Burroughs

B 6500

MASTER CONTROL PROGRAM

REFERENCE MANUAL

Burroughs Corporation
Detroit, Michigan 48232

$4.00

Printed in U.S. America

8-69

1042447

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Sales Technical Services, Burroughs Corporation, 6071 Second Avenue, Detroit, Michigan 48232.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

INTRODUCTION

In the early days of computer programming, each program was expected to perform its own input/output operations. With the advent of symbolic assembly languages, it became apparent that independent input/output routines for programs were time consuming to write and normally required the same functions. This led to the development of operating systems with input/output routines to simplify the programming task and avoid unnecessary duplication of efforts.

Eventually, it was realized that the relatively slow I/O operations could be made to occur in parallel with computation and thereby reduce the time required to complete a given job. This resulted in the incorporation of buffered I/O routines in the operating system for both assembly languages and the higher level "problem oriented" languages.

Even with buffered I/O, the Central Processing Unit of most computers still spent a significant percentage of time waiting for an I/O operation to be completed in order to continue processing data. Rather elaborate attempts were made by programmers to overlap the processing of data with the I/O operations. Many times, however, due to the nature of the data processing application, complete utilization of the computer was still an unrealizable goal.

At about the same time, computers were becoming fast enough (and expensive enough) to cause concern over the amount of idle time required for the operator to load programs and complete the set up required to run the next job. Consequently, automatic program loading routines began to appear as part of operating systems in order to decrease the idle time between jobs.

As various alternatives to the problem of system utilization were considered, it was realized that programs being executed in serial fashion did not really require all of the program to be in main memory all of the time. If it were possible to share memory space,

then more than one program could be ready to run at a given time. This would allow greater utilization of the computer system since the Central Processing Unit would be computing on one job while another job was waiting for the completion of an I/O operation.

This mode of operation is referred to as multiprogramming. A logical extension of the principle of multiprogramming was to add a second CPU which would allow simultaneous execution of jobs. This results in considering processors, memory, I/O channels, and peripheral I/O units as resources to be allocated among the programs which are being executed. This philosphy was implemented in the operating system and design of the B 5000 computer. Later, on the B 5500, the implementation of re-entrant programs allowed program code segments to be shared between different executions of the same program. The B 6500 Master Control Program (MCP), like its B 5500 predecessor, is designed to perform automatic resource allocation and multiprocessing as the normal mode of operation. In addition, the B 6500 MCP features automatic storage control which automatically performs information transfers between main memory, disk, and library tape.

The B 6500 MCP provides the interface between object programs and the B 6500 System. The design specifications for the B 6500 MCP include the following:

Multiprocessing as the normal mode of operation.
Re-entrant object program code.
Parallel execution of independent object program sections.
Dynamic storage allocation.
Independent compilation of procedures and subroutines.
Multidimensional tree structured disk directory.
Exclusion of assembly language programming.

The normal mode of operation of the B 6500 MCP assumes the existence of multiple jobs or "processes" running concurrently. The object of running processes concurrently (multiprocessing operation) is to maximize the utilization of the B 6500 System resources, thereby increasing the throughput of jobs.

In order to obtain the greatest throughput of jobs in a multiprocessing environment, it is essential to minimize the amount of MCP overhead required to execute the jobs currently in progress. In order to minimize overhead, the B 6500 MCP controls storage allocation for each process according to its current requirements.

By bringing program segments into memory only when they are needed, memory is assigned in an efficient manner. In the event that several processes require more memory than is currently available, the MCP reallocates memory for each job as required and the least-used segments which are present in memory are overlayed. Data segments which are overlayed must be written on the disk since the data may have been modified while it was in memory. Program segments and read-only data segments which are overlayed need not be saved, since the original copy of the segment is still present on the disk.

The primary purpose of this document is to provide a description of the operational characteristics of the B 6500 MCP. However, because of the B 6500 hardware-software interrelationship, a description of the MCP can proceed only under the assumption that the reader is at least basically familiar with the operational characteristics of the B 6500 System. It may also be helpful, but not necessary, for the reader to be familiar with the problem-oriented languages of the B 6500.

SECTION 1
SYSTEM INITIALIZATION

GENERAL.

The B 6500 MCP is designed as an integral part of the B 6500 System. Since the B 6500 System is intended to serve a wide range of installations and users, provisions have been made to adapt the operation of the MCP to the particular requirements of those installations. This has been accomplished by incorporating different options in the MCP which may be changed either by specification at the time of system initialization or by operator key-in messages.

In order to place the MCP in control of the system, the MCP must be loaded onto disk, and the option list must be initialized on the disk. These functions are performed by an initialization program called INITIALIZER. The hardware DISK LOAD SELECT function expects the MCP code file to be located at disk address zero. Therefore, INITIALIZER is used to load the MCP code file from tape (or elsewhere) onto disk, beginning at disk address zero. In addition, the MCP option list is written or revised on disk as indicated by INITIALIZER data cards.

The basic functions of INITIALIZER are:

a. Loading the MCP to disk address zero from tape or disk.
b. Writing or revising the MCP option list on disk.
c. Specifying the status of the disk directory to the MCP.

These functions may be specified individually or in various combinations by the data cards which INITIALIZER reads. At the conclusion of initialization, the first 8192 words of the MCP are read from disk address zero into main memory, and control of the system is transferred to the MCP.

The data cards for INITIALIZER specify the type of initialization to be performed. The INITIALIZER functions are selected by the following four types of data cards:

a. Cold start.
b. Cool start.
c. Load from tape or disk.
d. Set or reset options.

COLD START.
The COLD START card causes the MCP to create an empty disk directory when initialization is complete.

COOL START.
A COOL START card causes the MCP to retain the disk directory which currently exists. If neither cold start nor cool start are specified, cool start is assumed.

LOAD.
The LOAD card causes an MCP code file to be read onto the disk beginning at disk address zero. The LOAD card specifies either tape or disk as the source unit.

OPTION CARDS.
Two types of option cards are permitted, set options and reset options. If all options are to be set, the SET card reads SET ALL. If all options are to be reset, the RESET card reads RESET ALL. Individual options may be set or reset in addition to or instead of a universal specification. If a universal specification is not made, any option which is not specified remains unchanged from its previous setting. If no previous setting exists, which would be the case if no valid MCP option list exists on the disk, any unspecified options will be reset.

STOP CARD.
The STOP card indicates that INITIALIZER has read all of the valid option specification cards. Any cards encountered between the STOP card and the END card are ignored.

END CARD.
The END card signifies the physical end of the INITIALIZER data deck. INITIALIZER flushes all cards following a STOP card through the card reader until the reader is empty or until an END card is found.

SECTION 2
I/O OPERATIONS

GENERAL.

All input/output operations on the B 6500 System are performed by the MCP. Certain information must be made available by the compilers in order for the MCP to perform I/O operations. This information is:

a. Symbolic file name.
b. Actual file name (label name).
c. File medium (card, tape, disk, paper tape, etc.).
d. Access type (serial or random).
e. File mode (alpha, binary, etc.).
f. Buffer size.
g. Number of buffers.
h. Logical record size.

The actual file name is the label name which is associated with the unit which contains the file, or the label name in the disk file header. The actual file name will be identical to the symbolic file name unless specified otherwise.

SYMBOLIC FILES.

FILE ASSIGNMENT.

The MCP automatically assigns peripheral units to symbolic files whenever possible in order to minimize the amount of operator attention required by each process.

Input files requested by a process cause the MCP to search its tables for the appropriate peripheral unit which contains the file requested. If the file name specified by the process is found on a particular unit, that unit is marked "in use" and assigned to the process.

In the case of a disk file, the file header is marked in use by increasing the user count by one, and the address of the file on disk is passed to the process.

Output files requested by a process are automatically assigned by the MCP if a suitable unit exists for the file. In the case of disk files, a disk file directory entry is made and the required disk space is allocated for the file.

In order to allow specification of actual file names for a file, the following tables are necessary:

a. A File Parameter Block.
b. A Label Equation Block.
c. A File Information Block.

FILE PARAMETER BLOCK (FPB).
A File Parameter Block is created by the control card routine of the MCP for all files in a process. A File Parameter Block contains the symbolic file name and any compilations or execution-time label equation information specified for this process.

LABEL EQUATION BLOCK (LEB).
The Label Equation Block is created by the compiler and maintained by the I/O intrinsic functions for each file in a process. The Label Equation Block contains the current label equation and other file attribute information for a particular file, including any programmatic specification of file attributes.

FILE INFORMATION BLOCK (FIB).
A File Information Block is also created by the compiler and maintained by the I/O intrinsic functions for each file in a program. A File Information Block contains frequently used information concerning the status of the file (and its attributes) such as the type of access required, type of unit assigned, physical unit being used, and other attributes which vary depending upon the particular type of unit assigned. Incorporation of the file attributes in the FIB and LEB allows modification of file specifications such as buffer size and blocking factors, at program execution time, without re-compilation of the program.

OBJECT PROGRAM I/O.

Object program I/O operations on the B 6500 System involve the automatic transfer of logical records between a file and a process. A logical record consists of the information which the process references with one Read or Write statement. The size of a logical record does not necessarily coincide with the size of the physical record or block accessed by the hardware I/O operations. When a physical record contains more than one logical record, the file is referred to as a blocked file.

Files may be blocked in order to conserve storage space in the file media, or to increase the rate of processing the data by reducing the number of file accesses required.

When a file is accessed by a process, a physical record is written from or read to a memory area. This memory area is called a "buffer" area for the file. If the file is blocked, the MCP maintains a record pointer into the buffer. This pointer is used by the process to access the current logical record. If the next record is not already present in a buffer, then the MCP automatically performs the required I/O operation.

The MCP attempts to keep all input buffers full and all output buffers empty for each process, regardless of status, thereby minimizing the time that a process is suspended waiting for an I/O operation to be completed.

PSEUDO UNITS.

When operating a system in a multiprogramming environment, there are frequently a large number of I/O operations occurring at the same time. If these I/O operations are concerned with card decks and printer files, the number of card readers, card punches, and printers available to the system can be, and frequently is, the limiting factor for system throughput.

In order to minimize this type of limitation, the B 6500 MCP can simulate the existence of additional card readers and line printers

with disk files. These disk files are referred to as "pseudo card readers" and "pseudo printers" in the following paragraphs.

PSEUDO CARD READERS.
The MCP will accept control cards and data cards from a disk or tape file which has been assigned to a pseudo card reader just as it would from a card reader. A separate program, Load Control, is used to load the system control decks to disk or tape. The MCP then assigns these control decks to pseudo card readers as the pseudo readers become available.

PSEUDO CARD PUNCHES.
The MCP will create a tape or disk "punch" file in lieu of a card punch when specified by the system options, the program file attributes, or the system operator. The punch file may be punched by the Punch Backup Program when a card punch is available.

PSEUDO PRINTERS.
A pseudo printer is a disk or tape file which contains special forms information for the file, and printer carriage control information for each print line. The MCP will build a pseudo printer file when specified by the MCP system options, the program file attributes, or the system operator. The pseudo printer file may be printed on a line printer by the Print Backup Program when a printer is available.

SECTION 3
SUPERVISORY AND CONTROL FUNCTIONS

GENERAL.

A convenient method of describing the supervisory and control functions of the MCP is to first consider the initiation of a single job or process. The functions which apply to multiprocessing will then be discussed by assuming the several processes are running concurrently.

PROCESS SCHEDULING.

When a card deck is placed in the card reader and the START button is pressed, the status of the card reader changes from Not Ready to Ready. The STATUS procedure of the MCP periodically uses the Scan-In Peripheral Status command to determine the status of the peripheral units. STATUS, upon recognizing that the card reader is now Ready, reads the first record and creates an independent process which calls CONTROLCARD, the control card procedure.

CONTROLCARD interprets the information contained in the record and determines that the job in the card reader is a program execution. An entry into the SHEET Queue is made by CONTROLCARD to schedule the process.

The SHEET Queue is a linked list of processes which are scheduled to be executed as soon as sufficient system resources, such as memory, are available. Each entry in the SHEET Queue is a partially built process stack. The information contained in the stack includes information concerning the following:

a. Estimated amount of main memory required by the process.
b. Priority.
c. Time of entry into the schedule.
d. Size and location of code segments.
e. Parameter block size and location.
f. Working storage stack size.
g. Size and location of the segment dictionary.
h. Size and location of the process stack information.

CONTROLCARD then reads the next record to determine if any file names are to be modified for this particular job or process. If the card obtained is a FILE (label equation) card, the label information is stored in the File Parameter Block of the process. The scanning of control cards continues until a terminal control card (such as a DATA, LABEL, or END card) is found, or until a card is found which is not a system control card. If anything other than a control card is encountered before a terminal control card is found, an error message is sent to the operator and all subsequent cards are ignored until the next END control card.

When a terminal control card is found, the CONTROLCARD process handles the control card and then terminates itself.

PROCESS INITIATION.

When SELECTION determines that sufficient system resources exist to allow another process into the mix, an independent runner process called RUN is started. The independent runner unhooks the SHEET Queue stack and makes the segment dictionary present in main memory. RUN enters the beginning of job (BOJ) time into the system log and the level 1 display register, D[1], is set to point at the segment dictionary. Then RUN makes the first segment of the process present and points the level 2 display register, D[2], at it. Finally, RUN links itself into the TERMINATE queue and transfers control of the process in which it was running to the process it initiated.

PROCESS EXECUTION.

As soon as control is transferred to the new process, a PRESENCEBIT interrupt occurs because the outer block code segment is not present in main memory. The PRESENCEBIT procedure of the MCP is entered and the following actions occur in order to make the segment present:

a. The PRESENCEBIT procedure calls the GETSPACE procedure to allocate an area in main memory for the code segment.

b. When an area is made available, PRESENCEBIT calls DISKIO, the disk input/output procedure, and waits on an event which indicates that the segment has been read in.

c. DISKIO links the request into I/O queue. When the I/O request comes to the head of the queue, the disk I/O is performed. At the completion of the disk I/O, the event is caused, thereby notifying PRESENCEBIT that the segment is now available.

PRESENCEBIT marks the segment descriptor present and exits back to the process at the point of interruption.

As the process runs, additional segments of program code and data will be required. The process stack contains the storage locations for simple variables and array data descriptors, but program code segments and array rows are assigned their own areas of memory. The assigning of separate memory areas for code segments and array rows allows segments and data to be absent from main memory until they are actually needed. In the B 6500 System, a reference to data or code through a data descriptor or a segment descriptor causes the processor to check the presence bit, bit number 47, in the descriptor. If the presence bit is OFF, an interrupt occurs which transfers control to the PRESENCEBIT procedure in the MCP, passing the non-present descriptor as a parameter. The PRESENCEBIT procedure reads the address field of the descriptor, which contains the disk address of the data or segment, for non-present descriptors. Then PRESENCEBIT calls GETSPACE to allocate a memory area of the size specified in the descriptor. GETSPACE returns the memory address of the area it has allocated and PRESENCEBIT causes the information to be read from disk into memory. When the disk read is finished, PRESENCEBIT stores the memory address of the information into the address field of the descriptor, turns the presence bit ON, and updates the descriptor in the process stack. PRESENCEBIT then returns control back to the process which was interrupted, and the process resumes execution. Now the information is present in memory, which is indicated by the presence bit being ON, and the information is accessed by the process in the normal manner.

For purposes of discussion, assume that the process expects to read a data file named INCARD. When the process performs a read operation on the file INCARD, the File Information Block (FIB) is accessed. A bit in the FIB indicates that the Label Equation Block (LEB) has not been initialized. The symbolic file name in the LEB is used to search the File Parameter Block (FPB) symbolic name list. Since no label equation of INCARD was made, no file named INCARD is found. Therefore, the symbolic name is used as the actual file name and the label table is searched to find the unit containing the file INCARD. Upon finding the unit, the card reader in this case, it is marked as being in use in the unit table. Memory is allocated for the required number of buffers, and the unit is assigned to the process.

PROCESS TERMINATION.

When a process execution is terminated, the following actions occur:

a. Any outstanding I/O requests are completed, if possible. Any OPEN files are closed, the units released, and the buffer areas are returned to the available memory table.

b. All overlayable disk areas allocated to the process are returned to the available disk table.

c. All process object code and data array areas of main memory are returned to the available memory table.

d. An EOJ entry is made in the system log for the process.

e. The process stacks are linked into the TERMINATE Queue.

If a TERMINATE independent runner is currently running, it is informed of the fact that another process has been introduced into the TERMINATE Queue. If TERMINATE is not running, the MCP initiates a TERMINATE independent runner.

LOGICAL PROCESSORS.

In order to discuss multiprocessing more concisely, it is advantageous to introduce the concept of a logical processor. A logical processor

is defined as a set of system resources which contain all of the information required by a physical processor to execute a process. Therefore, multiprocessing may be thought of as the establishment of a queue of logical processors which share the physical processors available to the system. As a result, the physical processors become a resource to be shared by processes in the same way that main memory, disk, and I/O channels are shared.

On the B 6500 System, the logical processor concept is implemented by assigning a stack for each process. The stack is a contiguous area of memory which is an extension of the processor stack registers. The stack provides storage for program variables and data references. In addition, the stack contains information pertaining to the dynamic history of the process. The process stack provides the storage locations for:

a. Simple variables (single or double-precision operands).

b. Indirect Reference Words which refer to arrays, variables, and procedures that are not locally declared.

c. Data descriptors which reference data arrays.

d. Program Control Words which refer to procedures or subroutines.

The dynamic history of the process is automatically recorded in the process stack by the hardware. The dynamic history is stored by linking the Mark Stack Control Words for each lexicographic level of the process.

RE-ENTRANT CODE.

The term "re-entrant code" is used to refer to object program instruction segments which may be executed by more than one process at the same time. A necessary condition for re-entrant code is that it is not modified during execution. Since each process on the B 6500 System has a stack for data storage, and since the object program code is not modified by execution, all object programs on

the B 6500 are re-entrant. Figure 3-1 shows the stack structure for two copies of the same ALGOL program being executed concurrently by two processors. The declaration of a procedure causes a program control word to be created in the stack at execution time.

The Program Control Word is used for procedure entry and exit. It contains the following information:

a. The relative location in the segment dictionary of the object code segment descriptor.

b. The procedure entry point references:

   1) The word index relative to the beginning of the code segment.

   2) The syllable index relative to the beginning of the entry point word.

If an object program is being executed by more than one logical processor, the user count in the lower part of the segment dictionary is counted up to represent the number of users, and the additional stacks are linked together. Since there is only one segment dictionary for a given program, the code segments are automatically shared among the various logical processors which are executing the program.

OVERLAY.

Even with the ability to share object program segments, there are times when all of the processes which are currently active require more main memory than is available to the system. When such a condition arises, it becomes necessary to remove some of the program and/or data segments from main memory. This action occurs only when a process accesses an absent program segment or data array row and there is no available area which is large enough to accommodate the absent information.

When the absent segment or data descriptor is accessed, a presence bit interrupt occurs, and control is transferred to the PRESENCEBIT procedure. The PRESENCEBIT procedure calls GETSPACE to allocate an

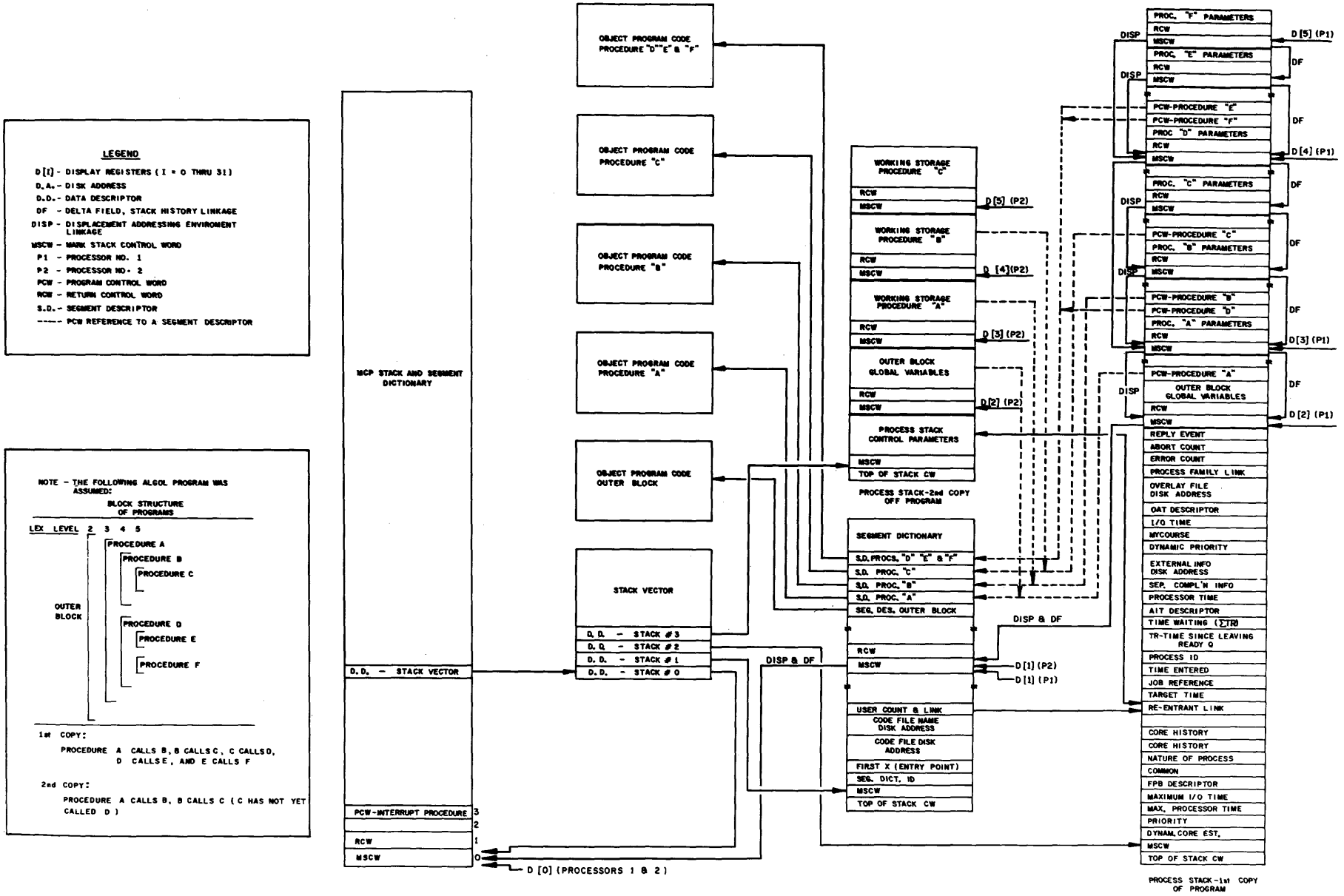


Figure 3-1. B 6500 Re-Entrant Program Stack Structure

area for the absent segment. GETSPACE first tries to locate an available area. If no available area is large enough, GETSPACE starts with oldest overlayable area in memory and determines if it, along with any adjacent available areas, is large enough to satisfy the request.

