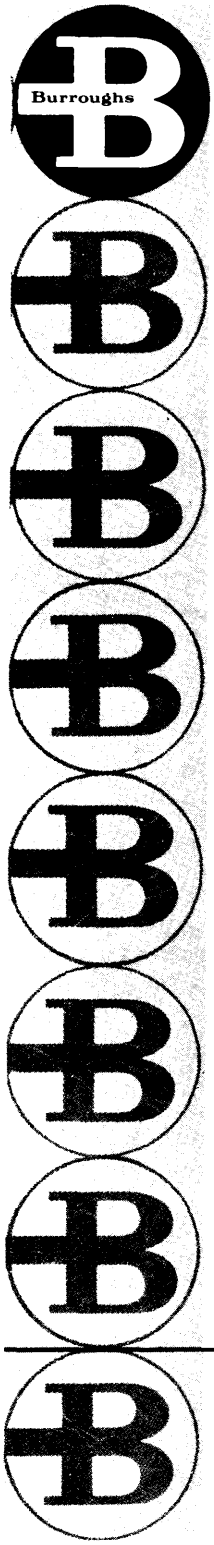
BJ-8

APRIL, 1966



B8500 SYSTEM

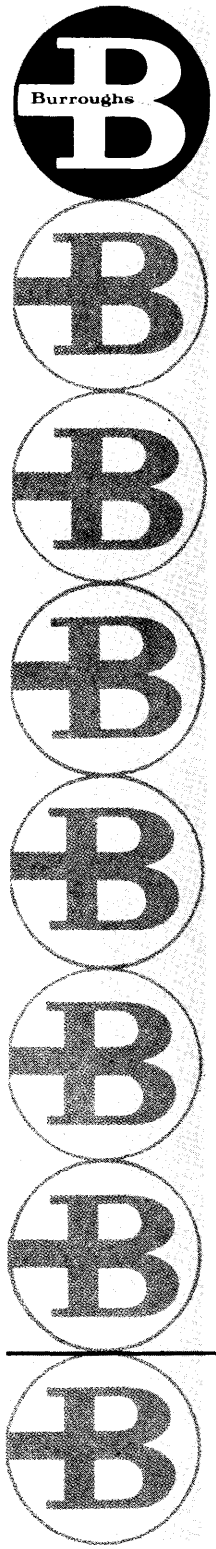
REFERENCE MANUAL

Burroughs Corporation

Defense, Space and Special Systems Group

BJ-8

APRIL, 1966



B8500 SYSTEM

REFERENCE MANUAL

Burroughs Corporation

Defense, Space and Special Systems Group

The information in this publication is subject to change. Revisions will be issued to advise of such changes and additions.

April 1966

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

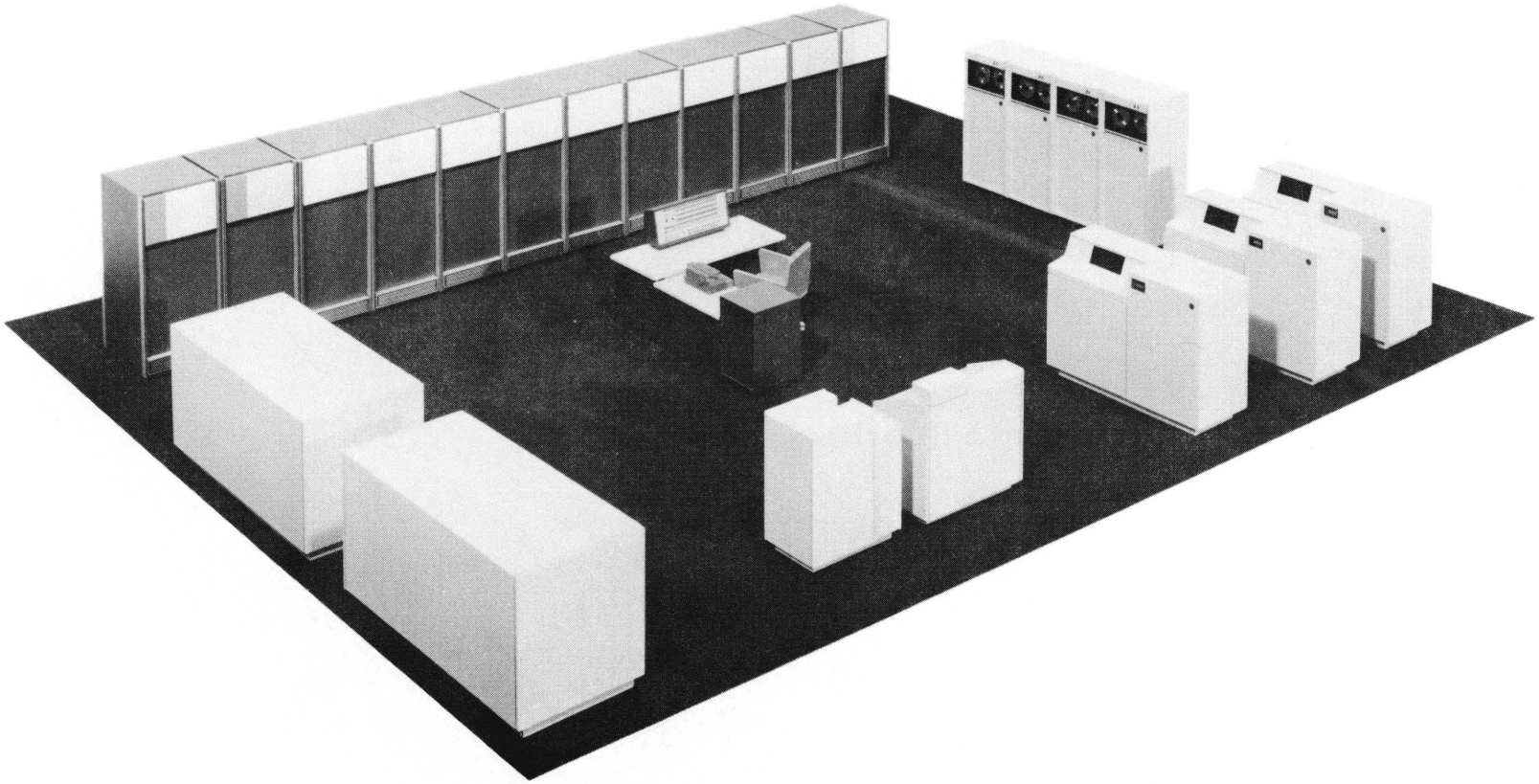


Figure 1-1. B8500 Data Processing System

INTRODUCTION

The Burroughs B8500 Data Processing System is a very large-scale, modular, high-speed, "third generation", information processor.

The unique, modular hardware and software organization of the B8500 provides a computing system which is flexible, self-regulating, and essentially free from "down-time". This modular organization, the various multi-processing and multi-programming features, and the powerful input-output capabilities of the system guarantee excellent performance over the broad spectrum of data processing tasks ranging from batch and "brute force" computation to conversational and on-line time-shared operations.

The B8500 employs the latest proven hardware techniques, including monolithic integrated circuits for the implementation of its logic and magnetic thin-films for the fully-shared random-access memory.

The information presented is intended for use as an introduction for persons unfamiliar with the B8500 System and as a general reference source for individuals engaged in marketing, programming, operating, and maintaining the B8500 System.

Listings and definitions of abbreviations, acronyms, and terms used in this manual are given in Appendices E (abbreviations and acronyms) and G (glossary).

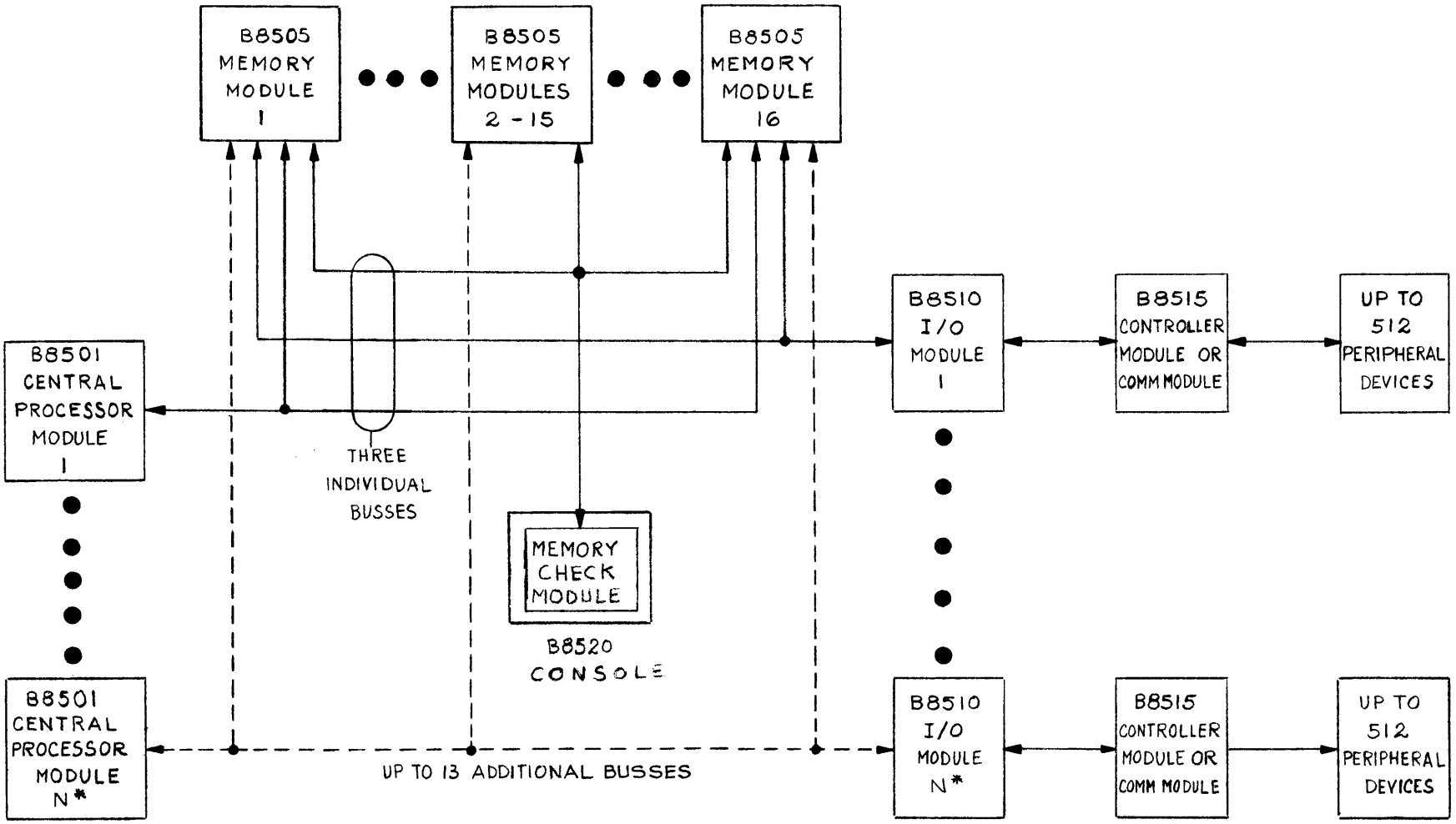


Figure 1-2. General Organization of the B8500 System

CHAPTER 1

SYSTEM DESCRIPTION

SYSTEM CONFIGURATION

The configuration of a B8500 Data Processing System depends upon specific applications and expected work loads. The B8500 is essentially a system of systems. Because the B8500 is a modular system, it can be tailor-made to fit a user's current or changing requirements. A typical installation of the B8500 System is shown in Figure 1-1.

Figure 1-2 shows the general organization of the B8500 System. The heart of the system is the Central Data Processing System which is comprised of the B8520 Console and configuration of B8501 Central Processor Modules, B8505 Memory Modules, and B8510 Input/Output (I/O) Modules. The three basic modules (processor, memory and I/O) are arranged into a central system cabinet configuration. A central data processing system may contain one or more processor modules, one or more I/O modules, and a maximum of 16 memory modules. The total combination of processor and I/O modules may be extended to 15 (e.g., if only one processor module is used, 14 I/O modules can be used).

A B8500 System also includes one or more B8515 Controller Modules, Communication Modules, and the necessary number of peripheral devices (such as disk file systems, magnetic tape systems, card readers, etc.) Each I/O module used in a system utilizes 512 simplex channels to handle peripheral devices. Table 1-1 lists the model numbers, names of equipment, and the maximum quantities which may be implemented.

SYSTEM CAPABILITIES AND FEATURES

Table 1-1 lists the basic capabilities of the equipment modules and peripheral devices used in the B8500 Data Processing System. The following paragraphs describe the general physical and functional features of the system.

Hardware Modularity

Three basic functions are implemented on the B8500 System: data processing functions, memory functions, and input/output functions. Each function is handled by a type of module specifically designed for its task (i.e., the B8501 Central Processor Module handles the data processing functions, the B8505 Memory Module handles the memory functions, and the B8510 I/O Module handles the input/output functions). All modules of one type are identical. Thus, a standardized interconnection network provides for freedom in the selection of a system configuration.

TABLE 1-1

B8500 DATA PROCESSING SYSTEM-EQUIPMENT CONFIGURATION AND OPERATIONAL CHARACTERISTICS

Model No.	Name	Maximum Quantity	Capability Features
B8520	Console	1	Operator's and test console with capability of automatically testing memory modules
B8501	Central Processor Module	14*	High-speed computation center of B8500 System. Performs arithmetic, control, data transfer, and interrupt service functions Contains four high-speed local memories 20-megacycle clock rate
B8505	Memory Module	16	Random access thin-film main memory Capacity of 16,384 52-bit words (four 52-bit words per location) 500 nanosecond cycle time (200 fetch, 300 store) 20 megacycle clock rate Maximum data rate: 69 million characters per second
B8510	Input/Output (I/O) Module	14*	Extremely high-throughput Input/Output device Thin film local memory with 1024 100-bit word capacity and 500 nanosecond cycle time, 1.7 usec service cycle
B8515	Controller Module	As required	Houses peripheral device controller submodules which provide the compatibility and buffering between I/O modules and peripheral devices if required
B470	Disk File System	20 per disk file controller	Fast access mass memory subsystem with 200-million 48-bit words and 30 millisecond access time (average). (Access time is substantially reduced by the use of the "queuer" described in this manual.)
B471	Disk File Electronics Unit	20 per Disk File System	
B475	Disk File Storage Module	100 per Disk File System	

* Maximum conbination of I/O and Processor modules is 15.

TABLE 1-1 (continued)

B8500 DATA PROCESSING SYSTEM-EQUIPMENT CONFIGURATION AND OPERATIONAL CHARACTERISTICS

Model No.	Name	Maximum Quantity	Capability Features
B425	Magnetic Tape Unit	Eight per Magnetic Tape Controller	Reads and records data at a density of 800 bits per inch and a speed of 120 inches per second 2.88-million 48-bit word capacity on each reel of magnetic tape Average transfer time of 9,000 words per second
B124	Card Reader	As required	800 cards-per-minute rate
B129	Card Reader	As required	1400 cards-per-minute rate
B303	Card Punch	As required	100 cards-per-minute rate
B304	Card Punch	As required	300 cards-per-minute rate
B329	Line Printer	As required	1040 lines-per-minute rate 64 characters (37 alphanumeric and 27 special) 132 print positions
	Communications Module	As required	Interfaces low-speed, voice-grade, and wide band communications lines with I/O modules

CENTRAL PROCESSOR MODULE

The choice of the number of central processor modules and I/O modules depends entirely upon the application of the B8500 System. There may be as many as 16 communications busses connected to the memory modules, and these busses can serve the console and any combination of central processor and I/O modules. When more than one processor module is used, the Executive Scheduling Program dynamically manages the job assignment of each processor module through the processor's control registers. Thus, any program can be executed by any processor module. There is no fixed master slave relationship between the processor modules. If more than one processor module is available and asynchronous segments of a program are indicated, the segments can be simultaneously processed on the multiple processor modules.

I/O MODULE

The most important feature of the I/O Module is its independence from the Central Processor Module. In effect, the I/O Module is a separate processor with its own local memory unit, logic and arithmetic functions, and communication capabilities. Because it provides channels for all data transfers between peripheral devices and the System Memory, the I/O Module executes transfers from one peripheral device to another peripheral device independent of direct Central Processor control.

The system operations of the I/O Module are programmed separately from the Central Processor Modules. This programming is coded in an assembly language designed solely for the I/O Module. Then, the symbolic program is assembled into an object program by means of an assembler program written in ALGOL. Input to the assembler is on punched cards. The output consists of a program listing, with relative memory assignments, and a tape that contains the assembled object program for printing and ultimate loading into the B8500.

The simple transfer function between peripheral devices and system memory is similar to the more conventional descriptor-controlled and buffered I/O Module in the sense that there is a descriptor generated by the Central Processor for each data transfer executed. However, in data transfers to and from peripheral devices, the I/O Module completely controls the I/O operation using parameters stored in the I/O Module's program data area. The execution of buffered I/O can now be treated by Central Processor programs in terms of tape, card, and print files, rather than by individual tape records, card images, and print lines. This means that I/O processing overlaps the Central Processor for long periods of time, rather than the more conventional technique of overlapping a single record. Using this overlap method reduces the number of interrupts to the Central Processor, thereby reducing the amount of software overhead associated with a Control Program.

MEMORY MODULE

As required by the workload at an installation, there may be as few as one and as many as sixteen B8505 Memory Modules in a B8500 System. Because each memory module can operate as an independent memory, system programs can be completely relocated to any memory module. All programs, therefore, are written with relative addressing methods to capitalize on the relocatability feature.

