETURNED BY SYSTEM IN ACS FIELD
= 1 — Write Access
= 2 — Read Access
= 4 — Execute Access
} union of bits allowed
} (to IOC)

MODE — This field must always be specified. (to IOC)
= Ø — open sequential (no minus page assumed)
= 1 — Virtual data (minus page assumed)
= 2 — Virtual code (minus page assumed)

W ≠ Ø — System will use virtual address from FILE INDEX and return it in the WORKING VIRTUAL ADDRESS field. Examined only for Sequential File being opened in Virtual Mode.

= Ø — System will use virtual address specified in WORKING VIRTUAL ADDRESS.

OPT — Bit Ø = preload, Bit 1 = kill (see pg. 3.4.8)
Bits 2-7 undefined.

SYSTEM CALL Ø3 - OPEN cont'd.

UT — Logical disc unit on which file exists. (Always filled in by system.)

ST — File Subtype - 0 = Normal
1 = Scratch
2 = Output
3 = Write Temporary
5-6 = Drop File
7 = Batch input

SS — Error Code.

= Ø normal completion

= 1

= 1 No name given or name not in FILE INDEX

= 2 No ALPHA word pointer or zero requests specified in ALPHA (2).

= 3 Virtual map overlap

= 4 IOC already in use

= 5 Illegal Type, Lok, ACS

= 6 Disc station error reading minus page

= 7 Virtual map full

= 8 Public or Pool Access Code greater than operating access code.

LENGTH — Number of 512-word sectors (small pages) comprising this file. (Always filled in by system).

WORKING VIRTUAL ADDRESS — is the virtual address to correspond with the first physical word of the file.

BUFFER LENGTH
BUFFER V. A. — If user wishes the file maps delivered to his program space and not mapped in. These specify the length and location of that buffer space where the system is to store the maps.

Purpose and Operation

Generally, to be able to accomplish any I/O, either implicit or explicit, a program must be attached to that I/O device through an IOC. The OPEN call is the mechanism by which a program attaches itself to an existing disc file.

Through this call, the IOC may be filled in from known information regarding the file or with selected options according to the user's wishes. The calling program may even permanently alter the type and access rights of his file through this call. Note that no modification or supercession of public

SYSTEM CALL Ø3 - OPEN cont'd.

file lockout, classification or access is allowed. Further, if a file is opened with the attribute "write temporary," the system will not allow the write access bit in the virtual map to be on, so the user need not declare it.

The calling program may use the file with all its attributes as is or may modify them for the time the file is open. File maps defining virtual space may be superceded even for PUBLIC files, for the duration of the IOC. The program may elect to look at those maps and re-define virtual space through MAP-IN calls.

If the calling program requests the virtual maps be delivered to its space, it must provide a BETA buffer of sufficient length to hold the entire map space. The system will store entries only until the buffer is full, however. But, no effort is made to squeeze out zero entries, so the user should always take the entire map. Currently, this is 40 entries or 80 words. These will be delivered to the program as contiguous 2 word entries. Note that this is a different format from that of the minus page which is made up of a 40 word table of first word entries and a 40 word table of second word entries.

The map format appearing in the user's BETA buffer will be:

<table>
<tr><td rowspan="2">Entry I</td><td>VOID
17</td><td colspan="4">VIRTUAL PAGE ADDRESS
32</td><td>NOT USED
15</td></tr>
<tr><td>PHYSICAL DISC ADDRESS
18</td><td>LENGTH (SECTORS)
16</td><td>IOC PTR
5</td><td>UNT
3</td><td>CONTROL
6</td><td>LOGICAL DISC ADDRESS
16</td></tr>
<tr><td rowspan="2">Entry 2</td><td colspan="6"></td></tr>
<tr><td colspan="6"></td></tr>
</table>

etc.

A program may open a "sequential file" in the "virtual" mode.

Constraints: contiguous virtual space is assumed; working virtual

SYSTEM CALL Ø3 - OPEN cont'd.

address must be given; only one entry will be made in the virtual space map; three BETA words will be assumed.

Note that whenever the FILE INDEX TYPE is changed by this call, the new TYPE is assumed to prevail for all the various functions of the call.

Trying to open a file at a level higher than operating results in SS=5 Error if the file space is declared to be write temporary or if read access is requested.

SYSTEM CALL Ø4 - MAP

The MAP call will be issued by a problem program to gain access to certain virtual space by relating that space to some area of an already opened disc file. Free virtual space, i.e., address space not bound to any disc file may be appended by this call. Release or redefinition of virtual space is also provided.

WORD ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		

BETA (1)	VIRTUAL PAGE ADDRESS 32		LOGICAL DISC SECTOR ADDRESS 32		
BETA (2)	LENGTH (SECTORS)16		IOC 8	CONTROL 8	SS 8

R — Response Code = # Ø Normal
= # 1 Error (see SS)

L = # FFFF — Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF — BETA buffer immediately follows word ALPHA (2) and contains L words.

C — Option Field

Ø= — MAP IN

1= — Complete Mapout

2= — Drop File Map-out only

F — Function code = Ø4 for MAP call

N — Not used

VIRTUAL ADDRESS — The first word virtual small page address of the space being defined.

SYSTEM CALL ∅4 - MAP cont'd.

LOGICAL DISC ADDRESS — The logical sector address within a disc file associated with the VIRTUAL ADDRESS above. If this field equal #FFFF, free space, as defined by VIRTUAL ADDRESS and LENGTH, will be appended.

LENGTH — The number of contiguous virtual address sectors (small pages) being defined (must be contiguous in the disc file if not a free space call).

IOC — A pointer to the IOC which defines the disc file being mapped. If the call is for a free space map-outor map-in, IOC=17 must be specified. If the call is for a source file map-out, IOC=16 must be specified.

CONTROL

1	2	3	4	5	6	7	8
C 1	C 2	Sm/Lg page	PRELOAD	KILL	C W A	A C	

C 1, C 2, not used.

Small/Large page bit = ∅ for small

= 1 for large.

PRELOAD bit will be copied to virtual map.

KILL = 1 kill these pages in memory upon map out,

= ∅; dump pages to disc

WA = ∅ Get access rights for virtual map entry from IOC.

= 1 Get access rights from AC.

AC = ∅ No Read or Write Access

= 1 Read Access

= 2 Write Access

= 3 Read/Write Access

Error field

= # ∅ No error

= # 1 virtual address overlap in bound virtual map

= 2 not used

= 3 map-out LENGTH greater than length in map

= 4 sector count not mod 128 for large page call

= 5 no IOC

SYSTEM CALL Ø4 - MAP cont'd.

Error field	= 6	virtual address same as existing advise call
	= 7	bound virtual map is full at map-in
	= 8	logical disc address and length is greater than disc file length
	= 9	A page requested for map out is locked down
	= A	Space undefined at map-out
	= B	MAP entry is large page, given virtual address is not
	= C	bound virtual map full at map-out
	= D	Wrong IOC number for Free Space grab
	= E	Free space map is full at map-out
	= F	Drop file of insufficient size to hold free space MAP-IN
	= 10	Can't find disk file index at map-in
	= 11	Virtual address overlap in free space map

The ERROR EXIT address will be executed for any SS≠Ø.

Purpose and Operation:

Generally, to define some virtual address range, previously undefined, as being part of program space.

Defining bound virtual space means associating some virtual address range with some physical disc range, on a contiguous word-to-word basis, in an already-open disc file. This might be the source code file itself or some other virtual data file.

Defining free space means appending some virtual address range to program space. Free space is not considered to be associated with any existing disc file, however, there must be sufficient space in the drop file to contain all defined free space and all modified pages that are not associated with virtual data files.

For any map-in there must exist sufficient room in the pertinent virtual space map to make a new entry. For bound virtual space, a new entry is required

SYSTEM CALL Ø4 - MAP cont'd.

for each virtual address discontinuity or change in IOC number or in access rights.

Mapping out virtual space means to release a virtual address range from defined program space and to make available the drop file space that represents it. The mapped-out range becomes eligible space for a map-in.

Mapping out space associated with a write-access virtual data file will cause any modified pages, in the map-out region, to be written to the parent file.

Mapping out any other virtual space, modified or not, causes total loss of all records associated with the space. No image is copied to any disc file. The space is no longer defined in any minus page map. Any previous image of that space which may have existed in the drop file is irrecoverable. The drop file disc space is available for re-assignment.

SYSTEM CALL Ø5 - CLOSE

The CLOSE call will be issued by a problem program to sever its connection with a disc file, but leave the file in existance. Modification of some File Index attributes is allowed.

ALPHA	(1)	R 16	L 16	C 16	F 16
ALPHA	(2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		

ALPHA BETA	(3) or (1)	IOC 8	DVC 8	TYPE 8	LOK 8	ACS 8	FLAG 8		SS 8
BETA	(2)	LENGTH 16		BASE VIRTUAL ADDRESS 48					

R — Response code = Ø Normal
= 1 Error (see SS)
= #211 Error (N=Ø)

L = # FFFF — Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF — BETA buffer is immediately follows word ALPHA (2) and is L words long.

C — not used.

F — Function code = Ø5 for CLOSE.

N — Number of CLOSES in this call (16 max.)

IOC — IOC number (Ø-15) of IOC to be closed.

TYPE — Expansion of TYPE field:

C_1	C_2	C_3	C_4	T
(1)	(1)	(1)	(1)	(4)

SYSTEM CALL Ø5 - CLOSE cont'd.

C_1 = Ø — Close file with no change to File Index file type.

C_1 = 1 — Close and change type in File Index to that given in field T.

C_2 = Ø — Close file with no change to File Index file access and/or lockout.

C_2 = 1 — Close file and change access and lockout in file and lockout in File Index to those given in LOK and ACS. (Note that lockout information is not currently being used).

C_3 = 1 — Install small page length in LENGTH into File Index. This will be drop file size for the code.

C_4 = 1 — Remove drop file length

T = Ø — Sequential data (no minus page assumed)

= 1 — Virtual data (minus page assumed)

= 2 — Virtual code (minus page assumed)

LOK — Lockout protection. (Currently not honored.)

= Ø — None

= 1 — Execute lockout

= 2 — Write lockout

= 4 — Read lockout

ACS = 1 — Write access

= 2 — Read access

= 4 — Execute access

*SS — Error code

= Ø — Normal completion

= 1 — Non-disc IOC (IOC not erased)

SYSTEM CALL Ø5 - CLOSE cont'd.

	= 2	IOC number out of range (IOC not erased)
	= 3	Access denied (try to modify a Public File Index entry). For SS-3 the file will be closed, i.e., the IOC is cleared but any changes indicated will not be accomplished.
	= 4	TYPE, LOK, or ACS value illegal
	= 5	A page of the file being closed is locked down.
FLAG	= Ø	Ignore BETA (2)
	= 1	Reset Base VIRTUAL ADDRESS in File Index to that given by BASE VIRTUAL ADDRESS.
	= 2	Change device type per DVC field.
	= 3	Both 1 & 2 are to be done.
DVC	= Ø	Normal disc file
	= 1	Scratch file
	= 2	Output file
	= 3	Write-temporary file
BASE VIRTUAL ADDRESS		is the virtual address corresponding to the first word of the file.

* Error exist will be taken for any SS=Ø

Purpose and Operation:

This call is generally issued for the purpose of erasing a disc IOC in the program minus page. Erasure of an IOC breaks the I/O connection with the device it represented. Virtual address space associated with the IOC being released is available for re-definition.

The system will accept the call only for disc IOC's numbered Ø through 15. The user may not erase his source or drop file IOC, numbered 17 and 17 respectively. Upon receiving the call, the system will examine and validate

SYSTEM CALL Ø5 - CLOSE cont'd.

the IOC. The file index will be searched for the represented file. If it no longer exists, the IOC will be erased and virtual space released and the normal return will be taken.

If the represented file has write access and the IOC is virtual, modified pages is the core-drum system will be written back to disc before the CLOSE is completed. If the file was write protected, any modified pages in the core-drum system and those represented in the drop file map will be deleted. Hence, closing a file causes erasure of the IOC, virtual maps and core-drum page tables. If the IOC was sequential, it and its accompanying sequential map entry are erased.

Permanent changes to the file index entry are allowed through this call. However, if the file is PUBLIC, privileged access rights must be obtained by the user. Note that all I/O to any disc file will be accomplished before any file index changes are effected. The file index entry will exist in its new state only at completion of the CLOSE call.

SYSTEM CALL Ø6 - TERMINATE

The TERMINATE call is issued by a problem program to signal the system that it has completed execution.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	RESUME ADDRESS 48		

R — Response code (not used)

L — L = Ø

N — (not used)

F — Function code = Ø6 TERMINATE

C — =Ø Erase problem program from core and preserve drop file.

=1 Erase problem program from core and delete drop file.

RESUME ADDRESS — If field is non-zero and (C=Ø, the RESUME ADDRESS will be stored in the program counter field of the IP in the drop file minus page.

OPERATION — All pages belonging to write access files which are open at time of this call will be returned to their disc images (if modified). All other modified pages will be written to the drop file (for (C=Ø).

If (C=1, all "scratch" disc files will be destroyed and all "output" disc files will be processed, it the latter have legal output names.

The program will not regain control after issuing this call.

SYSTEM CALL Ø7 - ADVISE

The ADVISE call might be issued by a problem program to inform the system of an expected need for some virtual space in an attempt to avoid faulting for the space. The space may be bound or free, i.e., associated with some bound virtual file, library space or simply an attachment of hitherto undefined space. The call may also be used to advise the system that it may immediately remove some pages from the core-drum system as the program will no longer use them.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT ADDRESS 48		

BETA (1)	SS 8	PGCT 8	VIRTUAL BIT ADDRESS 48

R		Response code
	= Ø	Normal completion
	= 1	See SS for specific error
L	= # FFFF	Left 16-bits of word ALPHA (3) equal length of remote BETA word. Right 48-bits word ALPHA (3) equal location of remote BETA word.
	≠ # FFFF	BETA follows ALPHA immediately.
C		(not used)

F Function code = Ø7 for ADVISE

N not used

PGCT Page control - expansion:

PIO (1)	PSZ (1)	PN (6)

SYSTEM CALL Ø7 - ADVISE cont'd.