If the area is large enough, the in-use area is overlaid by the OVERLAY procedure and GETSPACE returns the address of the required area to PRESENCEBIT. If the area located is not large enough, GETSPACE looks at the area which preceeds the current area being investigated. GETSPACE includes that area if it is either available or in-use but overlayable.

This process continues until the size of the area is sufficient to satisfy the request or a non-overlayable area is encountered. Once the area located is of sufficient size to satisfy the request, GETSPACE calls OVERLAY for each overlayable area.

If the area is too small when a non-overlayable area is encountered, GETSPACE locates the next oldest overlayable area which has not been encountered, and attempts to find enough area again, as described above.

Whenever OVERLAY is called by GETSPACE, it must determine what type of information is contained in the area of memory before it knows what action to take. If the area contains a program segment or read-only data, the only action required is to store the disk address of the program or data segment, which is in the second word of the memory link, into the descriptor. If the area contains data, it may have been modified after it was brought in from the disk. Therefore, the data must be written out to disk before the area can be made available.

EFFECT OF MULTIPLE BUFFERS.

Originally, multiple buffers were used to increase system efficiency by overlapping I/O operations with processor computations. Since

multiprocessing allows overlap of I/O operations and processor computations between different processes, much of the original need for multiple buffers would seem to be obviated.

In the B 6500 System, however, the existence of synchronous I/O multiplexor channels allows multiple buffers to still be effective in increasing throughput for processes which require groups of physical records at a time. Since the MCP performs all object program I/O action, a process with multiple buffers allocated for a file allows the MCP to perform I/O operations independent of the status of the process.

The determination of the number of buffers required for efficient execution of a process depends upon many factors. These factors include:

a. The type of files being used.
b. The particular hardware configuration being used.
c. The processing characteristics of the process.
d. The memory requirements of the process.
e. The mix of processes which are typically multiprocessed.

Particular note should be made of the fact that the use of excessively large buffers or an excessive number of buffers for processes can cause unnecessary overlays of program. This, in turn, will result in reduced system throughput and poor system performance.

DYNAMIC STORAGE ALLOCATION.

The B 6500 MCP performs dynamic storage allocation for all system storage media: Main memory, magnetic disk, and system library magnetic tape. As a result of considering the different system storage media as a hierarchy of memory, the MCP controls allocation and deallocation of all system memory, regardless of the type.

The MCP dynamically allocates the use of main memory as a resource among the current processes. If a process needs more memory than that which is currently available, the MCP will select a suitable contiguous area and overlay any in-use areas in order to make room for the process.

In addition to allocating main memory, the MCP also allocates disk areas. If a process or the MCP requires more disk area than is currently available, the MCP will select the oldest disk files which are contained in a suitable area and proceed to automatically create a system library tape containing the files which are to be overlayed. When the area has been cleared, the MCP will adjust the disk directory information to show that the overlayed files reside on system library tape, and then reallocate the area to the process requiring it.

In order to be able to recall the files which are located on system library tapes, the MCP requests volume serial numbers for the tapes. The volume serial number is used for MCP/operator communication in denoting which library tape to load for future recalls of the files.

SECTION 4
MCP FUNCTIONS

STORAGE CONTROL.

MAIN MEMORY.

The two visible memory allocation procedures are GETSPACE and FORGETSPACE. Available memory links are an ordered threaded list with the order based on:

a. The overlayability of the area that physically precedes the available area in memory.

b. The size of the available area.

In allocating an area of memory, the two arguments that are used are the overlayability of the desired area and the required size. These arguments and the ordering of the available list tend to keep non-overlayable and overlayable memory in separate continguous areas of memory. This is especially true when an area of the desired size is in the available list.

MEMORY LINKS.

Main memory links consist of in-use links and available links, each of which contain sufficient information for a single hardware operator to find the next link, and all succeeding links.

IN-USE LINKS. In-use links consist of at least four words that contain:

a. Word 1:

1) Stack number of the requesting process (10 bits).

2) Available bit (1 bit).

3) Back-link (length of area - 17 bits).

4) Left-off link (points to last previously allocated in-use area - 20 bits).

b. Word 2:

1) Disk address code (20 bits).

2) Address type (BOS relative or Abs. - 1 bit).

3) Address of MOM (20 bits).

4) Usage (data, program, separately compiled procedure, etc. - 7 bits).

c. Word 3: Reserved (normally used for I-O control word - 48 bits).

d. Word n:

1) Unused number of words in in-use area (Delta - 6 bits).

2) Left-off pointer (points to next allocated in-use area - 20 bits).

3) Overlayable bit (1 bit).

4) Available bit (1 bit).

5) Front-link (length of area - 17 bits).

6) Reserved (3 bits).

Words 1 and 2 are assigned within the area at addresses that are closest to zero. Word n is assigned to the last word of the area so that the back-link is self-relative and points to word n, and the front-link points to word 1.

AVAILABLE LINKS. Available links are three words long and consist of:

a. Word 1:

1) Previous area overlayability bit (1 bit).
2) Reserved area (6 bits).

3) Available bit (1 bit).
4) Back-link (length of area - 20 bits).
5) Link to next available area (20 bits).

b. Word 2. Link to last available area (20 bits).

c. Word n:

1) Available bit (1 bit).
2) Front-link (length of area - 20 bits).

Comments under in-use links pertaining to words 1 and n are applicable to available links.

MEMORY ALLOCATION.

Memory allocation is done by the GETSPACE procedure. Parameters are supplied to GETSPACE so that an adequate-sized continguous area of memory may be reserved for a particular stack number. The storage may be allocated at the front or rear of an adequately sized area and marked as overlayable or non-overlayable.

When an in-use area is allocated, it is linked to the previously allocated in-use area by the left-off link and pointer fields in the memory links. These fields comprise the LEFT-OFF LIST. A reference word pointing to the oldest entry in the LEFT-OFF LIST allows the chronological history of in-use memory areas to be determined.

If sufficient memory is not available, an effort may or may not be made to overlay adjacent areas of memory until an adequate sized contiguous memory area is available to satisfy the current request.

If the area found is larger than the required size, the unused portion is made available by linking it into the available list. This is accomplished by GETSPACE who calls FORGETSPACE, passing a negative address as a parameter. If sufficient memory cannot be made available, the request may be put in the WAIT or SPACEQUEUE until the request can be satisfied. In the case of an unsatisfied request, an appropriate operator communication may be generated.

MEMORY DE-ALLOCATION.

Memory de-allocation is done by the FORGETSPACE procedure. The parameter to FORGETSPACE is the absolute memory address of an area that is to be returned to the available list.

Adjacent available areas are consolidated into a single contiguous area. An available area may be that portion of an in-use area (DELTA) that was too small to be returned to the available list.

The available list is ordered with an argument that consists of:

a. Overlayability of the area that immediately and physically precedes the area being made available.

b. The size of the area.

This results in the available list being ordered by size within overlayability.

The ordering of the available list allows GETSPACE to find an area of the correct type with a minimum amount of searching.

OVERLAY OF MEMORY.

When there is insufficient available memory to satisfy a particular request, the overlay mechanism is invoked. The left-off list is searched, starting at the overlayable area that has been allocated for the longest period of time.

If this area, combined with any adjacent available area, is adequate to satisfy the request, it is overlaid. Otherwise, allocated areas with lower starting addresses are considered until one of the following occurs:

a. The request is satisfied.
b. A non-overlayable area is encountered.

If the request is not satisfied, the next oldest overlayable area is obtained, and the left-off list is searched as described above. This

process is repeated until the left-off list has been exhausted. If at this time the request has not been satisfied, a No Memory condition exists.

DISK.

DISK OVERLAY.

On some systems, especially those with a large number of system files, situations may arise where more disk space is required than is currently available. In such a situation, the MCP will find or request a system scratch tape and unload the oldest access date files onto the system library tape until sufficient space is available. The MCP will then record the appropriate volume and reel number into the directory for each file. Subsequently, if one of the absent files is required, the system will automatically reload the file from the system tape.

OPERATOR-MCP COMMUNICATIONS.

Communication with the MCP is accomplished with a combination of display units (CRT devices), control units (display units with associated keyboards), and control cards (special cards recognized by the MCP). The following discussion is based on a system with one control unit and one display unit, although a system may have any combination from a minimum of one display unit to a maximum of thirty display and control units combined.

Terms enclosed in the character pair ⟨ ⟩ are defined either immediately after use, or in appendix A.

DISPLAY OF STATUS.

The status of the system and of the processes in progress is presented on the display units. Various tables may be called for display by entering the appropriate keyboard input messages. In addition, specific questions requiring short answers may be entered in the keyboard. These questions and answers are displayed as they occur. The display tables are described below.

MIX TABLE. The MIX table is displayed continuously except for brief periods when it is replaced by another table. Each job being executed has an entry, the contents of which depend on whether the job is active or suspended.

ACTIVE ENTRY. If a job is being executed normally or was terminated between the two most recent updates, its entry contains the following information:

a. MI - mix index.

b. Job - the first five characters of each of the first three levels of the ⟨job name⟩.

c. P - priority.

d. C - compiler code:
   1) A - ALGOL.
   2) C - COBOL.
   3) E - ESPOL.
   4) F - FORTRAN.
   5) Blank - not a compilation.

e. S - status:
   1) B - Beginning-of-Job.
   2) R - Running.
   3) E - End-of-Job.
   4) D - Discontinued.

f. CU - core used (tenths of percent of usable core).

g. PT - processor time used as of last update in minutes.

A typical active entry is as follows:

MI	Job	P	C	S	CU	PT
13.093	=LITTL/BADGE/OPTIO	:5	:A	:R	:113	,1.6

SUSPENDED ENTRY. If a job is suspended for any reason, its mix entry changes from active to suspended and contains the following information:

a. MI - mix index.

b. P - priority.

c. Reason - an output message giving the reason for suspension.

d. Action - abbreviation for one or more input messages required to reactivate processing.

A typical suspended entry is as follows:

MI	P	Reason	Action
13.093	5	NO FILE=VOLID/FILID:OF,UL,IL,DS	

SCHEDULE TABLE. Following the entry of the input message SCH at the control unit, the schedule table will replace the mix table for a period of time, the length of which depends upon the number of entries. The entry for a job in the schedule contains the following information:

a. SI - schedule index. This will become the mix index upon entry into the mix.

b. Job - job name.

c. P - priority.

d. C - compiler code (see mix table).

e. S - status:
   1) S - scheduled.
   2) M - entered mix between two most recent updates.

f. CR - core required (tenths of percent of usable core).

g. ST - time in minutes since entry into schedule.

A typical schedule table entry is as follows:

```
 SI      Job      P C S  CR  ST
13.097=CORPO/PAYCH:5: :S,192,5.7
```

PERIPHERAL UNIT TABLE. This table is called with the input message PER, and has an entry for each peripheral unit in the system. An entry contains the minimum information necessary for determination of the status and content of a given unit as follows:

a. Unit - unit mnemonic.

b. S - status:
   1) I - In use by.
   2) L - Labeled.
   3) N - Not Ready.
   4) S - Scratch.
   5) U - Unlabeled.

c. MI - mix index of job using this unit.

d. File - file label of file associated with this unit.

e. RDC - RDC of tape reel on this unit.

A typical peripheral unit table entry is as follows:

```
 Unit S    File      RDC
MT002:L:CHECK/DEPOS:3,69002,1
```

LABEL TABLE. This table is called with the input message OL and contains an entry for each I/O unit of the designated type which is on line. If no units of the designated type are on line, the output message NULL ⟨unit mnemonic⟩ TABLE will appear. An entry may contain extensive information about the status and content of the specified unit, possibly including a complete listing of the label part of the file associated with the unit. An appropriate subset of the following information will be included:

a. Unit - unit mnemonic.

b. S - status:
   1) I - In use by.
   2) L - Labeled.
   3) N - Not Ready.
   4) S - Scratch.
   5) U - Unlabeled.

c. MI - mix index of job using this unit.

d. JOB - job ID of job using this unit.

e. FILE - file label of file associated with this unit.

f. RDC - RDC for the file associated with this unit.

g. LBLINFO - other label information to be specified.

A typical label table entry is as follows:

```
Unit S  MI    File          RDC
CD002:I:13,027:INVEN/RECVD:2,69002,1
```

DISK DIRECTORY TABLE. This table is called with the DIR input message. It displays all file labels in the disk directory which are contained in the set specified by the input message. If the specified set is empty, the output message NULL ⟨file set specifier⟩ will appear.

A typical directory table in response to the input message DIR DIRI DIRID1/=" is:

```
DIRID1?
VOLID1/FILID2,DIRID2/VOLID3/FILID4
VOLID2/FILID1,DIRID3/VOLID7/FILID5
VOLID2/FILID2,VOLID1/FILID1
```

JOB TABLE. This table is called with the JOB input message specifying any job in the mix. It contains detailed information about the job as follows:

a. Mix table entry.
b. Listing of control cards.
c. Correlation of physical units with file names.
d. Associated processes (⟨mix index⟩ SUFFIXES).

A typical job table is as follows:

```
13.097=CORPO/PAYCH:5: :R,184,3.2
EXECUTE CORPORATIONX/PAYCHECKWRITER.FOR
WEEK ENDING 1-3-69

FILE CARD=HOURLY
FILE DISK=PAYROLLINFO/HOURLY

FILE NEWDISK=PAYROLLINFO/HOURLY/UPDATED
FILE LINE=LINE PRINT OR BACKUP
CD010=CARD
LP002=LINE
.1,.2,.2.1,.3
```

CONTROL CARDS.
Information may be passed to the MCP through the use of punched cards called control cards. These cards are made distinguishable from other cards by an invalid character in column 1. Control information and comments are punched in columns 2-80. The format for this information is free field with the exception that the proper order must be maintained. All metalinguistic variables and constants must be separated by a space. If a period appears in a control card, the information following it is ignored by the MCP.

Normally, but not necessarily, one control card contains one control statement. Two or more control statements may be punched on a single control card provided they are separated by semicolons. No invalid character is required or accepted following a semicolon.

A control statement may be punched on more than one control card by terminating all but the last card with a hyphen, provided an identifier is not divided. Only the first card of such a group may contain an invalid character.

Control statements may also be entered at the control unit (see input messages).

The following paragraphs describe the format and function of each control statement accepted by the MCP.

COMPILE STATEMENT. The COMPILE statement must contain the following information:

⟨invalid character⟩ COMPILE ⟨program name⟩ ⟨comment⟩ ⟨compiler name⟩ ⟨comment⟩ ⟨disposal⟩ ⟨comment⟩

where:

⟨disposal⟩::= ⟨empty⟩ | LIBRARY | SYNTAX

The COMPILE statement designates the compiler to be used and the type of compile run to be made. This must be the first control statement in a compilation job. The three forms are:

a. COMPILE AND EXECUTE (⟨disposal⟩ ← ⟨empty⟩).
After an error-free compilation, the compiled program is scheduled for execution but the program name is not entered in the disk directory. The disk space used by the program is released after the execution is terminated.

b. COMPILE FOR LIBRARY (⟨disposal⟩ ← library).
The object code from an error-free compilation is left on disk and the program name is entered in the disk directory. The compiled program is not executed.

c. COMPILE FOR SYNTAX CHECK (⟨disposal⟩ - SYNTAX).
The compiled program is not executed and the program name is not entered in the disk directory. The disk space used by the program is released upon completion of compilation.

EXECUTE STATEMENT OR RUN STATEMENT. The EXECUTE or RUN statement must contain the following information:

⟨invalid character⟩ EXECUTE ⟨program name⟩ ⟨comment⟩
⟨invalid character⟩ RUN ⟨program name⟩ ⟨comment⟩

The designated library program is called from the disk and executed. This must be the first control statement in a job not requiring compilation.

REMOVE STATEMENT. The REMOVE statement is of the form:

⟨invalid character⟩ REMOVE ⟨file set list⟩

The specified file labels are removed from the disk directory and the associated disk space is released.

DUMP STATEMENT. The DUMP statement is of the form:

⟨invalid character⟩ DUMP TO ⟨volume id⟩ ⟨file set list⟩

A library tape will be generated containing the files in the file set list. The ⟨volume id⟩ is the library tape name.

LOAD STATEMENT. The LOAD statement format is:

⟨invalid character⟩ LOAD FROM ⟨volume id⟩ ⟨file set list⟩

The files with the specified file labels from a library tape with the specified volume ID will be written on disk and their file labels will be entered in the disk directory.

CHANGE STATEMENT. The CHANGE statement format is:

⟨invalid character⟩ CHANGE ⟨change list⟩

where:

⟨change list⟩ ::= ⟨change element⟩ |
⟨change list⟩,⟨change element⟩
⟨change element⟩ ::= ⟨file label⟩ TO ⟨file label⟩

The file specified by the first file label in the change element is relabeled using the second file label.

DATA STATEMENT. The DATA statement must contain the following information:

⟨invalid character⟩ DATA ⟨file label⟩

The information on all cards following this control statement, and until another control card is encountered, will be designated as data and be placed in a file called ⟨file label⟩. This file label must be the same as the file name used in the program, or must be label equated to it. The DATA statement must be the last control statement before the actual data.

DATABCL STATEMENT. The format of the DATABCL statement is:

⟨invalid character⟩ DATABCL ⟨file label⟩

The information on all cards following this control statement is treated as above except the cards following contain BCL data. BCL data must be followed by an ENDBCL control card.

ENDBCL STATEMENT. The ENDBCL statement format is:

⟨invalid character⟩ ENDBCL

This statement signals the end of BCL data and must follow each BCL data deck. It does not designate End-of-File action.

END STATEMENT. The END statement format is:

⟨invalid character⟩ END

This statement designates End-of-File information for a particular program and is required whenever a program is terminated for any reason while it has card information yet to be read. Consequently, if an END statement appears, it must be the last card in a deck

pertaining to a program. However, an END statement is not necessary to denote the end of a data file. An attempt to read any control card as data will cause an End-of-File notification. Therefore, if a program requires more than one card file, the end of one file will be denoted by the DATA statement for the next.

PROCESS TIME STATEMENT. The PROCESS time statement must contain the following information:

⟨invalid character⟩ ⟨optional compiler name⟩ PROCESS ⟨comment⟩ ⟨integer⟩

This statement specifies the maximum process time in seconds for the object program or the compiler. If the process time exceeds that specified, the job will be terminated.

IO TIME STATEMENT. The IO time statement must contain the following information:

⟨invalid character⟩ ⟨optional compiler name⟩ IO ⟨comment⟩ ⟨integer⟩

This statement specifies the maximum I/O time in minutes for the object program or the compiler. If the I/O time exceeds that specified, the job will be terminated.

STACK SIZE STATEMENT. The STACK size statement must contain the following information:

⟨invalid character⟩ ⟨optional compiler name⟩ STACK ⟨comment⟩ ⟨integer⟩

This statement specifies the number of words to be assigned in primary memory for the working stack of the compiler or object program. If no STACK size statement appears, the working stack size will be 512 words.

PRIORITY STATEMENT. The PRIORITY statement format is:

⟨invalid character⟩ ⟨optional compiler name⟩ PRIORITY
⟨comment⟩ ⟨integer⟩

This statement specifies the priority to be assigned to a compilation or an object program execution. Priorities may range from 0 to mm where 0 is the lowest priority and mm is the highest priority. Unless otherwise specified, a priority of mm/2 will be assumed. For a COMPILE AND EXECUTE job, a priority assigned to the compilation will also apply to the execution unless a specific priority is assigned with a control statement at execution time.