STORAGE FACILITIES

Storage facilities in the B8500 Data Processing System include a hierarchy of memories ranging from 50-nanosecond cycle scratch pad memories to reels of magnetic tape. The storage facilities consist of the following:

- Each B8505 Memory Module has 16,384 52-bit words and a full cycle time of 500 nanoseconds. Words are stored or fetched in four-word groups (208 bits) to enable a maximum data transfer rate of 416 million bits per second for each module.

- The local memory in each I/O Module has 2048 52-bit words and a full cycle time of 500 nanoseconds.

- The local integrated-circuit memories in each processor module have 32 52-bit word and 24 70-bit word capacities and 50 nanosecond cycle times. A 28-word (18 bits per word) associative memory is also contained in each processor.

- The disk file subsystems, each have a capacity of 200 million 48-bit words and an average access time of 30 milliseconds. Transfer rates average 8 million bits per second (166,666 48-bit words per second).

- The Magnetic Tape Units have storage facilities for serial files on 2400-foot reels of magnetic tape. Each reel of tape has a capacity of 2.88 million 48-bit words. The transfer rate possible is 9,000 words per second.

Programs and data are stored in the high-speed (500 nanosecond), thin-film main memory storage and are vailable when required by operational programs. Data required for actual execution of program steps is automatically fetched, ahead of time, from the 500-nanosecond thin-film memory in a memory module and is stored in the 50-nanosecond thin-film memory in a processor module. The disk file system backs up the 500-nanosecond memory. In general, the Executive Scheduling Program controls the transfer of data from the slower speed peripheral storage devices to the disk file system before allowing a program to run. As much information as is needed at any one time is then transferred into the 500-nanosecond memory by the Executive Scheduling Program.

System Availability and "Fail Soft" Reliability

The reliability of the system is enhanced by designing the interconnections between equipment modules as a group of parallel networks. This means each module of a specific type is independent of every other module of the same type, both in power supply and data flow.

If a component in a basic equipment unit (i. e., unit at the end of the network) should fail, obviously that particular unit is not operable, and the medium being operated upon will have to be relocated to another basic unit. An example of this might be a magnetic tape transport, where upon component failure, the reel of tape would have to be removed and mounted on another tape unit. If a component in the Magnetic Tape Controller should fail, however, the tape units normally serviced by the defective Controller would remain in operation. They would be serviced by another Controller via a second data path. The Controller assuming this new load would still handle the tape units originally assigned to it. The result is that the system would remain operative, but data flow would be at a reduced rate. Similarly, at the next level in the hierarchy of equipment, if a component in an I/O Module should fail, then an alternate path is available from the Magnetic Tape Controllers to a second I/O Module. In this fail-soft environment, therefore, jobs with high priority will not be materially affected by the failure of an individual component.

Because of the modular concept and interconnection network, on-site modifications for continuing quality improvement, can be made to equipment modules without interrupting the operation of the system. For the same reasons, unscheduled maintenance of the equipment modules can be performed while the system remains on-line.

Software Modularity

UNIFORM PROGRAM STRUCTURE

The B8500 operating system consists of an Executive Scheduling Program (ESP), service programs (such as an I/O procedure), and compilers for ALGOL, COBOL, and FORTRAN IV (ASA). Software modularity is achieved by requiring all program segments to conform to a uniform structure. Compilers, as well as the programs they compile, use the uniform program structure.

The Executive Scheduling Program itself uses the uniform program structure. The service routines surrounding ESP have the uniform program structure, but they also enjoy direct access to some of ESP functions.

Multiprogramming And Multiprocessing

Two of the most important features of the B8500 System are its multiprogramming and multiprocessing capabilities.

Multiprogramming is the concurrent processing of several programs on a "time-sharing" basis. For example, a program is normally made active and run when (1) all of its necessary input information has been assembled on the disk file system, and (2) the required output units are available. While a specific program is waiting for inputs, etc., other programs will have already been running. Therefore, the waiting time on one job is not wasted, and an effective match is made of the fast internal speed and the slower peripheral devices.

Multiprocessing, which is the simultaneous execution of two or more object programs, is possible on a B8500 System having two or more processor and I/O modules. Thus, when enough memory and peripheral devices are supplied, two processor modules can practically double the throughput of a system. Input/output operations occur nearly independent of the Processor, and are in parallel with processor module operations. Memory/processor operations can also occur independently of other system operations, because each memory module has its own list of descriptors which it can execute. An example of how concurrent operations may occur in a B8500 System, using only one central processor, one memory module, and one input/output module in conjunction with some peripheral equipment, is shown in Figure 1-3. The three modules are on the left half of the drawing, and the peripheral equipment is on the right half. As indicated in the upper right corner of the drawing, concurrent or parallel data processing occurs in three locations:

Central Processor Operations

Peripheral Operations

Input/Output Operations

CONCURRENT CENTRAL PROCESSOR OPERATIONS

The B8500 System goes beyond the conventional technique of buffering I/O peripheral devices with processing operations, commonly known as "reading while writing while computing". It even goes beyond the usual overlapping of a fetch of an instruction from memory with the store of data to memory. The B8500, in addition to using the above time-sharing method, also overlaps the execution of several machine instructions. This advanced degree of time sharing is made possible on the B8500 through the use of multiple "stations" in the Central Processor. Each station has an ultra-fast memory and queues that enable it to partially or completely perform a program instruction while another station is doing likewise. This "compute-while-compute" time-sharing technique, in concept, is analagous to the Store and Forward method used in communications systems.

CONCURRENT PERIPHERAL OPERATIONS

Concurrent peripheral operations are possible in a B8500 System because the peripheral device controllers contain the necessary logic, storage registers, etc. to allow each controller to process data going to or coming from one of the devices it controls. The Disc File Controller, shown in Figure 1-3, can be assembling an 8-word byte for transmission to the I/O Module, while the Magnetic Tape Controller can be unpacking a 4-word byte for storing on tape. At the same time, the Communications Module can be buffering the interchange of data between the I/O Module and a peripheral device.

CONCURRENT INPUT/OUTPUT OPERATIONS

As indicated in Figure 1-3, an I/O Module can interface with peripheral devices over any one of 512 simplex channels at a maximum data exchange rate of 286,000,000 bits per second. Such high-speed data transfers are possible when, for example, the I/O Module is communicating with another data processing system. In general, the number of concurrent operations possible in the peripheral equipment controllers connected to one I/O Module is determined by the maximum rate at which the Data Service Unit can handle data, and by the number and types of peripheral devices required by a particular system configuration.

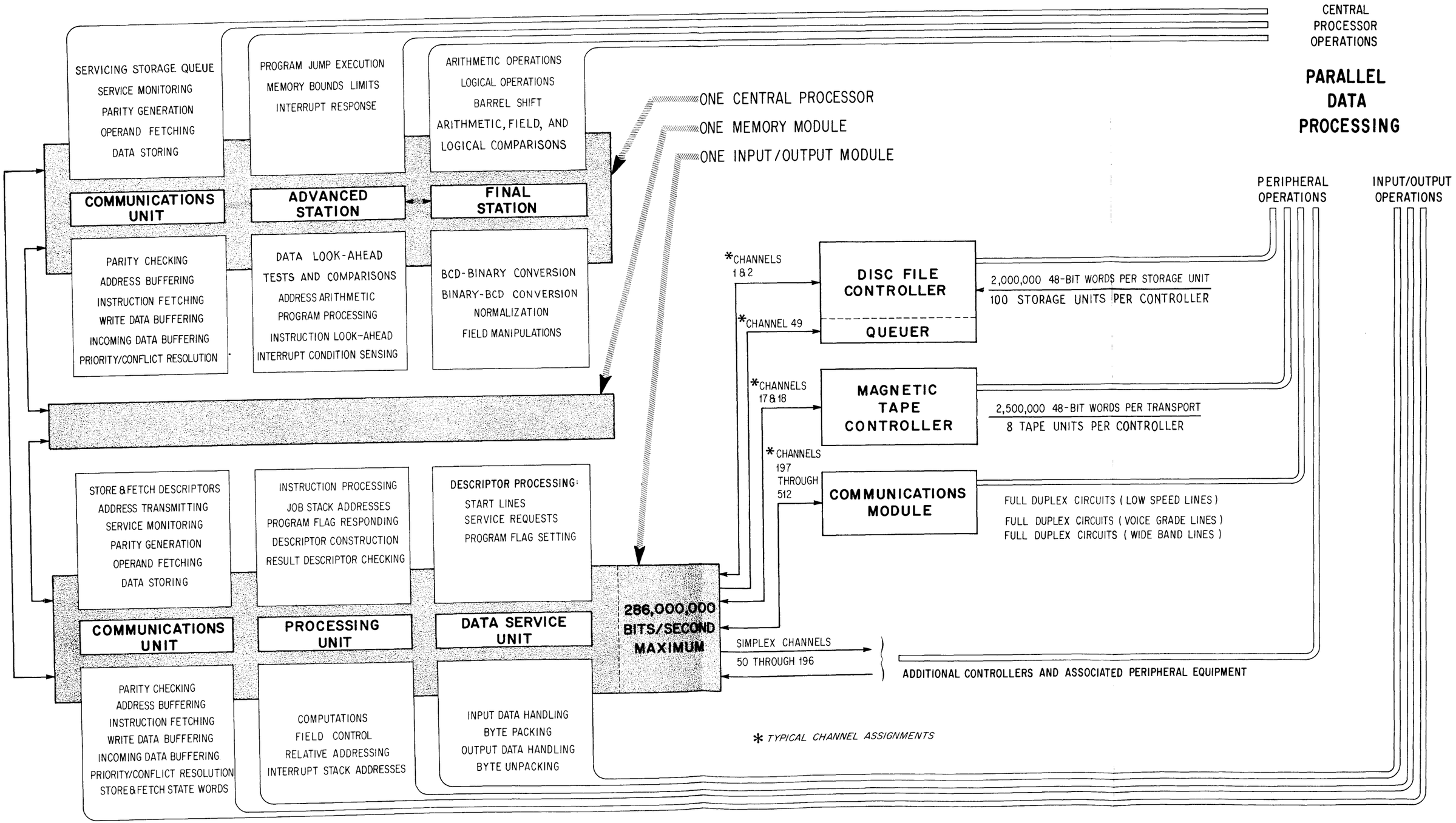


Figure 1-3. Concurrent Operations in B8500 System

Computerized Input/Output Module

The B8510 I/O Module is capable of simultaneously servicing up to 512 simplex channels and contains an independent processing capability which permits it to perform routine duties without the need to interrupt the processing modules. One of the primary functions of the I/O Module is to automatically (without Central Processor intervention) buffer, into high-speed disk files, the low speed data arriving over communications lines from remote stations. The Central Processor can then service all remote stations from the high-speed disk files and is not inefficiently tied to a low-speed device, such as a card reader.

The most important feature of the I/O Module is its independence from the Central Processor. In effect, the I/O Module is a separate computer with its own local memory, logic, arithmetic, and communications capabilities. Because the I/O Module provides channels for all transfers of data, such as transfers between the peripheral storage devices and Main Memory, or transfers between two peripheral devices, data transfers are effected independently of direct control from the Central Processor.

ALLOCATION OF I/O CHANNELS

The numbers and types of devices that can be connected to an I/O Module depend to some extent on how the 512 simplex (one way) channels are utilized. Data is transmitted over a channel one "byte" at a time. A "byte" can consist of 8 words, 4 words, one word, or less than one word (i. e. groups of bits that are submultiples of 48). The 512 simplex channels consist of 256 input channels and 256 output channels, and are allocated as follows:

Channel Number	Byte Size
1 through 16	8 words
17 through 48	4 words
49 through 512	1 word or less

The disk file controllers use 8-word bytes and therefore exchange data over channels 1 through 16. Each disk controller requires one input and one output data channel, plus an additional output channel for the disk queuer. The queuer channels may be from number 49 to 512, as the byte size is not more than one word. Thus one I/O Module can accommodate eight disk file controllers and 800 disk storage units for a total disk file storage capacity of 1.6 billion (48 bit) words. Additional disk file storage requirements are met by using additional I/O Modules.

The magnetic tape controllers use 4-word bytes, and exchange data over channels 17 through 48. Because the magnetic tape system requires simultaneous transmission of data in two directions (duplex), each tape controller requires one input and one output data channel. One I/O Module can accommodate up to 16 tape controllers and 128 tape transports.

System Expansion

The expansion of the B8500 is enhanced by the modular design concepts implemented in both equipment and programming areas. Additional Processor, Memory, and I/O Modules can be added without modifying existing programs or equipment. Programming systems are designed to utilize whatever equipment is available, and, if more equipment is made available, it is used with a consequent increase in information throughput.

Executive Scheduling Program

DYNAMIC ALLOCATION OF EQUIPMENT

One of the principal objectives of the Executive Scheduling Program is to dynamically allocate the equipment modules-processors, memory, I/O channels, etc., to a constantly changing set of jobs and thereby achieve a high degree of utilization for all modules. As defined on page 1-6, sharing equipment modules among many programs is called multiprogramming. The ESP goes a step further by multiprogramming a set of jobs that consist of both user requests and operating system functions. As an example, the service functions for jobs that are running or being readied to run are entered in the set of jobs to be done (Hot Job Table) along with the main job entries, and the whole set is multiprogrammed (and multiprocessed if more than one processor is in the system). The segmentation of programs permits a job to be executed with limited memory and the job's requirements for equipment modules are handled dynamically through the course of its execution.

MEMORY PROTECTION

Memory protection is also included in ESP. Although a job may consist of several software subroutines stored in independent memory locations, both read and write memory protection are provided by data descriptors. Actually much of this protection is provided by hardware design, but it is set up and controlled by ESP.

CALL-BY-NAME FILING SYSTEM

The filing system of ESP provides reference to files and file items by name rather than by storage medium or absolute address location.

Interrupt and External Request Capabilities

A comprehensive interrupt system is implemented in the B8500 System to service error conditions and/or requests. The interrupt system provides the Executive Scheduling Program (ESP) with the facility to maintain control of all system functions. ESP includes an interrupt processor program which determines what action is to be taken for each interrupt condition. The interrupt system utilizes a 70-bit condition register and a 70-bit mask register. All interrupts are maskable, i.e., a bit in the condition register is set whenever a particular condition exists, but the processor will not respond to the setting of the condition bit unless the corresponding mask bit is set. The mask, then, serves as a multilevel control device. When the processor is operating in the normal mode of operation, it may be interrupted by any unmasked interrupt condition. Once the processor is operating in the control mode of operation, it can only be interrupt conditions (e.g., parity error, no access to memory, etc.) regardless of the mask register settings.

External requests are recognized by the I/O Module. Once the requests are recognized, the I/O service programs may begin the necessary response to the request and accept all further data from the requesting peripheral equipment. A large number of external requests are treated solely by the I/O Module and do not require any further processing. If the request requires central processor handling, entries are made into the job tables for ESP, and the request is scheduled and serviced as a standard I/O procedure. Thus, the normal service to other peripheral equipment is not disrupted. Only a small number of external requests result in immediate interrupts of the central processor.

Compilers

The ALGOL compiler uses the compiling technique known as recursive descent syntactic analysis. This technique compiles quickly, makes modifications easier, and produces efficient object code. The ALGOL compiler implements virtually all of ALGOL-60 and provides extensions for I/O operations, partial word operations, string manipulation, and diagnostics.

The FORTRAN IV compiler implements the A.S.A. FORTRAN IV language. The compiling is done in one pass using conventional precedence scan techniques. The hardware stack (p. 3-11) used in the B8500 makes this type of analysis very fast.

The COBOL compiler implements D.O.D. COBOL-61, extended. Data segmentation and the ability to independently compile program segments are also included.

Source language programs written in ALGOL, COBOL, or FORTRAN, can be placed in the system's library, and called upon for insertion in other source language programs. The borrowing program need not be written in the same language as the borrowed program.

CHAPTER 2

PROGRAMMING FEATURES

EXECUTIVE SCHEDULING PROGRAM

Introduction

The Data Processing Operations Manager, in today's working environment, is faced with the following situation:

1. He has a large number of programs that need to be processed with different priorities and that are scheduled at various times of the day or week or month.

2. He wants to make the most efficient use of a limited number of peripheral devices, but he is willing to sacrifice efficiency if required to meet schedules. For example, he has a large processing work load and uses several complete computer systems to meet his schedules, even though he knows he could save money using one large system with the capability of parallel processing, multiprocessing, and multiprogramming.

3. If a peripheral device or a unit of the computer malfunctions, he loses the use of the computer until he can substitute an alternate unit, or an alternate job, if possible.

4. He needs to know in advance when the system will require data inputs so that valuable computer time will not be wasted while "setting up".

To meet the challenge of situations like the one above, the B8500 System has an internal, nonhuman controller, supervisor or operating manager. We call this internal overseer the Executive Scheduling Program; ESP for short. The name is aptly chosen, because it is the administrative program to which all other "worker" programs report for guidance as to their execution. In earlier Burroughs data processing systems, the forerunners of ESP were called MCP and AOSP. No matter what it is called, it is most important to know what it is and what it does. Let us delve further and examine how this nonhuman operations manager, in accomplishing its many functions, does meet the needs of today's operating environment.

What Is The Purpose Of ESP?

The purpose of the Executive Scheduling Program can be summed up in one sentence. ESP schedules all services (such as memory allocation, loading of memory, assignment of system components both equipment (hardware) and programs (software)), to achieve the maximum utilization of all components of the System. To illustrate, a large number of active programs requiring various services are present in the System, and their current status and required service are recorded. As some component of the System becomes available, e.g., processor, memory space,

or a peripheral device, the component is assigned to the active job of highest priority requiring it. An important concept to remember is that each component of the system provides a 'unit' of service, and then goes on to service another program. The main function of the Executive Scheduling Program, then, is to keep track of the services required by programs and to schedule these services as components (hardware or software) become available.

Of What Does ESP Consist And Where Is It Located?

ESP is a master program (software), comprising many special routines that provide automatic control over those functions listed below. These special routines can be logically divided into two groupings: Executive Routines and I/O Routines.