PIO	= Ø	attach or load pages
	= 1	dump pages from core-drum
PSZ	= Ø	small page(s)
	= 1	large page
PN		page count, maximum of 8 for small pages = 1 for large pages
VIRTUAL BIT ADDRESS		Starting address for this call
SS		Error code
	= Ø	Normal Completion
	= 1	Space violates system boundary
	= 2	Page count too large
	= 3	Page is locked down, can't remove it (PIO=1)

Purpose and Operation:

The advise call can be thought of as a pre-load mechanism. If the call is made referencing already-defined virtual space, a message will be sent from the call processor to the page fault processor indicating the address and length of the advised-for space. From then on, things will be treated much as a page fault. If more than one small page is indicated, the pages requested should be contiguous on disc. This will result in only one disc read instead of several. For large page advises, the system will allow only one per call. If the virtual address mentioned in the system call is not defined in any virtual map, it will be considered to be a definition of new free space and an appropriate entry will be made in the drop file map, as well as allocating core space. This is true for any size page.

If the call is to advise the system that some space is no longer required to be in the core-drum system, the system will write all modified pages in the mentioned space back to their appropriate disc file. Unchanged pages or pages with KILL bit on will simply be deleted from the core-drum maps. This option has the effect of increasing available space in the core-drum system.

SYSTEM CALL Ø8 - GIVE FILES

This call will be issued by a problem program to give one or more of its private, inactive files to another user.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT ADDRESS 48		

BETA (1)	NAME 64		
BETA (2)	SS 8		USER 48

R — Response code
= Ø — Normal completion
= 1 — See SS for specific error
= #211 — N = Ø (error)

L = # FFFF — Left 16-bits of word ALPHA (3) equal length if remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF — BETA buffer immediately follows word ALPHA (2) and contains L words.

C = Ø — Give to a private user number
= 1 — Give to pool named in BETA(2) (8 characters maximum)

F — Function code = Ø8 for GIVE

N — Number of files to be given (maximum=16)

NAME — ASCII File name

USER — ASCII user number to which the file is to be given.

SYSTEM CALL ∅8 - GIVE FILES cont'd.

SS	Error Code
#	
= ∅	Normal completion
= 1	File of same name already there
= 2	Name same as Public File name
= 3	No file of given name exists to give
= 4	User number given does not exist
= 5	Output file has illegitimate name
= 6	File is active (outstanding IOCS)
= 7	Specified PUBLIC user number
= 8	Trying to give source or drop file
= 9	Access greater than pard,giving to private (non-pool) user number.
=#A	Illegal Pool (C=1)

Purpose and Operation

The call allows a program to give one of its private files to another user number. The file must be closed, i.e., no IOCS active against it, before giving. Initially, it may be that the receiving user will have to be logged on. Files being given to user 999999 for output processing must have names beginning with one of the characters, P,p,H,h,D,d,B,b. File names beginning with character H,h or D,d will be examined for family membership. The output processor routines will run only on demand. The decision to run them is made by this call processor. Family files will be backlogged until the end name is recognized. Others will be processed at once.

The method of giving a file within the system is to unchain it from the giving user's list of files and to chain it in to the recipients file list.

No file may be given to the PUBLIC list. No file having the same name as a PUBLIC file may be given.

A file which is given will retain, in the file index, the account designator of the originating user until such time as the recipient user references the file. At that time, the recipient user's account designator replaces the original. Hence, the originating user remains liable for any charges against the file until the recipient user uses the file.

Files for output may have names involving upper and/or lower case characters.

SYSTEM CALL Ø8 - GIVE FILES cont'd.

Files for RJET output must have 8 character names beginning with RP or rp. Files for the FR80 must have 8 character names beginning with F or f. Both must be given to user 000002 for processing.

SYSTEM CALL ∅9 - LIST FILE INDEX OR SYSTEM TABLE

This call allows a problem program to get a copy of its private file index list or to get a copy of the public file index list, or certain other system tables.

<table>
<tr><td>ALPHA (1)</td><td>R
16</td><td>L
16</td><td>C
16</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>N
16</td><td colspan="3">ERROR EXIT ADDRESS
48</td></tr>
</table>

<table>
<tr><td colspan="8">FILE NAME
(64)</td></tr>
<tr><td>DEVICE
(8)</td><td colspan="3">DISC SECTOR ADDRESS
(24)</td><td>UNIT
(8)</td><td colspan="3">LENGTH (WORDS)
(24)</td></tr>
<tr><td colspan="5">▨ (40)</td><td>TYPE
(8)</td><td>CLS
(8)</td><td>RW
(8)</td></tr>
<tr><td colspan="4">TORG
(32)</td><td colspan="4">TLR
(32)</td></tr>
</table>

R Response code

= ∅ Normal completion

L = # FFFF Left 16-bits of word BETA (1) equal length of remote BETA buffer. Right 48-bits of word BETA (1) equal location of remote BETA buffer.

≠ # FFFF BETA buffer immediately follows word ALPHA (2) and contains L words.

C Option Control, ∅ = get Public Index
1 = get Private Index
2 = Timecard buffer
3 = Statistics buffer
4 = Bank update table
5 = Miscellaneous Table
6 = Batch input files

F Function Code = ∅9 for LIST FILE INDEX or system table.

N Is the maximum number of file index entries to be delivered. The BETA buffer must have a length at least equal 4*N for file index. For option = 2 through 5, N may be any number specifying program buffer size.

SYSTEM CALL ∅9 - LIST FILE INDEX OR SYSTEM TABLE

NAME	Filled in by system with ASCII File name.	
LENGTH	Filled in by system with the length (measured in 64-bit words) of the file.	
CLS	Security access code	
RW	Filled in by system with the Read/Write access of the file; 1 = write access 2 = read access 4 = execute access	
TORG	Time file was originated	microsecond clock
TLR	Time file was last referenced	shifted right 24 bits
TYPE	Filled in by system with file type: ∅ = Sequential data 1 = Virtual data 2 = Virtual code	
UNIT	Logical disc unit number on which file exists.	
DEVICE	= ∅ Normal, private disc file = 1 Scratch disc file = 2 Output disc file = 3 Write temporary disc file	

Disc Sector Address is absolute sector address at which file begins on disc.

SYSTEM CALL #ØA - RELEASE FILE SPACE

This call may be issued by a problem program to reduce the length of an existing private disc file. The reduction occurs at the largest absolute address end of the file.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT ADDRESS 48		

BETA (1)	FILE NAME 64		
BETA (2)	SS 8	LENGTH (WORDS) 24	

R — Response code
= Ø Normal completion
= 1 See SS for error

L = # FFFF — Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF — BETA buffer immediately follows word ALPHA (2) and contains L words

C — Not used.

F — Function code = #ØA for File Size Cutback.

N — Not used.

NAME — ASCII name of file whose size is to be cutback.

LENGTH — Is the user-supplied new length of the file in 64-Bit words which will be rounded up to the nearest 200 (Hex) by the System.

SYSTEM CALL #∅A - RELEASE FILE SPACE cont'd.

SS		Error code
	#	
	= ∅	Normal completion.
	= 1	New length given is larger than existing length.
	= 2	Name not in File Index.
	= 3	An active problem program has the file open.

The error return will be taken for any SS ≠ ∅ and the file length will not have been changed.

The file being cut back must be completely "closed", i.e. There may be no active IOC's for the file. This includes the requesting program.

SYSTEM CALL #ØB - CHANGE NAME

This call allows a problem program to change the name of an existing private disc file, or to change current account number.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT ADDRESS 48		

BETA (1)	NAME 1
BETA (2)	NAME 2

R — Response code

= Ø Normal completion

= 1 Name error

L = # FFFF — Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF — BETA buffer immediately follows word ALPHA (2) and contains L words.

C — = 0 change file name

= 1 change account number

F — Function code = # Ø B for CHANGE

N — Not used.

BETA (1) (C=Ø) — Old file name (ASCII, right justified)

BETA (2) (C=Ø) — New file name (ASCII, right justified)

BETA (1) (C=1) — New account number

The only errors which might occur are that the old file name doesn't exist or the new file name already exists, or new account number is invalid.

SYSTEM CALL #∅C - GIVE TAPE ACCESS TO CONTROLLEE

A problem program controller may give its controllee access to one or more tapes currently existing in the controller's IOC area.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT 48		

BETA (1)	SS 8	IOC 8		TAPE NAME 40

R Response code

= ∅ Normal completion

= 1 See SS field for specific error

= #211 N = ∅

L = # FFFF Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF BETA buffer immediately follows word ALPHA (2) and contains L words.

C = ∅, Give access

= 1, Recall access

F Function Code = # ∅ C for GIVE TAPE ACCESS

N Number of tapes to be given (16 max.)

TAPE NAME ASCII name under which tape exists in IOC. (5 characters max.)

IOC IOC number for this connection

SS Error Code

#	
= ∅	Normal Completion
= 1	No name given
= 2	Wrong IOC word
= 3	Controllee already owns a private tape in given IOC number
= 4	No controllee

SYSTEM CALL #ØC - GIVE TAPE ACCESS TO CONTROLLEE cont'd.

= 5	Controller doesn't own named tape.
= 6	Controllee IOC already in use.

Operation:

The controllee will gain access to the tape through the same IOC number as the controller uses. The controllee's IOC will not have the tape name filled in, however. So, the controllee will not be able to destroy such a tape.

SYSTEM CALL #13 "LIST CONTROLLEE CHAIN"

This call is used by a problem program to get a list of the CONTROLLEE chain. The list contains the problem program level and descriptor number, source file name, drop file name, and other information.

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>J
8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">BL
16</td><td colspan="4">VIRTUAL ADDRESS OF REMOTE
BETA BUFFER 48</td></tr>
</table>

F — Function code = #13

L = # FFFF — BL' contains the maximum number of words to be returned in the REMOTE BETA buffer. The right-most 48-bits of ALPHA (3) contains the location of the remote BETA buffer.

≠ # FFFF — L contains the maximum number of words to be returned in BETA. The BETA buffer begins in the word following ALPHA (2).

R — Good return means the controllee chain was stored. R contains the number of words returned in BETA.

Error Return means the call was not processed. R contains the error number.

= 1 Length of BETA is zero.

= 2 Illegal option

J — The calling problem program's level in the controllee chain, returned by the system.

B — The descriptor number, a unique number associated with the calling program, returned by the system.

M = ∅ List all controllees in the chain

= 1 List only this problem program (four BETA words returned by system)

= 2 List only this problem programs controller. (Four BETA words returned by system).

SYSTEM CALL #13 - "LIST CONTROLLEE CHAIN" cont'd

M = 3 — List only this problem program's controllee(four BETA words returned by the system).

BETA — There are 4 words per entry. For M=ØØ, the controllees are listed in ascending order, starting with the problem program directly attached to the teletype.

BETA (1)	S 8	T 8	C 8	24	N 8	D 8
BETA (2)	K 8	▨	▨	▨	▨	56
BETA (3)	SOURCE FILE NAME IN ASCII 64					
BETA (4)	DROP FILE NAME IN ASCII 64					

S — The level of the problem program whose name is in BETA (3), the level is a number 2-6.

T — Contains the descriptor number which is associated with the problem program in BETA (3).

C — Contains the descriptor number of this problem programs controller. (returned by the system)

N — Contains the descriptor number of another PP in the chain. The PP in BETA (3) has informed the system that messages from above not specifically directed to him should be sent to N. (returned by the system; may be zero) see P. 4.20.1)

D — Contains the descriptor number of another PP in the chain. The PP in BETA (3) has informed the system that messages from below not specifically directed to him should be sent to D. (returned by the system; may be zero). see P. 4.20.1)

K — Contains the descriptor number of this problem programs controllee (returned by the system; may be zero).

SYSTEM CALL #13 - "LIST CONTROLLEE CHAIN" cont'd

Remarks:

1. The descriptor number may be used in the Send Message system calls to specifically direct a message to a particular PP, or in the Get Message system calls to determine the identity of the sender.

2. The descriptor number is unique and is associated with the PP until it is disconnected.

3. There are a maximum 5 problem program controllee levels, starting with 2 which is the level directly under the teletype. The teletype is level 1.

4. The calling program may associate J & B in ALPHA (2) with S and T in BETA (1) to get his place in the controllee chain.

SYSTEM CALL #14 - "SEND A MESSAGE TO CONTROLLER"

This call may be used by a problem program to send a message to a problem program controller or the teletype.

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>C
8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">BL
16</td><td colspan="4">VIRTUAL ADDRESS OF REMOTE DATA
BUFFER 48</td></tr>
</table>

F — Function Code = #14

C — Control Field

= ØØ — Send a message to controller and if problem program controller, stop running this PP (pages drift out) and start running the controller. If teletype controller, keep running this PP.

= Ø1 — Send a message to controller and if problem program controller, write all of this PP's pages on disk before starting the controller. If teletype controller, keep running this PP.

= Ø2 — Send a message directly to teletype and keep running this PP.

M — Replace, Notify or Wait Option

= ØØ — Replace Option: If the teletype is logged out then replace any existing message. If the teletype is logged in but the buffer is full, then stop running this PP (pages drift out) until the buffer is free.

= Ø1 — Notify Option: Return to the error exit address if unable to send message. Check the R field.

= Ø2 — Wait Option: If unable to send message, stop running this PP (pages drift out) until the message can be sent.

SYSTEM CALL #14 - "SEND A MESSAGE TO CONTROLLER" cont'd.

L = # FFFF — BL contains the number of bytes in the message. The rightmost 48-bits of ALPHA (3) point to the remote BETA buffer where the message is stored.

≠ # FFFF — L contains the number of bytes in the message. The BETA buffer begins in the word after ALPHA (2).

L or BL must be >Ø & ≤ 4096.