FILE (LABEL EQUATION) STATEMENT. The FILE statement, often referred to as the label equation statement, must contain the following information:

⟨invalid character⟩ ⟨optional compiler name⟩ FILE
⟨file name⟩ = ⟨file label⟩ ⟨options⟩

The FILE statement is used to associate the file name used in the program with a particular data file for execution. The FILE statement may also be used to specify various options for input/output files as follows:

⟨options⟩ to be specified.

COMMENT STATEMENT.
To be specified.

I/O UNIT STATEMENT. The I/O unit statement format is:

⟨invalid character⟩ UNIT ⟨unit mnemonic⟩ ⟨comment⟩ ⟨file label⟩

This statement associates a file label with a particular I/O unit. It may be used when an input file does not have a label and operator intervention is not required.

CORE REQUIRED STATEMENT. The CORE required statement format is:

⟨invalid character⟩ ⟨optional compiler name⟩ CORE ⟨comment⟩ ⟨integer⟩

This statement allows the operator or programmer to override the compiler's estimate of the amount of core storage, in words, required for efficient execution of the program.

SAVE STATEMENT. The SAVE statement must contain the following information:

⟨invalid character⟩ SAVE ⟨comment⟩ ⟨integer⟩ ⟨comment⟩

This statement specifies the number of days from last access for which a program is to be saved in the disk library.

MESSAGES.
The operator communicates directly with the MCP through the use of input/output messages. All input messages and certain output messages are displayed as they occur. They will also appear in the system log.

OUTPUT MESSAGES. Output messages which appear only as answers to direct questions will be described with the corresponding input message. The remainder of the output messages appear below as they are displayed. Following each message is a brief description of its meaning and any required operator response.

There are four types of output messages as follows:

a. Messages giving information but requiring no operator action. In the following description, these messages contain no prefix.

b. Messages which require operator action appear as suspended entries in the MIX display. In the following description, these messages are prefixed with the # character.

c. Messages which signal discontinuation of the program prior to EOJ. In the following description, these messages are prefixed with the - character. These messages will normally appear only in the system log.

d. Messages which relate to the Breakout and Restart facility. In the following description these messages are prefixed with the -- character pair.

⟨mix index⟩ ACCEPT
An object program executed an ACCEPT statement. A ⟨mix index⟩ AX input message is required.

- ARG .GT, MAX F
To be specified.

BAD LIBRARY TAPE
A library tape has irrecoverable parity errors and cannot be loaded.

⟨unit mnemonic⟩ BUSY
An I/O operation was attempted on the specified unit, and the unit was apparently busy.

⟨file label⟩ CHANGED TO ⟨file label⟩
The MCP has performed an operation specified in a CHANGE control statement.

#CONTROL CARD ERROR ⟨unit mnemonic⟩ {information from control card}
The MCP expected to read control information from the designated I/O unit but has found the information to be in error.

CP RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩.
A program needs a card punch and none is available.

⟨unit mnemonic⟩ / ⟨I/O operation⟩ DA = ⟨integer⟩; # SEG = ⟨integer⟩;
RTRY = ⟨integer⟩; # TRNS = ⟨integer⟩
Retries had to be made on the disk file. The I/O operation is an R if it was on a Read, W if on a Write. The integer appearing after

DA is the disk address, the integer appearing after SEG is the number of segments read or written, the integer after RTRY is the number of retries necessary, and the integer after TRNS is the number of disk transactions since the last Halt-Load operation.

- DATA STMT ERR.
To be specified.

DECK ⟨integer⟩ REMOVED.
The specified control deck was removed from disk because of either job completion or an input message.

DISK FAILURE - ⟨unit mnemonic⟩.
An error occurred on disk access. If this message is not followed by by a ...#RTRY=... message, then a Halt-Load operation must be performed.

DISK PARITY ON LIBRARY MAINTENANCE.
Library maintenance was not completed successfully.

- DIV BY ZERO ⟨job name⟩, ⟨terminal reference⟩
An object program attempted a divide operation using a zero divisor.

DIV BY ZERO BRANCH ⟨job name⟩ , ⟨terminal reference⟩
A divide by zero occurred, but a programmatic recovery feature was used.

⟨file label⟩ DUMPED
The MCP has performed the operation that was specified in a DUMP control statement.

DUP FIL ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩ ⟨file list⟩
The object program attempted to access an input file but the MCP found more than one file with the specified file label. The condition can be corrected by making only one of the files available, then entering either a ⟨mix index⟩ OK, a ⟨mix index⟩ IL, or a ⟨mix index⟩ UL message.

DUP LIBRARY ⟨file label⟩ : ⟨job name⟩
An attempt was made to enter a file in the disk library when its file label was identical to a file label already in the disk directory. The condition may be corrected by using a CHANGE or REMOVE control statement followed by a ⟨mix index⟩ OK message, or by entering a ⟨mix index⟩ RM message.

--END OF REEL ⟨unit mnemonic⟩. BREAKOUT IN PROCESS WILL BE RESTARTED ON NEW REEL.
End-of-Tape has been reached on a BREAKOUT tape. The BREAKOUT will be restarted on a new reel.

- EOF NO LABEL ⟨file label⟩ : ⟨job name⟩ , ⟨terminal reference⟩
The end of an input file was reached with no specification as to action to be taken.

- EOT NO LABEL ⟨file label⟩ : ⟨job name⟩ , ⟨terminal reference⟩
An object program has reached the end of the designated files declared areas, as on disk.

⟨file label⟩ EXPIRED
This message refers to files on disk at Halt-Load time for which the SAVE period has expired.

- EXPON OVRFLW ⟨job name⟩ , ⟨terminal reference⟩
An object program performed an operation which caused an exponent overflow to occur.

EXPON OVRFLW BRANCH ⟨job name⟩ , ⟨terminal reference⟩
An exponent overflow occurred, but a programmatic recovery feature was used.

FACTOR = X, CORE AVAIL = Y, CORE IN USE = Z
This is the response to a TF input message.

⟨unit mnemonic⟩ ⟨I/O operation⟩ FAILURE - D ⟨integer⟩
One of the following errors persisted after 10 retries:

⟨integer⟩ = 19 - parity error between I/O control and core or disk file control.

⟨integer⟩ = 20 - parity error on transfer from disk.

- FILE UNOPENED ⟨job name⟩ , ⟨terminal reference⟩
An object program attempted to write a file that has not been opened.

FM RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program is ready to open a file for which the FROM option has appeared on a Label Equation card. An FM input message is required to continue processing.

-FRMT ERROR
To be specified.

- INTGR OVRFLW ⟨job name⟩ , ⟨terminal reference⟩
An object program performed an operation which caused an integer overflow to occur.

INTGR OVRFLW BRANCH ⟨job name⟩ , ⟨terminal reference⟩
An integer overflow occurred but a programmatic recovery feature was used.

- INVALD ADRSS ⟨job name⟩ , ⟨terminal reference⟩
An object program performed an operation which addressed a non-existent memory location.

INVALD ADRSS
An invalid address occurred in control state and it could not be associated with a particular program in the mix. A Halt-Load operation may be required.

- INVALID EOJ ⟨job name⟩ , ⟨terminal reference⟩
A COBOL or FORTRAN program attempted to execute the END statement, or a sequence error occurred.

- INVALD INDEX ⟨job name⟩ , ⟨terminal reference⟩
An object program attempted to index out of the range of an array being referenced.

INVALD INDEX BRANCH ⟨job name⟩ , ⟨terminal reference⟩
An invalid index occurred but a programmatic recovery feature was used.

⟨unit mnemonic⟩ INV CHR IN COL ⟨integer⟩
An invalid character has appeared in a position other than column 1 of a control card. The integer is the column number.

INV KBD {typed-in information}
The MCP was not able to recognize a message entered from the keyboard.

⟨unit mnemonic⟩ I/O INV ADDR
An invalid address occurred when data was to be transferred between an I/O channel and primary memory. The MCP recognizes the error condition and, if possible, rectifies the errors. The primary purpose of this message is to draw attention to a condition which could denote a hardware failure.

⟨unit mnemonic⟩ I/O MEM PAR
A parity error occurred when data was to be transferred between an I/O channel and primary memory. The MCP recognizes the error condition and, if possible, rectifies the errors. The primary purpose of this message is to draw attention to a condition which could denote a hardware failure.

⟨file label⟩ LIBRARY MAINTENANCE IGNORED
The MCP was not able to perform the library maintenance operation specified in a control card.

- LIST SIZE ERR
To be specified.

⟨file label⟩ LOADED
The MCP has performed the operation specified in a LOAD control statement.

LP BACKUP ON ⟨unit mnemonic⟩
A printer backup tape is on line. If the tape is to be printed, a PB message must be entered.

LP, PBT MT RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs a line printer or a printer backup tape, and neither is available. The situation will be remedied if a line printer, backup tape, or scratch tape becomes available. An OU message may be entered if desired.

LP RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs a line printer and none is available. The condition will be remedied when a line printer becomes available, or it may be changed with an OU message.

MT RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs a scratch tape for a tape file.

- NEGATIVE BASE XTOI
To be specified.

- NEGTV ARGMENT LN ⟨program name⟩ , ⟨terminal reference⟩
A program attempted to pass a negative argument to the LN intrinsic.

- NEGTV ARGMNT SQRT ⟨program⟩ , ⟨terminal reference⟩
A program attempted to pass a negative argument to the SQRT intrinsic.

NEW PBT ON ⟨unit mnemonic⟩
A new printer backup tape was opened.

- NMLST ERR
To be specified.

NO DISK AVAIL
A disk area was required, but sufficient disk space was not available.

NO FILE ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs an input file which is apparently unavailable. If the file is labeled, it must be made available. If the file is not labeled, an IL message is required. If it is a COBOL optional file, an OF message is required. If a COBOL program has read the final volume of a multi-volume unlabeled file, an FR message is required.

NO FILE ⟨vol id⟩ / FILEOOO
An attempt was made to load files from a library tape which was not available to the system.

NO FILE ON DISK ⟨file label⟩ : ⟨job name⟩
A program needs a file it expected to find on disk. The file must be made available, and then an OK message must be entered.

⟨mix index⟩ NO MEM
The MCP was unable to obtain required primary memory. Subsequent periodic attempts are made while other processing continues. If an area is obtained, the OK MEM message appears. If the mix index equals 0, the memory required was for the MCP and a Halt-Load operation is required.

⟨file label⟩ NOT IN DIRECTORY
A control statement referenced a file which was not in the disk directory.

⟨file label⟩ NOT LOADED (NOT ON TAPE)
A LOAD control statement referenced a file which was not in the disk directory.

⟨file label⟩ NOT LOADED (NOT ON TAPE)
A LOAD control statement referenced a file which was not on the specified library tape.

⟨unit mnemonic⟩ NOT READY
An I/O operation was attempted on a unit that was Not Ready.

⟨unit mnemonic⟩ NOT READY EU
An I/O operation was attempted on a disk file electronics unit that was Not Ready.

⟨mix index⟩ OK MEM
The condition indicated by a NO MEM message no longer exists.

OPRTR ST-ED ⟨job name⟩
The specified job was suspended in response to an ST input message. An OK message is required to continue processing.

- OPRTR DS-ED ⟨job name⟩ , ⟨terminal reference⟩
The specified job was discontinued in response to a DS input message.

PARITY ON ⟨unit mnemonic⟩
The MCP received an irrecoverable parity condition while reading the label or scanning down a multi-file volume.

- PAR NO LABEL ⟨file label⟩ : ⟨job name⟩ , ⟨terminal reference⟩
An irrecoverable parity occurred on the designated file, and no programmatic recovery facility was specified.

PBT MT RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs a scratch tape for a printer backup file. The condition will be remedied when a scratch tape is made available, or may be changed with an OU input message.

⟨unit mnemonic⟩ PG-ED
A tape was purged by an input message or a program.

⟨unit mnemonic⟩ PRINT CHECK
A print check error occurred during printing of a line on a line printer. Processing continues normally.

PP RQD ⟨file label⟩ ⟨rdc⟩ : ⟨job name⟩
A program needs a paper tape punch, and none is available.

⟨unit mnemonic⟩ PUNCH CHECK
A punch check error occurred during the punching or a card. Processing continues normally.

⟨unit mnemonic⟩ READ CHECK
A read check error occurred on a card reader. The last card in the stacker must be reproduced, if necessary, and run through again.

READ ERROR FOR {control card information}
A read error, probably irrecoverable parity, occurred during the reading of a control deck for the disk. The control card which is printed denotes the deck which will be deleted. The decks that follow will be loaded normally.

⟨file label⟩ REMOVED
An operation specified in a REMOVE control statement has been completed.

⟨unit mnemonic⟩ RW/L
A tape has been rewound and locked.

- SELECT ERROR ⟨file label⟩ : ⟨job name⟩ , ⟨terminal reference⟩
An object program attempted to perform an invalid operation on the designated file, e.g., rewind a card reader.

- STACK OVERFLOW ⟨job name⟩ , ⟨terminal reference⟩
The operations performed by an object program have caused its stack to overflow its limit, and the MCP was unable to extend it.

TILT
B 6500 MCP LEVEL ⟨level number⟩ , ⟨patch number⟩
A Halt-Load operation is required.

- TYPE ERR
To be specified.

UNEXP IO ERR
The MCP encountered an unexplained I/O error that coult not be directly associated with a particular program. A Halt-Load operation is required.

- UNEXP IO ERR ⟨job name⟩ , ⟨terminal reference⟩
The MCP encountered an unexplained I/O error associated with the specified job.

- ⟨unit mnemonic⟩ WRITE LOCK
A program attempted to write on a magnetic tape with no write ring, or on a disk file which has been locked out by a lockout switch.

- ⟨unit mnemonic⟩ WR PARITY
An irrecoverable parity error occurred on the designated unit.

ZIP ERROR
To be specified.

- ZERO ARGMNT LN ⟨program name⟩ , ⟨terminal reference⟩
The designated program attempted to pass an argument of zero to the LN intrinsic.

INPUT MESSAGES. Information may be supplied to the MCP through the use of input messages entered in free-field format at the control unit keyboard. These messages are not intended to provide detailed information about individual programs, e.g., the settings for registers or the contents for memory locations.

To enter a message, the operator must first depress the LOCAL key then the STX key. After keying in the message, he must depress the ETX key, then the HOME key, and then the TRANSMIT key. If the message is not recognizable, the MCP will not act upon it except to give an INV KBD output message.

The input messages appear below with their required spelling. Following each message is a brief description of its purpose and effect. Messages which may result in the display of a table have three letter mnemonics.

⟨mix index⟩ AX ⟨program name⟩
This message is entered in response to a program name ACCEPT output message.

? ⟨control statement⟩
Any control statement allowed on a control card may be entered. Multiple control statements may be entered on a line by separating them with semicolons. The last control statement must be an END statement.

CL ⟨unit mnemonic⟩
All exception flags maintained by the MCP for the specified unit will be reset (cleared). If the specified unit is a pseudo card reader, the deck it contains will be eliminated.

NOTE

Clearing of a unit assigned to a job will result in immediate discontinuation of the job.

DIR =
or
DIR ⟨file set specifier⟩

The disk directory table will be displayed on the unit where the message is entered. If the form DIR= is used, the entire disk directory will be displayed.

⟨mix index⟩ DS
The specified program will be discontinued.

DR ⟨integer⟩ / ⟨interger⟩ / ⟨integer⟩
The date used by the MCP will be reset to the one specified. The three integers are month (1 to 12), day (1 to 31), and year (0 to 99), respectively.

⟨mix index⟩ ES
The specified job in the schedule will be eliminated.

EXP=
or
EXP ⟨file set specifier⟩

All expired disk file labels belonging to the specified set will be listed.

⟨mix index⟩ FM ⟨unit mnemonic⟩
This message must be entered in response to a FM RQD message. The unit mnemonic specifies the unit to be used for the subject file.

⟨mix index⟩ FR
This message specifies that the input reel, the reading of which was just completed, was the final reel of an unlabeled file.

⟨mix index⟩ IL ⟨unit mnemonic⟩
This message is entered in response to a NO FILE message, and specifies the unit on which the required input file is located. The file may be either labeled or unlabeled.

⟨mix index⟩ JOB
The Job Table for the specified job will be displayed on the unit where this message is entered.

LD DK
or
LT MT
The LDCNTRL/DISK Program will search for a tape or card file with a file label of CONTROL/DECK. If found, the file will be placed on disk as a pseudo card deck for DK, or on magnetic tape for MT.

MIX
or
MIX SC ⟨integer⟩
The MIX table will be displayed on the specified display unit. If no display unit is specified, the one at which the message is entered will be assumed.

⟨mix index⟩ OF
This message may be entered in response to a NO FILE message if the file is a COBOL optional file. The specified program will then proceed without it.

⟨ mix index⟩ OK

The MCP will reactiviate a job which was suspended because of the condition designated by either a DUP LIBRARY, NO USER DISK, NO FILE ON DISK, or OPTR ST-ED output message.

OL ⟨unit mnemonic⟩

The Label Table will be displayed on the unit where this message is entered.

⟨mix index⟩ OU ⟨output code⟩

This message may be entered in response to either a LP RQD, LP PBT MT RQD, or PBT MT RQD output message. The output code may be empty, or may contain one of the following two letter codes: LP = line printer, MT = magnetic tape (printer backup tape), DK = disk (printer backup disk). The subject line printer file must be produced on the specified unit. If the output code is empty, either LP or MT may be used.

PB ⟨unit mnemonic⟩
or
PB ⟨pbd number⟩

The printer backup file on the specified unit will be printed. If a specified tape is not a printer backup tape, the message NOT PRINTER BACKUP TAPE will be displayed. The pbd number is the integer part of a PBD output message.

PCD

The MCP will display the name and first card image of each pseudo card deck on disk. If there are no pseudo card decks on disk, the message NO DECKS ON DISK will be displayed.

PER
or
PER ⟨unit type mnemonic⟩

where:

⟨unit type mnemonic⟩ ::= ⟨unit mnemonic⟩ | CD | CP | CR | LP | MT | MTX | PP | PR | SP

The specified peripheral table will be displayed on the unit where this message is entered. If no unit type is specified, all peripheral units will be displayed.

PG ⟨unit mnemonic⟩
The tape on the specified magnetic tape unit will be purged if the unit is Ready, in Write Status, and not in use.

⟨mix index⟩ PR = ⟨priority⟩
The priority of the specified job in the mix or schedule will be set to ⟨priority⟩.

RD =
or
RD ⟨deck list⟩
where:

⟨deck list⟩ ::= ⟨deck number⟩ | ⟨deck list⟩, ⟨deck number⟩
⟨deck number⟩ ::= #⟨integer⟩

The specified pseudo card decks will be removed from disk. If the form RD = is used, all pseudo card decks will be removed.

⟨mix index⟩ RM
This message may be used in response to a DUP LIBRARY output message. The disk file with the label specified in the DUP LIBRARY message will be removed.

RN
or
RN ⟨integer⟩
The integer specifies the number of pseudo card readers to be used. The number specified at Halt-Load time is zero. If this message requires that pseudo readers be turned off, they will complete the handling of pseudo card decks in process, if any, before being turned off. If no integer is included, the current number of pseudo card readers will be displayed.

RO -- (see SO)

RW ⟨unit mnemonic⟩

A rewind and lock action will be performed on the file on the specified magnetic tape unit. If the unit is in use, the action will be performed upon completion of the operation being performed.

RY ⟨unit mnemonic⟩

The specified unit will be made ready for use if it is in Remote status and is not in use.

SCH
or
SCH SC ⟨integer⟩

The schedule table will be displayed on the specified display unit. If no display unit is specified, the one at which the message is entered will be assumed.

SO ⟨option specifier⟩
or
RO ⟨option specifier⟩
or
TO ⟨option specifier⟩

The specified option will be set, reset, or displayed respectively. The options and mnemonics are to be specified.

SF ⟨decimal number⟩

where:

⟨decimal number⟩ ::= ⟨integer⟩ | ⟨decimal fraction⟩ |
⟨integer⟩ ⟨decimal fraction⟩

⟨decimal fraction⟩ ::= ⟨integer⟩

The multiprocessing factor will be set to the decimal number. The multiprocessing factor is normally 1.00, but may be set to any number from 0 to 100.

⟨mix index⟩ ST

The specified job will suspended temporarily. It may be reactivated with an OK message.

SV ⟨unit mnemonic⟩
The specified unit will be made inaccessible as soon as it is not in use. It may be made accessible with an RY message or a Halt-Load operation. The message ⟨unit mnemonic⟩ TO BE SAVED or ⟨unit mnemonic⟩ SAVED will be displayed, as appropriate.

TF
The multiprocessing factor will be displayed (see output messages).

⟨mix index⟩ TI
The following output message will be displayed:

⟨mix index⟩ : ⟨processor time⟩ IN FOR ⟨elapsed time⟩

where processor time is the time used and elapsed time is the time since the job entered the mix. Both are given in minutes and tenths of minutes.

TO -- (see SO)

TR ⟨integer⟩
The time will be reset to that specified by the integer which must be four digits. The first two digits specify the hour (0 to 23), and the last two specify the minute (0 to 59).

⟨mix index⟩UL⟨unit mnemonic⟩
This message may be used in response to a NO FILE message in order to designate the unit on which an unlabeled file is located. The subject file may be either labeled or unlabeled. All records including the label, if any, will be read as data. (This message differs from the IL message in that with the IL message the label is not read as DATA.)