FUNCTIONS OF THE EXECUTIVE ROUTINES

The Executive routines perform the following functions:

1. Handle analysis and service of internal (operational, error, and power) interrupts.

2. Schedule the computer and peripheral equipment, including remote terminals.

3. Handle program loading, file retrieval, and file maintenance.

4. Handle system protection and recovery, including memory and file protection, reruns, individual program recovery, and report on the current status of programs being executed and being scheduled.

FUNCTIONS OF THE I/O ROUTINES

The I/O routines perform the following functions:

1. Perform accessing commands, including scheduling, error checking, return facilities, and direct the transfer of data to file handler buffers.

2. Process the file handler functions.

3. Perform communications operations, including receiving input communications data, activating a user's program to process the data, accepting output communication data from the user program, and transmitting this data.

ESP resides in three separate locations in the B8500 System. It is stored on magnetic tape as the primary source for changes made to it. The complete program is also in disk storage, and serves as the primary source while operating. Segments of ESP are always in Main Memory for program execution. Main Memory contains enough of ESP to call forth from disk storage whatever other segments are required to perform a task.

What Can ESP Accommodate?

ESP is written to operate on the minimum B8500 System configuration, or to accommodate the maximum B8500 System configuration. This approach permits easy inclusion of new modules to the system, and permits the dynamic readjustment of the system in the event of a module failure.

What Does ESP Do?

As stated before, ESP is composed of special routines that provide automatic control over such functions as scheduling, program loading, allocation of memory, input/output operations, and time sharing of processors in the multiprocessing of user programs. In doing these ESP automates many of the functions that were previously performed by an operator. ESP controls the environment and operation to the extent that the major action of the system operator is one of response to system needs rather than offering direction to the system, as shown in the following examples:

SCHEDULING

One of the operator's tasks which has been automated is the scheduling of the processing system. ESP schedules jobs automatically according to priorities, time-of-day, deadlines, etc. ESP even informs the operator of the requirement for data (files) or special forms (checks, etc.) in advance of actual processing start time. The operator can then ensure that the data and forms are available to the system

when required. ESP can make such advance requests because, as it schedules devices and programs, it is aware of the requirements of all programs that are to be run. Due to the advanced software approach of the Executive Scheduling Program the operator need not be aware of precisely which jobs are being executed during a given instant.

ERROR RECOVERY

Another task taken from the operator is error recovery. Files that have been altered by a program and are subsequently found to be in error, are restored by ESP with a minimum of operator intervention. Copies of all altered records are placed on an audit trail tape for reference and available for reconstruction using standardized recovery routines.

USAGE ACCOUNTING

Still another function assumed by ESP is accounting and reporting. Processor usage statistics are accumulated by ESP. ESP maintains records of user program processing times for billing and accounting.

Uniform Program Structure

Program segmentation is a basic requirement of a time sharing system having many users, because it permits a large number of active programs to be present in the system, concurrently. A B8500 program may be considered as the output of one compilation, consisting of: program segments, data segments, an operand stack extension, a working storage segment, a program reference table, file list, CHORE (CHain Of Runs Executive) working area, cold job table, and a hot job table (Figure 2-1).

An active program (one that is in the hot job table) must retain in main memory one program segment, an operand stack, a working storage segment, and a program reference table. A large program may also require additional program and data segments during its execution. Whatever segments are required the allocation of these memory segments is provided at the time the program or data segments are referenced, through descriptors in the program reference table. The descriptors define the segments as they appear on disk storage. When the segments are allocated in main memory, the descriptors are altered to provide a means of communication between the separately allocated segments within main memory.

PROGRAM SEGMENTS

A program segment is a logical, independent grouping of instructions and constants which is reentrant. The maximum segment size is 4096 words. Therefore, the output of a compilation may consist of many program segments. The compiler must provide a description of the linkage between program segments and external procedures called by the object code. All computer-generated program segments are independent of position in main memory. They can be placed anywhere in main memory and be executed relatively from the starting position of each segment. The Base Program Register (BPR) contains the absolute address of the starting point of the program segment currently being executed. At any time, the program step within the segment being executed is defined by the Program Count Register (PCR). The PCR designates not only the word but also the syllable (all relative to the BPR) that contains the operator presently being executed in the Advance Station within the Central Processor

When a program jump to another segment occurs, the BPR will be altered to reference the starting position of the newly called segment. The PCR will contain the relative address of the first instruction to be executed within the new segment.

Program segments can only be read and are protected against accidental modification by placing them outside the area bounded by the Memory Bounds Registers, which define the read/write areas for a program. Program segments are referenced by program descriptors (in the program reference table) which must have an appropriate tag configuration. This prevents reading the program segments as data objects.

DATA SEGMENTS

A data segment is a block of storage which is accessed through a PRT descriptor having alternate bounds tag configuration. Data segments can be shared on a system or run basis. They can be read only or read/write objects. Data segments are used to represent arrays, I/O buffers, and similar constructs which require contiguous memory locations. Data segments are addressed relative to the lower boundary of an alternate bounds descriptor. The alternate lower bounds (ABL) value points to the starting point of the Data Segment, the alternate bounds upper (ABU) value points to the last entry in the segment. These registers provide both read and write memory protection.

OPERAND STACK EXTENSION

The stack extension, located in main memory, is a logical supplement to the central processor's hardware operand stack, and uses the push down concept for evaluation of Polish strings. The memory allocated to the stack extension is dependent upon the amount requested by CHORE. One stack is allocated and will be used by all runs within a single CHORE compilation. Memory protection is provided by the stack bounds upper and stack bounds lower limits. Any attempt to read or write data outside of the allotted area will cause a bounds violation interrupt.

WORKING STORAGE SEGMENT

The working memory segment contains a user's temporary and Global data. The working memory segment is addressable relative to either the Base Data Register (BDR) or the Base Index Register (BXR). The BDR is the reference point for the Global (Common) data area. Global or Common refers to that data which can be used by all program segments of all active programs. The BXR is the base address for data that is local to the segment of the object program that is currently being processed. When a jump to another segment is executed, the BXR is incremented to provide the new segment with its own local working area. Parameters to be passed to the called segment, prior to the execution of the jump, are stored relative to the new BXR. During the implementation of the jump operation, a return control word is placed in location zero relative to the new BXR. The return control word contains the information required to return to the caller and resume processing. This structure provides the method for subroutine nesting and recursion.

PROGRAM REFERENCE TABLE

The program reference table (PRT) is a read-only segment and contains program descriptors that will provide links to all segments generated by the compilation. All descriptors within the program reference table are relative to the PRT base register and bounded by the Program Reference Table Limit (PRTL) to prevent a reference beyond the area allocated to the PRT. The compilers generate a PRT entry for each reference to a program segment or a data segment. Each PRT entry consists of a program descriptor, description, and a link.

Descriptor

The descriptor contains the information required to access a segment and is either a jump control word (used to access program segments) or an alternate bounds word (used to access data segments). The alternate bounds word also applies bounds around the data object, allowing the data object to be allocated outside the normal working area. The alternate bounds descriptor also designates the segment as a read-only or a read/write object.

Description

The description contains the information required to obtain, establish and control the associated object. Some of the information contained in the description is object name, file name, position in file, size, and type of descriptor required to access the segment.

Link

The PRT contains one link word per descriptor and is required to link the descriptor to the description of the object. Since all that is required to run a program is one program segment, a PRT, stack and a working storage area, it is very likely that the next segment that is required will not be located in main memory. All references to segments other than the segment currently in process must be made via the PRT. If a segment is not in main memory its descriptor will be tagged "not present" and upon any reference to the descriptor the processor will be interrupted, transferring control to ESP. ESP first deactivates the calling program, keeping track of the program step on which this occurred. Then, utilizing the link word, ESP will find the description of the called object, and through interpretation of the description, ESP will allocate space, and the referenced segment will be loaded into main memory. The proper descriptor for this new segment will be formed and placed in the PRT. The calling program will be reactivated to make a second call on the segment, via the PRT. This time the segment will be present, and processing will continue.

FILE LIST

The compilers will produce as part of their output a file list. The file list will contain the names and characteristics of all files referenced by the compilation. When the compilation is

readied an entry is placed in the generated PRT that points to the file list. The file list, although generated by the compiler, is inaccessible to the user. When the compiler-generated code references a file (for reading, opening, etc.) it will reference through the file list descriptor in its PRT. Parameters such as relative position in file list, record name, block name, etc., are placed in the parameter area prior to execution.

There are three levels of file lists; Compilation, Run, and CHORE. The output of a compilation is a series of program units which can not be further subdivided, for example, a subroutine in ALGOL, or a paragraph in COBOL. A Run may consist of one or many compilations. A CHORE is a collection of Runs which performs a job from start to finish, i.e., "Salary Payroll".

The CHORE file list contains entries for all files used by a CHORE. When a Run is established as being active in the system, the subset of the CHORE file list belonging to the Run is entered into Main Memory and pointers to it are entered in the Hot Job Table (HJT). The Run file list is further subdivided on a compilation basis.

CHORE WORKING AREA

The CHORE working area is a segment of memory which is common to all runs within a CHORE. The size of the working area is specified at the CHORE level. This working area is for passing parameters and data from CHORE to Run, Run to CHORE, and Compilation to Compilation. It is a data segment accessible only to user compilations which are part of the series of runs called CHORE. The first word of the working area is reserved for use as the "Sequence Switch". The remainder of the area is free to be used as the need dictates.

At the object program level, this working area is a data segment, with a fixed name and is of the type "CHORE job's" (global). For a CHORE job's segment, in addition to each run being a user of the segment, CHORE is also a user, so the segment remains intact between runs. This segment will always be supplied by CHORE and will be available during the execution of the object (worker) program. It is the responsibility of the compiler, or user, if he desires, to ask for use of the area via a PRT entry.

At object program time, the CHORE working area can be completely modified, if desired. The area is treated the same as any other data segment belonging to the run.

At the source language level, the working area can be operated on using whatever mechanisms available in the particular source language. "STATUS-SWITCH-1, ON STATUS IS . . . " is an example of its use in COBOL.

COLD JOB TABLE

Cold jobs are descriptions of potential candidates for execution. A collection of these cold jobs is maintained in mass storage and called the cold job table. A cold job entry remains in the System from the time it is introduced for running (by CHORE) until its outputs are delivered (completed by the System). Information contained in the cold job entry includes the method of determining its priority, the class of job (conversational, batch, deadline, real time), estimated running time, amount of storage required, predecessor cold job links, age, data files, maximum core segment size, etc. To summarize, the information contained in the cold job table is what is required to introduce jobs into the system, based on current and anticipated computer load matched against the priority of the job to be performed.

HOT JOB TABLE

It is convenient for descriptive purposes to consider the Hot Job Table as logically divided into three parts: Start, State, and Control/Accounting.

Start: Within the program structure and other objects which constitute the requirements for a run, the start portion of the HJT is the link between the external structures of interrupt responders, allocators, filing, etc. The start portion of the HJT is nothing more than a PRT consisting of a few entries corresponding to the run's stack, file list and first program structure.

State: The information which must be stored in order to obtain a "picture" of an active object program so that, if it is interrupted, it can be resumed at a later point in time. The interrupted process is resumed by restoring the saved information so that it appears that the process has never been interrupted. This information consists mainly of processor registers and conditions of flip-flops.

Control/Accounting: This information includes such items as status, I/O counts, time counts, abnormal condition exits (overflow, EOF), priority, links to its associated Sleep Table, Cold Job Table, and File List, etc.

Interrupt System

The B6500 central processor has a very comprehensive interrupt system, detecting 70 error and system control conditions. (See Table of Interrupts.) Upon interrupt or an ESP call the central processor is transferred from User program control to ESP program control, i.e., from Normal mode (User mode) to control mode 1 (ESP Mode). The central processor has three modes of operation, Normal mode, Control mode 1, and Control mode 2 (the latter two are ESP modes). Normal mode has a restricted instruction set and a maximum number of allowable interrupts. The control modes have a complete instruction set and minimum number of allowable interrupts. A control mode program can attempt to determine the cause of a normal mode interrupt by minimizing the possibility of further interrupt. During the processing of a User or worker type program in Normal mode the interrupt conditions that are allowed to interrupt the User's execution are restricted by the ESP. This is accomplished by controlling the content of the User's interrupt mask register.

Each time an allowable interrupt occurs and the processor is placed in a control mode, entries must be made in the Hot Job Table or Sleep Table describing the temporarily suspended program structure. A Hot Job Table entry is made upon transfer of processing from a User program to ESP. The Hot Job Table contains three sections; start, control and accounting, and state. For each interrupt the state section contains a record of the central processor's registers at the time of interrupt and a status indication of the reason this User structure has been suspended.

The sleep table entry space is allotted at the same time space is allocated for the rest of the program structure. A sleep table entry is one created upon the temporary suspension of an ESP process, and contains all of the values required to resume the suspended ESP process. There is at least one sleep table space assigned to each HJT. One sleep table entry is made for each ESP process that is waiting to be resumed. Each sleep table entry contains a status field indicating whether it is ready to run, whether it is waiting for an I/O operation, etc. Each sleep table entry is linked to its corresponding HJT, and to subsequent sleep entries, if any; otherwise, it is marked as the last sleep table entry in the chain connected to the HJT. Sleep table entries are also linked to all other sleep table entries in order to facilitate scanning (see figure 2-1.)

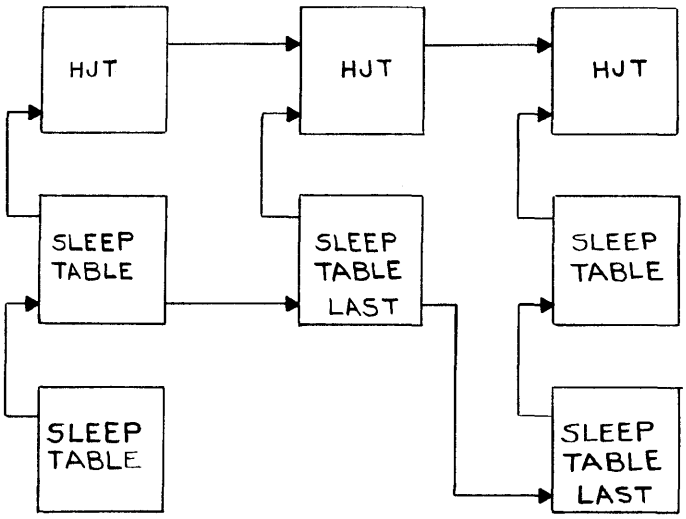


Figure 2-1. Hot Job Table and Sleep Table Linking

After entering control mode 1 (see Figure 2-2), a processor may be interrupted regardless of the mask register setting: by computer no-access-to-memory condition, computer parity error, stop instruction, etc. An interrupt occurring during control mode 1 will cause a processor to enter control mode 2. The occurrence of any of the above interrupt conditions while in control mode 2 will cause the processor to halt.

HARDWARE FUNCTIONS AT INTERRUPT

The transfer from User program execution (Normal Mode) to ESP interrupt service program execution (control mode 1) is accomplished by the central processor logic, after all Instructions in the final instruction queue (FINQ) have been executed. The current base program register (BPR), program count register (PCR), ADVAST address register (AAR), and specific control flip-flops are placed in the operand stack. The current value of the base interrupt register number 1 (BIAR1) is placed in BPR. The value of BIAR1 is determined by ESP (at the time the User structure is actuated) and designates the base address of the interrupt service routine. The program count register (PCR) is set to zeros. The next instruction to be executed is taken from the memory location indicated by BPR and PCR. The interrupt service routine will make a HJT entry for the User structure, link the HJT entry into the HJT priority chain, test for the interrupt condition, and transfer control to the ESP subroutine indicated by the interrupt. Upon completion of interrupt service the HJT status field is set to "ready to run". The processor then looks for something to do. It scans the sleep tables and the hot job tables and selects the program with the highest priority that is ready to run. At this time the loading of the processors is also evaluated to determine if more programs can be scheduled on the System.

MULTIPROCESSOR INTERRUPTS

The B8500 System is a multiprocessor system with all processors physically identical. There is no master-slave type of relationship, therefore special care is taken to insure that not more than one processor will be interrupted by the same system control interrupt, i.e., I/O-complete. This is accomplished by ESP controlling, for each processor the individual mask registers that determine which conditions are allowed to interrupt that processor. Also, critical areas of the interrupt service routines of the ESP are "locked out" with software controls. For example, if two processors attempt to use the same "critical" interrupt service routine at the same time, the first processor will set a software "lock" upon entrance to the routine, and the second processor, upon finding the routine locked, will be diverted to another function.

Functions Of ESP

INTRODUCTION

Although CHORE, Collector/Scheduler, and Interpreter/Controller routines are considered part of ESP, they are implemented as user programs. Hence, these three routines are called "External ESP" programs. Communications between these programs and the rest of ESP, called "Internal ESP" (Filing, I/O, Allocation, etc.), is accomplished as follows:

External to External-Procedure jump, segment jump as appropriate, tables accessible to routines communicating with each other, i.e., identical to methods used by normal user programs as produced by ALGOL, FORTRAN, etc.

Internal to Internal-Procedure jump, segment jump, or special transfers where required (e.g., switch BDR/BXR), via PRT global area, and tables or queues.

External to Internal-By macro call. It is highly desirable that this interface be the same as that for any user program and the ESP. Any special interfaces required are kept to a minimum. System tables are used both internally and externally.

Internal to External-Internal answers to external macro calls, common tables.