B — The message is sent to the controller whose descriptor number is in B. If C=Ø2, or this problem program is level two, B is ignored. The B field may be zero in which case the message is sent to the next high level controller (may be teletype).

R — Error Response Field

= 1 BETA byte count > 4096 or = Ø.

= 2 Illegal option.

= 3 For B non-zero, no controller exists by that descriptor number.

= 4 Teletype not logged in. (M=01)

= 5 Teletype logged in under a different suffix. (M=01)

= 6 System output buffer full. (M=01)

BETA — Contains the message.

BETA (1)

C_1 8	C_2 8	C_3 8	C_4 8	C_5 8	C_6 8	C_7 8	C_8 8
C_9 - - - - - - -							
C_{L-2} 8	C_{L-1} 8	C_L 8					

C is an 8-bit ASCII character.

Remarks:

1. For output messages to teletype using the replace or wait option (M=ØØ,Ø2), if the teletype is logged in, the system buffer will hold up to 5 messages or 4096 bytes, whichever occurs first. If the teletype is logged out, the buffer will hold only one message.

2. Output messages to teletype are grouped in blocks of 151 characters and sent one block at a time to the teletype. If the last block is less than 151 bytes, an end-of-message character (#17) is added after the last message byte.

SYSTEM CALL #15 -"SEND A MESSAGE TO CONTROLLEE"

This call may be used by a problem program to start its CONTROLLEE with or without a message. The CONTROLLEE must have been previously initialized.

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>C
8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">BL
16</td><td colspan="4">VIRTUAL ADDRESS OF REMOTE BETA BUFFER
48</td></tr>
</table>

F — Function Code = #15

C — Control Field

= ØØ — Stop running this problem program (pages drift out) and start the controllee immediately.

= Ø1 — Write all of this problem program's pages to disk before starting the controllee.

M — Is the Message Option Field

= ØØ — Start controllee with a message and if the controllee already has a message from controller, replace it with the message in BETA.

= Ø1 — Start controllee with a message, but if the controllee already has a message from controller, return to the error exit address.

= Ø2 — Start the controllee without a message.

L = # FFFF — BL contains the number of bytes in the message. The rightmost 48-bits of ALPHA (3) point to the Remote BETA buffer.

≠ # FFFF — L contains the number of bytes in the message. The BETA buffer begins in the word after ALPHA (2). L or BL must be >Ø & ≤ 4096.

SYSTEM CALL #15 - "SEND A MESSAGE TO CONTROLLEE" cont'd

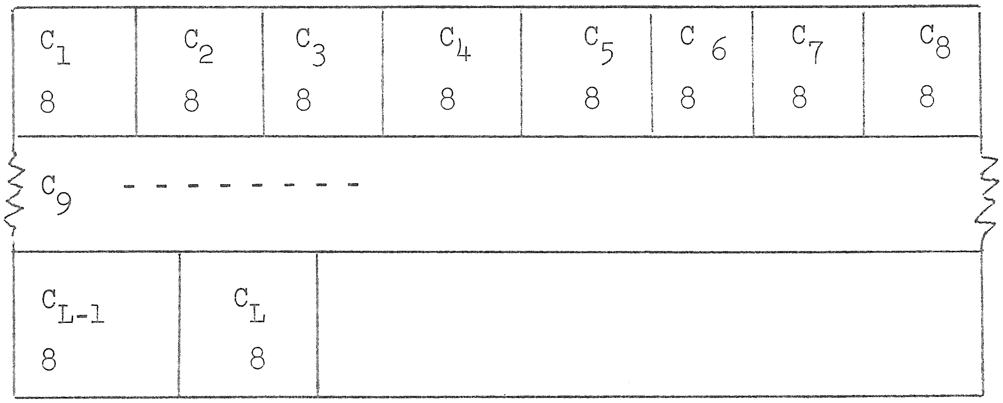
B — The message is sent to the controllee whose descriptor number is in B. The B field may be zero, in which case the message is sent to the next lower level controllee.

R — Error Response Field

= 1	BETA byte count > 4096 or = ∅
= 2	Illegal option.
= 3	For B non zero, no controllee exists by that descriptor number.
= 4	For B = ∅, no controllee exists.
= 7	Controllee already has a message. (M=01)

BETA — Contains the message

BETA (1)



C is an 8-bit ASCII character

SYSTEM CALL #15 - "SEND A MESSAGE TO CONTROLLEE" cont'd

Remarks:

1. Sending a message causes the system to copy the message from the problem program virtual space into a system buffer and to start the controllee.

2. If a controllee is running, a message from teletype sent to its controller will start the controller and stop the controllee.

3. If any controllee other than level two issues the SYSTEM CALL #16, "GET A MESSAGE or SYMBOLS FROM CONTROLLER" with the wait option, and there is no message from controller waiting, then the next higher level controller problem program will be started and the controllee will stop running and be put in a state of waiting for a message from controller.

SYSTEM CALL #16 - "GET A MESSAGE OR SYMBOLS FROM CONTROLLER"

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>C
8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>J
8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">BL
16</td><td colspan="4">VIRTUAL ADDRESS OF REMOTE BETA BUFFER
48</td></tr>
</table>

F Function Code = #16

C Control Field ("Wait" Option = ØØ & Ø2 and "Notify" Option = Ø1, Ø3)

= ØØ If a message from controller is not there, stop running this PP until a message arrives. Return the message to the PP buffer and release it from the system buffer.

= Ø1 If there is no message from controller, return to the error exit address. If there is a message return it to the problem program buffer and release it from the System buffer.

= Ø2 If a message from controller is not there, stop running this PP until a message arrives. Return the message to the PP buffer, but do not release it from the system buffer.

= Ø3 If there is no message from controller, return to the error exit address. If there is a message return it to the problem program buffer but do not release it from the system buffer.

M Message Format Option

= ØØ Return in message format.

≠ ØØ Return in symbol format.

= Ø1 Delimiters are space and control characters. Symbols are blank filled

= Ø2 Delimiters are space, control characters, comma, period, semicolon, left and right parenthesis, left and right brackets. Symbols are blank filled.

= Ø3 Delimiters are programmer specified. The number of delimiters is stored in the left most of 16-bits of BETA (1). The right most 48-bits of BETA (1) point to the Delimiters. The Delimiters are stored left to right, byte by byte in the buffer. The number of delimiters must be ≤ 2ØØ. The symbols are stored starting in BETA (2). Symbols are blank filled.

= Ø4 Delimiters are any character not a letter, digit, or period. Symbols are blank filled.

= Ø9 Same as Ø1 except symbols are null filled.

= ØA Same as Ø2 except symbols are null filled.

= ØB Same as Ø3 except symbols are null filled.

= ØC Same as Ø4 except symbols are null filled.

L = # FFFF BL contains the maximum number of bytes (M=ØØ) or words (M≠ØØ) to be delivered as a result of this call. The right most 48-bits of ALPHA (3) point to the Remote BETA buffer.

≠ # FFFF L contains the maximum number of bytes (M=ØØ) or words (M≠ØØ) to be delivered as a result of this call. The BETA buffer begins in the word after APLHA (2).

L or BL must be ≥ Ø and ≤ 4096.

R Good return means the message was stored. R contains the number of bytes (M=ØØ) or number of words (M≠ØØ) returned in BETA.

Error return means no message was stored. R contains the error number.

= 1 Byte (M=ØØ) or word (M≠ØØ) cound bad. L=Ø or > 4096.

= 2 Illegal option.

= 3 No message from controller waiting.

= 7 For M=Ø3 and =#ØB the delimiter count was greater than 2ØØ.

SYSTEM CALL #16 - "GET A MESSAGE OR SYMBOLS FROM CONTROLLER" cont'd.

J The system will store the level of the sender.

B The system will store the descriptor number of the sender.

Remarks:

1. Getting a message causes the system to copy the message from the system buffer to the problem program buffer. No end-of-message is added by the system. If the number of bytes in the message is greater than the number requested, only the number of bytes requested will be delivered. If the number of bytes in the message is less than the number requested, the entire message will be delivered and the remaining portion of the PP buffer will be cleared.

2. Getting symbols causes the system to crack the message into symbols and copy them into the problem program buffer. A delimiter, other than a blank or null is stored as a symbol. For M=Ø1,Ø2,Ø4, Ø9, #ØA, and #ØC, blanks and nulls are squeezed out. For M=Ø3, and #ØB blanks and nulls are squeezed out only if set as delimiters. If the number of symbols is greater than the number of words requested, only the number of words requested will be delivered. If the number of symbols is less than the number of words requested, all the sumbols will be delivered. (No end-of-message is added by the system.)

3. A symbol is defined to be less than 9 characters. Symbols are right-adjusted in the BETA word and blank or null filled in the leftmost part of the word if less than 8 characters.

4. If the message was sent from teletype, J will be set to one and B will be set = # FF.

SYSTEM CALL #17 - "GET A MESSAGE OR SYMBOLS FROM CONTROLLEE"

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>C
8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>J
8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">16</td><td colspan="4">VIRTUAL ADDRESS OF REMOTE BETA BUFFER
48</td></tr>
</table>

F Function Code = # 17

C Control Field

= ØØ Return the message to the PP buffer and release it from the system buffer.

= Ø2 Return the message to the PP buffer, but do not release it from the system buffer.

M Message Format Option.

= ØØ Return in message format.

≠ ØØ Return in symbol format.

= Ø1 Delimiters are space and control characters. Symbols are blank filled.

= Ø2 Delimiters are space, control characters, comma, period, semicolon, left and right parenthesis, left and right brackets. Symbols are blank filled.

= Ø3 Delimiters are programmer specified. The number of delimiters is stored in the leftmost 16-bits of BETA (1). The rightmost 48-bits of BETA (1) point to the Delimiters. The Delimiters are stored left to right, byte by byte, in the buffer. The number of Delimiters must be ≤ 2ØØ. The symbols are stored starting in BETA (2). Symbols are blank filled.

= Ø4 Delimiters are any character not a letter, digit, or period. Symbols are blank filled.

= Ø9 Same as Ø1 except symbols are null filled.

= #ØA Same as Ø2 except symbols are null filled.

= #ØB Same as Ø3 except symbols are null filled.

SYSTEM CALL #17 - "GET A MESSAGE OR SYMBOLS FROM CONTROLLEE" cont'd.

L	= # FFFF	BL contains the maximum number of bytes (M=ØØ) or words (M≠ØØ) to be delivered as a result of this call. The rightmost 48-bits of ALPHA (3) point to the Remote BETA buffer.
	≠ # FFFF	L contains the maximum number of bytes (M=ØØ) or words (M≠ØØ) to be delivered as a result of this call. The BETA buffer begins in the word after ALPHA (2). L or BL must be > Ø & ≤ 4096.
R		Good return means the message was stored. R contains the number of bytes (M=ØØ) or number of words (M≠ØØ), returned in BETA. Error return means no message was stored. R contains the error number.
	= 1	Byte (M=ØØ) or word (M≠ØØ) count bad. L = Ø or > 4096.
	= 2	Illegal option
	= 3	No message from controllee waiting.
	= 4	There is a message from controller waiting.
	= 5	Available
	= 6	PP was started because the controllee whose level and descriptor number is stored in J and B is waiting on a message from controller.
	= 7	For M=Ø3 or M=#ØB, the delimiter count is > 2ØØ

SYSTEM CALL #17 - "GET A MESSAGE OR SYMBOLS FROM CONTROLLEE" cont'd.

J — The system will store the level of the sender.

B — The system will store the descriptor number of the sender.

Remarks:

1. Getting a message causes the system to copy the message from the system buffer to the problem program buffer. No end-of-message is added by the system. If the number of bytes in the message is greater than the number requested only the number of bytes requested will be delivered. If the number of bytes in the message is less than the number requested, the entire message will be delivered and the remaining portion of the PP buffer will be cleared.

2. Getting symbols causes the system to crack the message into symbols and copy them into the problem program buffer. A delimiter, other than a blank or null is stored as a symbol. For M=Ø1, Ø2, Ø4, Ø9, #ØA, and #ØC, blanks and nulls are squeezed out. For M=Ø3 and =#ØB blanks and nulls are squeezed out only if set as delimiters. If the number of symbols is greater than the number of words requested, only the number of words requested will be delivered. If the number of symbols is less than the number of words requested, all the symbols will be delivered. No end-of-message is added by the system.

3. A symbol is defined to be less than 9 characters. Symbols are right-adjusted in the BETA word and blank or null filled in the leftmost part of the word if less than 8 characters.

4. The level and descriptor number returned by the system will have no meaning if the controllee who sent the message has been disconnected.

SYSTEM CALL #18 - "MESSAGE CONTROL"

This call may be issued by the problem program to inform the system that messages sent to this problem program should be directed to another CONTROLLEE or CONTROLLER in the chain.

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>M
8</td><td>C
8</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>J
8</td><td>B
8</td><td colspan="4">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
</table>

F — Function Code #18

C — Turn on or off bypass

= ∅ Turn off bypass

= 1 Turn on bypass

M — Option Field

= 1 Means set input bypass (C=∅1) or turn off input bypass (C=∅∅). If the input bypass is set, then messages from controller not specifically directed to this PP should be sent to this PP's controllee.

= 2 Means set output bypass (C=∅1) or turn off output bypass (C=∅∅). If the output bypass is set, then messages from controllee not specifically directed to this PP should be sent to this PP's controller.

= 3 Means set both bypasses (C=∅1) or turn off both bypasses (C=∅∅).

= 4 Means look only at J.

= 5 Means look only at B.

= 6 Look at both J and B to decide where messages should be directed.

SYSTEM CALL #18 - "MESSAGE CONTROL" cont'd.

L Not used.

R Error Response Field

= 1 No controller or controllee exists by that descriptor number.

J Descriptor number of CONTROLLEE to whom messages from above are to be directed.

B Descriptor number of CONTROLLER to whom messages from below are to be directed.

Remarks:

1. The controllee may direct messages from below to teletype by setting B = # FF.

2. This call does not redirect those messages which are sent specifically to this problem program, that is, the descriptor number was set in the B field in system call #14 and #15 (see p. 4.16.1 and p. 4.17.1).