WD
The MCP will display the date currently being used by the system. The date is given in the format MM/DD/YY.

WT

The MCP will display the time of day at the time the message was entered. The time is given in hours and minutes based on a 24 hour clock.

⟨mix index⟩ XS

The specified job in the schedule will be entered into the mix regardless of how full the mix is, and how inefficiently it will run.

SEPARATELY COMPILED PROCEDURES.

BINDING.

The B 6500 compilers are designed to allow separate compilation of program units, where a program unit is a logical subdivision of a program such as a FORTRAN subroutine or an ALGOL procedure. A program is a collection of any number of program units. For a program to function, it is necessary that its program units be tied together. This tieing together is called binding and is performed by the binder on object code files. The binding of code files can occur in three different ways:

a. Requesting a compiler to bind external program units at compile time.

b. Explicitly requesting the binder to bind two or more compiled program units.

c. Implicit binding at run-time (presence-bit binding).

COMPILE-TIME BINDING. This method of binding can be used when compiling a program unit that calls upon, or is called by, a prevvously compiled program unit. To illustrate this type of binding, assume that there resides in the library the object code file Y/A for a FORTRAN source subroutine named ALPHA. A FORTRAN main program that calls ALPHA is to be compiled to library. At this time, the main program may be compiled to library with or without binding a subroutine for ALPHA.

If compile-time binding is desired, compiler control cards would specify the object code file, Y/A for example, to be used for subroutine ALPHA. After successful compilation, the binder will bind the two program units forming one complete program. The file Y/A still remains in the library and is available to be bound with other program units. If compile-time binding is not performed, the main program will be compiled to library with the reference to subroutine ALPHA marked as unbound.

EXPLICIT BINDING. The binder program can be initiated by instructing the MCP to do so with the proper MCP control information. Data submitted to the binder will direct it to perform the binding operation on the specified code filer. For example, assume that the main program unit in the previous example has been compiled to library without binding its subprogram. The main program's code file was compiled to the library using the identification X/MAIN. The main program calls on subroutine ALPHA which is on disk as Y/A. The binder is executed to bind X/MAIN with Y/A for ALPHA. The result is a complete program capable of being executed. Both program units are still available on the disk to be bound to other separately compiled program units. Improper data to the binder, such as a file being specified which is not object code or a file being specified which is not on the disk, will yield an error message.

EXECUTION-TIME BINDING. The third method of binding will occur if a program is submitted for execution with one or more of its separately compiled program units unbound. When the program attempts to call the unbound program units, the MCP will perform automatic run-time binding. This method of binding does not generate permanent binding of program units. However, once a program unit is called and bound, it remains bound until the completion of this execution. When the execution terminates, no bound version of the complete program will remain on disk. Only the unbound program units that existed before the execution are retained.

Execution-time binding occurs when a process references a separately compiled program unit which has not been bound to the program. Since program unit object code files on the disk are referenced by their disk library file names, the program unit names may be label-equated to their corresponding object code file name. If no label-equation action is specified, all but the last part of the main program file name becomes a library specifier to be used in locating object code files by default.

For the following example, assume that:

a. The main program, X/MAIN, has been compiled with three program units (ALPHA, BETA, and GAMMA) specified as externally compiled.

b. BETA is label-equated to BETA/GREEK.

c. ALPHA and GAMMA are not label equated to a file specifier.

d. An object code file named X/ALPHA exists on disk.

e. No object code file named X/GAMMA exists on disk.

The first execution of a reference to BETA will cause execution-time binding of BETA/GREEK to the program X/MAIN. Then BETA (BETA/GREEK) is entered and execution is continued.

The execution of a reference to ALPHA will cause the library directory X to be searched for X/ALPHA. When X/ALPHA is found, the code file is bound to the program X/MAIN, entry to ALPHA occurs, and execution is continued. Further executions of references to ALPHA or BETA cause normal procedure of subroutine entry action, provided that the level at which ALPHA and BETA are declared is not exited.

Finally, the execution of X/MAIN is completed and the MCP performs its normal End-of-Job termination functions. Note that even though the object code file for GAMMA did not even exist in the system,

the program X/MAIN was still executable. If the program unit GAMMA has been accessed, a system message notifying the operator of the NO PROGRAM ROUTINE condition would have been generated.

Execution-time binding is less efficient than compile-time or explicit binding and, consequently, should not be the normal mode of operation on frequently run programs.

DISK LIBRARY.

DISK FILE STRUCTURE.

Each disk address references a disk segment which is an area of disk containing room for 30 words of information. A disk file consists of a file header and a number of areas which are not necessarily contiguous with each other. Each disk area is an uninterrupted sequence of segments, and all of the areas for a given file have the same size. A file header is an uninterrupted sequence of disk segments of variable length depending upon the number of areas used by the file. The header contains various information as given below:

a. The interlock field is used to solve disputes which may arise in accessing the file. It is set or checked whenever a requestor attempts to access the file.

b. The disk address field of the file header contains a zero if there are no current users of the file. If there are users, then a copy of the header resides in main memory and the address field of the file header points to the copy.

c. The volume serial number contains either the volume serial number of the library tape containing the latest copy of the file, or the disk address of the area if it is on the disk.

d. The update code indicates whether or not this file has been altered since it was loaded onto disk.

e. The open count indicates the number of users currently accessing this file.

f. The file type indicates whether the file contains data, object program, object subroutine, etc.

g. The security code defines the class of users for which this file may be accessed.

h. The data format specifies the structure of the data within the file. A user, if he specifies a format, is not allowed access to the file if his format is incompatible with this format. If the user's format is compatible, or if the user does not specify a format, he is allowed access. The MCP will automatically unblock the data into records as required.

i. NUMROWS is the current number of disk areas in use by the file.

j. AREASIZE is the number of logical records per area.

k. EOF count is the relative number of the last logical record in the file. The count is -1 when the file is empty.

l. SAVEFACTOR is the number of days the file is to be saved on disk from the date of last access.

m. CREATION DATE is the date on which the file was created.

n. LAST ACCESS DATE is the date on which the file was last used. The MCP may use the information in this field to automatically select a file, or set of files, to be dumped to a library tape in the event that an overlay of disk files is required.

o. Disk residence time is used by the log routine to determine how much disk a file used, and how long it was used. The date is entered in the system log and is available to the user for accounting purposes.

p. The area address fields are a set of fields which contain the initial disk address of each area in use by the file. If an area is not allocated, the corresponding address field will contain a zero.

The records in the file are addressed relative to the beginning of the file, where 0 is the first record and n is the last record in the file. Assuming that there are 1000 records per area and record 2345 is requested, the disk area required is area number 2 (2345 divided by 1000). Knowing the disk area number, the initial disk address of this area is obtained from the appropriate address field. The disk address of the segment containing the beginning of the record is then computed by adding the area initial address to (2345 modulo 1000)xK, where K= (the number of words per logical record +29) divided by 30), the number of segments required for a single record.

DISK DIRECTORY.

All files on the B 6500 System are referred to by an actual file name or label. The actual file name is a sequence of identifiers separated by a /. Each identifier may be of arbitrary length, but if the identifier is longer than 17 characters, only the first 17 will be used. Any number of identifiers may be used to construct a file name.

Correspondingly, the disk directory is really a collection of files organized as a tree structure. Such a file will be referred to as a directory. The directory at the origin of the tree structure will be referred to as the master directory. The directory body is composed of records which are 90 words (three segments) long. Each record contains a list of entries and ends

with a link to another record in the same directory. Each entry in a given record has several parts which are as follows:

a. The identifier part contains an identifier which is 17 characters in length.

b. The address part contains the address of the header of a file which may or may not be another directory.

c. The file type part indicates the nature of the file to which the address part points.

d. The volume number is used to specify which tape is needed if the file has been dumped to tape.

In order to reference a disk file by means of the actual file name, each identifier is used to find a directory which is reached through the preceding identifiers. A directory header contains the information as to how the directory is to be indexed. The main directory will be indexed by scrambling the identifier. Other directories will be indexed or searched, depending on the size of the directory at that level.

If a directory is scrambled, the directory record will be indexed and then searched for a matching identifier.

If the end of the record is found, the link indicates the next record to be searched. If a match is found, the address part of the entry will specify the location of the file which is to be searched next. If this file is a directory, then another label is expected. If, however, the file is not a directory file, the location is the disk address of the header of the file being referenced.

PROCESS SCHEDULING ALGORITHM.

The B 6500 MCP incorporates a dynamic scheduling algorithm. A programmer-defined priority may be assigned to any job that is to

be executed by the system. If no priority is specified, a default priority value will be assigned by the MCP. When a job is first introduced into the system, an entry for that job is made in a queue called the SHEETQUE by the MCP control card routine. After the control card routine completes its tasks, the entry is moved from the SHEETQUE to a queue called the READYQ.

The READYQ is divided into two parts, an active part and a passive part. At the time that the job is moved from the SHEETQUE to the READYQ, the resources that are required for efficient execution of the job are checked against the resources that are avilable to the system and are not currently in use. If sufficient resources are not currently available, then the job is placed in the passive part of the READYQ and temporarily assigned a priority of zero. If sufficient resources are currently available, then the job is placed in the active part of the READYQ.

The active and passive parts of the READYQ are ordered according to priority. The calculation of the priorities is performed in a well isolated section of the MCP. Thus, the user may easily modify the priority algorithms that are supplied for the active and passive parts of the READYQ to suit his own needs.

A typical scheduling algorithm that would be satisfactory for both the active and passive parts of the READYQ may be described as follows.

A default value of approximately 25 jobs (mm) in the active portion of the READYQ is assumed by the MCP. The value mm may be changed at each installation with an appropriate message to the MCP. The value for mm may be viewed by the MCP as the maximum number of jobs that it will allow in the active portion of the READYQ at any one time. If the MCP detects that the system is being overloaded, either by noting an excessive number of overlays or some other undesirable condition, the MCP may reduce the value of mm.

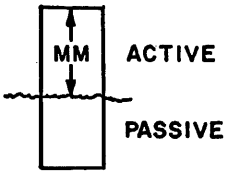
When an interrupt occurs, the MCP compares the priority of the job that was running at the time of the interrupt with the priority of the job that was at the top of the READYQ. If the job that was running at the time of the interrupt has a priority which is greater than or equal to the job at the top of the READYQ, then the MCP returns to the job that was running at the time of the interrupt. If the priority of the job at the top of the READYQ is greater than the priority of the job that was running at the time of the interrupt, then the MCP will insert the job that was running at the time of the interrupt into the proper section of the READYQ. It then removes the job that was at the top of the READYQ and proceeds to execute it.

The active portion of the READYQ is periodically rearranged according to a priority algorithm. A default rearrange factor causes the active portion of the READYQ to be rearranged approximately once every second. Each installation may override this default with a message to the MCP stating that the queue is to be rearranged once every n milliseconds, or once every n interrupts.

Only jobs that are actually ready to run appear in the READYQ. When a job is waiting for some event to happen, such as I/O buffer to be filled, it will not be entered in the READYQ. It will be entered into some other queue in the system. When the event occurs upon which the job was waiting, the job will be moved from that queue to the READYQ. When a job in the active portion of the READYQ terminates, the MCP normally rearranges the passive portion according to a priority algorithm and moves the job at the top of the passive portion to the bottom of the active portion.

Priority for the active and passive parts could be determined by the formula shown in figure 4-1.

The first term in the priority equation is A1xD. This priority equation is composed of the summation of six coefficients and eight different factors. The first of these factors is the



PRIORITY = A1 x D	D - DECLARED PRIORITY
+ A2 x TR	TR - TIME IN READY QUEUE
+ A3 x TE	TE - ELAPSED TIME
+ A4 x TW/(TP+1)	TP - PROCESSOR TIME TW - TOTAL WAIT TIME (TE - TP)
+ A5 x C	C - CORE USAGE
+ A6 x 1/(TT-CT)	TT - TARGET TIME CT - CURRENT TIME

Figure 4-1. B 6500 Ready to Run Queue

priority declared (D) by the user, thus giving the user control over the scheduling algorithm. The second term in the equation involves the TR factor. It is a function of the time since the most recent entry was made. This factor may be included to ensure that a job does not become permanently entrenched at the bottom of the READYQ.

The third term in the equation is the TE factor. It is the time elapsed since the job was first entered into the passive part. In an effort to decrease turn around time, this factor may be included so that the longer the job has been in the system the higher its priority will become.

The fourth factor in the equation is TW. TW is the total wait time which is the total amount of time this job has spent waiting in some queue in the system. This factor may be included to ensure that each entry progresses to the top of the READYQ at some point in time.

The fifth factor in the equation is TP. This represents the total amount of processor time that has been assigned to this job. The priority algorithm may include this factor to ensure that a processor-bound job does not utilize all of the available processor time.

The sixth factor in the equation is C. This represents the amount of core storage currently in use. Initially, C is produced as an estimate by the compilers. This core estimate is based on the total segment size, the size of the segment dictionary, the number of buffers, the size of the buffers, and the amount of array storage required. This estimate may be changed by a control card. After the job has moved from the passive to the active portion of the READYQ, the C factor is changed by the MCP to indicate the amount of core that the job is actually using. It is generally felt that smaller jobs should be given top priority so that they will clear the system. Thus, the coefficient A5 will have a default value which is negative. A default is provided for each of the coefficients A1 through A6. Any of these coefficients may be changed at the user's installation through the use of a suitable message to the MCP. Any one of the factors could be eliminated from consideration by setting the appropriate coefficient to zero.

The next factor in this equation is the difference between a target time and the current time. Thus, priority for a job could increase exponentially as the target time is approached.

This discussion is not intended to dictate the scheduling algorithm that one may select for an installation. It is only an example of the type of algorithm that may be incorporated. All of the factors discussed here, and many more, are maintained by the MCP for each job. This information is necessary to perform functions such as logging and storage allocation, so no penalty is being paid for the benefit of the scheduling algorithm. The user is free to use or ignore whichever factors he chooses.

MULTIPROGRAMMING CONSIDERATIONS.

Multiprocessing on the B 6500 System consists of having more than one job or "process" running at a given time. This is accomplished exclusively by multiprogramming (interleaved execution) on a single processor system and by a combination of multiprogramming and parallel processing on a multiple processor system.

MULTIPROGRAMMING.

Multiprogramming operation on a computer system requires the ability to interleave execution of processes. This means that a processor is not exclusively allocated to a process for the entire execution of that process. Multiprogramming on the B 6500 System is implemented by queuing processes in the READYQ Queue, and allocating (or reallocating) processors to the processes with the highest priority.

Since a processor is not allocated exclusively to a process, the process must contain all of the information necessary to describe its status when it is not being executed by a processor. This is accomplished by creating a separate "stack" for each process that is to be executed.

A process may have one of the three following states:

a. Active.
b. Inactive.
c. Suspended.

A process is active when it is being executed by a processor. An active process may be made inactive by the MCP if a higher priority process needs a processor. An active process may cause itself to be suspended by executing a WAIT or HOLD, or by requesting an I/O operation which results in a WAIT on the I/O complete event.

If an active process is made inactive by the MCP, it is placed in the READYQ. The READYQ contains only those processes that are "ready to run" and are waiting only for a processor.

If an active process suspends itself, the subsequent action of the MCP depends upon the types of queues with which the process is associated. If the suspended process is linked to an event wait queue, the process will get put into the READYQ when that event is caused.

If a process is linked into a software interrupt queue (event interrupt queue), the process will be moved to the READYQ upon the occurance of the event if it is suspended, or it will be interrupted if it is active.

If a suspended process is not linked into an event queue or the READYQ, it can not be reactivated. For example, when a process is terminated, the process is suspended, any queue entries are delinked, and the process is linked into the terminate queue.

PARALLEL PROCESSING.

Parallel processing occurs on B 6500 Systems which have more than one processor. The fact that multiple processors are available does not preclude the multiprogramming of processes. It merely means that processes may be executed simultaneously to increase the throughput of the system.

The structure of the MCP is such that processors are considered a resource to be allocated like other system resources. Therefore, the only additional requirements for parallel processing are the inclusion of some additional MCP LOCK variables to prevent simultaneous execution of exclusive MCP functions. For instance, it would not be desirable to have two processors simultaneously trying to make an absent program segment present in memory. This circumstance is prevented by an MCP LOCK which is set and tested by the presence bit procedure. The first processor entering presence bit locks all others out until it is safe for them to enter the procedure.

THE STRUCTURE OF OBJECT PROGRAMS.

Object programs reside on the disk where they are referenced as

code files by the MCP through the disk directory. The reference will be the result of either an EXECUTE request or the GO part of a COMPILE AND GO. In either case, the code file will have been constructed by a compiler or the binder program.

The main function of a compiler is to convert symbolic source statements into object machine language code. However, efficient utilization of memory in a multiprogramming environment requires that object code files be segmented so that during execution of an object program the only portion of the code file resident in core is that segment currently being processed. Segmentation of the object code file is an additional function of the compiler.

The length of a program segment is variable, depending on the program logic and language used. ALGOL program segmentation is based on the block structure of the source program, where each block is compiled into a code segment. COBOL programs are segmented by section level, unless specified otherwise by the programmer. FORTRAN program segmentation is by program unit (subroutine or main program) level and, if necessary, these units are further segmented to optimum segment size. Segmentation of FORTRAN programs may also be effected by programmer option.

The code file consists of a number of variable-length records, each record being some multiple of 30-word disk segments. The first record in the file is one disk segment in length. It contains linkage and object program information for use by the MCP at job initiation. Following this record is a group of logical records where each is a program segment, for all but the outermost segment of the program. Additional logical records contain code related to formats, lists, and other compiler-generated data. Also included is compile time information for program input/output files.

The object code for the programs outer-most segment follows next. Since compilation is in most cases one pass, the outer-most segment, which encloses the entire program, cannot be written onto

disk until compilation is completed.

The final logical record in the code file contains a directory referencing all previously written logical records. Each entry in this record is one word in length and contains the relative record address and the size (in words) of the logical record it references. These words are defined as segment descriptors.

When the logical record of segment descriptors is read into core by the MCP at job initiation time, it is referred to as the segment dictionary. In conjunction with display register D1, the segment dictionary becomes the basis for accessing the object code segments within the code file. The code segments are read into core by the MCP as a result of presence bit interrupts incurred by accessing the segment descriptor for the code segment. The frequency and order in which the code segments are processed is determined by the dynamic flow of the object program.

RE-ENTRANT CODE.

Re-entrant code is common object code which may be accessed by several programs in a multi-processing environment. Implementation of this concept requires that object code must not be modified during its execution. In the B 6500 System, this requirement is met by separating working storage and object code, and by programming in higher level languages which do not generate self-modifying code.

Working storage is assigned a separate memory area as a push-down stack which is unique for each process in the mix. These stacks contain temporary results, simple variable values, data array descriptors, and program control information. Object code segments are referenced by segment descriptors located in a segment dictionary.

In addition to software protection of object code, the B 6500 System provides hardware code protection. Object code words are also memory protected by the tag bits associated with each word

in memory. Any attempt to programmatically modify code words in memory causes a memory protection interrupt which results in termination of the process attempting to modify the code. Since object code segments are referenced through a segment dictionary, the code segments for all programs accessing a common code file are made re-entrant by linking these programs to one common segment dictionary. This also ensures that only one copy of a required code segment will be present in memory.

Identification of re-entrant code users is effected by creating a word in the base of a segment dictionary which points to a link word in the working stack of the first associated process. This link word will, in turn, point to a link word in the working stack of a second concurrent or parallel common process, and so on.

The stack pointer word in the segment dictionary also contains a count of current and latent users of a common segment dictionary. The current user count defines the number of process stacks linked together through a common segment dictionary. The latent user count defines the number of processes which have accessed the segment dictionary, but which are not currently using it. These processes have not terminated and may return to the current user list. When all users have terminated, the memory space for the segment dictionary and all remaining code segments is de-allocated.

This re-entrant capability is also extended to include data which does not change in value such as literal strings and read-only data. This is accomplished by placing their associated descriptors in the segment dictionary.

COMPILER/MCP INTERFACE.

A compiler is a special purpose computer program which accepts, as input, source statements in the language for which the compiler was written. The output of a compiler is a file on disk consisting of object code.

A compilation followed by immediate execution of the resultant object code is referred to as a Compile and Go. Implementation of this type of operation requires certain functions to be performed by the compiler and the MCP which involves:

a. Recognition of a compiler by the MCP.
b. Communication between the compiler and the MCP.
c. Construction of a standard object code file by the compiler.

A compile card is a request to the MCP to schedule a particular compiler for execution and provide special handling for this program. The schedule entry for this execution will also reflect the option associated with the compilation (Compile and Go, Compile for Syntax, Compile to Library). The schedule entry for a Compile and Go execution is appended with a skeleton schedule representing the GO part.

A word in the working stack of the compiler is used for communication between the compiler and MCP. This stack location corresponds to the first symbolic declaration in the source language or the compiler, thus this common word is known to both the compiler and the MCP. The compile code (GO, SYNTAX, LIBRARY) is stored in this word by the MCP as a compiler enters the job mix. This code value may direct the compiler to suppress code generation if the compilation is for syntax only. During compilation of the source language, the common word also accumulates the count of syntax errors. If the source file contains no syntax errors and the compilation is not for syntax only, the compiler writes the object code to disk and closes this file with lock, through the MCP. The MCP recognizes the calling program as a compiler and, if the code value in the common word specified GO, the object code file header is written to scratch disk and this disk address is stored in the common word. Otherwise, the code value specified LIBRARY and the file header is entered in the disk directory, making the code file permanent.