EXTERNAL ESP PROGRAMS

External ESP consists of the following functions:

1. CHORE (CHain Of Runs to be Executed)
2. Collector/Scheduler (C/S)
3. Interpreter/Controller (I/C).

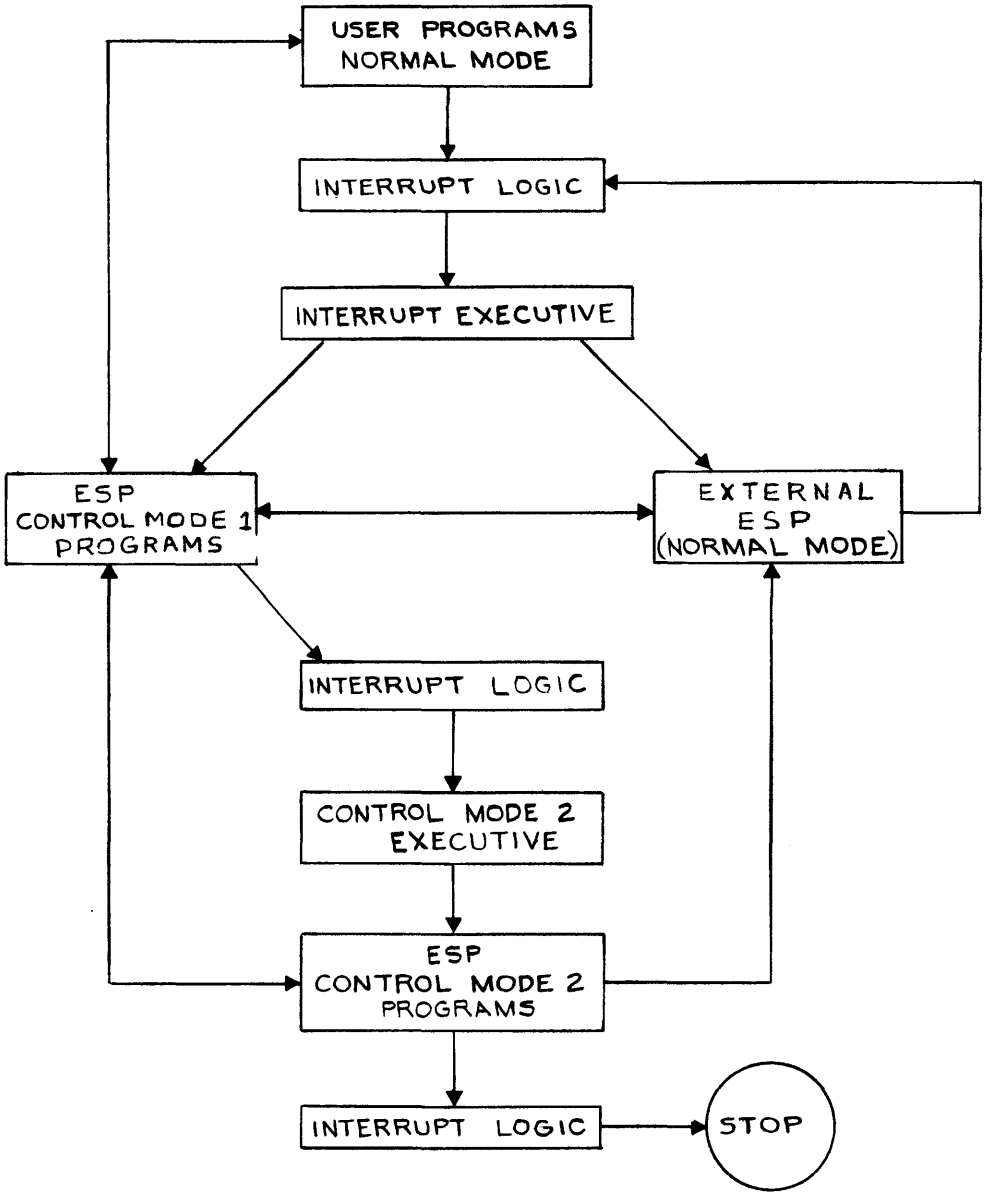


Figure 2-2. Interrupt Flow of Control

CHORE

The scheduling concepts implemented on the B8500 Computing System require that the system has information concerning the acquisition and processing of computer tasks. The conventional method of using control cards becomes inefficient, because most information about tasks can be prestored, and can be called automatically with or without parameters. Using this method, control cards are only needed to supply parameters to prestored tasks and to supply data regarding deviations from normal processing.

A run is defined as the execution of the combined outputs of one or more compilations. It is desirable to connect runs (which could have originated from different source languages) into a single process. It is also desirable to have decision making and parameter passing as allowable functions in the mechanism which combines runs.

A language has been developed in which a chain of runs is specified to the Operating System (ESP) so that the chain of runs can be scheduled and executed efficiently. Specifications in this language are called CHORE (CHain OF Runs Executive). The CHORE compiler accepts a source routine written in the CHORE language and produces a directive list that is used by the Operating System to schedule and control the described chain of runs.

A CHORE routine is compiled from source language and filed on a master CHORE file, along with many other CHORE routines. A CHORE routine is activated in one of three methods:

1. Automatically by the system based on a cycle specification in the CHORE file.

2. On command from an external device such as a card reader or teletype.

3. On call from a worker program or ESP.

In any of the above methods, the CHORE routine and associated parameters are merged to form an executable CHORE task which is introduced to ESP for scheduling, collection, and execution.

A CHORE routine is divided into three basic sections. The sections are: IDENTIFICATION, DATA, and PROCEDURE.

The Identification Section identifies the CHORE routine to the system. Information such as deadline, priority, and repetition cycle are defined to the system. Parameters required by the CHORE routine on a command call (worker or external device) or a cyclic call are also defined.

The Data Section provides a list of files required to initiate the CHORE routine. Included are all input files and program files used by the CHORE routine.

The Procedure Section contains the logic to be followed in executing the CHORE routine. The Procedure Section can be divided into segments to implement overlay. The Procedure Section is logically divided into run segments to provide the sequencing of control through the runs in the CHORE routine. Statements that control file input, output, and printing are provided, along with statements for the creation of files. Logical statements for transferring control, testing, communication with runs, and overlaying segments are also provided in this section.

An example of processing a CHORE task (a series of runs hereafter called CHORE), using all three functional programs of External ESP follows.

A CHORE is written and punched on cards in the CHORE source language. The source language cards are placed in a card reader and a button at the card reader is depressed, signaling the presence of work to be processed. The I/O PAC Routine of ESP, on detecting the signal, causes a card responder program to be activated by the Collector/Scheduler. After scanning the initial cards, the Responder calls on C/S requesting the activation of the CHORE which calls for execution of the CHORE compiler. The request from the card reader is taken as a command and the CHORE for the CHORE Compiler is entered in a table, from which it will be ultimately scheduled.

The example used above to illustrate the process of activating a CHORE from the Card Reader is the CHORE for the CHORE Compiler. The action taken is the same for any CHORE, and so as not to confuse the names, the CHORE Compiler is hereafter called Compiler C.

The CHORE for Compiler C is executed when convenient by the C/S program. The files required by Compiler C are acquired, disk space allocated, etc. The input card file (the source deck to be compiled) is associated with the file list for Compiler C. When all file requirements have been satisfied, the CHORE for Compiler C is entered in a list where collected CHORES await introduction by the C/S. When sufficient memory space and intermediate storage are available, the C/S calls on ESP to introduce the Interpreter/Controller, with the CHORE for Compiler C as a parameter.

When the I/C program is activated, it interprets the CHORE for Compiler C. The runs specified in the CHORE for Compiler C will be introduced by the I/C for processing by the ESP. Compiler C will be readied, and control passed to it. It will proceed to compile the input file passed to it and will produce (assuming correct syntax) as output:

1. A directive list representing the CHORE. This CHORE will be appended to the file containing all CHORES.
2. An entry to the cyclic schedule list, if the CHORE compiled is cyclic.

When the runs for Compiler C are completed, ESP initiates "end-of-run cleanup" (release the space occupied by Compiler C, etc.). ESP then returns control to the I/C which initiates any further action required for Compiler C (e.g., print output file). When I/C has completed processing the CHORE for Compiler C, it initiates "end-of-CHORE cleanup".

The CHORE compiled by Compiler C is now on the CHORE file and in the future can be called either automatically, by means of the cyclic schedule list, or on command from a card reader (just as Compiler C itself, was called.)

Collector/Scheduler

The functions of the Collector/Scheduler Program of the External ESP are as follows:

1. Automatically introduce and process repetitive work.
2. Accept and process dynamic requests for computer service.
3. Control the deposition of system resources to individual tasks.
4. Provide facilities for the acquisition of remotely stored files and for the connection of the obtained file with the appropriate requestor.
5. Act as a look-ahead for the System, providing a backlog of processable work, but still exercise control over what is being processed so as not to overload the System.
6. Exercise control under the general guidelines of priority, time of data arrival, deadlines, and current load.
7. Provide interaction with external environment.

The Collector/Scheduler functions are interrelated and it is difficult to separate one from the other. Collection is usually concerned with acquisition and connecting functions, whereas scheduling is generally concerned with control functions. What is to be collected must have been scheduled for collection, and what is to be scheduled must have been collected for scheduling.

The technique used for Collector/Scheduler is predicated on the characteristics put forth above. Collector and Scheduler are as the teeth on two meshed gears driving a shaft; each doing its job, setting up for the next one, and moving on out of the way. In the discussions of the Collector/Scheduler Program, the functions are treated as a single entity and referred to as C/S.

Interpreter/Controller

The Interpreter/Controller (I/C) is the program which dynamically interprets the directive list (CHORE) produced by the CHORE compiler. The routine is called by Collector/Scheduler to process a CHORE which has been partially collected (left to be obtained are memory space and intermediaries). The method of call is as follows:

1) The stack and HJT are allocated by Collector/Scheduler.

2) The PRT portion of the HJT is initialized to call I/C.

3) A call is made on the Internal ESP.

On return from the call, Collector/Scheduler has been disconnected from the new job just established and continues in an "across the board" fashion for other jobs. The ESP (Internal) completes the setup of the HJT, performs its own functions such as linking, allocation of sleep table, etc., and transfers control to the Start Program. The Start Program then accesses the PRT line in the HJT for the I/C, at which time the I/C is readied. When readied, control is transferred to the I/O.

INTERNAL ESP PROGRAMS

Internal ESP consists of the following functions:

Filing
Memory Allocation
Disk Allocation
Input/Output Processing

These functions can be referred to, collectively, as the Data Management System of the B8500.

Filing

STRUCTURE OF THE FILES

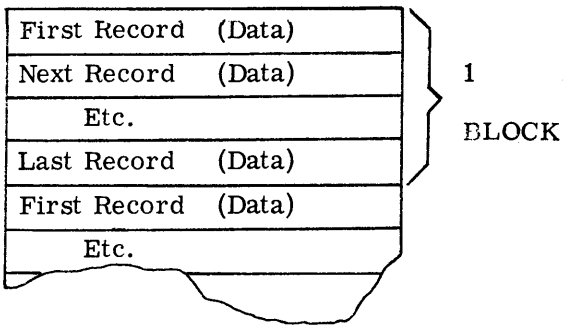
The types of files are individually described below.

SERIAL ACCESS FILES

A. Fixed length records.

This type of file contains fixed length records (all records contain the same number of B8500 words). The file description contained in the system directory entry is sufficient to allow serial access of the data, and includes:

1. Starting address of the file
2. Size of the records
3. Total size of the file



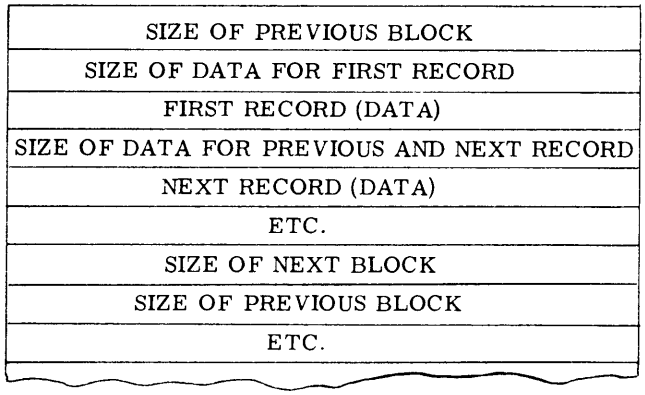
As can be seen above, the data portion of this file does not contain any System-inserted information, such as pointers or counters. Accessing records on this type of file requires one disk access per block. The blocking factor will be determined by the record size and economical read-write unit sizes, in multiples of eight B8500 words.

B. Variable length records.

This type of file contains records that are variable in size, but all records must contain an integer number of B8500 words. If the records are blocked, the blocks must contain an integer multiple of eight words. The system directory entry for this type of file contains sufficient information to describe the file, such as:

1. File starting address
2. Maximum record size.
3. File size.
4. Size of first block of data.
5. Size of last block of data.
6. Maximum block size.

The user data for this type of file is contained in blocks with a variable number of records per block. The blocks also contain entries to enable the filing system to step through the file one record at a time. The first block is accessed via the system directory by utilizing the file starting address and the size of the first block of data. Following blocks are accessed by maintaining a pointer to the current block address and increasing or decreasing the pointer (from information contained in the block) for the next block to be accessed. The blocks will be constructed as follows:



From this construction it is possible to access the file in either a forward or backward direction. As described above, accessing records on this type of file requires one disk access per block. Blocking factor will be determined by the record size and economical read-write unit sizes.

RANDOM ACCESS FILES

A. Fixed length record called by record number.

This type of file contains fixed length records. All records must contain an integer (modulo 8) of B8500 words. The file description contained in the system directory entry is sufficient to allow random access of the data such as:

1. File starting address
2. File size
3. Record size

The data portion of this file does not contain any system inserted pointers or counters. Access to any record on the file is accomplished by taking the desired record number and multiplying it by the record size. Add the product to the starting file address and then access the record at that address. Accessing any record on this type of file requires one disk access.

B. Fixed or variable length record called by block name and record name.

This type file contains records that may be variable in size but all records must contain an integer number of B8500 words.

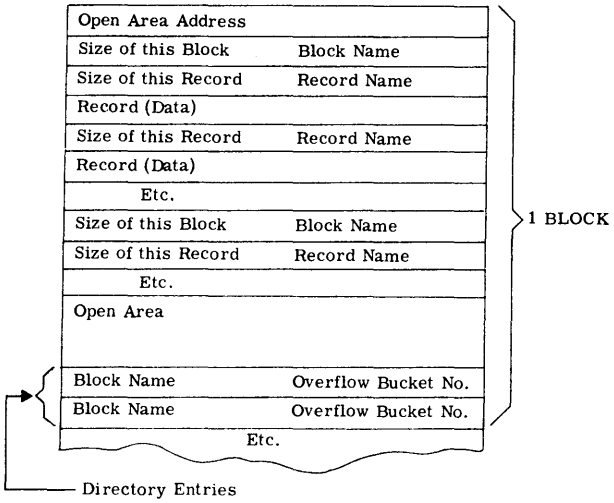
This file has a unique structure that permits, the majority of times, the retrieval of any record on the file with one disk access, and never more than two disk accesses will be required.

The disk area required for this type of file is prestructured by an I/O utility routine. This routine requires the following parameters:

1. Size of the file
2. Maximum size block
3. Average size block

From these parameters, the total disk space is calculated and allocation is requested. After allocation, the disk area is divided into equal-sized areas (hereafter called buckets). The size and number of these buckets is determined by the size of the records, the number of records, and economical read-write sizes. The buckets are then segregated into two groups. The first group is called the directory-and-data buckets, while the second group is called the overflow buckets.

The directory-and-data buckets will be formatted to contain the user's data in the following manner:



This type of bucket is used primarily for data, and until such time as the bucket is filled with data, no directory is necessary. When the bucket is filled with data and it is required to put more data in that bucket, an overflow bucket is used and the data is stored there. The block key and the identity of the overflow bucket is put in the directory-and-data bucket so that the entire block may be retrieved with two disk accesses. As more demand is placed on the directory-and-data bucket to store blocks of data, and there is no more open area in which to put the directory entry, blocks of data which were originally contained in the directory-and-data bucket will be moved to overflow buckets, permitting additional directory entries.

The system directory entry for this type of file will have the following type of entries to facilitate file access:

(1) Starting address of the data for the file

(2) Total size of the file

(3) Number of directory-and-data buckets

(4) Size of each bucket

(5) Address of overflow buckets

(6) Number of overflow buckets

(7) Map of available overflow data buckets

In servicing a request on this type of file, the system directory entry for the file is accessed. The block name along with the number of directory and data buckets from the system directory is used as an input parameter to an algorithm whose output is the bucket number where the block should be. This number is multiplied by the size of the buckets and the product is added to the starting address of the data to develop the address where the desired bucket is located on disk. An input-output routine is then activated to read the desired bucket. When the bucket is in Main Memory, it is scanned until the desired block name is located. If the desired block name is in the bucket, the block is scanned to locate the desired record. If the block name is not located in the data portion of the bucket, the directory section of the bucket is scanned to locate the block name. When the name is located, the overflow bucket number in the directory is used to access the bucket where the block is located (as described above).

VALIDATION OF FILE REQUESTS

Every request for file service references the System Directory to validate the acceptability of the request as follows:

(1) Requests to open a file cause a check of the following:

(a) Security indication; if present, further tests will be performed to see if this job should be granted access to this file.

(b) Lockout indication; if present, this request is deferred in a request-waiting table until the file is available.

Note: If the Open requests lockout, a scan is made of the file-in-use table to determine if the file is currently in use. If it is, the request is deferred, as above, until the file is available.

(c) The structure of the file is checked to see if the intended use will be allowed. If yes, the type of access is filled in from the request.

(2) Other types of requests are checked to ascertain that they are consistent with the type of access designated.

At this point the information in the system directory entry is held until further needed in processing the request.