SYSTEM CALL #19 - "WRITE CONTROLLEE PAGES TO DISK"

<table>
<tr><td>ALPHA (1)</td><td>R
16</td><td>L
16</td><td>16</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>N
16</td><td colspan="3">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td>BL
16</td><td colspan="3">VIRTUAL ADDRESS OR REMOTE BETA BUFFER
48</td></tr>
</table>

F — Function Code = #19

N — Control Option

= ØØ Write next lower controllee pages to disk.

= Ø1 Write controllee pages whose descriptor number is in BETA (1) to disk.

= Ø2 Write this problem program's pages to disk.

L = # FFFF BL contains the number of words in BETA. The rightmost 48-bits of ALPHA (3) contains the location of the remote BETA buffer.

≠ # FFFF L contains the number of words in BETA. The BETA buffer begins in the word following ALPHA (2).

R — Error Response Field

= 1 No controller or no controllee by that descriptor number.

BETA	B 8	▨ 56

B — The controllee whose descriptor number is B will be written to disk. (N = Ø1)

Remarks:

1. The CONTROLLER stops running and is put in a "WRT CNTE" state until all the CONTROLLEE pages are on disk. (N = ØØ, Ø1)

SYSTEM CALL #1A - "SEND A MESSAGE TO THE OPERATOR"

A problem program may use this call to communicate to the operator.

ALPHA (1)	R 16	L 16	16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		
ALPHA (3)	BL 16	VIRTUAL ADDRESS OF REMOTE BETA BUFFER 48		

F Function Code #1A

N Control Field

=ØØ If unable to send message, stop running this problem program until the message can be sent.

=Ø1 Return to the error exit address if unable to send message.

L =# FFFF BL contains the number of bytes in the message. The rightmost 48-bits of ALPHA (3) point to the remote BETA buffer.

≠# FFFF L contains the number of bytes in the message. The BETA buffer begins in the word after ALPHA (2).

L or BL must be > Ø and ≤ 8Ø.

R Error Response Field.

= 1 Byte count bad, L = Ø or >8Ø.

= 2 System buffer full (N = Ø1).

Remark:

1. The operator's teletype is always logged in. Therefore, the only reason a message could not be sent at the time the call is issued is because the system buffer is full.

SYSTEM CALL #1B - "INITIALIZE OR DISCONNECT CONTROLLEE"

This call is used by a problem program to initialize another problem program as controllee. This call may also be used by a problem program to disconnect a previously initialized controllee.

<table>
<tr><td>ALPHA (1)</td><td>R
16</td><td>L
16</td><td>▨
16</td><td>F</td></tr>
<tr><td>ALPHA (2)</td><td>N
16</td><td colspan="3">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td>BL
16</td><td colspan="3">VIRTUAL ADDRESS OF REMOTE BETA BUFFER
48</td></tr>
</table>

F		Function Code = #1B
N		Control Field
	= ØØ	Initialize controllee and restart this PP.
	= Ø1	Initialize controllee and start controllee immediately. Stop running this PP.
	= #10	Disconnect Controllee and restart this PP (BETA not used)
L	= # FFFF	BL contains the number of words in BETA. The rightmost 48-bits of ALPHA (3) contains the location of the remote BETA buffer.
	≠ # FFFF	L contains the number of words in BETA. The BETA buffer is located in the word after ALPHA (2).
R		Error Response Field
	For N≠ 1Ø	
	= 1	Controllee already present
	= 2	Illegal option
	= 3	File not there
	= 4	Not enough time in bank to run controllee
	= 5	Illegal Priority
	= 6	System drop file create error
	= 7	Controllee file is not executable
	= 8	Disc trouble
	= 9	System tables full, cannot initialize controllee at this time.

SYSTEM CALL #1B - "INITIALIZE OR DISCONNECT CONTROLEE" cont'd.

= # A   Source or drop file IOC anomaly.

= # B   5 levels of controllees are already present.

= # D   Controllee drop file is too small

for N = # 10

= # C   No controllee present.

BETA

BETA (1)	SOURCE FILE NAME OF CONTROLLEE (ASCII)		
BETA (2)	B 8	8	TIME LIMIT IN MICROSECONDS 48

B   Contains the descriptor number of the controllee (returned by the system).

If time limit is zero, the controller's time limit is used.

Remarks:

1. Five levels of problem program controllees are maximum.

2. The descriptor number is a unique number associated with the controllee. If the controllee is disconnected and re-initialized, this number might change.

SYSTEM CALL #1C - "PROBLEM PROGRAM INTERRUPT"

This call may be used by a problem program to inform the system that it wants to be interrupted or it does not want to be interrupted by a message.

<table>
<tr><td>ALPHA (1)</td><td colspan="2">R
16</td><td>L
16</td><td>
16</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>J
8</td><td>B
8</td><td colspan="3">ERROR EXIT VIRTUAL ADDRESS
48</td></tr>
<tr><td>ALPHA (3)</td><td colspan="2">BL
16</td><td colspan="3">VIRTUAL ADDRESS OF REMOTE BETA BUFFER
48</td></tr>
</table>

F Function Code = #1C

L = # FFFF BL contains the number of words in. BETA. The rightmost 48-bits of ALPHA (3) point to the remote BETA buffer.

≠ # FFFF L contains the number of words in BETA. The BETA buffer begins in the word after ALPHA (2).

R Error Response Field

= 1 Interrupt address greater than $2^{47} - 1$.

= 2 Illegal option.

B Option

= ∅ Means any message will interrupt this PP.

= 1 Means any message preceded by a left adjusted CTRL-E i (#0549) will interrupt this PP.

= 2 Means this PP no longer wants to be interrupted by the arrival of a message.

J = ∅ Type interrupt option

Means the B filed refers to messages from controller

SYSTEM CALL #1C - "PROBLEM PROGRAM INTERRUPT"cont'd

BETA	Contains the interrupt address, the virtual bit address where the PP is to be started when a message arrives.

Remarks:

1. When the "Problem Program Interrupt" system call is issued for options B = ØØ and 01, the problem program will be interrupted by all succeeding messages or CTRL-E i messages until the problem program terminates or issues the call with option B=Ø2.

2. There will always be a message waiting if the problem program is sent to the interrupt address.

3. The problem program interrupt is treated like an I/O interrupt. In order to release the interrupt the problem program must issue the system call, "Return from Interrupt" described on P. 4.29.1.

4. For B= Ø1, the CTRL-E i is stripped off the message. The message is repositioned at the beginning of the word.

5. The CTRL-E i interrupt will cause any output message(s) to be released.

6. The CTRL-E i will interrupt the highest level controllee who has issued the interrupt system call with option BB = Ø1. The input bypass is ignored.

SYSTEM CALL #23 - USER DIRECTORY MODIFICATION

This call is used to add, delete or modify an entry in the User Directory. It will be used to update bank accounts, user combinations, etc. It also contains a means for donating time between pool accounts. The call is restricted to privileged users.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N 16	ERROR EXIT ADDRESS		

BETA (1)	D 32	E 1	P 1	S 1	R 1	U 20

BETA (2)	SS 8	G 8		Z 40

R Response Code

= Ø Normal Completion

= 1 See BSS

= 2 Error form service station

L = # FFFF Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.

≠ # FFFF BETA buffer immediately follows word ALPHA (2) and contains L words.

C Option Control

= Ø modify existing accounts, or add if not already in user directory.

= 1 add new account (s).

= 2 delete existing account (s).

= 3 donate from pool to pool (1 per call)

SYSTEM CALL #23 - USER DIRECTORY MODIFICATION cont'd.

F — Function code = #23 for UD Mod.

N — The number of accounts to be processed with this call.

BETA WORD FORMATS:

C = Ø — The first BETA word is considered to have the update time, the second BETA word to have the bank name, the third BETA word to have the minimum priority, the fourth BETA word is reserved for future use. The actual bank account data begins in BETA (5).

BETA (5) :

G = maximum percentage donation from the divisional repository allowed for this user.

D = division code (ASCII)

E ≠ Ø reset DEBIT and PERCENTAGE fields for all users of this repository

= Ø no such reset as above

P ≠ Ø repository account

= Ø private user account

S ≠ Ø time returns to pool (snap-back)

= Ø user eligible to keep time

R ≠ Ø OK to execute priority job

U = binary user number (not used for P ≠ Ø).

BETA (6) — BSS - error code*

Z = The bank account quantity to be installed (ignored if P = Ø) in microseconds.

SYSTEM CALL #23 - USER DIRECTORY MODIFICATION cont'd.

BETA (7)		As BETA (5)
BETA (8)		As BETA (6)
etc.		
C = 1		Format as for C=Ø, but start with BETA (1) being the first account. No presumptions made about first four BETA words.
C = 2		Only one BETA word per account is required.
BETA (1)		G - error code*
	D =	division code (ASCII)
	E =	not used
	P =	as for C = Ø above
	S =	not used
	R =	not used
	U =	as for C = Ø above
C = 3		Three BETA words required. Only one donation is allowed for each call.
BETA (1)	G =	error code*
	D =	division code (ASCII) for Donor
	E =	not used
	P =	not used
	S =	not used
	R =	not used
	U =	not used
BETA (2)	G =	not used
	D =	division code (ASCII) of Donee
	E, P, S,R,U	not used
BETA (3)	Z =	microsecond donation

SYSTEM CALL #23 - USER DIRECTORY MODIFICATION cont'd.

*	Error Codes
= ∅	Normal
= 1	Trying to add an already extent number
= 2	Division mismatch
= 3	No such account to update
= 4	UD full when trying to add a new entry
= 5	Trying to modify non-pool account
= 6	Percentage greater than 100
= 7	Not enough time in donor pool
= 8	Illegal option
= 9	Illegal table name
= 1∅	Table size too big

SYSTEM CALL #24 - MISCELLANEOUS

This call allows a Problem Program to manipulate its time limit and/or that of its controllees. It further allows the problem program to a variety of miscellaneous information about itself, its controller(s) and its controllee(s).

<table>
<tr><td>ALPHA (1)</td><td>R
16</td><td>L
16</td><td>C
16</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>N
16</td><td colspan="3">ERROR EXIT ADDRESS
48</td></tr>
</table>

<table>
<tr><td>BETA (1)</td><td></td></tr>
<tr><td>BETA (2)</td><td></td></tr>
</table>

etc...

R		Response Code
	= ∅	Normal completion
	= 1	Error
L	= # FFFF	Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.
	≠ # FFFF	BETA buffer immediately follows word ALPHA (2) and contains L words
C	#	Control field
	= ∅	Modify time limit BETA (1) = new time limit from program (microseconds) BETA (2) = existing time limit from system (microseconds)
	= 1	Get user ID number and bank account BETA (1) = User ASCII number from system BETA (2) = Bank time (integer microseconds)
	= 2	Change priority of calling program - - Call has been disallowed - -

SYSTEM CALL #24 - MISCELLANEOUS cont'd.

= 3 Get time limit and priority
BETA (1) = Existing time limit (microseconds) from system
BETA (2) = Existing priority from system

= 4 Change priority of calling program and controllees
- - Call has been disallowed - -

= 5 Change priority of controllees
- - Call has been disallowed - -

= 6 Get controllee name
BETA (1) = Source file name
BETA (2) = Drop file name

= 7 Get controller name
BETA (1) = Source file name
BETA (2) = Drop file name

= 8 Who am I?
BETA (1) = Source file name
BETA (2) = Drop file name
BETA (3) = Suffix, Level in controllee chain, ID (ASCII)

= 9 Get elapsed execution time and page fault count
BETA (1) = CPU, Memory times
BETA (2) = System Call, Implicit I/O times
BETA (3) = Explicit I/O, Remote I/O times
BETA (4) = Page fault count, page faults to drum

= A BETA (1) = LRL Master clock
BETA (2) = HR:MI:SE (ACII)
BETA (3) = MO/DA/YR (ASCII)
BETA (4) = Millisecond station clock
BETA (5) = CPU microsecond clock

= B Restricted to User-1 codes, erase system output tape in numbers BETA(1) = HSP, CRT or bb8∅ (ASCII)

SYSTEM CALL #25 - RECALL

This system call allows a problem program to suspend itself for a time period in the interval (30 sec $\leq$ Tsus $\leq$ 30 min.). The system will recall the program to active status at the end of the suspension interval. The program is not allowed to own tape(s) or to be connected to a problem program controller or controllee. System privaleged user numbers are exempt from the tape ownership restriction.

ALPHA (1)	R 16	L 16	C 16	F 16
ALPHA (2)	N	ERROR EXIT ADDRESS 48		

BETA (1)	▨ 32	TIME 32

R		Response Code
	= Ø	Normal Completion
	= 1	Error - call not allowed
L	= # FFFF	Left 16-bits of word ALPHA (3) equal length of remote BETA buffer. Right 48-bits of word ALPHA (3) equal location of remote BETA buffer.
	≠ # FFFF	BETA buffer immediately follows word ALPHA (2) and contains L words
C		not used
F		Function code = #25 for RECALL
N		not used
TIME		Suspension period given in integer microseconds, 30 sec. $\leq$ time $\leq$ 30 min. Any time period given outside the allowed interval will be set to the nearest interval limit.

SYSTEM CALL #50 EXPLICIT I/O

Two forms of input and output are available on STAR, Implicit and Explicit. Implicit I/O is accomplished by the user defining a 1 to 1 correspondence between segments of disk space and equal length segments of virtual address space (see calls CREATE, OPEN, MAP). This mapping information is stored in the executing program's minus page and IOC's (see STAR minus page format). Given this Map, a reference to a virtual address not already in the memory drum system can be transformed into a disk address via the map provided and the system can do the necessary I/O to obtain the required page. References to pages which do not exist in the map cause pages to be assigned to the user. These pages are called free space and the system automatically catalogues them in the free space map of the drop file. The user may also obtain free space in blocks larger than one page using the ADVISE call.