When compilation terminates, the MCP again recognizes that the calling program was a compiler and, if the common word specified GO, the skeleton entry is updated from the first record of the code file and entered into the schedule. If the common word value indicates the occurrence of syntax errors, the skeleton schedule entry is discarded, thus suppressing the GO part.

In addition to generating the object code file, the compilers are responsible for supplying scheduling information to the MCP. This information, in the first record of the code file, includes the core estimate, stack size, and pointers into the code file for locating the segment dictionary, the file parameter block, and the first executable code segment. A compilation to the library may require the MCP to modify this information if program parameter cards had been included with the compile request. These changes will be in effect for all subsequent executions of the object code file.

INTRINSIC FUNCTION.

Historically, the programmer was responsible for writing an entire program, including those parts which performed input/output operations and arithmetic function calculations. Later, as it became apparent that a large amount of programming effort was being spent writing similar functions for different programs, one set of standard routines to perform these operations were written for all users. The required routines were included in each program and became a part of the program object code file.

In a multiprocessing system, the inclusion of these common functions in each program causes multiple copies of the functions to be present in main memory. In order to make more effective use of main memory, the code for these routines is included in the operating system and is accessible, as intrinsic functions, to all user programs.

Since all programs in the B 6500 system are written in high-level languages, the use of intrinsic functions is implemented by the compilers.

Each compiler recognizes the names of those intrinsics that are allowable in each language. An intrinsic name which occurs in a source language statement is processed by a compiler as a pre-compiled procedure. Each compiler is responsible for verifying that actual parameters agree with the formal parameters specified for each intrinsic.

For each intrinsic required by the object program, the compiler emits a Make Program Control Word (MPCW) syllable into the object program outer block code segment. When the object program is run, each MPCW syllable is executed as part of the code which initializes the programs D[2] stack. Execution of the MPCW instruction causes a program control word (PCW) to be generated. This PCW is stored in a stack location within the D[2] addressing environment which makes it addressable from any level within the program.

The PCW for an intrinsic contains a lexicographic level field of zero which refers to the base of the D[0] (MCP) stack. The index field locates the segment descriptor within the D[0] stack. This descriptor contains the memory/disk address for the required intrinsic code. Since the D[0] stack is global to the total addressing environment, any segment descriptor in this stack is accessible from any program which references a PCW containing a lexicographic level field of zero. Because there is a single segment descriptor in the D[0] stack for each intrinsic, only one copy of the object code is present in memory. Thus the intrinsics are re-entrant.

The intrinsics are written in an ALGOL-like language and compiled as a set of disjoint procedures into a single code file. This file is bound into the MCP code file by the binder program. As a result of the binding process, the intrinsics are included as procedures in the MCP, and therefore are accessible through the D[0] stack.

INTERRUPTS.

The interrupt handling mechanism of the MCP deals with two classes

of interrupts, hardware interrupts and software interrupts. The hardware interrupts are generated automatically by the B 6500 System and are handled by the MCP interrupt procedure. Software interrupts are programmatically defined for use by the MCP and object program processes. Software interrupts allow processes to communicate with each other and with the MCP.

The B 6500 processor interrupt system is the primary interface between the MCP and the hardware. Because of the importance of this interface, the relevant features of the B 6500 processor will be described along with the discussion of interrupt handling.

An interrupt is a means of discontinuing a process subject to the occurrence of certain conditions. In order to fully understand the operation of B 6500 interrupts, an understanding of the concept of control state is required.

A given processor may operate in one of two distinct states: normal state or control state. The primary difference between normal state and control state is that external interrupts are disabled while a processor is in control state. Also, there are certain operators, such as some forms of Scan Out (SCNO) which can only be executed by a processor in control state.

A processor in normal state may enter control state by executing a Disable External Interrupts (DEXI) instruction, or by entering or exiting to a control state procedure. Similarly, a processor in control state may enter normal state by either executing an Enable External Interrupts (EEXI) instruction, or by entering or exiting to a normal state procedure.

It should be noted that while in control state, a processor can selectively mask out any or all I/O multiplexor (MPX) interrupts before executing an EEXI. The result is a processor in normal state which does not receive the masked MPX interrupts.

HARDWARE INTERRUPTS.

When an interrupt condition occurs in a processor, the processor enters control state, marks the stack, and inserts three words in the top of the stack. These three words are the Indirect Reference Word (IRW) pointing to D[0]+3, followed by two interrupt parameters, P1 and P2, which contain information indicating the nature of the interrupt condition. D[0]+3 should contain a Program Control Word (PCW) pointing to the MCP hardware interrupt procedure. However, an IRW or IRW chain pointing to a PCW is a legitimate condition. At this point, the processor enters the procedure pointed to by the PCW passing P1 and P2 as parameters. When the processor enters the MCP hardware interrupt procedure, it also enters control state. This is accomplished by generating the interrupt procedure PCW with the control bit on.

Upon entry to the hardware interrupt procedure, the parameter P1 is analyzed to determine the type of interrupt which occurred. For some interrupts, such as presence bit interrupts, P2 contains additional information to be used by the interrupt procedure.

The action to be taken for each interrupt is described in the following paragraphs. The description covers the following three classes of hardware interrupts:

a. Syllable dependent interrupts.
b. Alarm interrupts.
c. External interrupts.

The stack structure prior to calling the interrupt procedure is shown in figure 4-2.

After entering the interrupt procedure, the program base register is pointing at the interrupt procedure, PIR and PSR are pointing at the interrupt procedure entry point, and the Return Control Word for the interrupt procedure's exit is pointing back to the object programs code as shown in figure 4-3.

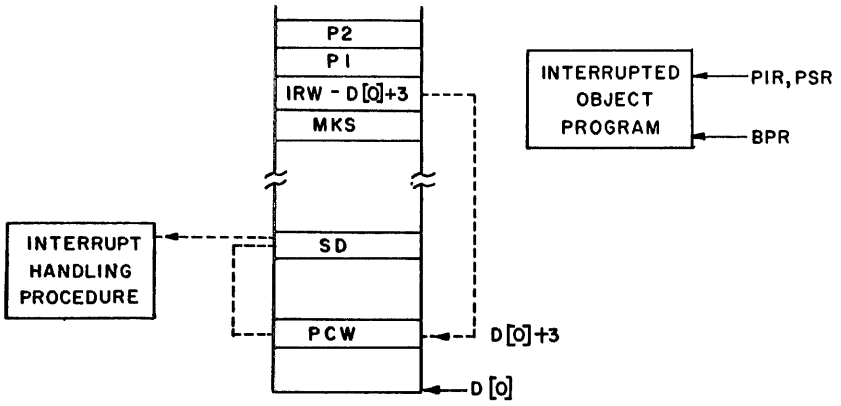


Figure 4-2. Stack Prior to Interrupt Procedure Entry

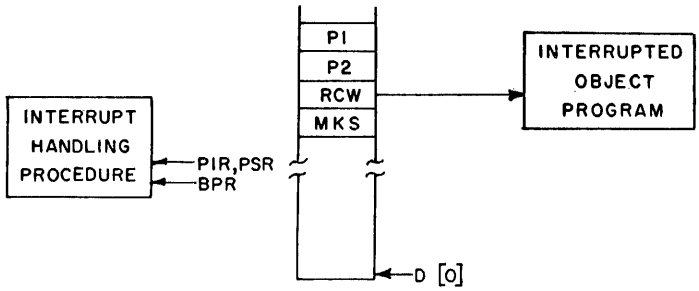


Figure 4-3. Stack Following Interrupt Procedure Entry

SYLLABLE DEPENDENT INTERRUPTS.
These interrupts are detected by the processor operator logic.

ARITHMETIC ERROR INTERRUPTS. This group includes the divide-by-zero, exponent overflow and underflow, invalid index, and integer overflow interrupts. Termination of the program will occur unless programmatic control of the interrupt is specified.

PRESENCE-BIT. This interrupt occurs when the processor accesses a data descriptor or segment descriptor with the presence-bit off, indicating that whatever the descriptor references is not present in memory. Upon detecting a presence bit interrupt, the presence bit procedure PRESENCEBIT is called by the interrupt procedure. It is clearly not desirable that two processes attempt to make the same segment or data present since this would ultimately require that the descriptor point to more than one location in memory. Consequently, PRESENCEBIT is equipped with a lock which assures that only one process will be executing it at a time.

However, this does not entirely solve the problem. In order to make the absent object present, PRESENCEBIT must initiate at least one I/O. While it is waiting for the I/O to be completed, PRESENCEBIT is not being executed. Therefore, it is desirable for PRESENCEBIT to "unlock" the presence bit lock while waiting for the I/O. However, doing this would make it possible for another process to attempt to make the same object present. This difficulty is overcome by putting a separate "lock" on the descriptor. Any process now attempting to make present the object pointed to by this descriptor will know that another process is already making it present and will wait for the descriptor to be "unlocked" before executing PRESENCEBIT.

If PRESENCEBIT finds that the original descriptor referenced by some copy is in fact present, it simply makes the copy present by inserting the memory address in the address field and turning on the presence bit.

MEMORY PROTECT. This interrupt occurs when the processor attempts to write in a memory location that currently has the memory protect bit of the tag field on. This bit indicates to the hardware that this word does not want to be written on. This interrupt results in a termination of the program.

BOTTOM OF STACK. This interrupt indicates that the processor tried to exit from the bottom of the stack. It results in a termination of the program.

SEQUENCE ERROR. This interrupt occurs when the processor receives data or instructions out of sequence. It results in a termination of the program.

SEGMENTED ARRAY. The occurrence of this interrupt indicates that the MCP has segmented an array row when storing it and has just attempted to index beyond the end of the current segment. The interrupt procedure must now replace the current segment by the proper segment, if one exists, and continue executing the process.

PROGRAMMED OPERATOR. This interrupt indicates that the current or active stack has attempted to execute an operator code which is not currently assigned. It allows the MCP to simulate the operator programmatically.

INVALID OPERATOR. This interrupt occurs when the processor attempts to execute a valid operator on data which is invalid for that operator. It results in termination of the program.

ALARM INTERRUPTS.
These interrupt conditions are not normally anticipated by the processor operator logic. They serve to inform the processor of some detrimental change in environment and can result from hardware failure as well as programming errors. They all result in termination of the program.

LOOP. This interrupt occurs when the processor has spent two seconds in the execution of one operator.

MEMORY PARITY. This interrupt indicates a faulty Read from Memory.

MPX PARITY. This interrupt indicates faulty reception of data from a multiplexor.

STACK UNDERFLOW. This interrupt occurs when the processor tries to delete through the base of the current stack.

INVALID ADDRESS. This interrupt indicates that the processor attempted to address a memory address which is not available to the system. The memory module may not exist or it may be inoperative.

INVALID PROGRAM WORD. This interrupt indicated that the processor has encountered a word which is supposed to be a program instruction word, but it isn't.

EXTERNAL INTERRUPTS.
These interrupt conditions are like the alarm interrupts in that they are not anticipated by the operator logic. However, they do not normally require immediate action and do not necessarily result in termination of the program. As mentioned above, none of the external interrupts can interrupt a processor in control state.

INTERVAL TIMER. This interrupt occurs at regular intervals of time when the interval timer for the processor has been set. If the timer is reset, it will no longer occur. It is intended for the use of the MCP in distributing MPX interrupts evenly among the processors.

STACK OVERFLOW. This interrupt occurs when the process stack has exceeded the estimated limitations. At present, this results in a termination of the program. In the future, however, it is expected that the hardware interrupt procedure will find more space for the stack, link things together, and restart the program. It will also have to modify the program estimate of stack size, and should inform the programmer that the stack estimate was too small.

PROCESSOR TO PROCESSOR. This interrupt occurs when one processor executes the HEYU operator which enables one processor to interrupt all other processors except those running in control state. If a processor is in control state, the interrupt is held in abeyance until it resumes normal state processing.

MPX. This interrupt group includes I/O Finish, MLC1, MLC2, MLC3, MLC4, GCA, and external MPX interrupts. These interrupts occur when a multiplexor wishes to communicate with a processor. They are handled in various ways depending on the specific type.

When an I/O Finish interrupt occurs, the hardware interrupt procedure calls the I/O Finish procedure which checks for errors which may have occurred. If no error is found, I/O finish initiates a new I/O. There are two queue structures related to the I/O operations: the WAITCHANNELQUES, one for each I/O channel, and the UNITQUES, one for each unit. When the I/O finish procedure has decided that it should initiate another I/O, it first checks the WAITCHANNELQUE for the channel it has just finished with and initiates the first I/O request in that queue. It then checks the UNITQUE for the unit it just used, removes the top entry from that queue, and **inserts it in the** WAITCHANNELQUE.

In order to prevent confusion, the WAITCHANNELQUE is not allowed to contain more than one I/O request for any given unit. If an I/O request occurs for a unit that is already in a WAITCHANNELQUE (for any channel), then the request is entered in the appropriate UNITQUE.

The Multiline Control (MLC) interrupts indicate that something like a data communications system wishes to communicate with the processor through a word interface of the multiplexor. The way this interrupt is handled depends on the nature of the device which is attempting to communicate with the processor.

GCA interrupts indicate that some sort of special control device (an analog device, a plotter, or some machine that the computer is controlling) wishes to communicate with the processor. Since there is only one GCA interrupt, it is clear that only one such device can be handled at a time. It is also evident that the handling of this interrupt is dependent on the nature of the device in question.

When a multiplexor is attached to the word interface of one of the system multiplexors, it becomes necessary to handle interrupts from the "external" multiplexor. This is the function of the external MPX interrupt which indicates that the processor must first interrogate the external multiplexor to determine the nature of the MPX interrupt.

SOFTWARE INTERRUPTS AND EVENTS.

Software interrupts allow a process to stop running (thereby releasing the processor) until a specified event occurs, or continue running and be interrupted if the event occurs. A software interrupt occurs when a process is interrupted by the direct action of some other process. In the following discussion, the implementation of this concept will be developed as it relates to the queues, the stack structure, and the MCP routines that concern themselves with software interrupts.

A process can be interrupted if it has an interrupt declaration within its scope.

Example:

```
INTERRUPT 12 : ON EVNT, A - A + 1;
```

When a process enters a block having an interrupt declaration, a Stuffed Indirect Reference Word and a Program Control Word are placed in the stack. This interrupt declaration must occur within the scope of its associated event declaration, as shown in figure 4-4.

```
EVENT EVNT;

PROCEDURE A;
INTERRUPT I1: ON EVNT, <STATEMENT>;
ENABLE (I1);
*

PROCEDURE B;
INTERRUPT I2; ON EVNT, <STATEMENT>;
ENABLE (I2);.
*HOLD;

PROCEDURE C;
*WAIT (EVNT);

PROCEDURE D;
*WAIT (EVNT);

PROCEDURE E;
CAUSE (EVNT);
```

Figure 4-4. Interrupt Declaration Example

Example:

```
EVENT EVNT;
```

An event declaration reserves two words in the stack and defines the identifier of a quantity which may be used to record an occurrence. The stack containing the interrupt declaration is linked into the EVENT INTERRUPT Queue by the event declaration as shown in figure 4-5.

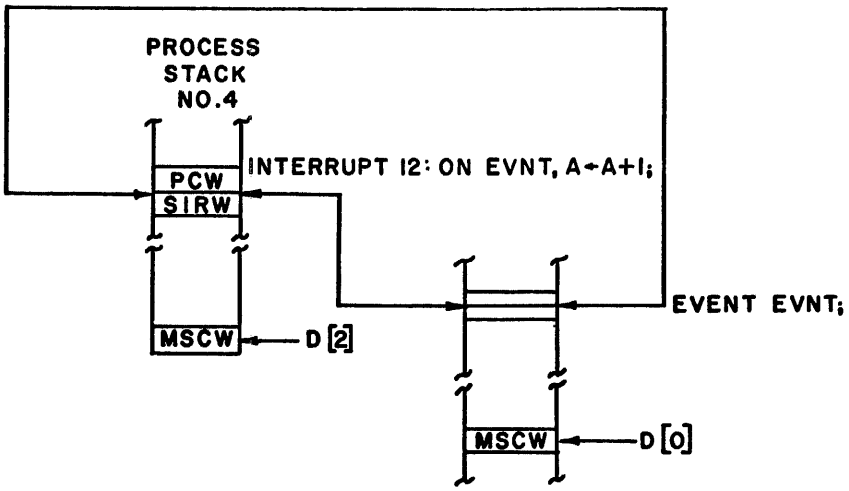


Figure 4-5. EVENT INTERRUPT Queue, Single Process

If a second process enters a block containing an interrupt declaration for the same event, then its stack is linked into the event interrupt queue as shown in figure 4-6.

The process in stack number 4 and stack number 2 will continue to run until the event occurs. When the event is caused, all of the process in the INTERRUPT Queue for that event are interrupted. The occurence of an event is invoked with the cause statement.

Example:

```
CAUSE (EVNT);
```

If a process causes the occurence of an event, the MCP scans the EVENT INTERRUPT Queue. As the MCP scans the EVENT INTERRUPT Queue, it will check to see if the interrupt has been enabled.

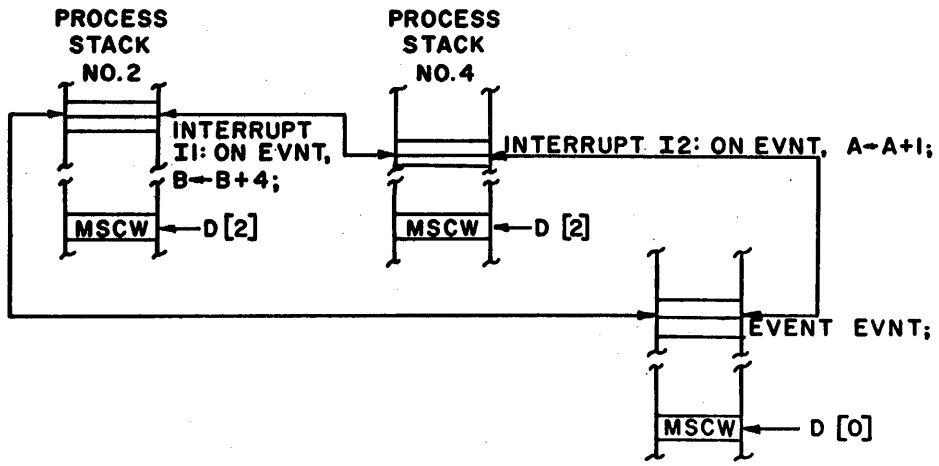


Figure 4-6. EVENT INTERRUPT Queue, Multiple Process

Example:

```
ENABLE (I2);
```

The enabling of an interrupt turns on the software interrupt enable bit (bit 46) of the Program Control Word of the two-word interrupt declaration mentioned previously. If an interrupt is not enabled and the event is caused, no action is taken by the MCP on that process, and it looks at the next process stack in the queue.

If interrupts are enabled in the next stack, the MCP makes an entry in the SOFTWARE INTERRUPT Queue. This queue is ordered by stack number. If the stack is active, i.e., another processor is working in the stack, the MCP will interrupt that processor with a processor to processor interrupt.

Next, the MCP forces a transfer of control to the statement related to the interrupt declaration. Upon completion of this statement, the process will return to its previous point of control unless a transfer of control is specified in the interrupt statement. In this case, the process will not return the point of

control before the interrupt, but will transfer control as specified in the interrupt statement.

As the MCP scans the EVENT INTERRUPT Queue finding enabled interrupts in inactive stacks, it makes an entry in the SOFTWARE INTERRUPT Queue doing nothing with that stack until it becomes active. Immediately after making the stack active, the MCP checks the SOFTWARE INTERRUPT Queue to see if there is an interrupt pointing to that stack. If an interrupt is found, the MCP forces a transfer of control to statement referred to by the interrupt declaration. Upon completion of the statement, control is transferred as described previously.

It is possible for a procedure to be entered, get linked into the EVENT INTERRUPT Queue, and either exit the procedure without enabling the interrupt or exit the procedure before the event is caused. In either case, this stack is delinked from the queue.

Having enabled a software interrupt, it is sometimes desirable to suspend further processing of the code until an enabled software interrupt occurs. This action is invoked with the HOLD statement.

Example:

```
ENABLE (12);
HOLD;
```

When the event is caused and the related interrupt statement executed, control will pass to the statement following the HOLD.

A process can be intentionally suspended with the execution of a WAIT statement.

Example:

```
WAIT (EVNT);
```

The parameter of a WAIT statement is an event whose scope includes the block in which the WAIT resides. Upon execution of WAIT, the stack of that process is linked to the event declaration following an EVENT WAIT queue as shown in figure 4-7.

Stacks are removed from the WAIT queue when another process executes a CAUSE statement.

Example:

```
CAUSE (EVNT);
```

Stacks removed from the WAIT queue are linked into the READY Queue. The stacks shown in figure 4-7 represent the EVENT INTERRUPT Queue and the WAIT Queue at a point in time when the procedures are at a place in their code string indicated by the asterisk (*) as shown in figure 4-2. Procedure A is running in process stack number 4, procedure B in stack number 2, procedure C in stack number 20, and procedure D in stack number 7. Both queues are linked to the event declaration in the D[0] stack.