INACTIVE FILE SERVICE

The inactive file service maintains the inactive file list and services requests from other parts of the filing system and ESP. The inactive file list contains the following type of information for every file known to the system:

(1) File name

(2) File size

(3) Frequency of use

(4) Security indicator

(5) Type of access

(6) Date of file creation

(7) Date of latest change

(8) File format description

(9) File retention period

(10) Reference to previous cycles

(11) Storage medium (with reel numbers and other pertinent data as required)

(12) Library locations

Maintaining the inactive file list is the function of a utility routine. When called, this routine purges all files according to their individual retention schedules, puts the scratched file reels on the list of available scratch tapes and directs the tape librarian to the library location of these tapes.

Servicing the rest of the system is accomplished by different parts of the inactive file control as enumerated below:

1. The original request (from either the CHORE or C/S routines of ESP) is to locate the requested file(s). The inactive file control locates those files carried on the inactive file list and passes information about them to the requestor. A request for file(s) not located on the inactive file list causes a message to be sent to the operator at the control station. File parameters may then be specified from the control station.

2. A subsequent request to the inactive file control causes a routine to find where the file(s) are located and, if the file(s) are contained on tape, prints a message to the tape librarian with the desired reel numbers and library locations. The librarian delivers the desired reels to the input tape operators. If the desired file(s) are not on the inactive file list, this routine makes skeleton entries for the file(s) from information previously input from the control station and requests that these files be made ready at the input station. The control station must then locate the file(s) and deliver them to the system.

3. The next request to the inactive file control is to make the originally requested file(s) active. The inactive file control will print mounting instructions to the input tape operators. It will also instruct the input operators to make all other designated files available to the system, i.e., cards, etc. As the input operators make the files available to the system, the inactive file control will verify that the input files are those requested, allocate sufficient disk space for each file, and activate I/O routines to transfer the file(s) from the external media to disk. Upon completing the transfer of all the requested input files to disk, the appropriate system directory entry is made for the file(s). The requestor (usually the C/S routine of ESP) is notified that the requested input files are now available to the system. The output file(s) requested will have sufficient disk space allocated and a system directory entry is made for each file. When this is completed, the requestor is notified that the requested output files are available for use.

4. Upon termination, the inactive file control is again called on to transfer the output files to their external storage media. This entails assigning output reel numbers and library locations, activating output routines to effect the physical transfer, and providing the output operators with the necessary labeling information. If the files are to be printed, the output scheduling is also handled by the inactive file control. When the output has been completed, both printed and an image of the file made on magnetic tape, an entry is made in the inactive file list and the system directory entry is removed from the system. The disk space that the file occupied is released for other use. The input files and the space they occupied on the disk are also released.

ACTIVE FILE SERVICE

The active file service part of the filing system services user requests for serial files, random access files called by file name, block name, record name, or random access files called by file name, record number. In order to service the various types of requests, the active file service utilizes a series of directories and tables. These directories and tables are used for locating specific blocks or records, and for control or validation of specific requests. The following paragraphs describe the internal functions of the active file service. By understanding the internal mechanisms, the source language programmer should be able to utilize the inherent flexibility of the data management system to solve his data handling problems.

SYSTEM DIRECTORY

The System Directory contains an entry for every file that is currently active in the system (i.e., a file is active when it is directly available to the System, such as being contained on disk or mounted on a tape unit). The individual entries for each file will contain the following information:

(1) File name.

(2) Type of access (How the file is currently being used, i.e., Random or Sequential).

(3) Structure of File.

a. Sequential file--fixed length record.

b. Sequential file--variable length record.

c. Random access--one record (fixed size) per block, called by record number.

d. Random access--one record (fixed or variable size) per block, called by block name and record name.

e. Random access--multiple records (fixed or variable size) in a block, called by block name and record name.

f. Randomly accessed block, with sequentially accessed record within a block--called by block name and NEXT record.

(4) Number of reads and writes (random access file only).

(5) Last date and time used.

(6) Security (yes/no indication).

(7) Lock out indication (file level).

(8) Address of the data for the file.

(9) Total size of the file.

(10) Record size.

a. Average record size

b. Maximum record size

(11) Size of area being used (size of the buckets and total used).

(12) Number of data buckets (random access only).

(13) Size of individual data buckets (random access only).

(14) Address of overflow data buckets (random access only).

(15) Number of overflow data buckets (random access only).

(16) Map of available overflow data buckets (random access only).

(17) User label indicator (serial access only).

(18) Retention schedule.

FILE-IN-USE TABLE

The file-in-use table contains an entry by CHORE for every file that is referenced by that CHORE which is currently active in the system. The individual entries will contain the following information:

(1) User Identification.

(2) File Name.

Successful attempts to open a file for a user will cause an entry to be made in this table. Attempts to lock out a file for a user will reference this table to ascertain that no other user is currently using the file. All other types of requests will reference this table to ascertain that the file has been opened for the requesting user.

REQUEST-WAITING TABLE

The request-waiting table contains an entry for every request that must be deferred because either the requested file or block is not available to fill the request. The individual entries will contain the following information:

(1) Either file identification or block identification.

(2) Requesting user identification.

(3) Current user identification.

All Open requests must provide lockout facilities for serial files requesting lockout and for serial access of random access files. This lockout can be made only if the file is not already locked out or, in the case of the random access file mentioned, only when there is no other current user of the file. If the request can not be filled, an entry is made in the request-waiting table. Whenever a user releases a file, this table is interrogated to see if anyone is waiting for this file. If there is someone waiting for it, the request is processed and the entry removed from the table.

All other requests that can not be processed because someone else is currently using the requested block are likewise defined in the request waiting table. When the block is made available, the request is processed and removed from the table.

Every time an entry is made in the request-waiting table, the table is scanned to see if the user that is currently being held up is, in turn, holding up the user that is causing the current holdup. This is known as a stalemate and appropriate action must be taken to back one off so that the other user can proceed.

BLOCK-IN-USE TABLE

The block-in-use table contains an entry for every block that is in high speed memory and is referenced by a user. The individual entries will contain the following types of information:

(1) User identification

(2) File identification

(3) Block identification

(4) Block location

(5) Block size

(6) Block lockout indication

When an access request is made, this table is scanned to see if the information is already in high speed memory. If an entry is found, the request is filled from the copy that is already present. Otherwise, further access for the desired data is required.

FILE RECONSTRUCT PHILOSOPHY

Files are handled in such a fashion as to make them readily reconstructable in case of a program or equipment failure. Dumps are taken at strategic points during the operation of the System, and audit trail tapes will monitor all changes to certain shared random access files.

Memory Allocation

As hardware and software techniques have advanced it has become possible for groups of program steps (segments) to be loaded into memory during or between the execution of other program steps. It is this dynamic capability that has allowed separate segments, or programs composed of several segments, to operate under the direction of a master program or executive. During the execution of a particular program, only the segments of the program that are actually being performed need be present in memory at any given time. The rest of the program need not be called into memory until it is to be performed, thus saving storage space. When a segment or a program has been completed a call is made on ESP to initiate an overlay, putting a newly required segment into memory. At the same time the segment areas no longer required are released for further assigment by ESP. These segment areas may contain groups of instructions or data. Since these segments are shared across programs (which may have been written in different source languages) a uniform program structure is required. How memory is allocated is explained in the following paragraphs.

The two basic functions of the Memory Allocation program are to obtain a block of available space in Main Memory in answer to a request and to monitor blocks of memory after they are relinquished. As areas in memory are assigned, the Main Memory becomes divided into blocks of memory in use and blocks of memory that are available for use. All blocks are linked in the order in which they lie in memory. In addition to this linkage, available blocks are linked by size, with the largest block linked around to the smallest block.

Allocation of space for a requestor is governed by the priority and class of the requestor, and the amount of space that has been committed previously to requestors of that class. The Allocation Routine first tries to allocate by scanning the available space map to find the smallest block that is large enough to fulfill the request. If a block of sufficient size cannot be found, control is transferred to the Overlay routine.

The Overlay routine searches for a used block of memory that can be reassigned to fulfill the new request. This is done by using the priority and class of the requestor. If a program segment is chosen, the related Program Reference Tables are updated to cause an interrupt on access by the callers of the segment. If a data segment is chosen, the data segment is saved in disk storage

before the block is reassigned. The PRT's of all users' programs referencing that segment are updated as appropriate.

If a request for space cannot be granted by these means, the request is deferred and put into an unallocated chain on a first-in-first-out-by-class basis. Periodically, this chain is scanned to allocate the deferred request. (There are four general classes of priority with sub-priorities within them. It is these sub-priorities that are on a first-in, first-out basis.)

As explained above, at any given time in the execution of a program, only the active segments require memory allocation. Large contiguous areas of memory are not required. Therefore, programs can be run with varying amounts of memory allocated. This is important in time-sharing applications where there are many concurrent users.

Disk Allocation

INTRODUCTION

The purpose of the disk allocation program is to allocate and release storage on disk modules in either logically or physically contiguous locations. The disk modules are used as the B8500 System's primary mass storage for input and output data files and for program files currently being referenced. Each disk module is logically divided into 20 units of 100,000 words each, for a total physical size of 2,000,000 words (a word being composed of eight 6-bit characters). Requests for disk storage are assigned space in a contiguous string.

GENERAL DESCRIPTION

The disk allocation program is composed of eight independent procedures, as follows:

1. Physical Allocation

Function: Allocates contiguous physical disk storage, and maintains tables of Available Disk Storage and Available Contiguous Areas.

2. Logical Allocation

Function: Allocates a range of logically contiguous locations and maintains a table which relates the logical to the physical disk address.

3. Logical Reallocation

Function: Assigns additional logical locations to a previous request, maintaining contiguity.

4. Release Physical Locations

Function: Places the released space in the Available Disk Storage Table, tests the table for newly created contiguous locations, and updates the Available Contiguous Areas Table.

5. Release Logical Locations

Function: Releases logical locations and their corresponding physical disk locations.

6. Allocate Empty Disk for Swapping

Function: Allocates a whole disk module for the purpose of moving data from one disk to another, and sets inhibit bits on both the sending and receiving disk modules to prevent allocation.

7. Allocate Back-up Disk

Function: Allows a user to specify the number of words of disk storage (maximum of one disk module) and the number of different disk modules required (maximum of three).

8. Remove Disk Module from System

Function: Tests the Assigned Storage Table to insure that the disk module is empty, and the inhibit bit is not set. If both conditions are true, the disk module is deleted from the Available Disk Storage Table.

A detailed description of each of the above procedures can be found in the reference manual on the ESP (Form BJ-4).

SUMMARY

Since the disk modules are assigned by means of a table in which logical units are related to physical addresses, a request for an area greater than one module requires physical allocation. If the request cannot be completely placed within one module then the Logical Procedure will fill the request by overflowing into another disk module. Any fraction remaining (less than 2,000,000 words) is assigned to another disk module, in the form of a contiguous chain of data.

This program also contains the facility to change allocation of one disk module to another disk module. When this option is selected, both disks are inhibited from being further allocated.

A special request to allocate an area on multiple disk modules is incorporated. This allows back-up of data.

When the disk modules are returned from file maintenance, an entry is placed in the Available Disk Storage Table. When a disk module is removed from the system, the Available Disk Storage Table is modified. If the disk module is not empty, an indicator is set.

A table of the assigned segments of disk is maintained in the Assigned Storage Table. A table of the physically contiguous areas (Available Contiguous Areas Table) is available and contains the total number of words available in the System, and the total number of empty disk modules.

A special request may place temporary restraints on selected disk addresses. Consequently, these disks will not be allocated unless all disk space is in use.

Input/Output Processing

INTRODUCTION

Input/Output(I/O) operations on the B8500 are initiated by the Central Processor but performed by the I/O Module. Thus, the Central Processor is free to continue at high speed, executing some programs while several other different programs may be awaiting the slower completion of I/O requests. This freeing of the Central Processor to permit greater time sharing across programs is the most important feature of the I/O Module.

Each I/O Module contains a separate processor, local memory unit, and communication capabilities. Because it provides channels for all data transfers between peripheral devices and the Main Memory, the I/O Module executes programs and controls the transfer of data from one peripheral device to another peripheral device independent of direct Central Processor control.

ESP'S I/O DRIVER ROUTINE (I/O PAC)

ESP makes four basic requests on the I/O PAC Routine:

1. Transfer one or more physical records (such as the contents of a punch card or a magnetic tape) between a peripheral device and System Memory.

2. Transfer one or more physical records between two peripheral devices.

3. Transfer data from or to one or two buffers located in System Memory to or from the disk files.

4. Process the Interrupt Stack.

In addition to these four basic requests, provisions for specialized requests are implemented as required. This includes such functions as scan column number one of input cards and interrupt Central Processor if a special character exists; otherwise, write the cards on disk. For each of these requests certain parameters must be established. Examples of some of the parameters inserted into the parameter area by ESP and the I/O PAC are:

Function

Device number

Addresses (Memory Module, Disk, Sleep Table Word, Buffers)

Number of Words read or transferred

Input and output devices

Operation

JOB STACK WORD ENTRIES

After the parameters have been established, a Job Stack Word entry must be inserted in the Job Stack Word Table (JSWT). There are five types of entries that can be made. Their functions and contents are listed below:

1. A NEW JOB entry causes the I/O processor to: (a) be assigned to the channel designated in the word; and (b) execute the program defined by the base registers contained in the word. The NEW JOB word contains complete state information for a channel.

2. A NEW PARAMETER BASE ADDRESS entry causes the I/O Processor to: (a) be assigned to the channel designated in the word; and (b) execute the program at the channel's current IBA, but with a new PBA.

3. A STOP entry causes the I/O Processor to stop operations on the designated channel. It accomplishes this by causing the active bit in the designated channel state word to be reset and the I/O Processor to be released. For a channel to use the I/O Processor the active bit must be set. The start line to the device for that channel is also reset, terminating the operation of the peripheral device.

4. A NEW JOB STACK ADDRESS entry causes the JSA to be loaded from the JSWT entry, and the I/O Processor to become available

to the channel with highest priority. The contents of this entry contains the new setting for the JSA register. By placing this type word in the last position of the current Job Stack, the I/O Pac defines the extent of the JSWT, and provides an address link to the next Job Stack area.

5. A NEW INTERRUPT STACK ADDRESS entry causes the Interrupt Stack control registers within the I/O Module (ISAR and ISLR) to be loaded, and the I/O Processor to be released. ISAR indicates the location, in Main Memory, of the I/O Module's Interrupt Stack, while ISLR indicates its limit (length).

CONTROL AND DATA FLOW DURING I/O PROCESSING

I/O processing can best be illustrated by an example. The various numbers in parentheses referenced in this example are shown on Figure 2-3.

During the processing of User Program-B an I/O operation is requested. A call (passing of program control) (1) is made on the ESP to perform the operation. ESP interprets this call and determines which peripheral devices and data areas are to be used. ESP loads into the Central Processor and Main Memory the programs that are required to perform this transfer, and then transfers parameters and control to the I/O Pac Routine (2). The I/O Pac uses the parameters to construct a new Job State Word containing a channel number, instruction base address, and parameter base address. The channel number specifies the device to be used in the I/O operation, the Instruction Base Address (IBA) specifies the I/O program that is to be executed, and the Parameter Base Address (PBA) defines the location of the parameter area (data area) which contains the parameters to be used with the I/O program. The new Job State Word (JSW) is placed at the bottom of the Job State Word Table (JSWT) (3). After a specific data transfer program has been initiated, the I/O processor scans the JSWT to select the next job to be initiated. The I/O processor, using the Job Stack Address Register (JSAR), scans the JSWT to select the JSW now appearing at the top of the table. The JSW corresponding to the JSAR is entered into the I/O processor and the I/O program, designated by the JSW, is activated to operate with the parameters specified (4). The program develops a descriptor that will control the transfer of data into, or out of, the buffer area specified by the parameters. The descriptor contains a byte count field, status field, and memory buffer address field. The descriptor is placed in a Local Memory location associated with the JSW channel number (5). As each byte is accepted from a peripheral device (6), the descriptor is removed from Local Memory, the byte count is decremented, and the descriptor is replaced (7). When the byte count reaches zero (no more data to transfer) the I/O program that initiated the data transfer on this channel is reactivated. The program will also be reactivated if the I/O Module detects an error, at which time all further data transfer is stopped (8).

The I/O program checks the status field of the descriptor to determine the reason for terminating data transfers on this channel. The status of the I/O operation on this channel is inserted into the parameter area, the Job State Word that initiated operation on this channel is inserted into the Interrupt Stack, and control is transferred to the I/O Pac (9). The I/O Pac interprets the information contained in the Interrupt Stack and the parameter area to determine the status of the terminated operation (10). If the status is "good", parameters and control are passed to the ESP (11). ESP determines which user program had requested the object data that is now available. That user program is reactivated on one of the Central Processors with pointers to the requested object (12).