Any input or output operations done by the system as a function of the virtual space - disk address correspondence mapping is called Implicit I/O since the user causes it to be done implicitly by virtual address referencing.

Tape I/O cannot be done Implicitly and only data blocks of 512 words (small pages) or 65K words (large pages) may be transferred implicitly. These two facts give rise to the need for the user to have the capability of requesting specific blocks of data to be transferred. This is accomplished through a system call with function code #100. The format of this call is the same as any other system call except the error exit address in the second alpha word is replaced by an interrupt address.

Up to 8 BETA words may be associated with each call and each BETA word may be either a window (buffer definition) operation or an image (data transfer) operation. There are two reasons for separating these two functions. First, the user often uses the same window (buffer area) for a succession of image (data transfer) requests. Hence, redefining the window for each image is a redundant operation. Second, it is anticipated

SYSTEM CALL #50 EXPLICIT I/O cont'd.

that the system should evolve such that a code stated in large pages utilizing explicit I/O should be capable of running in a small page demand paging mode with no changes to the code when an insufficient number of large pages are available. It appears necessary to separate window and image requests to achieve this goal.

In order to allow double buffering without requiring an inordinate amount of window requests, the system provides two windows per file(IOC).

Note that if one wants to simulate an IOD as used on the Frost and Floe systems, it requires three beta words, the first an open window, the second an image and the third a close window.

The Format for a STAR I/O call is:

A	R	L		F	
1	15	16	16	16	ALPHA (1

SYS	B	INTERRUPT ADDRESS	
8	8	48	ALPHA (2

BETA LENGTH	BETA ADDRESS	
16	48	ALPHA (3)

FC = # 50_{16} for an I/O call.

BP contains the BETA index for the I/O request which just completed and caused the interrupt. B is stored by the system.

L = # FFFF means ALPHA (3) contains the address and length of the BETA buffer.

≠ # FFFF means the BETA buffer immediately follows ALPHA (2).

L is then the number of words in the BETA buffer.

R is the response code filled in on completion of the call.

= Ø	NO ERRORS
= 1	ILLEGAL INTERRUPT ADDRESS
= 2	MORE THAN 8 REQUESTS
= 4	ERROR IN ONE OR MORE I/O REQUESTS

A - A bit cleared by the system when the ALPHA and BETA words are no longer required.

SYS temporary storage for system.

Each I/O request is ONE word having one of the following two formats.

<table>
<tr><td>OP
8</td><td>SUBOP
8</td><td>BUSY
1</td><td>7</td><td colspan="2">IOC
8</td><td colspan="2">CENTRAL ERROR
8</td><td>STATION ERROR
24</td></tr>
<tr><td>MODE
8</td><td>TAPE MODE
8</td><td>ISSUED
1</td><td>7</td><td>7</td><td>RETRY
1</td><td>7</td><td>TAPE FLAG
1</td><td>FILE ADDRESS
24</td></tr>
</table>

<table>
<tr><td>OP
8</td><td>SUBOP
8</td><td>BUSY
1</td><td>7</td><td>IOC
8</td><td>CENTRAL ERROR
8</td><td>24</td></tr>
<tr><td>WINDOW LENGTH
8</td><td colspan="3">24</td><td colspan="3">WINDOW ADDRESS
32</td></tr>
</table>

The first format is for IMAGE requests. The second format is for WINDOW requests.

OP =	1	READ
	2	WRITE
	3	FUNCTION
	4	WINDOW

SUBOP for OP = 1 or 2

SUBOP =	1	WINDOW 1
	2	WINDOW 2

SUBOP for OP = 3

SUBOP =	3	REWIND
	4	UNLOAD
	5	WRITE END OF FILE
	6	FOREWARD SPACE RECORDS
	7	READ TO END OF FILE
	8	BACKSPACE RECORDS
	9	BACKSPACE FILE

A	SET DENSITY
B	SEEK
C	ERASE
F	READ STATUS

SUBOP for OP = 4

SUBOP =	1	OPEN WINDOW 1
	2	CLOSE WINDOW 1
	3	OPEN WINDOW 2
	4	CLOSE WINDOW 2

IOC - Input/Output Connector

MODE = $000xxxx1_2$ interrupt on good completion.

$000xxx1x_2$ interrupt on error.

$000001xx_2$ process this request and all previous requests before issuing the next request. Control is returned to the PP.

$000010xx_2$ process this request and all previous requests before issuing the next request. Control is NOT returned to the PP.

$000011xx_2$ process this request before issuing the next request. Control is returned to the PP.

$000100xx_2$ give up CPU until this request is complete.

$000000xx_2$ proceed with next request immediately.

FILE ADDRESS - logical page (sector) address at which data transmission is to begin.

BUSY - cleared when the request is complete.

WINDOW ADDRESS- Starting VIRTUAL PAGE ADDRESS where image requests are to deposit or obtain information

WINDOW LENGTH - The length in pages (sectors) of the VIRTUAL RANGE to be associated with the WINDOW.

TAPE FLAG - For TAPE READ, = Ø means truncate record to 16-bit word boundary.

= 1 means transmit entire record.

RETRY = Ø	Standard Recovery Procedure
= 1	No Retry on Error

TAPE MODE Ø = BCD 1 = BINARY 2 = BINARY ASCII

CENTRAL ERROR - Set when an error is found by central before the request is sent to the station.

= Ø	No error found by central
= 1	Non-existent IOC
= 2	Window size greater than 24 small pages
= 3	Not sequential disk file
= 4	Density not Ø, 1, or 2 for FUNCTION # A
= 5	Illegal OP or SUBOP
= 6	Illegal tape mode or mode field
= 7	No WINDOW assigned
= 8	FILE ADDRESS out of bounds
= 9	Illegal access (r/o or w/o file)
= # A	Interrupt requested with no interrupt address specified
= # B	Over 128 small pages for this I/O call
= # C	Window crosses large page boundary.

ISSUED - a bit set by the system when no central error was found and the request has been sent to the station.

STATION ERROR - The following error conditions are returned by the station. Multiple conditions are or'ed together.

xxxxx1	Device not ready
xxxxx2	Parity error
xxxxx4	Data exceeds programmer's buffer
xxxxx8	End of Tape
xxxx1x	End of file condition
xxxx2x	Attempt to write file-protected tape
xxxx4x	Channel failure
xxxx8x	Lost data on Tape record
xxx1xx	Attempted backspace at load point
xxx2xx	Disk positioning Error

NOTES ON I/O REQUESTS

1. For TAPE READ or WRITE operations, the FILE ADDRESS FIELD when NON-ZERO will specify the number of 16-bit bytes to come from or go to tape, otherwise the full IMAGE of the specified WINDOW will be transmitted.

2. After a TAPE READ operation, the FILE ADDRESS FIELD will contain the number of 16-bit bytes in the physical record. If the record is larger than the WINDOW, only as much data as will fit in the WINDOW will be transmitted to the WINDOW.

3. The FILE ADDRESS FIELD will contain the RECORD COUNT for the FOREWARD SPACE and BACKSPACE operations. If an END OF FILE is encountered the spacing operation will stop and the number of records actually spaced over will be returned in the FILE ADDRESS FIELD.

4. The FILE ADDRESS FIELD will contain the tape UNIT STATUS after the READ STATUS operation. The STATUS bits are:

000XX1	Ready
000XX2	Busy
000XX4	Write Enable
000XX8	End of File
000X1X	Load Point
000X2X	End of Tape
000X0X	200 BPI Density
000X4X	556 BPI Density
000X8X	800 BPI Density
0001XX	Lost Data
0002XX	End of Operation
0004XX	Parity Error
0008XX	Reserved

5. For ERASE DISK FUNCTION, the second half word in the second BETA word will be treated as two 16-bit fields, the left most 16-bits containing a sector count and the rightmost 16-bits containing a patteren to be written. A sector count of zero will imply the whole file is to be erased.

6. The FILE ADDRESS FIELD will contain the number of erasures to be performed for the ERASE TAPE FUNCTION. (∿ 6 inches of blank tape/ erasure.)

7. No interrupt routine may be interrupted. The interrupts are stacked and processed one at a time. The zero level will be started only after the "Return from Interrupt" call has been issued for the last interrupt in the list.

8. The maximum WINDOW size for disk and tape is 24 small pages unless the buffer is in a large page.

9. After a TAPE READ operation, if the RECORD length is not equal to the WINDOW length, the remainder of the WINDOW will be undefined.

10. If there is a central or station error in one of the Beta requests and this request does not have the mode bit "interrupt on error" set, all following requests will be processed normally. If the "interrupt on error" bit is set, the requests following the one in error will be processed up to and including the request with any of the contingency bits set (the three left adjusted mode bits); the rest of the Beta requests will NOT be issued.

11. No window may cross a large page boundary.

12. Only 6 I/O calls may be processed at any one time. If six #50 calls have I/O outstanding and the PP issues another I/O call, the PP will disconnected until one call completes.

13. For FUNCTION #A, the DENSITY should be placed in the FILE ADDRESS FIELD. It should be set = ∅, 1, or 2 for 200 BPI, 556 BPI, or 800 BPI, respectively.

14. A WINDOW may be closed as soon as the IMAGE request has been issued to the station. Note the I/O does not have to be completed.

SYSTEM CALL # 51 - "RETURN FROM INTERRUPT"

This call is used by a problem program to terminate an input/output interrupt routine or a problem program message interrupt routine.

ALPHA (1)	R 16	L 16	16	F 16
ALPHA (2)	N 16	ERROR EXIT VIRTUAL ADDRESS 48		

F Function Code = # 51

L Not used

R Error response field

= 1 Already at zero level

N Option

= Ø Release the current interrupt and return to the zero level at the point of interruption or take the next interrupt in the list. All zero level registers are preserved.

= 1 Release the current interrupt and make this the zero level which will be restarted at the Good Return for this system call. The zero level will be started immediately if no additional interrupts have been stacked or after the "Return From Interrupt" call has been issued for the last interrupt in the list.

SYSTEM CALL #102 - "GIVE UP CPU UNTIL I/O COMPLETES"

This call is for use with the Explicit I/O call, function code = #50 It allows the problem program to give up the CPU until all or part of its I/O is complete

<table>
<tr><td>ALPHA (1)</td><td>R
16</td><td>L
16</td><td>16</td><td>F
16</td></tr>
<tr><td>ALPHA (2)</td><td>N
16</td><td colspan="3">ERROR EXIT VIRTUAL ADDRESS 48</td></tr>
<tr><td>ALPHA (3)</td><td>BL
16</td><td colspan="3">VIRTUAL ADDRESS OF REMOTE BETA BUFFER 48</td></tr>
</table>

F — Function Code = # 52

L = # FFFF — BL contains the number of words in BETA. The rightmost 48-bits of ALPHA (3) contains the location of the remote BETA buffer.

≠ # FFFF — L contains the number of words in BETA. The BETA buffer begins in the word following ALPHA (2).

R — not used

N — Option

= 0 — Give up CPU until all I/O is complete. Note NO BETA words are required.

= 1 — Give up CPU until those I/O calls specified in BETA are complete.

BETA — Contains the virtual Bit address of the explicit I/O call, that is the location of ALPHA (1) for the system call #50 (N = ∅1)

SYSTEM CALL #52 - "GIVE UP CPU UNTIL I/O COMPLETES" cont'd.

Remarks:

1 The system only keeps a record of the I/O which is outstanding. Therefore, if any virtual address in BETA does not point to one of the problem program's I/O calls, then the system will consider that the I/O has already completed. Note it is the user's responsibility to set BETA correctly.

APPENDIX A

STAR REGISTER FILE CONVENTIONS

The register file is subdivided into five major portions.

1. Temporaries -- space used by any subroutine for temporary residence of addresses or data. This space is never saved by the caller. This space is chosen large enough to permit execution of many lowest level subroutines (such as, SIN, COS, etc.) completely within the temporary space, obviating the need for saving and restoring any of the caller's permanent registers. The choice of low numbered registers permits their use for both full and half word temporaries.

2. Globals -- registers whose contents are universal to all programs within a specific execution/language system (the constant "1", or an operating system entry point, for example). These cells are assumed by all modules within a given system and are not usually loaded by called modules. Likewise, these registers are not saved and restored by program modules. The values in these registers are unique to a given operating environment, thus if a module from a different environment is to be called, it is the caller's responsibility to establish the correct values for the callee in the proper registers.

The total (temporaries + globals) space begins at register 3 and continues through register 19. Thus the number of temporaries available is dependent upon the number of globals required by a specific environment.

The Global registers defined by LLL are:

14 Contains constant #800 -- Used for initializing the register file during a programs prologue (register #20).

15 Contains constant #680 -- Used for saving the register file's environment and working registers (register #1A).

16 Contains the constant 1.

17 Parameter descriptor -- Contains the number of the parameters being passed during a call. The number is contained in the length portion. The address portion contains zero if the parameters are in the register file, and the address of the parameter list if the parameters are in memory.

18 Function return -- The function return value is a two word

19 Pair (See parameters).

3. Environment -- The environment registers consist of the minimum set of registers needed to support the general requirements of recursive, re-entrant execution with dynamic linking. The environment registers are . . .

1A Return register -- contains the bit address of the location in the caller to which the calle normally returns.

1B Dynamic space pointer -- contains the bit base address of the next available free location in the dynamic stack. (In an ascending sequence from the value of the dynamic space pointer can be found the only known unused space for stacking or allocating dynamic data). The dynamic space pointer is always advanced prior to storing data into the region or before addresses pointing to that region are calculated.

1C Current stack pointer -- contains the bit base address of the region in the dynamic stack for storing the register file. The minimum length of that region will be the maximum number of registers that the caller will need to have saved, plus the length of the region required for dynamic working storage for the program. During call sequences the caller will set the length portion of the current stack pointer to the number of registers to be saved by the callee. The current stack pointer is set up by the caller, but the registers are saved by the callee. The minimum number of registers that the caller can indicate are to be saved is the number of environment registers (six).