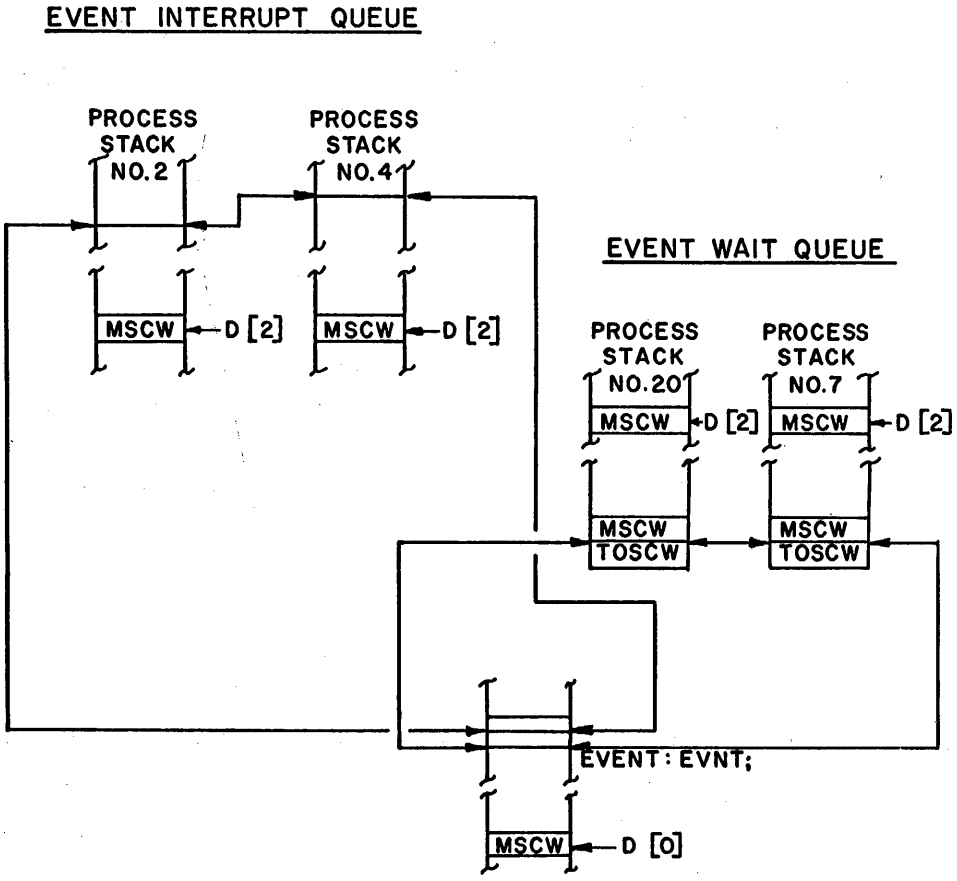


Figure 4-7. Event Queues

If the CAUSE statement in procedure E is executed at this point in time, stacks 20 and 7 are linked into the READY Queue and removed from the WAIT Queue. Pointers to the interrupt statements in stacks numbers 2 and 4 are entered into the SOFTWARE INTERRUPT Queue.

FILE CONTROL.

Since the B 6500 compilers allow the use of symbolic files, the MCP must be able to recognize the physical files present on the peripheral units and assign the units to a symbolic process file. The file control functions of the MCP consist of recognizing the existence of a file on a peripheral unit and assigning the peripheral unit to the appropriate process.

FILE RECOGNITION.

There are two types of physical files recognized by the B 6500 System, labeled files and unlabeled files.

Labeled files are those files which contain a label record (or records) as the first record(s) of a file. Since the label record contains a file label name, the MCP can recognize the existence of a labeled file and associate the appropriate peripheral unit with a symbolic process file.

Unlabeled files, however, must be assigned by the operator at the time that a process requires access to the file.

The format of file labels for various types of peripheral units are described in the following paragraphs. The physical file naming system used allows file names to be formed by a sequence of file identifiers separated by slashes. A file identifier is delimited by a blank or a slash (/) and may be of any length, but only the first 17 characters are used if the identifier exceeds 17 characters in length.

The following are examples of file names:

A
B/C
D/E/F
G/H/I/J

where:

a. A, C, F, and J are file identifiers.
b. B, E, and I are volume identifiers.
c. D, H, and G are file directory identifiers.

The organization of files is dependent on the I/O devices holding the file, each of which is discussed individually.

CARD FILES.
The format of card files is as follows:

LABEL CARD
(data deck)
END CARD

The format of the label card is:

⟨i⟩ DATA ⟨file name⟩ . ⟨any comment⟩

or

⟨i⟩ DATABCL ⟨file name⟩ . ⟨any comment⟩

The i represents an invalid character and must be in column 1. DATA indicates the data deck is punched using the EBCDIC (8 bit) character set. DATABCL indicates the data deck is punched using the BCL (6 bit) character set. Except for the invalid character in column 1, the card is free-field.

Example:

⟨i⟩ DATA DATACARD

The format of the end card is:

⟨i⟩ END ⟨any comment⟩

PRINTER FILES.

Upon opening a labeled printer file, the operating system will:

a. Skip to the top of the page.
b. Write a header label record(s).
c. Skip to the top of the page.

Upon closing a labeled printer file, the operating system will:

a. Skip to the top of the page.
b. Write a trailer label record.
c. Skip to the top of the page.

Header and trailer label record formats are identical. The format is:

LABEL ⟨file name⟩ ⟨program name⟩ ⟨comment field⟩

The comment field from the EXECUTE or COMPILE card used to execute or compile the program is saved and copied into the printer label. This aids the computer operator in locating the owner of printer listing. For example, the card EXECUTE MATRIX/INVERT JOHN J. JONES was used to put in execution a program which generated a printer file output. The printer label record would look like:

LABEL OUTPUT MATRIX/INVERT . JOHN J. JONES.

CARD PUNCH.

The format of a card deck produced in the card punch is:

LABEL RECORD
(data deck)
LABEL RECORD

The format of the label record is:

LABEL ⟨file name⟩

Example:

LABEL PUNCH/DECK

PAPER TAPE.

Paper tape files are always considered as unlabeled. For handling of unlabeled files, see unlabeled tape below.

UNLABELED TAPE FILES.

Unlabeled tape files are those which do not have any way of being self-identified. The system assumes for input or generates for output the following data formats:

a.	Single file volumes	⟨data⟩ **
b.	Multi-file volumes	⟨data⟩ *--------* data **⟩

where * denotes a tape mark.

The source languages can specify that input and output files are to be unlabeled. To produce multi-file volumes, the source program must close with no rewind, then open output with no rewind for each file on the volume (close with no rewind produces a tape mark). When a single file volume or multi-file volume is closed completely, the system produces the double tape mark at the end. When, in the process of creating the file, and when physical End-of-Tape is encountered, the operating system writes the double tape mark, rewinds the file, and assigns another tape.

When an unlabeled file is requested for input and no UNIT control statement has been seen, the operator is notified by a ⟨mix⟩ NO FILE ⟨file name⟩ message. The operator must mount the file and enter the ⟨mix⟩ UL ⟨unit designate⟩ message. If a UNIT control statement was specified, the specified unit will be assigned to the file. If a single tape mark is encountered, the object program is notified via an End-of-File condition. To read the file following a single tape mark, the object code must close with no rewind, then open input with no rewind. When the operating system encounters a double tape mark, it will automatically close the file and inform the operator via the NO FILE message to load another reel of the file. (The operating system has no way to determine if a file is a single or multi-volume file.) In response to the NO FILE message, the operator can either mount the file and UL it

in, or enter the message ⟨mix⟩ FR. If the FR (final or only reel) is entered, the object code is given an End-of-File condition.

LABELED TAPE FILES.

The operating system will recognize two labeling conventions for tape input files, the B 5500 label record and the proposed USASI Standard Tape Label for Information Interchange (USASI).

The system will only produce the USASI label format for labeled output tapes. The formats of the various records of the proposed USASI label are shown in tables 4-1 through 4-3.

Table 4-1
Volume Header Label Format

Field	Name	Starting Character	Character Length	Description
1	Type	1	3	Must be VOL.
2	Number	4	1	Must be 1.
3	VSN	5	6	Volume serial number. Used to indicate physical storage location.
4	ACS	11	1	For volume security. Blank means unlimited access.
5	VOLN	12	17	Volume identifier.
6	OP1	29	3	Reserved for operating system.
7	RFE	32	6	Reserved for expansion.
8	Owner	38	14	Identifies file owner.
9	RFE	52	28	Reserved for expansion.
10	LSL	80	1	Always 1.

Table 4-2

First File Header Label Format

Field	Name	Starting Character	Character Length	Description
1	Type	1	3	Must be HDR.
2	Number	4	1	Must be 1.
3	FID	5	17	File name.
4	Set ID	22	6	Last six characters of volume name.
5	FSN	28	4	File section number. 1 for first volume of file, 2 for second file in volume, etc.
6	FSEQ	32	4	File sequence number. 1 for first file in volume, 2 for second file in volume, etc.
7	GEM	36	4	Generation number. 1 for first copy, 2 for second copy, etc.
8	GV	40	2	Generation version.
9	CDATE	42	6	Creation date in form bYYDDD.
10	EDATE	48	6	Expiration date in form bYYDDD.
11	ACS	54	1	For file security. Blank means unlimited access.
12	Block Count	55	6	Always 0.
13	Record Count	61	7	Always 0.
14	System ID	68	6	Always bB6500
15	RFE	74	7	Reserved for expansion.

Table 4-3

Second File Header Label Format

Field	Name	Starting Character	Character Length	Description
1	Type	1	3	Must be HDR.
2		4	1	Must be 2.
3	RF	5	1	Record format: F - fixed size records. D - first four characters of record are the record size in decimal. V - first four characters of record are the record size in binary. U - unknown. L - links. I - internal. F - FORTRAN links.
4	Block size	6	5	Maximum block size.
5	Record size	11	5	Maximum record size.
6	Density	16	1	Density: 0 - 200. 2 - 800. 3 - 1600.
7	VSF	17	1	0 - if first volume of file, else 1.
8	Mode	18	1	0 - alpha (standard), 1 - binary (non-standard).
9	OPI	19	32	Reserved for operating system.
10	OFS	51	2	Size of system control field in front of each record.
11	RFE	53	28	Reserved for expansion.

FIRST END-OF-FILE LABEL.

This is identical to the first file header label except that:

a. Field 1 must be EOF.
b. Field 12 contains the number of blocks of this file on the volume.
c. Field 13 contains the number of records of this file in this volume.

SECOND END-OF-FILE LABEL.
This is identical to second file header label except that field 1 must be EOF.

END OF VOLUME LABEL.
This is identical to the first End-of-File label except field 1 must be EOV.

USER'S HEADER LABELS.
The optional user's header label format is provided in table 4-4.

Table 4-4
User's Header Label Format

Field	Name	Starting Character	Character Length	Description
1	Type	1	3	Must be UHL.
2	Number	4	1	Must be 1 through 9.
3	Use	5	76	User's portion.

USER'S TRAILER LABELS.
This is identical to user's header label except field 1 must be UTL.

The user can specify the creation of single file volumes or multi-file volumes. In addition, the operating system will, for either of the above cases, do volume switching when the data being written exceeds the capacity of a volume. It will also do automatic volume switching on input when required. The tape format is shown as follows (* denotes tape mark):

a. Single file, single volume:

VOL1 HDR1 HDR2 * DATA * EOF1 EOF2

b. Multi-volume file:

```
VOL1  HDR1  HDR2  *  first volume data * EOV1 **
VOL1  HDR1  HDR2  *  last volume data * EOF1  EOF2 **
```

c. Multi-file volume:

```
VOL1  HDR1  HDR2  *  file 1 * EOF1  EOF2  *
HDR1  HDR2  *  file 2 * EOF1 * EOF2 **
```

d. Multi-file, multi-volume:

```
VOL1  HDR1  HDR2  *  file 1 * EOF1  EOF2 *
HDR1  HDR2  *  first part file 2 * EOV1 **
VOL1  HDR1  HDR2  *  part of file 2 * EOV1 **
VOL1  HDR1  HDR2  *  remainder file 2 * EOF1  EOF2 *
HDR1  HDR2  *  File 3 * EOF1  EOF2 **
```

User header labels may appear immediately after HDR2, and users trailer labels may appear after either EOF2 or EOV1.

To create or read multi-file volumes, the user must specify the same volume name for all the files in the set. Only one file in the set can be opened at a time. To create a multi-file volume, the user must CLOSE NO-REWIND, the current file in the set, and use OPEN OUTPUT NO-REWIND for the next file in the set.

To handle input, the operating system will give back to the object code an END-OF-FILE condition when an EOF label is encountered. The user then must CLOSE NO-REWIND on the current file, and OPEN INPUT NO-REWIND on the next (or some other) file in the set.

The EOV label, when encountered on input, is the sentinel by which the operating system can detect when volume switching is required. This is done by locating the next volume or requesting the operator to load a volume which has the same volume name as the current volume, and has a file section number (in HDR1) one greater than the current volume.

As stated previously, a user can specify the creation of a single file volume by specifying only a file identifier, or the creation of multifile volume by specifying both a volume identifier and a file identifer. In addition, the user may specify one or more directory identifiers. This will cause the operating system to keep track of the physical location of the file in the file directory.

It is intended that the volume serial number in the VOL1 label be used as a physical location number. When an empty reel of tape is presented to the system, the operator must indicate that the tape is available for output, and what volume serial number is to be associated with the tape by entering PG ⟨unit designate⟩ ⟨volume serial number⟩. This will cause a scratch label containing the volume serial number to be written on this tape. Later, when file control assigns an output file to the unit containing a scratch label, the volume serial number is read and placed in VOL1 label of the volume being created. If the user has also specified a file name containing one or more directory identifiers, the hierarchical structure for the volume and the volume serial number is entered into the directory. Later, if the file is requested for input, the operator can be notified as to the physical location of the volume containing the file, if it is not already mounted on a tape drive.

Volume serial number 0 (zero) can be used for tapes generated on the system, but which are to be used elsewhere. For this reason, volume serial number 0 cannot be used for tapes which are to be controlled through the directory.

FILE ASSIGNMENT.

In order to assign peripheral I/O devices to symbolic process files, the MCP must know the status of the peripheral devices. Therefore, a pheripheral directory is maintained by the MCP STATUS procedure which keeps track of all I/O devices other than disk. A hardware-software interface causes STATUS to be called periodically to determine whether any peripheral I/O device changes its local/remote state.

When STATUS detects that a device has gone into local, the peripheral directory entry for that device is marked Not Available. When the MCP looks for a unit upon which to assign or locate a file, those which are Not Available are ignored.

Several things occur when STATUS detects a device going from local to remote. First, STATUS marks the unit Available. Then it will try to determine if the unit is to be an input or output device and mark it accordingly. The action taken depends upon the type of unit.

Output only devices (card punch, paper punch) are marked as output.

If the device is a card reader, the first card is read. The MCP CONTROLCARD routine is called, passing the unit designation of the card reader. CONTROLCARD checks to see if the first record contains systems control information (i.e., a control card). If so, the card reader is marked In Use and CONTROLCARD performs the appropriate actions until a LABEL Control statement is encountered. Then, the file name is saved and the card reader is marked Available and labeled as input.

If the device is a magnetic tape, the first record is read. If the record is not a valid label, then the unit is marked as Available, Unlabeled, and Input. If the record read is a label, then the label is checked. In addition, the unit is interrogated for a write ring. If the unit has a write ring and a scratch label, the unit is marked Available, Labeled, and Output. If the unit has a scratch label and no write ring, the operator is notified (a scratch label is created in response to a purge tape message), and the unit marked Not Available. If the unit does not have a write ring and has a valid label, or it has a write ring and a valid label and the expiration date in the label is greater than the current date, the unit is marked as Available, Labeled, and Input. If the unit has a write ring and a valid label, and the expiration date is less than the current date, the unit is marked as Available to be purged.

For each file marked as Labeled and Input, STATUS obtains the file label from the file label records and saves it in a LABEL table. This LABEL table is used at file-open time to associate input file names with the actual hardware device upon which the file is mounted.

The relationship between a file name and a file label can be established by source language statements at compile time, or Label Equation cards at run time. In addition, certain MCP messages can associate a file with a process. To do the association, the compilers generate a Label Equation Block (LEB) and a File Information Block (FIB). The logical association of the file name and file label is made utilizing the FPB.

FILE PARAMETER BLOCK (FPB). The FPB is a 2-dimensional array which is created and maintained by the MCP CONTROLCARD routine. The FPB is an array in the form ⟨file name⟩ = ⟨file label⟩. The compilers create one for each file. In the absence of any label equation statements in the source language, the compiler will enter the file name in both sides in the FPB, thus, if the file name and the file label are identical, further label equation is not needed.

A Label Equation card in the form:

⟨i⟩ FILE ⟨symbolic file name⟩ = ⟨actual file label⟩ ⟨file attributes⟩

is normally used to associate a file label (or actual file name) with a symbolic file name. This card appears immediately following the COMPILE card or the EXECUTE card. When the MCP routine CONTROLCARD routine sees Label Equation cards, it saves the information in the FPB. This information is used later to modify file FPB and FIB entries when the file is first opened.

LABEL EQUATION BLOCK (LEB). The Label Equation Block contains the current label equation and file attributes information for each file in process. Both the LEB and FIB are referred to by a descriptor in the working stack which allows the dynamic specification of file attributes to be implemented in an efficient manner.

FILE INFORMATION BLOCK (FIB). An FIB is created by each compiler for each file in a process. The usage of the FIB is indicated by the following definitions of its contents.

a. The STATUS field reflects the current status of a file, i.e., opened input, closed, rewound, at End-of-File, etc.

b. The R FORMAT contains the programmers description of data in the file, i.e., record size, blocking, etc.

c. The file attributes are:

   1) Output media type.
   2) Optional file indicator.
   3) Standard or non-standard indicator.
   4) Access indicator (serial, random).
   5) Record count and block count (number of records or blocks from beginning of file).
   6) Designated unit and type of unit.

d. The disk attributes are:

   1) Pointer to file header.
   2) Area size.
   3) Number of areas.

e. The printer attributes are:

   1) Line count.
   2) Line limit (page size).
   3) Page count.

FILE OPEN.

The first function performed by the operating system when requested to open a file is to map the file names and file attributes from Label Equation cards into the FIB and FPB. This allows association of a file name with a file label. The file attributes part of the label equation card allows altering the source language description of a file's attributes in a way that such things as file blocking, output file device, etc., can be modified at execute time without

recompiling the source language. For example, a program written to produce its output on the card punch can be label-equated to produce its output on a line printer or a blocked magnetic tape without recompiling.

The second step in opening a file is to assign a device to the file. The action taken depends upon the following types of files.

a. If it is an output file and is not a disk file, locate a device of the correct type marked Available and Output, determine its unit designation, write label records and user label records, and execute user USE routines, if any.

b. If it is an output file and a new disk file, generate a file header in memory and assign actual disk space for the first disk area.

c. If the file is a pre-existent file on disk (input or output), locate the file in the file directory, read its disk header into main memory, and check the attributes specified in the FIB against attributes specified in the file header. If the attributes are incompatible, terminate the process.

d. If it is an input file and not a disk file, locate the file in the label table which also indicates the unit designation, and read the file labels and users labels (executing appropriate users USE routines). Compare the file attributes in the FIB with the file attributes in the file label. If the attributes are not compatible, terminate the program.

The last steps in assigning a file to a program are:

a. Allocate memory space for the buffers.
b. Construct I/O control words and IOAREA control areas.
c. For all input files and blocked output files on pre-existant disk files, preload the buffers with data.

OBJECT PROGRAM I/O.

Object program input/output operations on the B 6500 System involves the automatic transfer of records between a file and a process. The concepts of concern to object program I/O are record size, block size, blocking, serial I/O, and random I/O. Each is described below.

RECORD SIZE.

The record size is the size of that set of data processed by each I/O statement in the source language.

BLOCK SIZE.

The block size is the size of a set of data that can be processed by the hardware on each actual hardware I/O operation. The limiting factor in size of a block is dependent on each hardware device. For example, card readers are fixed at 80 characters per block, tape is variable in increments of 1 to 16,767 words, and disk block size is variable in increments of 30 word segments.

BLOCKING.

Blocking involves the capability of specifying one or more records per block. That is, one hardware I/O operation will transfer one or more logical records. The purpose of blocking data is to conserve storage space and to increase processing speed.

BUFFERING.

Buffering involves the use of one or more buffers (memory areas) which provides the interface between the hardware device and the source language I/O statements. There must always be at least one memory area to be used as a buffer for each file. The hardware must process one block of data on each I/O operation. Therefore, the memory area should be at least as large as one block. The records must then be supplied one at a time from the block to the program. The use of more than one buffer can also be used to increase the processing speed of data since multiple buffers allow I/O to be performed on one buffer at the same time that a logical record is being processed in another buffer.

SERIAL I/O.

Serial I/O operations require the system to supply the next record in sequence to the program on each I/O operation. All I/O devices on the system are capable of serial I/O operations.

RANDOM I/O.