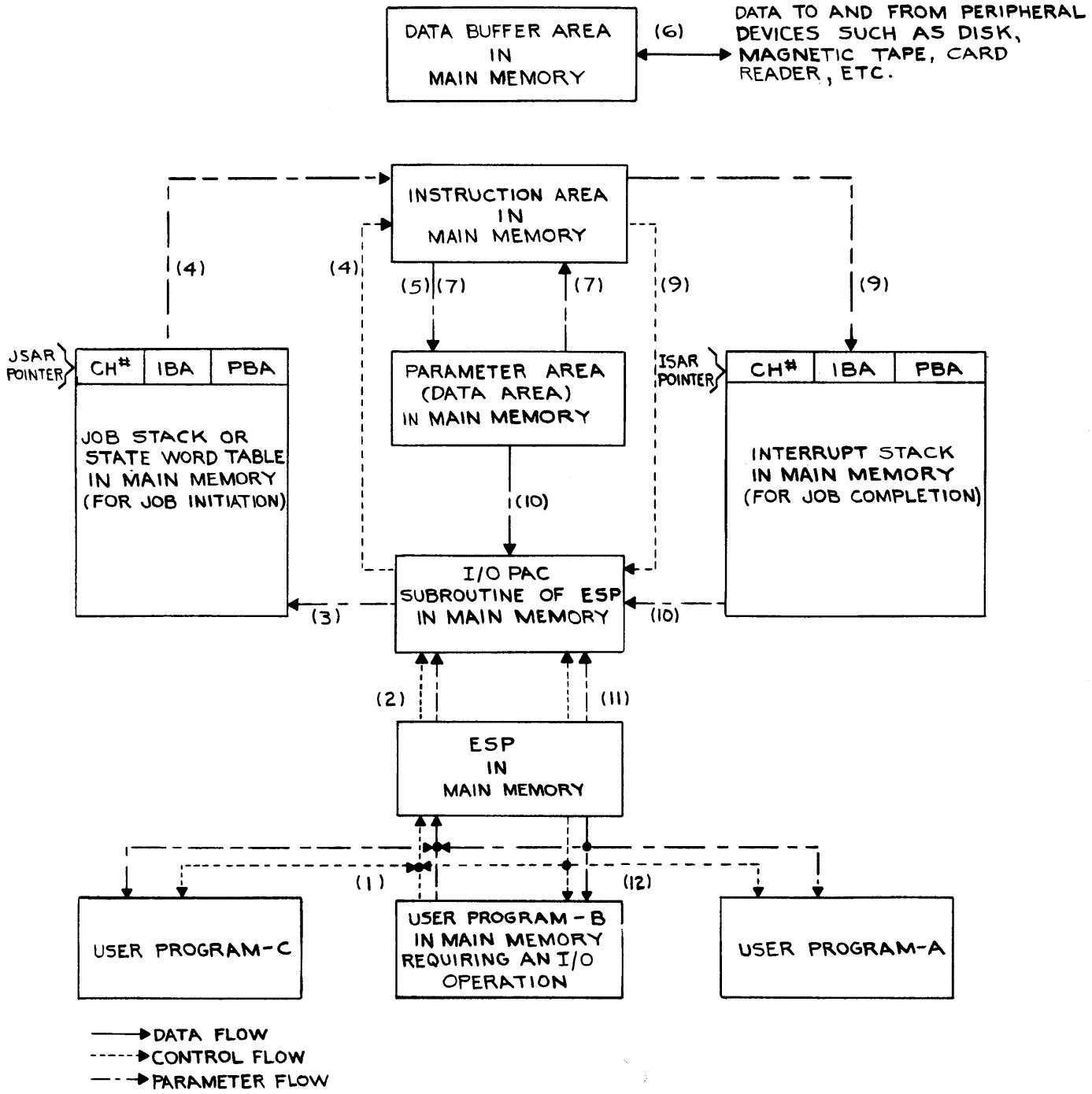


Figure 2-3. Example of I/O Processing

SOURCE LANGUAGE COMPILERS

System Compatibility

Uniform program structures allow subroutine intermixing among ALGOL, COBOL, and FORTRAN. Subroutines, each in a different language, can be nested in the calling program.

B8500 extensions to ALGOL permit full system versatility when interfaced with ESP.

TOOL Compiler

INTRODUCTION

TOOL (The Only lOgical Language) was created for the development of the system programs for the B8500 Data Processing System. The language is based on Extended ALGOL 60. TOOL implements much of Extended ALGOL with additional extensions for the Executive Scheduling Program (ESP) functions. Although TOOL is a problem oriented compiler language, it also provides for the necessary hardware manipulation that would otherwise be accomplished through assembler or machine language programming. The ESP and the compilers for ALGOL, FORTRAN, and COBOL are written in TOOL.

FEATURES OF TOOL

Through the use of the External Declaration, library or file procedures are available. Concatenate and Partial Word expressions facilitate word, character, and bit manipulation. For a field within a word that is to be used in an arithmetic or logical operation, the partial word designator gives the starting bit and the number of bits. The concatenate expression designates field starting bits in multiple words and the corresponding field bit lengths, thereby permitting word building.

ESP-Mode instructions provide a firm control over hardware functions. For the ESP applications, addressable processor registers can be specified and treated as simple variables. Declarations for special memory allocations and specification of absolute addresses are also provided for ESP application.

ALGOL Compiler

INTRODUCTION

The B8500 ALGOL compiler implements virtually all of ALGOL 60. In addition, extensions to the language have been provided to permit communication between the object program and the Executive Scheduling Program (ESP) to handle input and output operations, editing of data, and source language program debugging.

The B8500 ALGOL compiler consists of a number of associated program and data segments, not all of which need be in memory for compilation. The ALGOL compiler runs under control of the ESP, as a user program.

Inputs to the compiler are source language statements, the B8500 System library, and job parameters. Optional outputs which may be specified include: descriptive syntactic and consistency error messages; storage map of segmentation and variable locations; and a cross reference table of variable symbols and statements.

During compilation, major errors, such as any syntactic error, an undefined label, reference to an undeclared array, etc., inhibits code generation while continuing with syntactic analysis. A minor error, such as an assignment of a Boolean variable to a non-Boolean variable, etc., is annotated but does not cause the compilation to be aborted.

The complete B8500 ALGOL compiler occupies approximately 12, 000 words of mass storage exclusive of tables, stack areas, input-output buffers, etc. During compilation the additional mass storage required for source language, intermediate outputs, object code, and listings, is requested from the ESP.

COMPILATION SPEED AND SPECIFICATIONS

Card load time to mass storage is indeterminate as most of the transfer time is overlapped with other work in the multiprocessing environment. Compilation rate is a function of system load; at maximum, it exceeds 9, 000 card images per minute.

Extensive MONITOR and DUMP features are provided for run time diagnostics.

Three types of compilation may be specified: compile and go, compile for library, and compile for syntax check.

Fortran IV Compiler

INTRODUCTION

The B8500 FORTRAN IV compiler is based on A. S. A. FORTRAN IV. Extensions and exceptions reduce the restrictions of the basic language and provide communications between the object program and the Executive Scheduling Program (ESP).

The compiler will operate under control of the ESP and appear to the ESP as a user program. During compilation, linkage between programs and subroutines in all approved languages is provided; however, the called subroutine shall contain only one language. This does not prevent the compilation of nested subroutines, each in a different language.

Source language statements, the B8500 System Library, and job parameters are inputs to the FORTRAN compiler. At compile time the following optional outputs can be specified: lists of source statements, object code, used but undefined data and procedure names; error statements indicating that part of the source statement which is in error; a storage map showing the beginning location of each subroutine, the total storage used by the program, and the location of each symbol (variable) name.

STORAGE REQUIRMENTS

The compiler requires approximately 10,000 words of mass storage excluding the tables, input-output buffers, etc. At compile time additional mass storage for source statements, intermediate outputs, object code, and listings is requested from the ESP.

COMPILER SPEED AND SPECIFICATIONS

The actual compiler load time is overlapped with other computer functions. Compile speed is a function of storage allocation and the maximum rate exceeds 10,000 card images per minute.

For optional diagnostic programming aids, symbol tables will be available at run time for symbolic dumps, traces, etc.

FORTRAN developed object code will make effective use of the B8500 System's hardware features.

Communication between the various source languages used in the system is provided by the Uniform Program Structure and ESP.

COBOL Compiler

PHILOSOPHY OF APPROACH

Many of the concepts of COBOL are outdated because the B8500 employs advanced techniques in both hardware and software design. An example of this in the B8500 software is CHORE, which handles the assignment of I/O units (File-Control) and scheduling of runs. COBOL, of course, handles the function of File-Control through the ENVIRONMENT DIVISION, and makes no attempt to handle the scheduling function.

COBOL, itself, is undergoing change, based on the experience of users. A. S. A. 's COBOL Information Bulletin #7 clearly shows that many elements of the language considered REQUIRED today, will probably be OPTIONAL tomorrow.

It is in this changing world that the B8500 COBOL compiler must bridge the old with the new. It must be able to compile existing source language programs with minimum changes, and yet fit them into the new and advanced scheme of doing things. The guiding philosophy for the B8500 COBOL Compiler has been to accept existing syntax to the fullest extent possible, only rejecting it where it is in direct opposition or totally unrelated to the B8500 software and hardware functions. Upon acceptance of these language elements, the compiler will generate the necessary machine code to:

a. completely handle the element, or

b. partially handle the element, passing on the necessary parameters to other software packages to complete the function, or

c. take no action to handle the element.

The above approach will permit conversion of existing programs with the least disturbance. Long range, however, those elements found to be redundant or vacuous can be stripped from the source language programs, thereby increasing the compiler's efficiency.

FEATURES OF COBOL

The B8500 COBOL compiler implements COBOL as defined by the COBOL Edition 1965. All required and most elective portions of DOD "COBOL-61 Extended" are included. B8500 COBOL extensions and electives provide:

1. SORT verb-Use of the tournament replacement technique coupled with a forward/backward read on a balanced merge, using random access devices, produces extremely fast sort times. Implementation permits sorting on single or multiple keys in descending or ascending order. Input and output procedures allow user program intervention before record buffer content destruction.
2. CORRESPONDING option of MOVE, ADD and SUBTRACT verbs- for dynamic operation on group items.
3. ADVANCING option of WRITE verb - to vertically position printer.
4. Report Writer - fully implemented with grouping techniques and complete structural controls on report format.

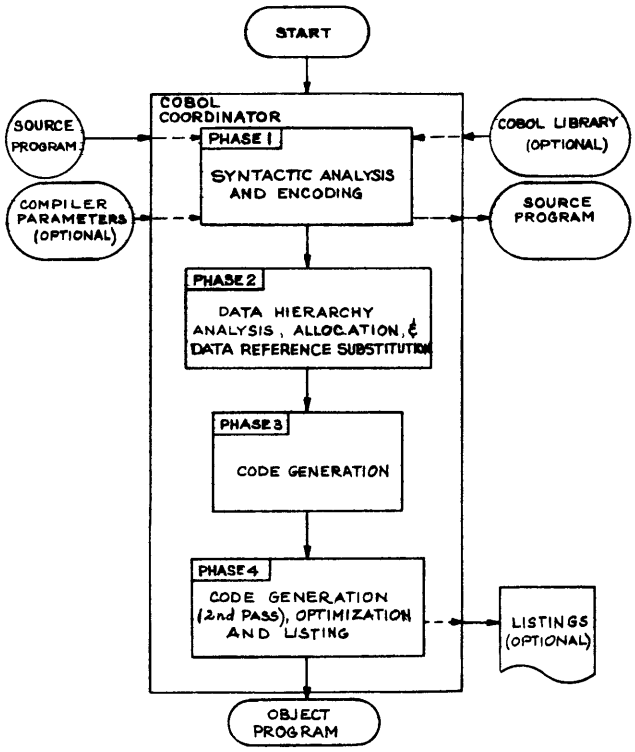


Figure 2-4. COBOL Compiler Structure

5. Table handling - including SEARCH and SET verbs for reference point establishment and location.

6. COPY and INCLUDE verbs for library facility - to economize programming effort through use of programs already written.

7. Full formula and relational operator sets: +, -, *, **, /, ≠, =, x, >, <, ≥, ≤, and their English language equivalents.

8. COMPUTE verb, with all arithmetic operations.

9. Mass Storage feature with asynchronous processing capability as provided in the DOD specification of 11/26/63. Extensions are implemented to interface with the filing system of the B8500.

10. Unlimited conditional, IF, nesting with automatic subject, relation, object and logical connector recursion.

11. Automatic sequence numbering of source code when omitted.

Continuing the B8500 modular concept, the COBOL compiler has been constructed in a highly segmented fashion for ease of maintenance (Figure 2-4). Four major segments provide the functions of: thorough syntactic analysis and encoding; data hierarchy analysis, allocation and data reference substitution; object code generation; and code optimization and listing.

Inputs to the compiler may be the source program, a COBOL library, and job parameters determining the system configuration and operating system interface. Specifiable outputs are: updated source and object programs; COBOL library, and listings including: data division memory map; cross reference listing; source program listing with syntactical, format and consistency error messages; and object program listing.

In the object program corresponding data and procedure segmentation contribute to minimal dynamic memory demands while allowing efficient object code execution. SECTION-names may be assigned priority numbers to mediate segment overlay procedures for optimal retention of high usage code. COBOL object programs may be made up of independently compiled segments and may call on subroutines written and compiled in ALGOL, COBOL, or FORTRAN.

Debugging is facilitated by extensive variations of the MONITOR verb which, when linked with the Executive Scheduling Program (ESP), will perform run time program diagnostics and provide the programmer with self-specified symbolic records of executed control points with dynamic value tracing.

Mass storage compile speed of B8500 COBOL exceeds 3000 card images per minute. Approximately 24k words of B8500 memory are required for compilation, the exact amount depending on the variable and program references. Several compilations may be executed from the same compiler copy in memory, under the direction of ESP.

Versatility of peripheral equipment is maximized through extended field sizes; alphanumeric literals, up to 132 characters, printable; numeric literals, 30 characters; numeric arithmetic fields, 18 characters; and display data, 2048 characters, minimum. Input-output operations are handled by passing control parameters to the ESP allowing device independent COBOL and multi-usage I/O routines.

Although the B8500 is a word machine, the data stream entering and leaving the system is, conceptually, character oriented for the COBOL programmer. Automatic handling by the compiler of the SYNCHRONIZE function, previously associated with word machines, makes it unnecessary for the programmer to account for word boundaries.

IDIOT RECORD FORMAT

Character Location	Field
1-132	EDITED-PRINT-LINE Complete character string already edited for printing.
133-134	TYPE-OF-LINE Code

Blank	Normal print line
00	L-B-P and L-A-P contain control information for the program, such as top and bottom margin spacing.
01	First body line of a page
02	Forms change record
03	Internal alignment line
04	Heading line(s). Will be retained and printed on top of every page.

135-137	LINES-BEFORE-PRINT Number of blank lines needed. However, with 00 in T-O-L, a "9" in location 135, followed by NN in 136 and 137 will indicate form length in sixth's; e.g., 00966 means a form 11 inches long.
138-140	LINES-AFTER-PRINT Number of blank lines needed. Normally, single spacing will always be given after a line is printed, indicated by blanks in TYPE-OF-LINE and LINES-AFTER-PRINT. Zeros in LINES-AFTER-PRINT, however, will suppress spacing, allowing "over-printing" by the following EDITED-PRINT-LINE.
141-147	REPORT-NUMBER The report number may be divided into as many categories as necessary to provide restarts from any given sub-report; e.g., Gross dollarization by Division (intermediate) within Plant (major).
148-152	PAGE-NUMBER A method to number pages can be furnished by the source language programmer, and if furnished, should restart on each change in REPORT-NUMBER. It is believed, however, that the task of keeping track of the number of lines on a page as well as page numbering should be a function of the IDIOT printing routine. Therefore, in the absence of any value in this field, page numbering will be furnished.

UTILITY PROGRAMS

Some of the utility routines that are provided are represented on the following pages. The list is not inclusive, but merely representative. The number of service programs available to the User will grow with the B8500 System.

IDIOT (IDeal Interface OuT) Print Program

This program offers the following advantages:

1. Automatic page numbering.

2. Source programs preparing printed reports will be oriented to forms movement rather than to specific hardware features. Programmers will not be required to keep track of where they are on a page. Therefore, programmers need not be concerned with the hardware, but furnish only the standard format outlined on page 2-26.

Program Functions of IDIOT

1. Gather, retain, and correlate for the individual report such information as top and bottom of page margin spacing, form size (lines per page), page numbering, and line count per report. Absence of the necessary parameters in a print file will cause IDIOT to apply established standards; e.g., a form size of 66 lines (11"), 5 blank lines as top margin, no header lines, 5 blank lines as bottom margin; and page numbering, starting with 00001 upon each change in Report-Number, and being printed in the center of the bottom page margin.

2. It is anticipated that a standard carriage control tape will be provided for each length of form and will need to be changed only when a form of different length is required. Channel one on the tape will always be punched for the first horizontal line immediately below the perforation at the top of the page (not necessarily the first printing line). The remaining channels will be punched starting with channel two on line 6 (6 lines to the inch), and repeating every 6 lines through Channel 11 at line 60. Channel 12 (overflow) will never be punched nor used. Line spacing, therefore, will be accomplished by Channel Skipping plus spacing the remainder through single or double spaces.

3. Restarts, through console interrupts, will be facilitated as follows:

   IDIOT will print out, upon any termination, the report and page or line number. Using these as basic references, the operator can:

   a. Print from and through any given report and page or line number. A special case of this general restart capability is the recovery from forms movement failure, when the operator can perform a "two page backup" and resfart, thus assuring at least one full page of overlap.

   b. A second special case of the general restart capability is the file passing option (to selectively sample print lines).

   c. Will provide for prematurely terminating a given print run, via console interrupt, and displaying a report number and page or line number from which to restart.

4. Will print the internal alignment line(s), and repeat their printing until operator, through console interrupt, terminates this function.

Source Program Maintenance

DESCRIPTION

Starting with a source program library of either COBOL, FORTRAN, or ALGOL apply changes, delete or add new programs, as directed by the control information. Output of this run consists of a file for subsequent input to a compiler for completion of any program(s) changed or added, and a listing of each. Since these facilities will be used for both maintenance of production programs and during new program testing, an updated library reflecting the changes will be produced on an optional basis. Associated with the new library will be a control listing reflecting all existing elements and the date of last change.

NOTE: Card columns 73-80 should contain identification for programmer's use, i.e., sequence number, date of change, or coding of changes.

OPTIONS

1. Request that the name only of a library program be changed.
2. Produce a new copy with changes under a new title from an existing library program.
3. Merge with selection, two or more source libraries.
4. Specify FROM and TO program names of those to be acted upon.