1D Previous stack pointer -- contains the bit base address and the number of registers where the caller's registers have been saved. The callee's previous stack pointer is an exact copy of the caller's current stack pointer.

1E Callee data base -- contains the bit base address of the static space which was allocated to the module by the loader. The caller passes the callee the address of the callee's static space in the callee data base register. If, at the time of the call, the caller has not been linked to the callee by the loader, the value of the callee data base will be the data base address of the loader. The exponent portion of the callee data base register will contain an ordinal used by the loader to determine which module is making the call.

1F On unit -- contains the bit base address of a stack of data in dynamic space which defines the action to be taken by interrupt and error handling routines for a given set of pre-defined conditions for the active modules. The regis' must be stored at each call in order to support the execution requirements of condition handling in block structured languages such as PL/I.

4. Register save area -- begins at register 1A and thus contains the environment registers. It defines the space to be saved and restored by called processors and therefore is the space wherein permanent variables and addresses would be allocated. The length of this area is dependent upon the usage of this area by the caller. The allocation of the environment registers at the beginning of the space ensures that they will appear at the beginning of every stack, thus facilitating unstacking or stack searching processes needed for block structured languages as well as non-standard Fortran call/return usage.
The working registers are the portion of the register save area that does not include the environment registers.
LRLTRAN has reserved two working registers for further environment information.

20. Program name - ASCII (left adjusted) name of the program currently in execution.

21 Current data base - contains the bit base address of the current executing programs data base. Upon entry into a program, the callee data base register (#IE) is copied in the current data base register.

5. Temporary -- these registers consist of those registers that are not to be saved and restored and do not contain parameters.

6. Parameters -- to permit a varying number of parameters to be passed via the register file (depending on the execution environment), the parameters are assigned from register FF upwards towards the end of the register save area. All parameters are either passed in the parameter section of the register file or in memory outside the register file area. As all other registers are accounted for, no registers other than the parameter registers may be used for passingparameters and values.

The allocation of the parameter registers in the above manner allows the parameters to be passed in even/odd register pairs. LRLTRAN makes reference to only the even part of the register pair.

The register pair will allow parameter passing of the following form:

A. Passing base addresses and offsets or pointer pairs for sparse vectors.

B. Passing type double or complex parameters.

By assigning the parameters backwards from FF the compiler can define the boundary he will accept dividing permanent registers from parameter registers. (LRLTRAN on the 7600 allows only a maximum of 5 arguments to be passed through the register file. Thus the maximum permanent register is #F5). Thus if the maximum number of parameters expected still leaves sufficient permanent registers for execution, all such parameters can be passed through the register file. Note that since the callee knows the use to which a parameter is going to be put (i. e., he will use only the value portion of the register pair), he may utilize one of the registers for temporary calculations.

REGISTER FILE LAYOUT

1		Machine Zero	
2		Data Flag Return	
3 . . . 13		Always Temporary	
14	O	#800	WR
15	O	#680	SR
16	O	1	
17		Parameter Descriptor	PD
18 19		~~FUNCTION RETURN~~	
1A		RETURN	RET
1B		Dynamic Space Pointer	DSP
1C		Current Stack Pointer	STACK
1D		Previous Stack Pointer	OLD-STACK
1E		Callee Data Base	LINK
1F		On Unit Stack Pointer	ON
20		Program Name	PRG
21		Current Data Base	OLD-LINK
22		Working Registers	
. . . F5		Temporary Registers	
F6 . . . FF		Parameter Registers	

APPENDIX B

OBJECT MODULE FORMAT FOR STAR

1) General Table Structure

An object module consists of a number of standard tables. Each of these tables begins with a stanardd two word header of the following form:

<table>
<tr><td colspan="2">ASCII Table Name</td></tr>
<tr><td>Length</td><td>Back Pointer</td></tr>
</table>

Word 1 -- The ASCII name of the table.

Word 2 -- (Length field) full word length of the table.

(Address field) pointer to the header table.

(relative with respect to the respective table).

2) Module Header Table

The module header is a standard table giving general information concerning the object module and providing a linkage to all the other tables in the module. The module header table is logically the primary table in the module.

ASCII Table Name	
Length	0
Module Name	
Date + Time Created	
T Leng	Processor
C Leng	Data Base Length
Type	Pointer
Type	Pointer

Word 1 -- The ASCII name of the table = ''$_\Lambda$MODULE$_\Lambda$''.

Word 2 -- The length of the table (length portion).

A back pointer of 0 (address portion).

Word 3 -- The ASCII name of the module, eight-character, left adjusted, and blank-filled.

Word 4 -- The date and time the module was created. This information is in packed decimal with a positive sign. The date and time are in this order: year, year, month, month, day, day, hour, hour, minute, minute, second, second, millisecond, millisecond, millisecond.

Word 5 -- The word length of the tables excluding the code. The ASCII name of the processor that created the module.

Word 6 -- The word length of the code. The bit length of the data base area.

Word 7 on -- Each word contains a table type and a pointer to a table of that type. The type is contained in the length portion. The pointer contains a bit address relative to the first word address of the header. By convention the first table described is the code, and the second is the external/entry table. If HEX type is "4", the pointer contains a bit address to the next module header.

Table Types --

HEX type	ASCII Name	Description
0001	"CODE"	Code Block Table
0002	"EXT ENTR"	External/Entry Table
0003	"REL CODE"	Code Relocation Table
0006	"SYMB TAB"	Debug Symbol Table
0101	"INT DATA"	Interpretive Data Initialization Table
0201	"INT RELO"	Interpretive Relocation Initialization Table

3) Code Block Table

The code consists of a standard table whose contents is the executable code.

ASCII Table Name	
Length	Back Pointer
Code	

Word 1 -- ASCII table name "CODE".

Word 3 on -- The code.

4) Code Relocation Table

This table describes relocation in the code itself.

ASCII Table Name	
Length	BackPointer
Current Base	
NBI	NI
I1, I2, I3, . . . IN	

Word 1 -- ASCII Table name "REL CODE".

Word 3 -- Current base - Current bit address at which this module is relocated

Word 4 -- NBI - The number of bits per index in the bit string starting in word 5, NI = The number of indices in the string.

Word 5 -- A bit string of indices, each is NBI bits long. Each index references a half word in the code to be relocated relative to the base address of the code.

As the result of processing this table, the bit base address of the code will be added to the 48 bit fields pointer to by the indices in the bit string.

5) External/Entry Table

The external/entry table contains definitions for all entry point, external symbols, and common blocks.

ASCII Table Name	
Length	Back Pointer
M	N
Entry Name 1	
Entry Name 2	
Entry Name M	
External Name 1	
External Name 2	
External Name (N-M)	
Entry Descriptor 1	
Entry Descriptor 2	
Entry Descriptor M	
External Descriptor 1	
External Descriptor 2	
External Descriptor (N-M)	

Word 1 -- ASCII Table Name "EXT ENTRY".

Word 3 -- M = number of entry point names in the table.

N = number of names in the table.

Word 4 through 3+N -- List of entry point names.

Word 4+N through 3+M -- List of external names.

Word 4+M through 3+M+N -- List of entry point descriptors.

Word 4+M+N through 3+M+M -- List of external descriptors.

The following types of entry points and external symbols are defined:

Entry Points

An entry point is a named value defined in the procedure, and is intended to be referenced as an external by an external procedure.

Common Blocks

A common block is a named alterable space referenced by one of more proceudres. A common block can be initizlized with relocatable data. Blank common is a common block with name of eight blanks.

External Procedure

The standard method of using an external procedure reference is in a call.

Having a symbol multiply defined as a common block and external procedure is specifically allowed.

All names are eight character, left-adjusted, and blank filled.

Each descriptor is of the following form:

Type	Value

The type field defines the HEX type of the symbol.

The value field contains information associated with the symbol.

Type = 1 [Entry point in code]. Value = a relative bit address in the code.

Type = 14 [External procedure]. Value = 0.

Type = 16 [Common block]. Value = bit length of the common block.

6) Interpretive Data Initialization Table

The interpretive data table contains information that, when processed by the loader, results in the initialization of areas of static space.

ASCII Table Name	
Length	Back Pointer
Data Item Descriptor Data Item	
Data Item Descriptor Data Item	
Data Item Descriptor Data Item	

Word 1 -- ASCII Table name "INT DATA".

Word 3 on -- Data item descriptor and item pairs.

0 15	16 31	32 39	40 63
ORDI	ORD2	Type	Chain

Data Descriptor Format

The data item descriptor contains the following fields:

ORDI -- The pseudo address vector ordinal of the static space to be initialized.

ORD2 -- The pseudo address vector ordinal relative to which relocation is to be done (relocation base).

Type -- Indicates what type of data item follows.

Chain -- Relative full word count to next data item descriptor (if any) in the table.

When ORD 2 is zero, the values in the item are stored directly into the destination field described by ORD1. If ORD2 is not zero, the relocation base described by ORD2 is added to the values before they are stored into the destination field described by ORD1.

Leng	Relative Address
Values	

Format 1

Leng	Relative Address
Value	
Leng2	Bit String

Format 2

Leng	Relative Address
Value	
NBI	NI
Bit String	

Format 3

Leng	Relative Address
Type	Number of Descriptors
Desc 1	Desc 2
- - -	Desc N
Value	

Format 4

The kinds of initialization area:

Type	Description	Data Item Format
1	Full Word Broadcast	1
2	Half Word Broadcast	1
3	Full Word Vector Transmit	1
4	Half Word Vector Transmit	1
5	Full Word Sparse Vector	2
6	Half Word Sparse Vector	2
7	Full Word Index List	3
8	Half Word Index List	3
9	Byte String	1
A	Bit String	1
B	Sparse Structure	4
C	Character Broadcast	1

<u>Full Word Broadcast</u>

Data Item Type -- 1

Item Format -- 1

Length -- Full word vector length

Value -- A full word to be stored in consecutive full words starting at the relative address in the section of static space.

Half Word Broadcast

Data Item Type -- 2

Item Format -- 1

Length -- Half word vector length

Value -- A left-adjusted half word to be stored in consecutive half word locations starting at the relative bit address.

Full Word Vector Transmit

Data Item Type -- 3

Item Format -- 1

Length -- Full word vector length

Values -- Full word vector to be transmitted to the relative address in the control section.

Half Word Vector Transmit

Data Item Type -- 4

Item Format -- 1

Length -- Half word vector length

Values -- Half word vector to be transmitted to the relative address in the control section.

Full Word Sparse Vector

Data Item Type	--	5
Item Format	--	2
Length	--	Number of values in item
Values	--	Full word values
Leng2	--	Length of control vector
Bit String	--	Control vector of length leng2

Half Word Sparse Vector

Data Item Type	--	6
Item Format	--	2
Length	--	Number of values in item
Values	--	Left Adjusted half word vector
Leng2	--	Length of control vector
Bit String	--	Left adjusted control vector.

Full Word Index List

Data Item Type	--	7
Item Format	--	3
Length	--	Number of values in item
Values	--	Full word values
NBI	--	Number of bits per index
NI	--	Number of indices
Bit String	--	A bit string of NI indices, each index is NBI bits long and contains a full word count.

Half Word Index List

Data Item Type 8

Format -- 3

Length -- Number of values in item

Values -- A left-adjusted half word vector

NBI -- Number of bits per index

NI -- Number of indices

Bit String -- A bit string of indices, each index is NBI bits long and contains a half word count.

Byte String

Data Item Type -- 9

Item Format -- 1

Length -- Number of bytes in value field

Values -- A left-adjusted byte string.

Bit String

Data Item Type -- A

Item Format -- 1

Length -- Number of bits in value field.

Values -- A left-adjusted bit string.

Sparse Structure

Data Item Type -- B

Item Format -- 4

Length -- Number of items is value field

Data Type -- Type of value (word, half word, byte string, bit string).

ND -- Number of descriptors

Descriptor -- Half Word descriptors

The details of this structure will be defined later.

Character Broadcast

Data Item Type -- C

Item Format -- 1

Length -- Length of byte string to be filled with a character.

Value -- Left byte contains a character

7) Interpretive Relocation Initialization Table

ASCII Table Name	
Length	Back Pointer
Relocation Item 1	
Relocation Item 2	
- - - - -	
Relocation Item N	

Word 1 -- ASCII Table name "INT RELO".

Word 3 on -- Relocation items - item formats are similar to data initialization table formats but do not contain values.

8) Debug Symbol Table

ASCII Table Name	
Length	Back Pointer
Debug Symbol Table	

Word 1 -- ASCII Table name "SYMB TAB".