Random I/O operations are allowed only on disk files. This concept involves supplying a specified record to the program. All records on disk are stored sequentially and are addressed 0 through n, where 0 is the first record of the file and n is the last record in the file.

MCP.

Associated with each file is a record pointer (see figure 4-8). All data is accessed by a program through the record pointer. This pointer contains a base and a maximum record size. The various languages can specify records of variable size and a maximum record size. Also, some languages depend upon the program to establish the record size. Certain hardware checks will cause program termination if the program establishes a record size exceeding its specified maximum record size. This establishes one level of system integrity, i.e., a program cannot alter or destroy data outside of the program data area limit.

Each I/O statement makes a record available to a program altering the base field of the record pointer. In the case of blocked records and another record in the block exists, then the next record can be obtained by incrementing the pointer base by the size of the previous record. In the case where all logical records in a block have been processed, the base can be set to either the next buffer, if multiple buffers are specified, or the front of the buffer if only one buffer is specified.

Any time the record pointer is set (rather than indexed), the address of the I/O control area IOAREA is passed to the MCP which activates an actual I/O operation on that buffer. At the same time, the event associated with the buffer to which the record pointer has been set is checked.

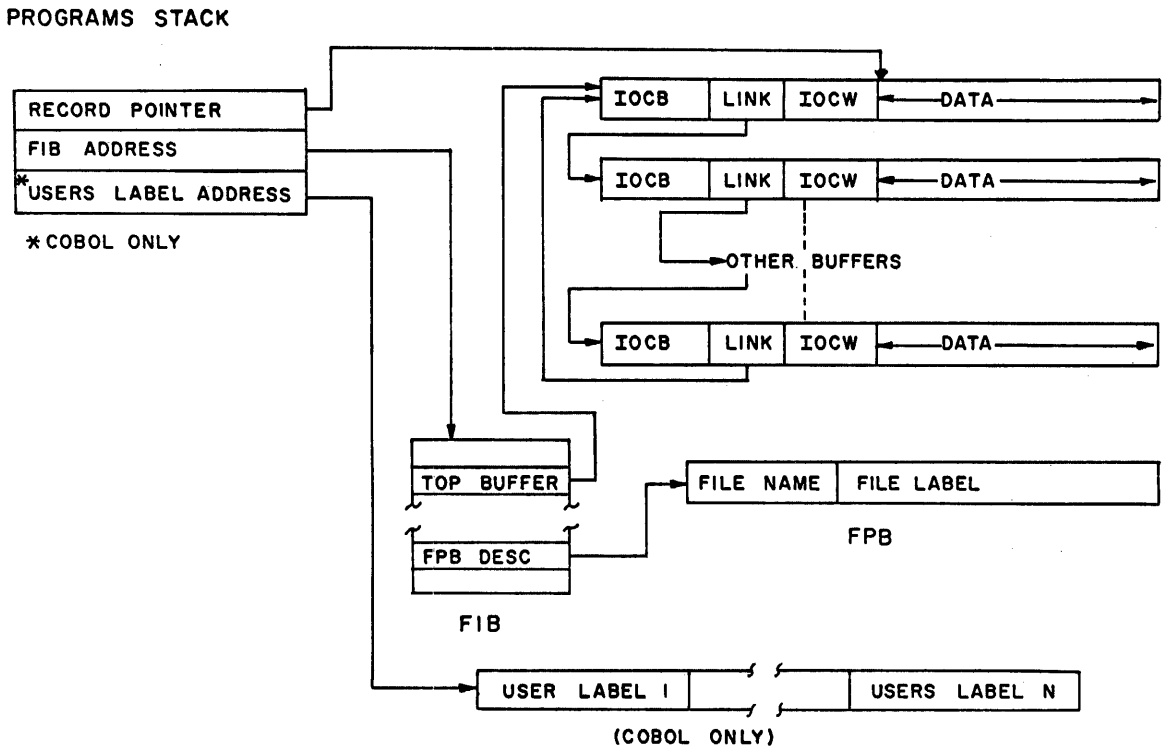


Figure 4-8. Normal State I/O

The buffer event serves to interlock the buffer with the program such that the program cannot reference a buffer which has an I/O in progress on it. When a buffer is passed to the MCP, its event is set to the state Not Happened. Upon completion of the I/O for that buffer, the MCP routine IOCOMPLETE sets the event to a state Happened. Prior to returning to the program after setting the record pointer to a buffer, its event is checked for the Happened state. If the event has not Happened, then a WAIT event is executed. This will cause the program to be moved from the READY Queue to the WAIT Queue, i.e., the program is suspended and another program in the READY Queue is started. Later, when IOFINISH causes the event to Happen, all of the processes waiting on that event are moved from the WAIT Queue to the READY Queue. The processes will be re-activated according to their priorities in the READY queue.

RANDOM RECORD ACCESS.

Since actual I/O operations may involve blocks of records when a read or update is required on a record, the entire block containing

the record must be read. If the I/O action is a random action which may be specified for files such as disk files, the record required may have been included in a block of records which was previously accessed. Therefore, in order to eliminate unnecessary I/O actions, the MCP remembers which logical records are currently held in each buffer. When a request is made for a particular record, the buffers are first checked to determine whether the record already exists in the buffer. If it is, then the record pointer is set to point at it and control returns to the object program immediately. If the record is not already in the buffer, then the MCP must be called to load the block containing the record, the program must be suspended until the record is loaded, and then the record pointer is set to point at it.

SEEK.

The SEEK function can be activated by the source language. Associated with the SEEK is a record address. SEEK has two purposes depending on whether or not the file is serial or random. SEEK on sequential files serves to reset the file to start sequential processing at the record indicated in the SEEK statement. For example, the first I/O statement after a SEEK RECORD n would make record n available to the program. The record being processed prior to the SEEK is not disturbed by the SEEK. That is, it is still available.

The SEEK statement on random access is intended to cause the system to prelocate the next record while the program is processing the current record. The SEEK first examines the buffers to see if the record already exists in a buffer. If the record is in a buffer, control is returned to the program.

If the record is not in a buffer and there is only one buffer, then control is returned to the program. It should be noted that the use of SEEK on files with only one buffer causes unnecessary MCP overhead and should be avoided. If there are multiple buffers, then the MCP may be called to load the block containing that record. Assuming the record pointer is pointing at buffer number 1,

then consecutive READ SEEKs are alternated through buffers 2 through n.

MCP I/O.

As previously stated, I/O control requests actual I/O by passing the address of an I/O area to the MCP procedure IOREQUEST. The primary purpose of IOREQUEST is to quickly set up an I/O request and return to the calling program. To set up an I/O request, several things must be considered.

First, since IOREQUEST is handling I/O operations on all buffers of all programs in the mix, each I/O must be associated with a particular buffer of a particular program.

A second consideration is that IOREQUEST must set up an I/O operation and return to the caller, even if the I/O request is on a device that cannot be initiated. The device may already be in use by a prior request, or all multiplexor channels may be busy performing I/O operations on other devices. This also implies the setup must include the capability of later sending the request to the multiplexor when the device does become available.

Finally, the setup must also include the ability to interlock the I/O buffer and later, when the I/O operation is complete, unlock the buffer. This interlocking must be transparent to the programmer. In addition, it must allow the program to run and be stopped only when the program attempts to process data in a buffer for which an I/O request has been made, but is not yet completed. The MCP utilizes the two queues in the handling of an I/O request, as shown in figure 4-9.

Each device in the system (each reader, tape, disk electronics unit, etc.) has a unique unit number and a unique I/O Queue.

The IOREQUEST functions are as follows:

a. The I/O area IOAREA is linked into the I/O Queue. If there is more than one entry in the queue, IOREQUEST

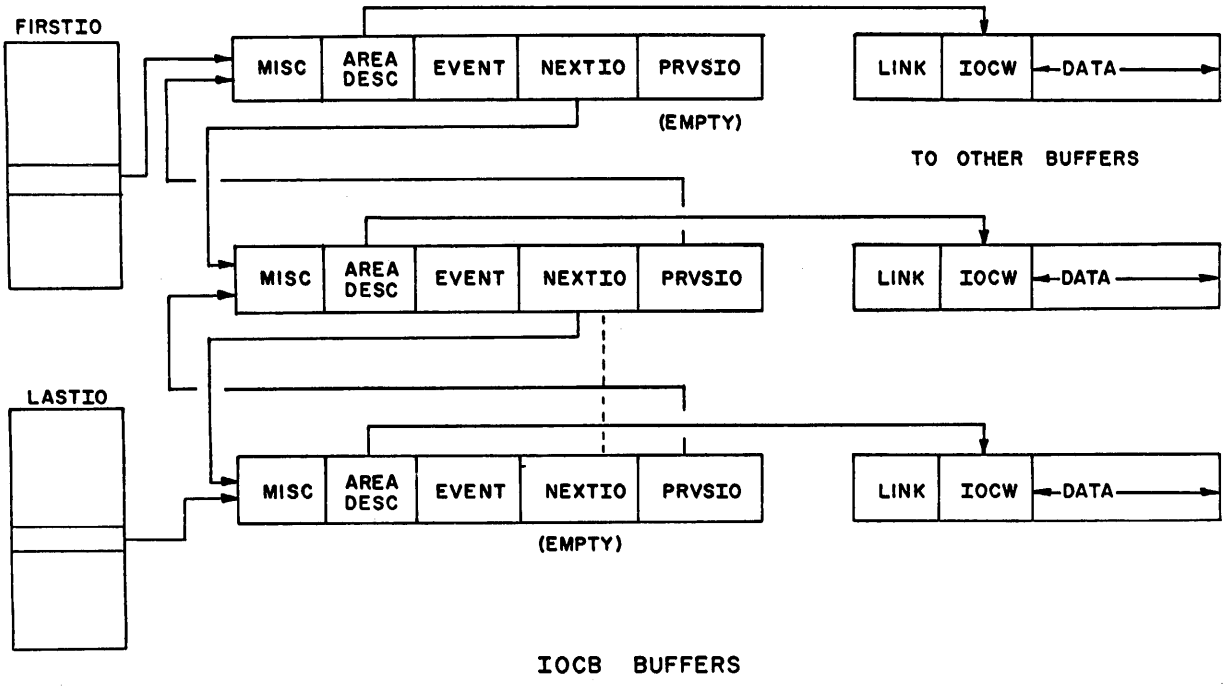


Figure 4-9. MCP I/O Queue

returns to the requesting process. Otherwise, IOREQUEST calls STARTIO. STARTIO makes up the UNITWORD which specifies the unit and multiplexor, and the hardware instruction which interrogates for an I/O path is executed. If a path (I/O channel) is available, then STARTIO calls INITIATEIO which causes the multiplexor to start transferring the information. INITIATEIO also records the initiate time which the IOFINISH routine uses to calculate I/O time for the process. Control is then returned to the process requesting I/O action.

b. If a path is not available, then the UNITWORD is entered into the WAITCHANNEL Queue as shown in figure 4-10. Control is then returned to the process requesting process.

The multiplexor, after obtaining an I/O request via a processor Initiate I/O instruction, proceeds to handle the request completely independent of the processor. In the process of doing the I/O, the multiplexor builds a result descriptor. Upon completion of the I/O operation, it generates an IOFINISH interrupt to the processor. The MCP routine IOFINISH is activated upon this interrupt.

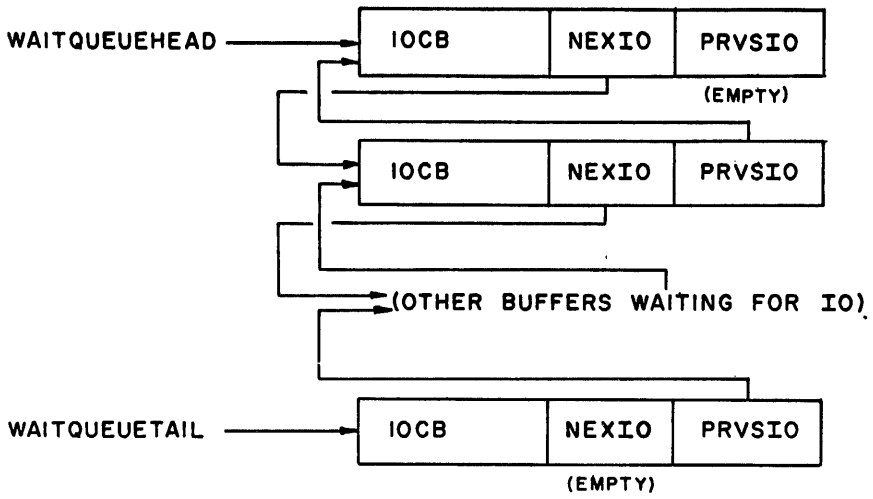


Figure 4-10. MCP Wait Queue

The first operation of IOFINISH is to execute the instruction Read Result Descriptor for the multiplexor. This instruction transfers the result descriptor from the multiplexor to the top of the stack in the processor. At the same time, it clears the interrupt mechanism in the multiplexor so that it becomes capable of generating another IOFINISH for some other device.

The result descriptor has three fields of concern: the unit number, error bit, and error field. The error bit is off if no errors were detected. If the bit is on, then the result descriptor is passed to the IOERROR. IOERROR analyzes the error field which denotes such errors as End-of-Page, End-of-File, Parity, Not Ready, etc. Depending on the type of error, IOERROR will take appropriate action.

Assuming IOERROR corrected the error, or there was no error, IOFINISH continues as follows:

a. The I/O just completed is removed from the I/O Queue. Since the result descriptor has the unit number in it, FIRSTIO ⟨unitnum⟩ points at the I/O to be removed from the IOQUEUE.

b. If the WAITCHANNEL Queue is not empty, NEWIO is called to initiate an I/O operation on the first unit waiting in the queue. This I/O Queue is then checked to see if

it is empty. If it is not empty, then the next I/O operation requested is placed in the WAITCHANNEL Queue.

c. If the WAITCHANNEL Queue is empty, the STARTIO is called to initiate the next I/O operation in the I/O Queue for this unit.

d. The user I/O time is recorded in the system log.

e. The I/O finish event is CAUSEd which causes the process in the events WAIT Queue to be moved into the READY Queue.

Since IOFINISH was activated by an interrupt rather than being called, the exit from IOFINISH is done by branching to a routine which will activate the process or program which is in the top of the READY Queue.

UTILITY OPERATIONS.

LOAD CONTROL.

The MCP provides a means whereby card deck information, including system control information, can be placed on the disk in the form of a "pseudo card file," and then used as though it were in a card reader. The three major parts of the load control system are the LDCNTRL/DISK Program, pseudo card readers, and pseudo card decks. The LDCNTRL/DISK Program can place either a magnetic tape file or a card file on the disk, and copy a file onto magnetic tape.

LOADING A CONTROL DECK FILE ONTO DISK. The primary function of the program LDCNTRL/DISK is to read a file with the multiple file identification CONTROL, and the file identification DECK, and to place that file on disk or a magnetic tape.

The normal mode of operation for LDCNTRL/DISK is to locate a card or tape file labeled CONTROL/DECK and place that file on disk as one or more pseudo card decks.

CARD READER CONTROL DECK FILE. If a control deck file is to be read from a card reader, the file must be preceded by a label card

to identify it. Also, the last card in the control deck must be an END CONTROL card containing the following:

⟨invalid character⟩ END CONTROL

MAGNETIC TAPE CONTROL DECK FILE. If a control deck file is to be copied from magnetic tape onto disk, the tape must be properly labeled and, as is the case with a control deck from a card reader, the last card image on the tape file must be an END CONTROL card. In addition to these requirements, the tape file must be properly formatted so that system control cards (i.e., system control statement and program-parameter statement cards) can be recognized. Specifically, the tape must have the following characteristics:

a. Each record containing a question mark card must be nine words in length.

b. Each record containing a card which is not a question mark card must be 10 words in length.

PSEUDO DECK ON DISK.
When the LDCNTRL/DISK program reads a control deck file, it places it on disk as one or more pseudo card decks. The number of pseudo decks created depends upon the number of END cards located within the control deck. That is, each time an END card is encountered, it is taken to denote the end of a deck. Creation of another pseudo deck is then initiated and, as each new pseudo deck is created, it is given an identification of the form #⟨integer⟩.

It should be noted that what is referred to as a pseudo deck is analogous to a single continuous deck that would be placed in a card reader. Therefore, if a pseudo deck contains more than one file, each file following the first will be recognized only when the file preceding it has been passed. Also note that there is no set limit to the number of cards that may be contained in a control deck file.

Due to each pseudo card deck that is placed on disk, the deck is

linked to the previous deck, forming a queue waiting to be used by a pseudo card reader. Because of the queue feature, the RD keyboard input message must be used to remove pseudo decks from the disk.

The secondary function of the LDCNTRL/DISK system is to read a file labeled control deck, delimited by an END CONTROL card, and to copy it onto magnetic tape. If the control deck being copied is a card file, the file will be copied onto tape in the required format specified above. If the control deck being copied is a magnetic tape file, a tape copy is performed.

The LDCNTRL/DISK may be called out either by a keyboard input message or control cards.

If LDCNTRL/DISK is used to place a control deck on the disk, either the keyboard input message LDDK or a control card containing ⟨invalid character⟩ EXECUTE LDCNTRL/DISK may be used.

If LDCNTRL/DISK is used to copy a control deck onto tape, either the keyboard input message LDMT or a control card containing ⟨invalid character⟩ EXECUTE LDCNTRL/DISK; COMMON = 1 may be used.

ERROR CHECK IN LDCNTRL/DISK.
If a parity error is encountered in a control deck file being read from magnetic tape, the remainder of that file is skipped and the file containing the parity is completely ignored.

PSEUDO CARD READERS.
To make use of pseudo card decks, the MCP contains logic which can, in effect, supply the system with up to 32 pseudo card readers. These pseudo card readers, in many ways, appear to be much like physical peripheral units. That is, system messages are typed for the pseudo card readers as though they were actual card readers, and keyboard input messages can reference the pseudo card readers. The pseudo card readers are identified by specifying CD ⟨unit number⟩ as shown below:

CD1
CD2
CD3
...
...
...
CD32

All pseudo card readers are turned off at Halt-Load time. The system operator may cause these readers to be turned on through the use of an RN keyboard input message. When an RN ⟨digit⟩ message is initially entered and the digit is not equal to zero, the MCP automatically searches for pseudo card decks to satisfy the need of the specified number of pseudo card readers. Thereafter, as long as pseudo card readers are on and pseudo card decks are available, the MCP will keep the readers full.

If the system operator wishes to turn off pseudo card readers, he need only type in an RN message that specifies the number of pseudo card readers he wants left on. The MCP will then turn off a sufficient number of readers to meet these requirements as soon as the readers complete processing their current decks.

If, for any reason, it is desirable to remove a deck from a pseudo card reader (e.g., a card file never opened by a program that was discontinued), the removal can be accomplished by entering an ED keyboard input message.

ERROR HANDLING IN THE PSEUDO CARD DECK.
If a pseudo card deck is being read and an error is detected in a control card or Program Parameter card, the MCP will remove the deck in which the erroneous card appears and will continue to the next available pseudo deck.

PRINT BACKUP.
Due to the relatively high cost of printers, some means is desirable to ensure that the printer is kept working at its maximum rated

performance. This goal may be accomplished by simulating printers with disk files or magnetic tape units. On the B 6500 System, information which is intended to be written on a printer may be routed to a disk file or tape called a printer backup (PB) file. When the PB file is closed and a printer is free, the PB file may be printed at the maximum rated speed for the printer.

There is one control record per print file containing the file identification, the name of the program creating the print file, a copy of the first nine words of the header card, a special forms flag, and recycle option flag. This record is the first record of the print file and is the first record of the block on the PB file. A control record is always the first record of a block. The remainder of the block is composed of data records. Each data record contains the output record created by the program followed by a control word. The control word contains the carriage control information developed from the original I/O descriptor.

A printer backup file on tape has the name PBTMCP/xxxx BACK-UP. It may contain more than one printer file and may span more than one reel. Additional disk areas are allocated as required. The name of a print backup disk (PBD) file is PBD/xxxxnnnrrr, where xxxx are the first four letters of the program creating the file, nnn is a serial number (in EBCDIC) corresponding to the print file, and rrr is the serial number of the backup file within the print file. Thus, each print file may be composed of more than one physical backup file on disk, all with the same nnn part.

FILE OPENING ACTION. If the file control card has recycle options, the auto-print option will bypass the print file because the user wishes to make multi-copies of his output, but doesn't want to create a backup tape. The number of copies required by the user may be generated in the following manner. Assume that five copies of output are required. The operator types in PBxxxx5, where xxxx is the identification number of the file and integer 5 is the number of times the file is to be repeated before the print file is removed. The printer backup routine will use the maxumum number

of printers available to get the five copies he requires.

SKIP OPTION.
The PB system also has a skip option feature. The skip options are specified as follows:

a. PBxxxx SKIP n = ⟨key⟩.
b. PBxxxx SKIP n.

The first option skips the first part of the document and only starts printing when it finds the key starting in print column n.

The second option skips n lines of output.

When a printer backup file is opened, a bit is set which notifies the MCP that this file is in use. Subsequent to Halt-Load time, the MCP will attempt to reopen the file, but will not be able to because the In Use bit is on. At this point, the MCP will communicate to the display that it needs a new PB command to continue the job. It may be desirable to use a skip option feature.