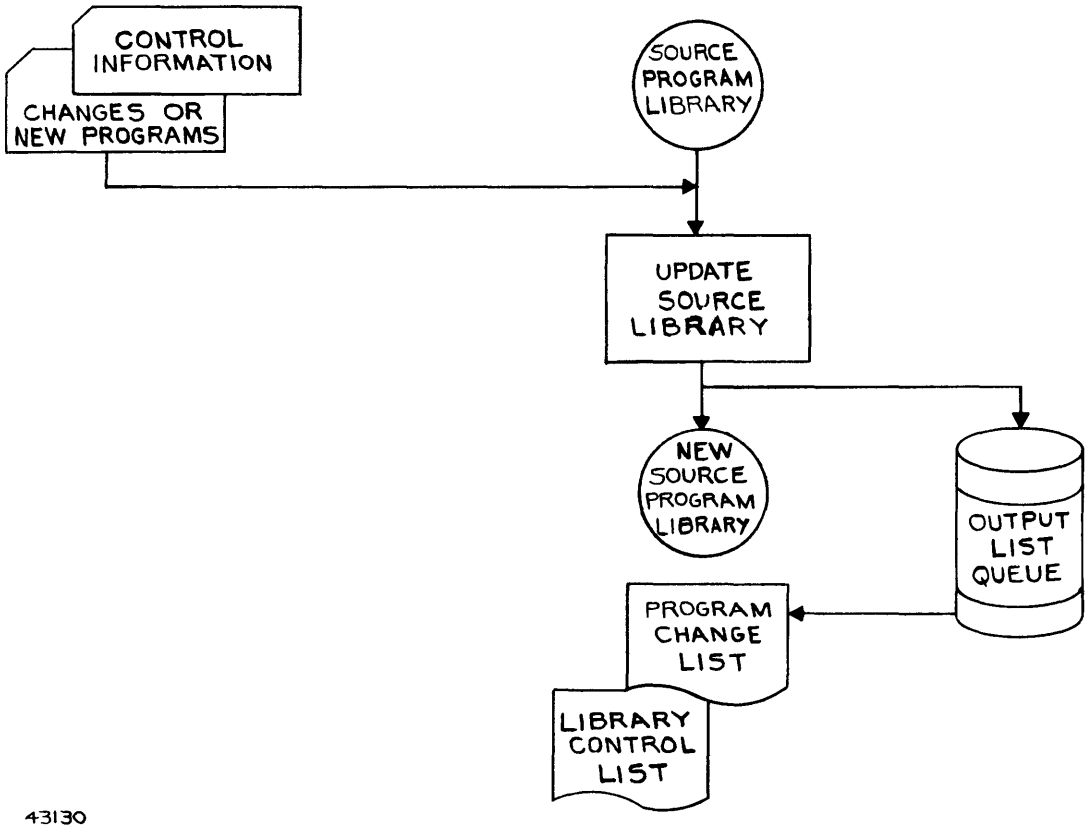


Figure 2-5. Source Program Maintenance

Generalized Sort/Merge

DESCRIPTION

In addition to the facilities provided by the COBOL compiler for sequencing or merging files, this routine will accomplish the same results based upon control parameters supplied in the job stream at object program run time. In this manner a single version of this program may be called upon to process any data files of the system with variable input/output information such as file identification, record type, record size, and control fields specified for a given execution only.

If it is desired to accomplish user own-coding during the input or output of the sort/merge, the COBOL facilities for generating a specific run should be used.

OPTIONS

1. According to control specification handle both multi-reel files and multi-file input.
2. Select specific files for input from multi-file reels.

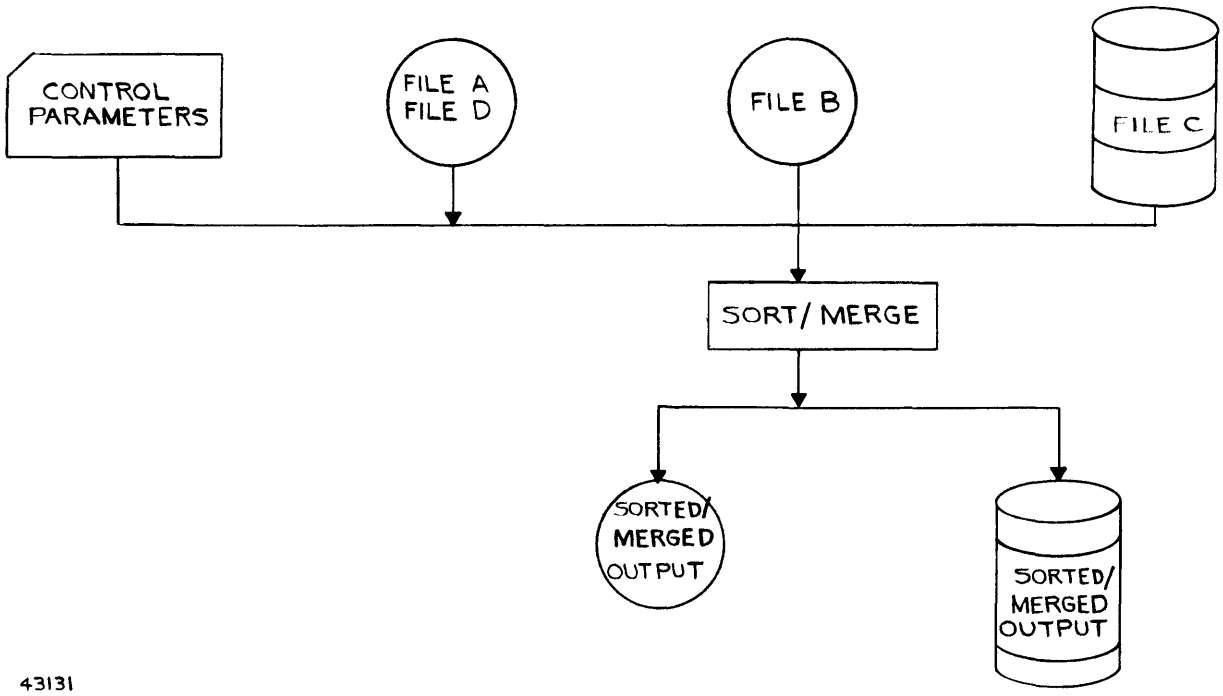


Figure 2-6. Generalized Sort/Merge

Data File Copy

DESCRIPTION

Copy either a permanent or temporary catalogued file with ability to interchange resident devices in the process. Basically, this function provides an exact duplicate of the original file. The following options are to be included.

OPTIONS

1. Handle multi-reel files and multi-file reels both in and out according to programmer specification.
2. Change density.
3. Change blocking factor.
4. Duplicate only a portion of the file.
5. Duplicate only selected records of the file on a key.
6. Duplicate with specified field change.
7. Duplicate original file but change catalogued file identification.
8. Provide a print listing of new file.
9. Record unreadable records with special character (?) insertion on an independent file for subsequent display and re-entry thru the merge function (see Generalized Sort/ Merge).

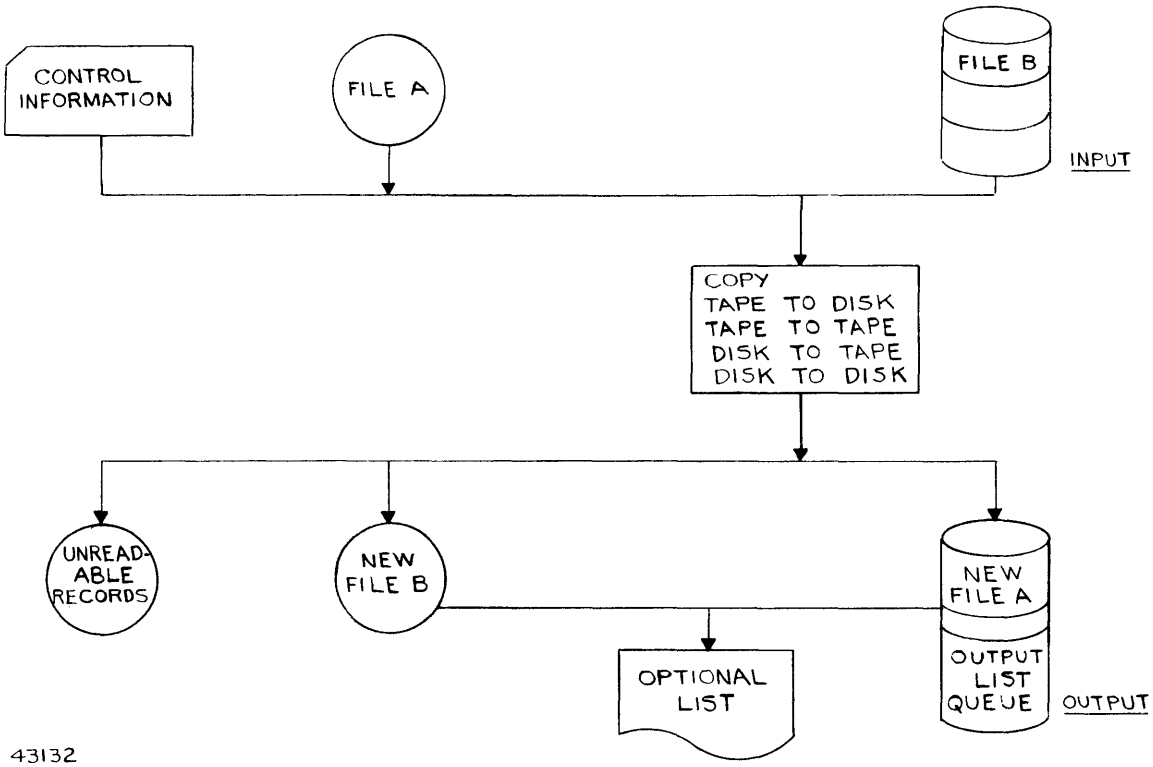


Figure 2-7. Data File Copy

Data File Compare

DESCRIPTION

Compare two independent data files and produce an output discrepancy file in standard print format for any unmatched items. Each item to be listed will be noted with record and block number. For any items in disagreement both records will be outputed for listing, indicating specifically the fields that do not compare.

OPTIONS

1. Specify starting point of either input file to begin comparison. This can be either block and record number, or a specific symbolic key.
2. Handle multi-reel files or multi-file reels according to control information.
3. Specify only portions of a record to be compared.
4. Specify a sequence key to compare only matched records for handling out of phase files with sequence check on both files.
5. Provide the ability to communicate with CHORE in case the results of the comparison may affect subsequent processing.
6. Specify fields of the records that are in computational format and need be converted to display format for discrepancy printing.

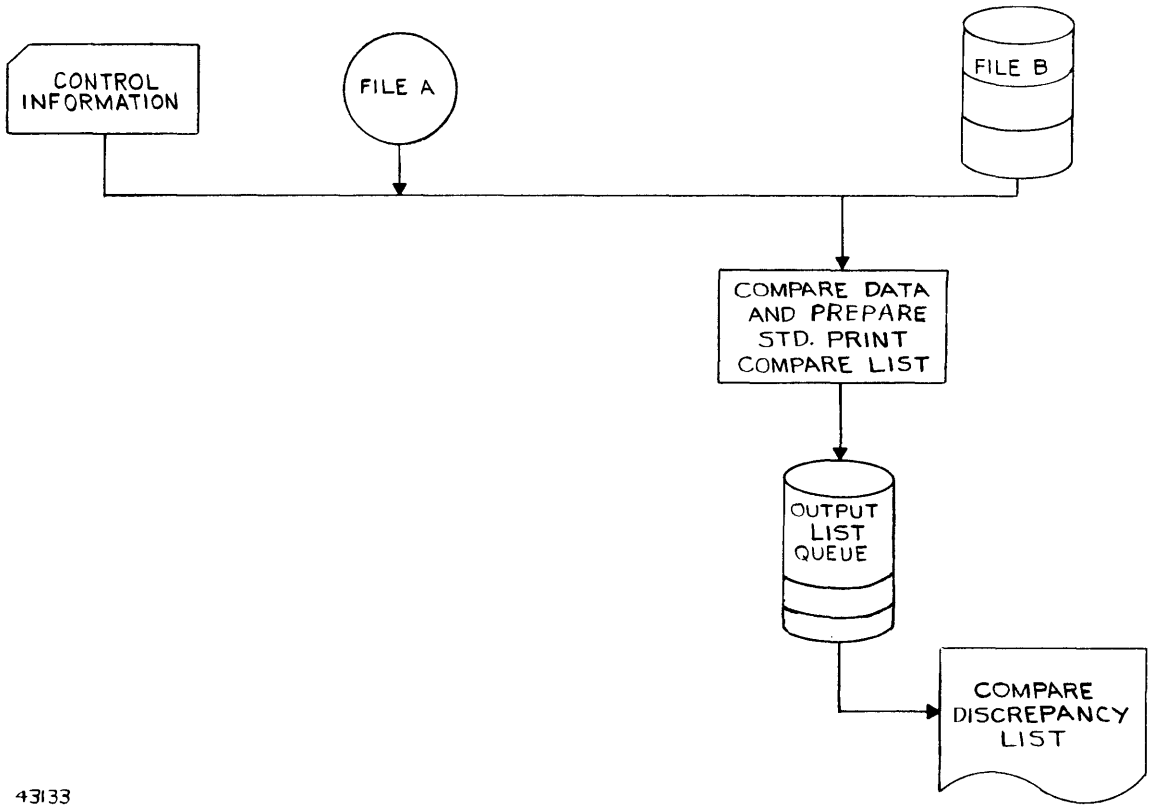


Figure 2-8. Data File Compare

Data File Print

DESCRIPTION

Produce a formatted record from a specified data file(s) to be queued on a disk buffer for subsequent concurrent printing. The output record will conform to the standard specifications for printing a uniform (120 or 132 characters per line) listing. Print-out will reflect appropriate record and block number. Each new record will start on a new line. Each file listing will be identified by file ID and CHORE title.

OPTIONS

1. Specify the desired starting point of the file to be printed according to block name and/or record number.
2. Specify the number of records or blocks to be printed.
3. Specify records to be printed in increments only, or on selected keys for sampling of files.
4. Specify spacing on output list.
5. Request that backup tape copy of the output listing be provided.
6. Specify whether computational fields are to be converted to display for printing or dump of total actual values must be provided.

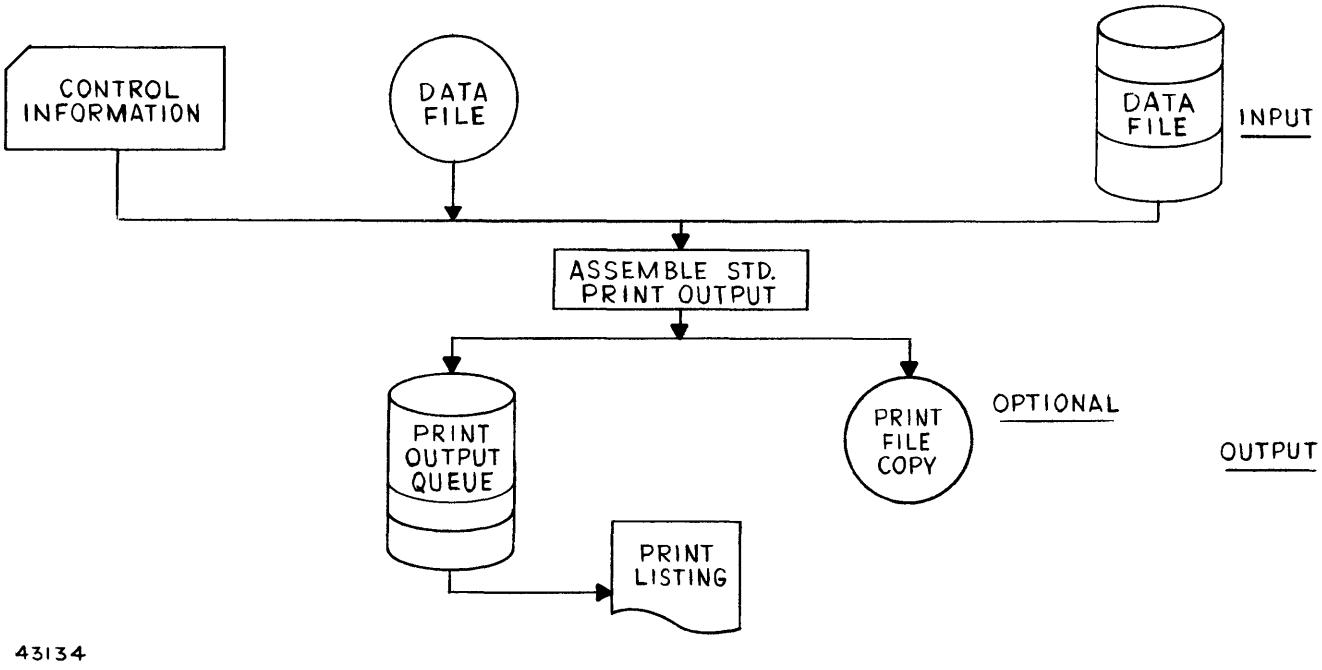


Figure 2-9. Data File Input

CHAPTER 3

EQUIPMENT SPECIFICATIONS

GENERAL DESCRIPTION

The equipment used in the B8500 Data Processing System is divided into two groups; the central data processing system and the peripheral equipment systems and devices. This chapter describes the general physical, functional, and operational characteristics of the modules and devices comprising an overall B8500 System.

Central Data Processing System

The Central Data Processing system comprises one or more B8501 Central Processors, B8505 Memory Modules, and B8510 I/O Modules, the necessary number of B8515 Controller and Communications Modules, and a B8520 Console. Except for the B8520 Console, the modules comprising the Central Data Processing System are housed in standard cabinets which are identical in size and appearance. (See Figure 3-1.) Each cabinet is approximately 83 inches high, 39 inches deep, and 34 inches wide. The weight and internal component configuration of each cabinet are determined by the type of module.

The standard cabinets, which are constructed of steel frames and panels, contain one or more hinged card rack assemblies, a bulkhead mounted power supply, a maintenance panel, a forced-air cooling system, and provisions for cable interconnections. Internal logic, control circuitry, and memory facilities are determined by the functions of the module housed in the cabinet. Interconnecting cables are routed through the bottom decks of the cabinets. All switch interlock cables are routed through the sides of the cabinets in order to minimize the lengths of cables in the interlock system.