9) Pseudo Address Vector - ordinal description

	0
Code Address	1
Data Base Address	2
External Address 1	3/4
External Address 2	5/6
	7/8
External Address N	2N+1, 2N+2

For Common

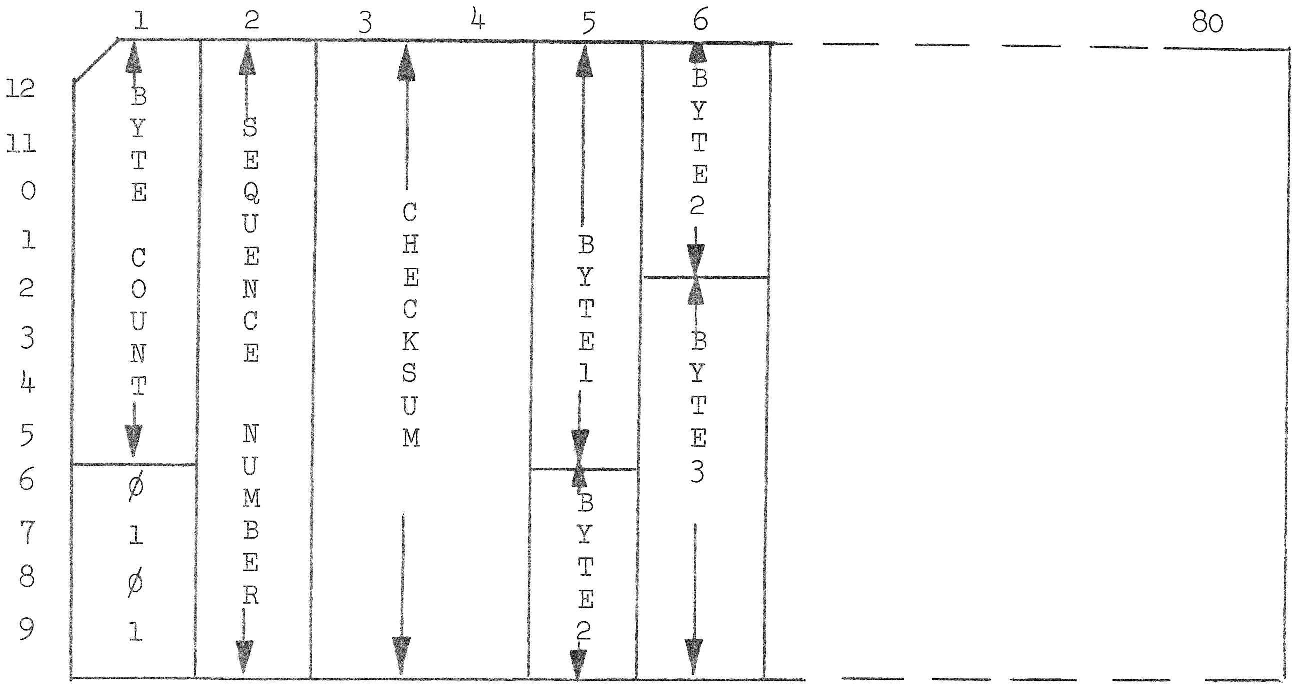
0	Address
0	Bit Length

For External Procedure

EORD	Entry Address
MORD	Data Base

APPENDIX C

STAR BINARY CARD FORMAT



BYTE COUNT is the number of 8-bit bytes starting in column 5.

SEQUENCE NUMBER is the sequence number of the card starting from 1.

CHECKSUM is the 24-bit arithmetic sum of the 8-bit data bytes.

BYTE1, BYTE2 are the 8-bit data bytes.

EOF is a card with 6-7-8-9 punches in column 1.

APPENDIX D

CREATED PAGES

Pages created in core will be initialized with the following pattern:

ØØØC1F1C (HEX) To be written into each half word

The leading zeroes will result in an interrupt to monitor mode if an attempt is made to execute the word. The 1F is an end-of-line sentinel, and 1C is an end-of-file sentinel for any of the out-put routines.

Definition of a "Created Page":

1. The user faults during execution for a virtual address not defined previously in his bound virtual map or drop file map.

2. The virtual address is found in the drop file map but is not in core, on drum or on disc. A bit in the drop file map will be "on" if it is on disc. These addresses were probably entered in the map via system call -04- "map in call."

3. One of the above two conditions occurred during an "advice call" - system call 07-.

Note: The act of writing the pattern will not constitute a "modified" page. The page becomes "modified" only when the hardware detects a write during "job mode" execution.

APPENDIX E

FATAL USER ERRORS FROM FAULT PROCESSOR

The error number will be found in the minus page along with the associated virtual bit address where applicable - in word 139 (10).

Error # (Hex) Message and Meaning

25 Virt. Add Dup. Advice Call

When making an advice call one specifies large or small pages. Upon finding one of the virtual addresses either in core, drum, drop map or virtual map the page size found did not match the page size specified.

27 WOP. Violation in System Call

In processing a system call, the system faulted for a user's page to write into it. The page was found to have a read/only protection.

28 WOP. Violation Direct Fault

During program execution the user attempted to write into a page with read/only protection.

29 Out of Bound Memory Reference

During program execution the user attempted to reference virtual address space reserved for the system (i.e., the upper quarter of virtual address space). System conventions as to address space are:

Ø to 2^{47} - 1 user space

2^{47} - 1 to 2^{48} - 1 lower half shared library space upper half system space

Drop File Map/Disc Overflow

When attempting to append pages to the drop file either the map was full or the physical disc space was full. The page fault routine appends pages to the drop file under the following circumstances:

1. The page wanted is here-to-for undefined space and so a page(s) is created in core.
2. A first write attempt is made into a source file page or a page from a "write temporary" file.

ERRORS DETECTED BY FILE MANAGEMENT

#2ø9 NO SOURCE FILE (non-fatal)

#21ø NO DROP FILE (non-fatal)

APPENDIX F

STAR CHARACTER SET

DEC.	OCTAL	HEX	USASC	KEYPUNCH	DEC.	OCTAL	HEX	USASC	KEYPUNCH
Ø	Ø	Ø	NUL	NONE	64	1ØØ	4Ø	@	¬
1	1	1	SOH		65	1Ø1	41	A	A
2	2	2	STX		66	1Ø7	42	B	B
3	3	3	ETX		67	1Ø3	43	C	C
4	4	4	EOT		68	1Ø4	44	D	D
5	5	5	ENQ		69	1Ø5	45	E	E
6	6	6	ACK		7Ø	1Ø6	46	F	F
7	7	7	BEL		71	1Ø7	47	G	G
8	1Ø	8	BS		72	11Ø	48	H	H
9	11	9	HT		73	111	49	I	I
1Ø	12	A	LF		74	112	4A	J	J
11	13	B	VT		75	113	4B	K	K
12	14	C	FF		76	114	4C	L	L
13	15	D	CR		77	115	4D	M	M
14	16	E	SØ		78	116	4E	N	N
15	17	F	S1		79	117	4F	O	O
16	2Ø	1Ø	DLE		8Ø	12Ø	5Ø	P	P
17	21	11	DC1		81	121	51	Q	Q
18	22	12	DC2		82	122	52	R	R
19	23	13	DC3		83	123	53	S	S
2Ø	24	14	DC4		84	124	54	T	T
21	25	15	NAK		85	125	55	U	U
22	26	16	SYN		86	126	56	V	V
23	27	17	ETB		87	127	57	W	W
24	3Ø	18	CAN		88	13Ø	58	X	X
25	31	19	EM		89	131	59	Y	Y
26	32	1A	SUB		9Ø	132	5A	Z	Z
27	33	1B	ESC		91	133	5B	[	[
28	34	1C	FS		92	134	5C	\	|
29	35	1D	GS		93	135	5D	]	]
3Ø	36	1E	RS		94	136	5E	∩	^
31	37	1F	US		95	137	5F	-	→
32	4Ø	2Ø	□	SPACE	96	14Ø	6Ø	"	NONE
33	41	21	!	∇	97	141	61	a	
34	42	22	"	Δ̲	98	142	62	b	
35	43	23	#	D̄	99	143	63	c	
36	44	24	$	$	1ØØ	144	64	d	
37	45	25	%	π	1Ø1	145	65	e	
38	46	26	&	&	1Ø2	146	66	f	
39	47	27	'	~	1Ø3	147	67	g	

STAR Character Set (continued)

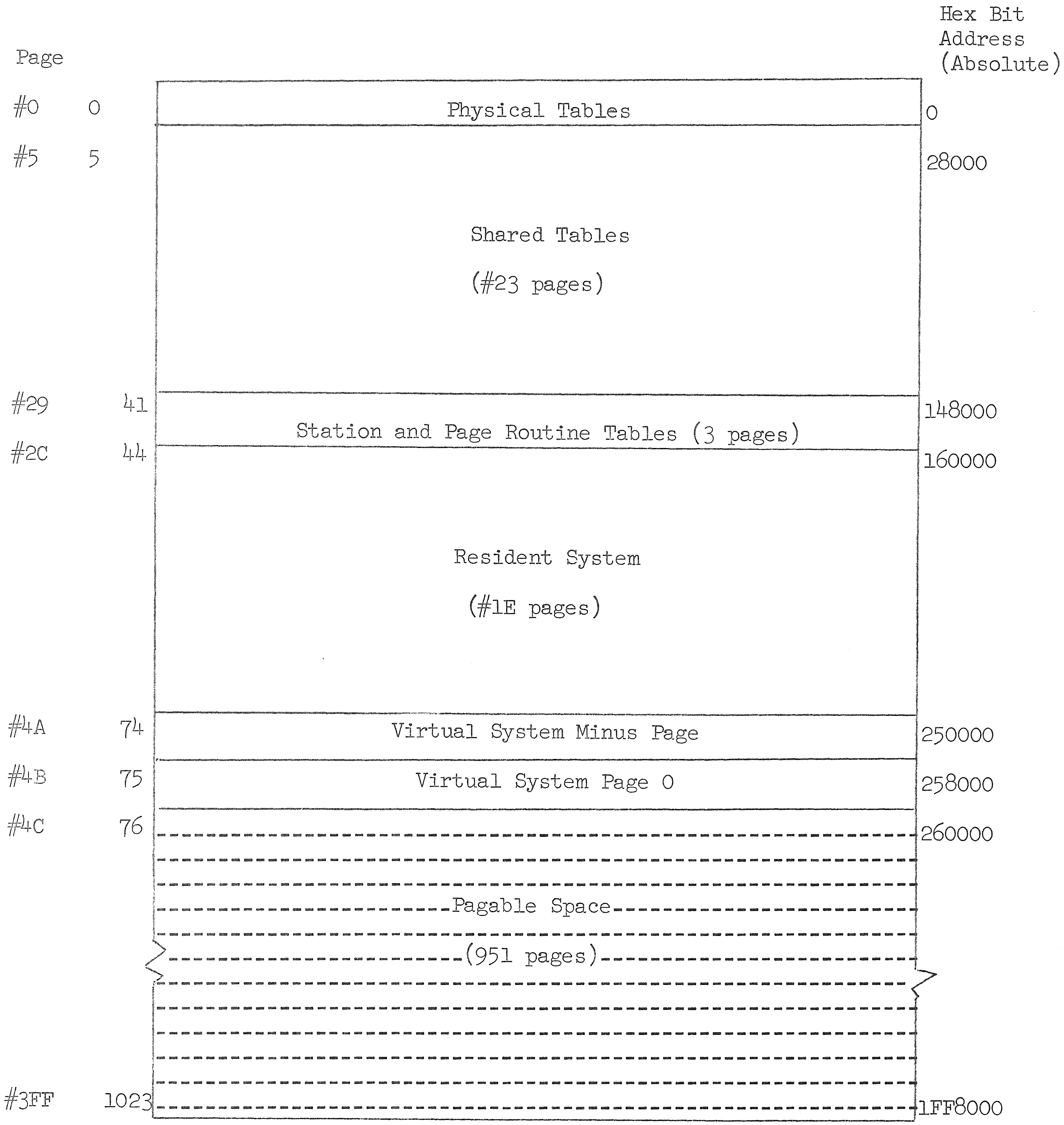
DEC	OCTAL	HEX	USASC	KEYPUNCH	DEC	OCTAL	HEX	USASC	KEYPUNCH
4Ø	5Ø	28	(	(	1Ø4	15Ø	68	h	NONE
41	51	29	)	)	1Ø5	151	69	i	
42	52	2A	*	*	1Ø6	152	6A	j	
43	53	2B	+	+	1Ø7	153	6B	k	
44	54	2C	,	,	1Ø8	154	6C	l	
45	55	2D	-	-	1Ø9	155	6D	m	
46	56	2E	.	.	11Ø	156	6E	n	
47	57	2F	/	/	111	157	6F	o	
48	6Ø	3Ø	Ø	Ø	112	16Ø	7Ø	p	
49	61	31	1	1	113	161	71	q	
5Ø	62	32	2	2	114	162	72	r	
51	63	33	3	3	115	163	73	s	
52	64	34	4	4	116	164	74	t	
53	65	35	5	5	117	165	75	u	
54	66	36	6	6	118	166	76	v	
55	67	37	7	7	119	167	77	w	
56	7Ø	38	8	8	12Ø	17Ø	78	x	
57	71	39	9	9	121	171	79	y	
58	72	3A	:	:	122	172	7A	z	
59	73	3B	;	;	123	173	7B	{	
6Ø	74	3C	<	<	124	174	7C	¦	
61	75	3D	=	=	125	175	7D	}	
62	76	3E	>	>	126	176	7E	∿	
63	77	3F	?	α	127	177	7F	DEL	

APPENDIX G

STAR MEMORY LAYOUT   Initial System

The first #5 pages of central memory will contain tables accessible to the central system. Shared tables will start at physical word address #ADD. They will occupy #23 pages in the initial system. These pages will be entered in the page table starting at word address #30000000000, i.e., the beginning of the upper quarter of memory , and will occupy #4800 words. The next 3 pages of physical memory will contain read/write areas shared between the Kernel, stations, and pager tables. Following this will be the resident system {Kernel and Pager}. This is expected to occupy #1E pages of memory. The remainder of memory will be available for user and virtual system pagable space. In order to allow for possible expansion of shared tables, the origin address for the possible expansion of shared tables, the origin address for the virtual system will be word address #30000008000. #100 pages will be reserved on the system drum for system pages; this puts the upper end of the system range at word address #30000027FFF. Any pages at addresses greater than this referenced by the system should be created by the system referencing them, locked down while in use, and then unlocked. The first 128 pages of this range are reserved for the minus page table, with one entry per DB. The next 128 pages are reserved for VPZTAB with one entry per DB. System I/O buffers will be assigned in the following space as needed.