The selection of a physical unit for a printer backup file is determined in the following manner. If the file may go to tape, an existing printer backup tape (PBT) file is used, if one is available. Otherwise, if a scratch tape is available, a new PBT is created and used.

If a unit is not found for the file, a message is displayed to inform the operator. If a unit of the specified type is made available, it is used. Otherwise, the operator may reply with an OU message to assign a different type of output unit such as a PB disk file, a printer, or a card punch.

SPECIAL FORMS. If the special forms feature is desired on a print file opened as a printer backup file, any special forms requirement is deferred until the backup file is printed. If the print file is opened on a printer, the operator is informed that special forms are required by the message # ⟨unit⟩ FM RQD----, or a special program generated message. The operator may then:

a. Load the forms onto that unit and key in ⟨mix⟩ FM ⟨unit⟩.

b. Key in ⟨mix⟩ OUMT or ⟨mix⟩ OUDK to force the chosen printer to be released to open a backup file. When a backup is printed which required special forms, the message #FM RQD ⟨unit⟩ FOR ⟨mfid⟩/⟨fid⟩ OF ⟨program name⟩ will be displayed to which the operator may reply with an OK, WY, or DS message.

CLOSING A PRINT FILE ON DISK. If the system option auto-print is set when a print file on disk is closed, it is scheduled to be printed unless it is a recycle option file. If auto-print is not set, a message is typed to inform the operator that a PBD exists and may be printed by the message PBD xxxx. When an output file is printed from a PB file, an entry is made into the log containing the header card information of the program.

LIBRARY MAINTENANCE.
The MCP provides a library maintenance process, LIBMAIN/DISK, to perform disk library utility operations. The LIBMAIN/DISK process is initiated either from the supervisory display unit or from a system control card. Some options available are REMOVE, DUMP, LOAD, UNLOAD, CHANGE, EXCHANGE, PRINT, and DISPLAY.

REMOVE OPTION. This option causes the specified file(s) to be removed from the disk directory and causes the disk space used by the file(s) to be made available for other use. The MCP files, SYSTEM/LOG and DIRECTORY/DISK, may not be removed by this option.

To specify what files are to be removed, file lists and/or multiple file names may be used. A multiple file name specifies one or more files that are to be removed.

DUMP OPTION. This option causes a copy of the specified disk files to be copied onto a magnetic tape. The file information written on the magnetic tape forms a multi-file library tape. The dump option facility does not remove the file from the disk

directory. The MCP files such as SYSTEM/LOG and DIRECTORY/DISK cannot be copied.

To specify what files are to be copied, file lists and/or multiple file names may be used. A multiple file name specifies one or more files are to be copied.

LOAD OPTION. This option loads the specified files from library tape to disk and makes the appropriate changes in the disk directory. At the completion of the load operation, the tape is rewound and locked.

UNLOAD OPTION. This option dumps the specified files onto a library tape. At the successful completion of the dump, the disk directory entries are modified to show that the files are no longer on disk and the disk space is returned to the available list.

CHANGE OPTION. This option is used to change the names of files which are on disk.

EXCHANGE OPTION. This option is used to exchange two file names of program files and/or data files which are on disk.

PRINT OPTION. The use of this option causes the MCP to write various information about a given file onto a line printer. Such information as hierarchical order, media, volume number, etc., is written for each file specified.

DISPLAY OPTION. The use of this option causes the MCP to display the specified file names and related directory information on a display unit.

SYSTEM LOG.

The MCP provides a log maintenance function which maintains a log or history of user program activity and time usage. The log, as compiled from the LOG SHEET Queue, is saved on the disk by the MCP.

The log information for a process run on a B 6500 System is written in a file on user disk. The log file occupies one area on disk,

and has the file name SYSTEM/LOG.

The SYSTEM/LOG file consists of 10-word logical records blocked three to a physical record. Word 0 is used to identify the message which appears in words 1-9. For the last record in the file, bit 47 of word 0 is set to 1 and the rest of the record is left empty. The system operator is kept informed of the size of the SYSTEM/LOG by the LOG ⟨percentage⟩ % FULL message which is printed in 5 per cent increments. The operator can create an empty SYSTEM/LOG with the LN message. When LN is typed, the name of the current SYSTEM/LOG is changed to ⟨number⟩ LOG, and a new file named SYSTEM/LOG is created. All ensuing display messages will be stored in the new SYSTEM/LOG and the ⟨number⟩ LOG file can be processed later.

⟨number⟩/LOG. ⟨number⟩ has 7 digits to identify the log information, where digits 0-1 contain the month, digits 2-3 contain the date, and digits 4-6 are the serial number of log to link each log file together. The first file is number 001, the second 002, etc. If SYSTEM/LOG becomes 95 per cent full, an LN is automatically initiated with the following message: LOG 95% FULL (AUTO LN). Table 4-5 provides a listing of the various system log message type codes.

The formats of the various messages are listed below:

The BOJ message:

⟨program specifier⟩/⟨user code⟩ = ⟨mix index⟩ BOJ ⟨time of day⟩

The EOJ message:

⟨program specifier⟩/⟨user code⟩ = ⟨mix index⟩, PST = ⟨time⟩ EOJ

where the time is processor time. DS-ED, ES-ED, etc., may appear instead of EOJ.

PBEOJ message:

PRNPBT FOR ⟨program specifier⟩, PST = ⟨time⟩, IOT = ⟨time⟩; EOJ

where PST is the processor time and IOT is the I/O time.

Table 4-5

System Log Message Type Codes

Code	Message Type
0	A message not applicable to logging.
1	A message typed in from the display.
2	BOJ message.
3	EOJ message.
4	PBEOJ message (printer backup End-of-Job).
5	File open message.
6	File close message.
7	Halt.
8	EOJ statistics.
9	File close statistics.
10	On message, indicates a successful log-in.
11	Off message.
12	Charge message.
13	Disk charge message.
14	Date message.
15	Time message.
16	Operator RSVP message.
17	ST-ED message.
18	Reserved for expansion.
19	Reserved for expansion.
20	Reserved for expansion.

File close message:

⟨unit mnemonic⟩ REL ⟨data file designator⟩ ⟨rdc⟩ = ⟨job specifier⟩

System initialization message:

B 6500 MCP LEVEL ⟨change level⟩

EOJ statistics have three words:

a. Word 1 contains the processor time in 60ths of a second.

b. Word 2 contains the I/O time in 60ths of a second.

c. Word 3 contains the number of words of core used.

File close statistics have five words:

a. Word 1 contains the multiple file identification.
b. Word 2 contains the file identification.
c. Word 3 contains the reel number and date.
d. Word 4 contains the cycle, number of errors, and the unit.
e. Word 5 contains the length of time the file was opened.

HALT message:

HALT ⟨date⟩ AT ⟨time⟩

DATE message:

DATE IS ⟨day of week⟩, ⟨month⟩ / ⟨day⟩ / ⟨year⟩

TIME message:

TIME IS ⟨time of day⟩

Operator RSVP message:

FOR ⟨user code⟩ ON/OK ⟨date⟩ AT ⟨time⟩

ST-ED message:

FOR ⟨user code⟩ ST/OK ⟨date⟩ AT ⟨time⟩.

SYSTEM RECONFIGURATION.

In the event that a hardware module fails or must be shut down for maintenance, the system must be reconfigured to eliminate the module which is no longer available to the system. The ability to reconfigure the system easily and efficiently is designed into the B 6500 so that the loss of a particular key module will not be catastrophic to the system unless that module is unique. For example, the failure of one processor on a two processor system would cause a degradation of system performance, but the system would still be operable. In the B 6500 system, there are very few system modifications which are not handled automatically by the MCP, and there are even fewer which cannot be handled with the aid of a Halt-Load.

The basic criterion for being able to shut down or disconnect a unit is whether it is currently in use by some process. If, for example, a memory module is shut down, it is clear that the information which is currently stored in that module is now inaccessible. Such a situation would almost certainly lead to an invalid address. However, if a unit which is not currently in use, such as a magnetic tape drive, is shut down, the system will continue to function as if nothing has happened. It is possible to issue a command to the MCP indicating that a particular unit is to be shut down, and that the MCP will respond by rearranging the system to avoid the use of the unit, at which point it will indicate that the unit has been detached from the system.

There are, however, certain major hardware modifications to the system which will require modification of the MCP. This is due to the fact that handling of DCP (Data Communication Processor) interrupts and the GCA (general control adapter) interrupts is contingent on the nature of the device involved. If, for instance, a data communication system or a substantially different data communication system is to be added, it will be necessary to alter the MCP. It also means that before connecting a device such as a plotter or analog interpreter, it will be necessary to specify the nature of the GCA interrupt. In other words, it will be necessary to specify how the MCP is to handle GCA interrupts by changing the MCP. Except for these two contingencies, however, reconfiguration of the system will not require modification of the MCP.

SECTION 5
DATA COMMUNICATIONS

GENERAL.

The Data Communications Processor (DCP) is a peripheral control device intended to cope with the routine problems and functions of a multiterminal data communications network.

In the B 6500, no terminal device directly interfaces with the central system. Instead, the sequence is: terminal device (teletype, for instance), communications line, line adapter.

This is followed by the adapter clusters, the DCP, then to multiplexor, and into the main memory.

Each adapter cluster can service 16 lines. Each DCP can service 16 adapter clusters. Each multiplexor can service four DCP's. Thus a two multiplexor B 6500 can service 2048 lines. More lines may be handled by chaining multiplexors. By chaining multiplexors, the maximum number of lines is limited only by the main memory access time.

The DCP contains a resident local instruction memory and logic for over 30 different hardware instructions. The resident program executes entirely in the DCP logic and does not require attention from the MCP for its operation. It controls the clusters and functions to modify and concatenate the adapter output into a form suitable for transmission to the central processors.

The adapter cluster controls all transmissions to and from the terminals connected to its lines. In effect, it determines what information is to go on each line, and when it is to be transmitted or received.

DATA COMMUNICATIONS SOFTWARE SYSTEM.

The B 6500 data communication software system is shown in the figure 5-1. Block number 1 represents the data communications processor (DCP) line control procedures. A DCP Program Generator

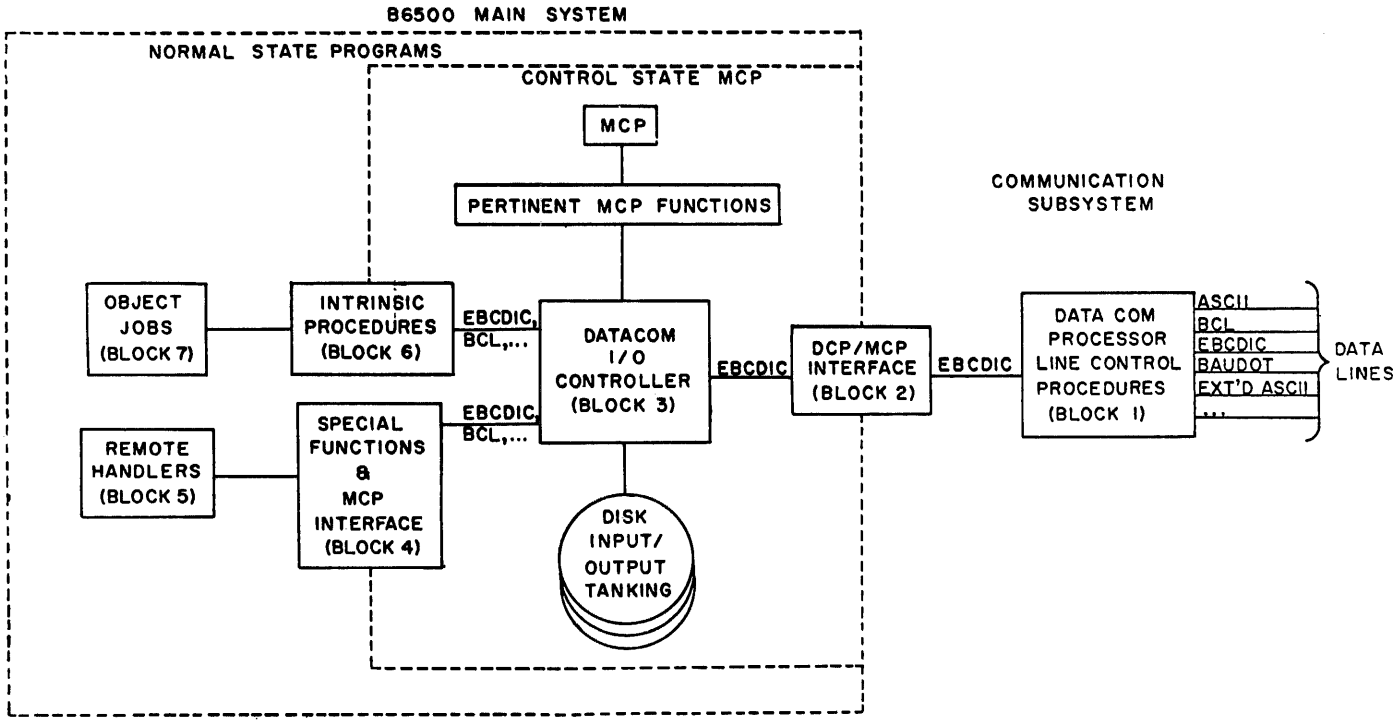


Figure 5-1. Data Communications Software System

will be provided for the purpose of producing a DCP program. This program is developed by combining selected elements from a modular set of sub-units that are provided by Burroughs.

It is the user's responsibility to convey to the Program Generator the types of devices to be attached to his installation's DCP. The user must also provide the form of the messages that are to be transmitted. By making use of the above information, the DCP program can be generated.

Depending upon the particular application and kind of devices on the line, the DCP program may send acknowledgements and negative acknowledgements (ACK's and NAK's) for messages successfully or unsuccessfully received. They also perform code translation and handle general line maintenance. Other functions may include

horizontal and vertical parity checking, and making use of redundancy for the purpose of reconstructing lost information.

Block number 2 represents the interface function. This function is partially developed from the DCP Program Generator sub-units and partially coded in the Executive System Programming Oriented Language (ESPOL). ESPOL is a high-level language which is used to program the B 6500 MCP. The two major functions handled by the interface routines are the maintenance of the DCP INITIATE Queues and the DCP COMPLETE Queue.

The DCP INITIATE Queues are a communication link between the MCP and the DCP program. When an object program requests an I/O operation to be performed, the MCP queues the request according to unit number or DCP number. Thus, there is an INITIATE Queue for each DCP.

Each time the MCP adds an element to an INITIATE Queue, it checks to see if the queue was previously empty. If it was empty, the MCP performs a Scan Out command indicating that DCP attention is required. The DCP recognizes that the queue is now active and starts emptying the queue as rapidly as possible. Emptying the queue simply means pulling each queue entry off the INITIATE Queue, and placing it on the end of a queue that exists for each line number. Each time a line becomes free, the DCP initiates an I/O operation for the top item in the queue for that line.

A new entry is made in the DCP COMPLETE Queue each time a message has been assembled by the DCP. If the COMPLETE Queue was empty when the current complete was placed in the queue, a signal is given notifying the MCP that the queue is now active.

Block number 3 depicts the Data Communication Controller which is an ESPOL Program that is the central switching unit for the entire DCP software subsystem. Some of the functions that may be performed are the linking of messages to the proper program, the combination of a set of buffers into one interlaced buffer, and the tanking of messages on disk.

Block number 4 depicts special functions which are built into the MCP for the benefit of remote handlers. These special functions are coded in ESPOL. The two classes of special functions are supplying of information and execution of unusual tasks. The number of jobs in the mix, the number of multiplexors on the system, the number of data communications processors, and the size of core memory are other types of information that may be supplied.

Block number 5 depicts the functions performed by remote handlers. Remote handlers may perform scheduling, error detection, and reconstruction of lost information through the use of redundancy. The user can code the remote handlers to suit the requirements of his installation.

Block number 6 depicts the intrinsic procedures which are coded in ESPOL. Intrinsic procedures provide the link between user's object programs and the Data Communications I/O Controller procedures of the MCP. Normal READ and WRITE statements in the user's object jobs generate code which calls the intrinsic procedures.

Block number 7 depicts the user's object jobs. The object jobs are coded by the user in any one of the languages provided by Burroughs.

APPENDIX A
DEFINITIONS

Several terms which are used in the discussion of the MCP are defined below. Semantics following syntax are enclosed in parentheses and begin on a new line. Where semantics are used to describe syntax, they are enclosed with the character pair {}. Logical OR is represented by a vertical line (|).

⟨letter⟩ ::= A | B | C | D | E | F | G H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨integer⟩ ::= ⟨digit⟩ | ⟨integer⟩ ⟨digit⟩

⟨space⟩ ::= {one or more blank card columns or equivalent}

⟨special character⟩ ::= [| . | $ | @ |] | , | - | (| : | + | & |) | < | " | % | = | > | * | /

⟨character⟩ ::= ⟨letter⟩ | ⟨digit⟩ | ⟨special character⟩

⟨reserved character⟩ ::= - | . | = | , | ;

⟨invalid character⟩ ::= {any card code which does not represent a character}

⟨identifier⟩ ::= ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩ (Only 17 characters of an identifier will be used by the MCP.)

⟨reserved word⟩ ::= ALGOL | ALPHA | BACKUP | CHANGE | COBOL | COMMON | COMPILE | CORE | DATA | DISK | DISPLAY | DUMP | END | ESPOL | EXECUTE | EXTERNAL | FILE | FORM | FORTRAN | FREE | IO | LIBRARY | LOAD | MINIMUM | PAPER | PRINT | PRIORITY | PROCESS | PROTECT | PUBLIC | PUNCH | RANDOM | READER | RELEASE | REMOTE | REMOVE | RUN | SAVE | SERIAL | SPECIAL | STACK | SYNTAX | TAPE | TAPE7 | TAPE9 | UNIT | UNLABELED | UPDATE | USE | USER

⟨comment⟩ ::= {any series of characters excluding reserved characters, reserved words, and spaces}

⟨optional compiler name⟩ ::= ⟨empty⟩ | ⟨compiler name⟩
(This is used to designate whether a control statement refers to the compiler or the object program. If empty,

it refers to the object program.)

⟨compiler name⟩ ::= ALGOL | COBOL | ESPOL | FORTRAN

⟨program name⟩ ::= ⟨identifier⟩ | ⟨program name⟩ | ⟨identifier⟩

(If a program is entered into the library, the program name becomes the file label for the file in which the program is stores.)

⟨job name⟩ ::= ⟨program name⟩

(Normally a program name is used as the job name.)

⟨mix index⟩ ::= ⟨integer⟩ | ⟨mix index⟩.⟨digit⟩

(The integer contains the hour of the day and serial number. The .digit indicates that this is a process initiated by the job represented by the integer.)

⟨file label⟩ ::= ⟨file id⟩ | ⟨vol id⟩ | ⟨file id⟩ |
⟨directory id⟩|⟨vol id⟩|⟨file id⟩

⟨file name⟩ ::= ⟨identifier⟩

(This is an identifier by which a program refers to a file. The file name is commonly label equated to a file label or another file name.)

⟨directory id⟩ ::= ⟨identifier⟩ | ⟨directory id⟩|⟨identifier⟩

(Used to form the hierarchical levels of the disk directory.)

⟨vol id⟩ ::= ⟨identifier⟩

(The vol id designates a particular volume. For a tape file, it refers to a tape reel.)

⟨file id⟩ ::= ⟨identifier⟩

(The file id designates a particular file in a volume.)

⟨priority⟩ ::= ⟨integer⟩

(The slowest priority is mm and the fastest is 0.)

⟨unit mnemonic⟩ ::= ⟨letter⟩ ⟨letter⟩ ⟨integer⟩

The unit mnemonics recognize by the MCP are listed below:

MT⟨integer⟩ ::= magnetic tape unit ⟨integer⟩
CR⟨integer⟩ ::= card reader ⟨integer⟩
LP⟨integer⟩ ::= line printer ⟨integer⟩
CP⟨integer⟩ ::= card punch ⟨integer⟩
SC⟨integer⟩ ::= supervisory console ⟨integer⟩

CD⟨integer⟩ ::= pseudo card reader ⟨integer⟩

PP⟨integer⟩ ::= paper tape punch ⟨integer⟩

PR⟨integer⟩ ::= paper tape reader ⟨integer⟩

⟨file list⟩ ::= ⟨file label⟩ | ⟨file list⟩,⟨file label⟩

⟨file set list⟩ ::= ⟨file set specifier⟩ |
⟨file set list⟩,⟨file set specifier⟩

⟨file set specifier⟩ ::= ⟨file label⟩ | ⟨file label⟩ | =
(The character = represents any identifier, hence A/= specifies the set of all file labels whose first level is A.)

⟨rdc⟩ ::= ⟨empty⟩ | ⟨reel⟩ | ⟨reel⟩,⟨date⟩ | ⟨reel⟩,⟨date⟩,⟨cycle⟩

⟨reel⟩ ::= {an integer of up to four digits}
(The reel is the file section number in a USASI label.)

⟨date⟩ ::= {a five digit integer}
(The first two digits specify the year, and the last three digits specify the day of the year.)

⟨cycle⟩ ::= {an integer of one or two digits}

⟨terminal reference⟩ ::= S = ⟨integer⟩, A = ⟨integer⟩
(S represents the segment number, and A the relative address within the segment of the last non-intrinsic syllable that was executed.)