PHYSICAL MEMORY LAYOUT



VIRTUAL SYSTEM MEMORY LAYOUT

# Page		Virtual Word Address
#0	VIRTUAL SYSTEM PAGE 0	00000000000
		00000000200
#1	USER SPACE	
#100000000	LIBRARY SPACE	20000000000
#180000000	SHARED TABLES	30000000000
#180000024	VOID	30000004800
#180000040	PAGABLE SYSTEM	30000008000
		30000028000
#180000140	MINUS PAGE TABLE	
#1800001C0	MINUS PAGE SYSTEM BUFFER	
		30000038000
#1800001C1	PAGE 0 TABLE	30000038200
#180000241	FREE SPACE	30000048200
#1FFFFFFFF		#3FFFFFFFE00

APPENDIX H

STAR OP CODES AND MNEMONICS

	ØX MONITOR	2X BRANCH	4X HALF REGISTER	6X FULL REGISTER	8X VECTOR	AX SPARSE VECTOR	CX SPECIAL VECTOR	EX BYTE-BIT	
ØØ	IDLE IDLE	JEQH JUMP = HALF	AUF ADD UPPER	AUF ADD UPPER	AUV ADD UPPER	AUS ADD UPPER	FEQ SELECT FIRST =	AB ADD BINARY	EØ
Ø1	**	JNEH JUMP ≠ HALF	ALH ADD LOWER	ALF ADD LOWER	ALV ADD LOWER	ALS ADD LOWER	FNE SELECT FIRST ≠	SB SUB. BINARY	E1
Ø2	**	JGEH JUMP ≥ HALF	ANH ADD NORM.	ANF ADD NORM.	ANV ADD NORM.	ANS ADD NORM.	FGE SELECT FIRST ≥	MB MULT. BINARY	E2
Ø3	**	JLTH JUMP < HALF	**	AXF ADD INDEX	AXV ADD INDEX	**	FLT SELECT FIRST <	DB DIV. BINARY	E3
Ø4	BKPT BREAKPOINT	JEQF JUMP = FULL	SUH SUB UPPER	SUF SUB UPPER	SUV SUB. UPPER	SUS SUB. UPPER	GCEQ COMPARE = ORD.	AD ADD DECIMAL	E4
Ø5	**	JNEF JUMP ≠ FULL	SLH SUB LOWER	SLF SUB LOWER	SLV SUB. LOWER	SLS SUB. LOWER	GCNE COMPARE ≠ ORD.	SD SUB. DECIMAL	E5
Ø6	FAULT FAULT TEST	JGEF JUMP ≥ FULL	SNH SUB NORM.	SNF SUB NORM.	SNV SUB. NORM.	SNS SUB. NORM.	GCGE COMPARE ≥ ORD.	MD MULT. DECIMAL	E6
Ø7	**	JLTF JUMP < FULL	**	SXF SUB INDEX	SXV SUB. INDEX	**	GCLT COMPARE < ORD.	DD DIV. DECIMAL	E7
Ø8	SETCF SET CHAN. FLAG	FEQC FIRST = CHAR.	MUH MULT. UPPER	MUF MULT. UPPER	MUV MULT. UPPER	MUS MULT. UPPER	GXEQ SEARCH = INDEX	KB COMPARE BINARY	E8
Ø9	EXIT EXIT FORCE	FNEC FIRST ≠ CHAR.	MLH MULT. LOWER	MLF MULT. LOWER	MLV MULT. LOWER	MLS MULT. LOWER	GXNE SEARCH ≠ INDEX	KD COMPARE DECIMAL	E9
ØA	TIMERM SET MTR. TIMER	LQL LOAD LENG. 16	**	**	**	**	GXGE SEARCH ≥ INDEX	MASKC MERGE/ BYTE MASK	EA
ØB	**	CL INC. LENG. 16	MSH MULT. SIG.	MSF MULT. SIG.	MSV MULT. SIG.	MSS MULT. SIG.	GXLT SEARCH < INDEX	EDIT EDIT MARK	EB
ØC	STAR STORE AR's	**	DUH DIV. UPPER	DUF DIV. UPPER	DUV DIV. UPPER	DUS DIV. UPPER	**	AC MODULO ADD	EC
ØD	LDAR LOAD AR's	**	LQH LOAD IMM.	**	**	**	LIH LOAD IMM. HALF 24	SC MODULO SUB.	ED
ØE	XLTINT XLATE EXT. INT.	**	AQH INC. IMM.	**	**	**	AIH INC. IMM. HALF 24	TAB TRANSLATE	EE
ØF	XLTADR XLATE VIRT. ADD	**	DSH DIV. SIG.	DSF DIV. SIG.	DSV DIV. SIG.	DSS DIV. SIG.	FUZZ ARITH. COMPRESS	TABT TRANSLATE TEST	EF
1Ø	RDB DEC. TO BIN.	**	INTH TRUNCATE	INTF TRUNCATE	INTV TRUNCATE	JEQX JUMP = INDEX	HSUM AVERAGE	XOR EXCLUSIVE OR	FØ
11	RBD BIN. TO DEC.	JIX JUMP INC. INDEX	ILH FLOOR	ILF FLOOR	ILV FLOOR	JNEX JUMP ≠ INDEX	MEAN ADJ. MEAN	AND AND	F1
12	LC LOAD CHAR.	JB BIT BRANCH ALT	IUH CEILING	IUF CEILING	IUV CEILING	JGEX JUMP ≥ INDEX	**	OR OR	F2
13	STC STORE CHAR.	JF DFR BRANCH ALT	ROOTH SIG. SQRT.	ROOTF SIG. SQRT.	ROOTV SIG. SQRT.	JLTX JUMP < INDEX	**	NAND NOT AND	F3
14	COMPB COMPRESS BITS	**	CSH ADJ. SIG.	CSF ADJ. SIG.	CSV ADJ. SIG.	JLEX JUMP ≤ INDEX	HDIF AVE. DIFFERENCE	NOR NOT OR	F4
15	MERGB MERGE BITS	JDX JUMP DEC. INDEX	CEH ADJ. EXP.	CEF ADJ. EXP.	CEV ADJ. EXP.	JGTX JUMP > INDEX	DELTA DIFFERENCE	IMPL IMPLICATION	F5
16	MASKB MASK BITS	JSX JUMP SAVE ADD.	**	RFH CONTRACT	RFHV CONTRACT	J JUMP	FMB SEARCH MASK BIT	INHB INHIBIT	F6
17	MERGC MERGE CHARS.	**	**	RFHN CONTRACT ROUND	RFHNV CONTRACT ROUND	SOW XMIT INDEX DEST	TABM XLATE MARK	IFF IF AND ONLY IF	F7
18	RCR BYTES RIGHT	RL REP. LENG.	RH XMIT HALF	RF XMIT	RV XMIT	REV XMIT REVERSE	MAX MAXIMUM	RC BYTES LEFT	F8
19	FRNEC SCAN RIGHT	CLOCK READ CLOCK	RAH XMIT ABS.	RAF XMIT ABS.	RAV XMIT ABS.	TRANS XPOSE MOVE	MIN MINIMUM	RKC BYTES LEFT COMP.	F9
1A	FFCI FILL CHAR. IMM.	TIMER SET TIMER	UH EXP R→T	UF EXP R→T	UV EXP A→C	REAP XMIT INDEX SOURCE	SIGMA VECTOR SUM	SCALE MOVE AND SCALE	FA
1B	FFC FILL CHAR.	FLAG DFR LOAD/STORE	PH PACK	PF PACK	PV PACK	MASKV MASK VEC.	PROD VECTOR PRODUCT	RZD ZONED TO DECIMAL	FB
1C	GMZ GEN. MASK Ø	MXH MULT. INDEX HALF	RHF EXTEND	UL XMIT LENGTH	RHFV EXTEND	COMPV COMPRESS VEC.	DOTV VECT. DOT PROD.	RDZ DECIMAL TO ZONED	FC
1D	GMB GEN. MASK 1	MXF MULT. INDEX FULL 16	RHX EXTEND INDEX	**	**	MERGV MERGE VEC.	DOTS SPRS. DOT PROD.	KC COMPARE CHAR.	FD
1E	CLE COUNT LEAD =	LQF LOAD IMM. FULL 16	LH LOAD	LF LOAD	**	LIF LOAD IMM. FULL 48	POLY POLYNOMIAL EVAL.	FMC SEARCH MASK	FE
1F	BSIGMA COUNT 1's	AQF INC. IMM. FULL	STH STORE	STF STORE	**	AIF INC. IMM. FULL 48	IOTA INTERVAL	FMC SEARCH MASK	FF
	1X BYTE-BIT	3X BRANCH	5X HALF REGISTER	7X FULL REGISTER	9X VECTOR	BX BRANCH/VECTOR	DX VECTOR MACRO	FX BIT/BYTE	

APPENDIX I

STAR SUBROUTINE LINKAGE CONVENTIONS

1) Call Sequence

The following sequence will be used as a standard call of an external procedure:

78YY00LK Load the link register with the address of the data base to be invoked.

36RR00EP Branch to the entry point of the called procedure and set a return location.

Where:

ST = Stack Register

LK = Link Register

RR = Return Register

EP = Entry Point Value

In the static case, LK and EP contain resolved address. However, to support dynamic linkage, LK will initially reference the data base of the loader, and EP will initially reference the entry point to the loader. The exponent of EP will contain the entry ordinal (EORD) of the caller and the exponent of LK will contain the module ordinal (MORD) of the caller.

EP	EORD.	ENTRY ADDRESS
LK	MORD .	DATA BASE ADDRESS

Note that LK is a canonical register while EP is not.

If the load is invoked because of an unresolved call, the following can be determined:

1. The exponent of the link register (LK) contains the module ordinal (MORD) of the calling procedure within the catalog of all resolved modules.

2. The "R" register of the "78" instruction preceding the value of return (see call sequence) tells the loader the register from which LK was loaded.

3. The "T" register of the "36" instruction preceding the value of return (see call sequence) tells the loader the register containing the entry ordinal (EORD) and entry address.

Using MORD, the identity and location of the caller may be determined. Applying EORD to the external list of the caller, allows the name of the entry to be invoked, to be determined. The loader must perform the resolution of the requested entry, update the data base of the caller, and update both EP and LK which are contained in the register file.

2) Prologue (entry) Sequence

The prologue of the called procedure has the following responsibilities:

```
38ST00SR                    Save the status of the register

980000SR                    file in the stack frame of the

000000ST                    caller.  (This is a store back).



 78ST00OS

78DP00ST                    Advance stack pointers

3FDPXXXX


2ASTXXXX                    Set number of registers to be

                            saved by a called program.


38LK00WR                    Block load register file with

980000LK                    initial values using LK as the

 000000WR                   source descriptor.
```

Where:

ST = Stack Register

SR = #680 Register

DP = DSP Register

OS = Old Stack Register

WR = #800 Register

3) Epilogue (return) Sequence

The epilogue of the called procedure shall be as follows:

```
38OS00SR
980000OS
000000SR
334000RR
```

A non-normal return will be carried out in a similar fashion except that the values of OS and RR will be obtained from known variables.

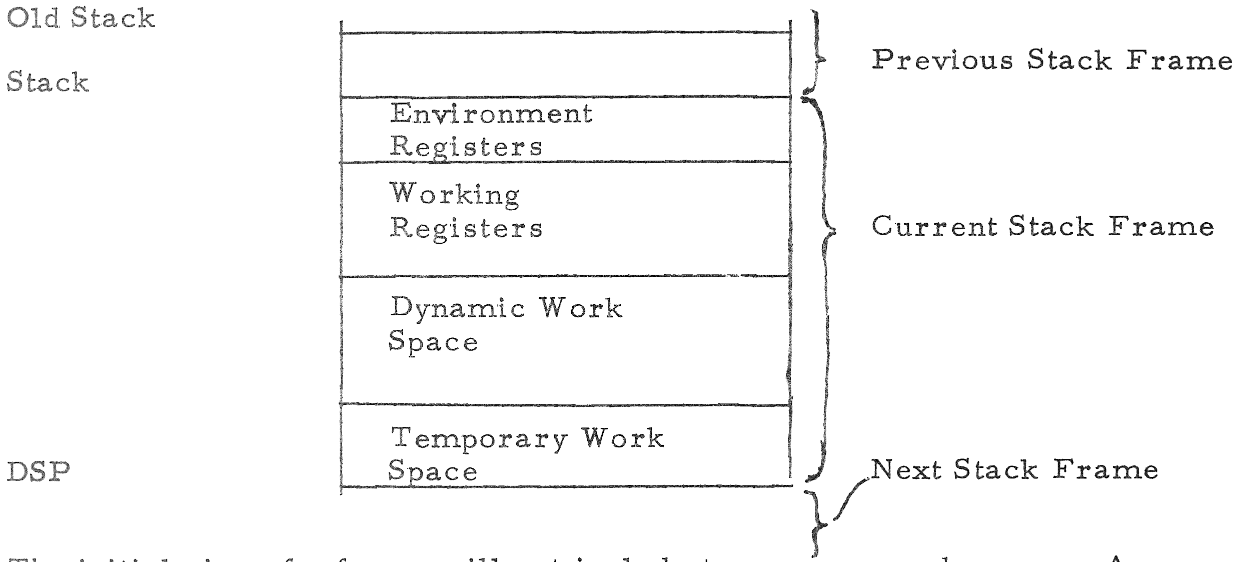
4) Rationale

1. Procedures may be statically or dynamically linked.

2. Addresses that are established at execute time are stored in static space and the procedure itself is not modified.

   Hence, the procedure may be maintained in write protected storage.

3. The mechanism for saving and restoring the register file is a conventional chained stack. This also allows for the creation of dynamic storage for block structured languages such as PL/I and ALGOL. Note that the environment registers are saved beginning at the top of a stack frame (prologue of caller). Thus a stack frame appears as follows:



The initial size of a frame will not include temporary work space. Any time temporary work space is needed, the program can increment the DSP (integral number of words) and obtain space. An entire frame disappears when returning to a calling program.

4. When an interrupt occurs, the entire register file must be saved.

This can be accomplished by obtaining a save area for 256 registers beginning at DSP. Resuming requires the entire register file to be restored and hence is not a normal return.

5. Note that the number of registers to be saved is set by the caller. The actual save is performed in the prologue of the callee (store back).

If the caller can do all its work within temporary (clobberable) registers, the registers need not be saved. This is only true for the lowest level module which will never invoke other modules.

DISTRIBUTION

LLL Internal Distribution

J. Requa	50
Addressograph F	848
TID File	75

External Distribution

W. Sembrat Control Data Corporation Livermore, California	25
Technical Information Center Oak Ridge, Tennessee	2

NOTICE

"This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights."

102664987

RAC/lh

☆ GPO 791-895