Solid-state components are used exclusively in all module circuits; monolithic integrated circuit chips are used to the greatest extent possible. Semiconductor integrated circuits are complete electronic circuits fabricated within monolithic bars of material, using diffusion techniques, and are used to form transistor, diode, resistor, and capacitor elements. The advantages of using integrated circuit modules include lower cost, improved system reliability, and reduced size, weight, and power consumption. Discrete solid-state parts are used primarily in power supplies and in memory, logic, and control circuits. All circuits are packaged on plug-in, printed-circuit boards which are installed in card rack assemblies.

The card rack assemblies are integrated assemblies complete with card connectors, card guides, backplane wiring, power busses, and fans. Card rack assemblies are independently hinged on each cabinet and swing outward to provide easy access to the components inside the cabinets.

The interior of a cabinet is cooled by fans which draw air through filters in the bottom of the cabinet. The air filters can be removed, cleaned and re-treated, and then replaced in the cabinets. The air is exhausted through louvered openings in the top of the cabinet.

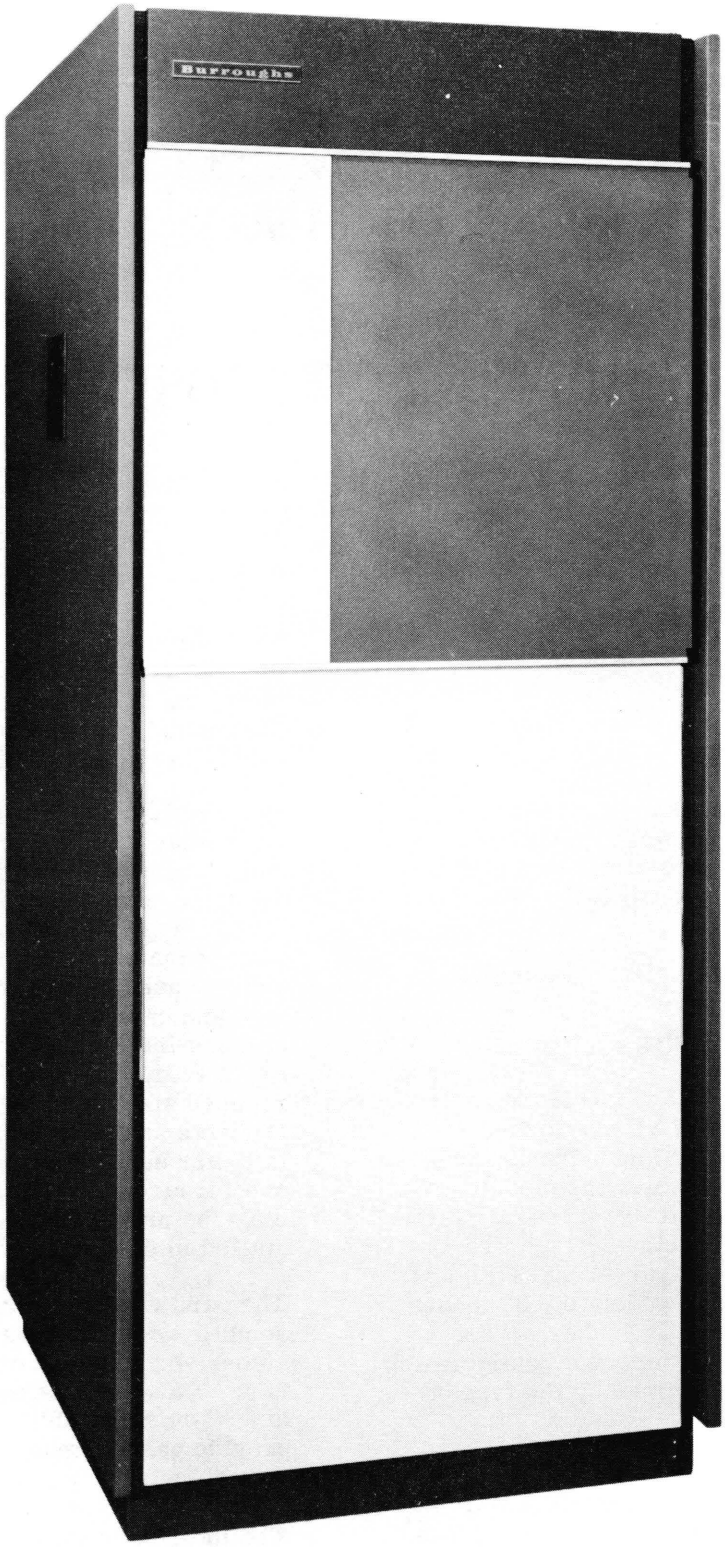


Figure 3-1. Module Cabinet

Peripheral Equipment Systems and Devices

The following common types of peripheral equipment systems and devices are used in conjunction with the central data processing system:

Disk File System

Magnetic Tape System

Card Readers

Card Punches

Line Printers

Other devices, depending upon the customer's requirements, can be used with the B8500. The diverse number of input-output devices available plus the engineering know-how of the Burroughs Corporation make such a tailor-made system not only feasible, but also economically available. The character set used in the B8500 System and associated with the devices is given in Appendix A of this manual.

The above named systems and devices are described under their respective headings in this chapter. Other devices, such as Teletype units, displays, plotters, communication links, etc., which may also be used with the B8500 System, are not described in this manual.

B8501 CENTRAL PROCESSOR MODULE

General Description

The Central Processor Module performs the computations in the B8500 Data Processing System. The characteristics of the module are summarized in Table 3-1. The primary functions of the processor module are to execute arithmetic calculations, control functions, data transfer operations, and interrupt services. The processor module is packaged in a single cabinet which also contains local ultra-fast memories, an associative memory, memory control and logic circuits, processor logic circuits, a power supply, a maintenance panel, and a forced-air cooling system.

Functional Description

The Central Processor Module consists of three stations: the communications unit (COMM), the advance station (ADVAST), and the final station (FINST). In general the interfacing, fetching, and storing functions are accomplished by COMM, the address arithmetic is performed at ADVAST, and the operations usually associated with data manipulation are performed at FINST. A simplified drawing showing the principal paths for the flow of data and control information between the three stations, together with a listing of the functions, is given in Figure 3-2.

The separation of the Central Processor Module into the three functional stations provides the distinct advantage of simultaneous operations within the Central Processor. Resulting multiprocessing and look-ahead techniques are thus achieved which improve data processing speeds to a dramatic degree. Independent operations within each of the three functional areas is a criterion of the basic design and results in extensive concurrent operations. An example of the simultaneity possible is the execution of an instruction by the final station, the initialization of a subsequent instruction in the advance station, and the acquisition of data by the communications unit from a memory module. The functions of each of the three stations are described below in greater detail, with the aid of the Central Processor Block Diagram, given in Figure 3-3.

TABLE 3-1

CENTRAL PROCESSOR MODULE CHARACTERISTICS

Arithmetic Stack

52-bit word:

48 bits for data
3 bits for control
1 bit for parity check

20-Megacycle Internal Clocks

Unlimited Indirect Addressing

Relative Addressing

Hardware-aided Jump procedures to allow Nested and Recursive Subroutines

Local high-speed and Associative Memories:

Name	No. of Words	Word size in bits
Instruction Look Ahead	12	52
Associative Address List	28	18
Index and Descriptor Queue	24	70
Storage Queue	4	52
Final Operator Queue	4	6
Temporary Operand Queue	4	52
Stack	12	52

Binary Arithmetic:

200 nanosecond Integer Add (single precision)
1.0 microsecond multiplication (single precision)

Variable Syllable Instructions

Classes of Instructions:

Floating point arithmetic
Full set of Boolean operations
Full masking availability
Variable length data manipulation
Character handling operations

Polish String Notation

Full Interrupt Capabilities

Extensive Indexing Capabilities

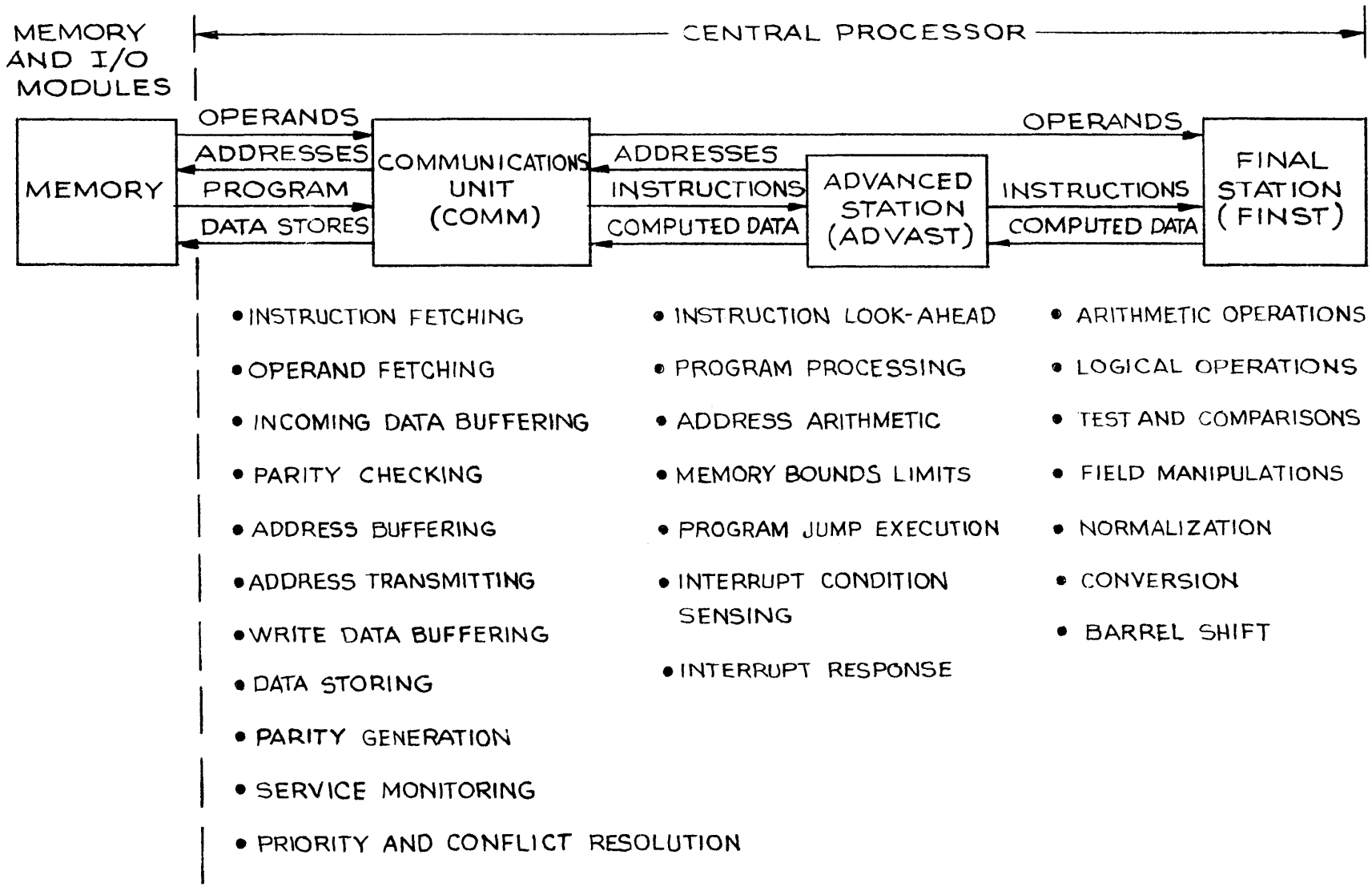


Figure 3-2. Functions of Central Processor Stations

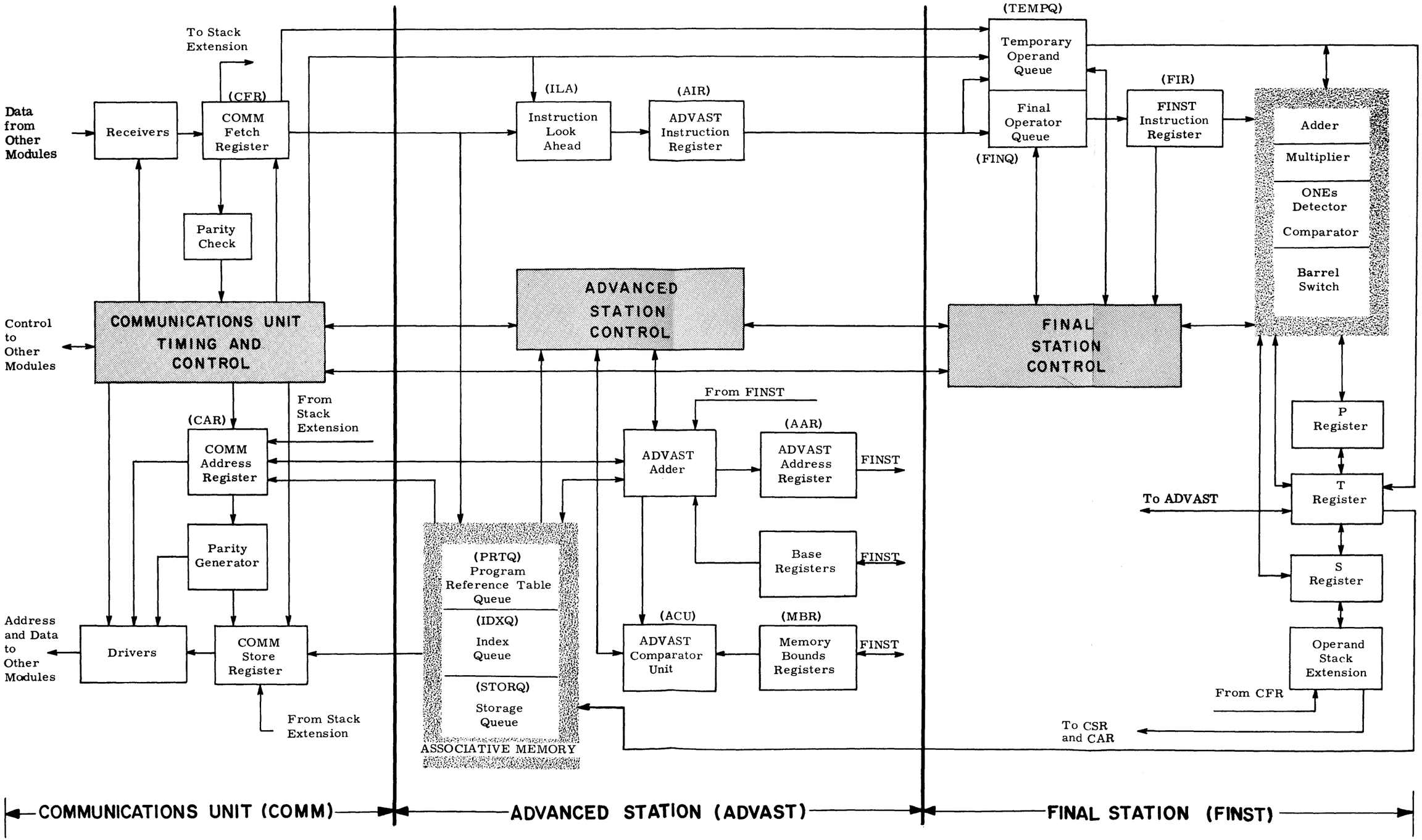


Figure 3-3. Central Processor Module, Block Diagram

COMMUNICATIONS UNIT (COMM)

The communications unit coordinates the transfer of information between the Central Processor Module and the main memory. Signal driving power and buffering are provided for, respectively, by the Drivers and Receivers.

Interface Characteristics

The Central Processor module interfaces with the main memory and I/O modules by means of the communications unit. Specifically the elements of COMM interfacing with the other modules are the Receivers, Drivers and the Communications Timing and Control Unit (Refer to figure 3-4). The function of each of these interface elements follows:

RECEIVERS-Accept and standardize incoming data from the Memory and I/O modules.

DRIVERS-Supply power to drive data going from the Central Processor to the Memory and I/O modules.

TIMING AND CONTROL-Directs sequential operations required by the communications process, including service monitoring, and priority and conflict resolution.

COMM constantly monitors the internal operations within the Central Processor Unit by serving as the sensor for units within the Advanced Station (ADVAST), and Final Station (FINST) areas. As memory accesses are required by these units, COMM provides the necessary interconnection between the Central Processor and the desired memory module.

Description of Logical Operation

A typical operation has its origin as a read operation from a selected memory. The instruction arrives at the Central Processor via the Receivers and is transmitted to the Communications Fetch Register (CFR). A parity check is performed on the incoming data and if the correct parity (odd) is sensed, the COMM Unit Control will enable data flow to one of four possible units depending upon the type of information which was received from memory. The four units within the Central Processor are:

1. The Stack Extension - a twelve word (52 bit word) local memory.
2. The Temporary Queue (TEMQ) - four data storage locations within the FINST.
3. The Instruction Look Ahead (ILA) - a twelve word local memory within ADVAST.
4. The Associative Memory-a 28 word local memory within ADVAST.

The Associative Memory is further sub-divided into three sections, the Storage Queue (STORQ), the Index Queue (IDXQ) and the Program Reference Table Queue (PRTQ).
The selection of the particular unit within the Central Processor which will receive the data from COMM is determined by COMM Unit Control.

Memory module linkages to the four units within the Central Processor Unit are grouped into two functional classes: "need" (automatic) and "demand" (Programmer control). The "need" linkage is a hardware implemented function which is not directly under program control. The "demand" linkage is directly controlled by Instructions, i. e., Fetch Memory to Stack (FMS), or Store Stack to Memory (SSM).
If the requested address involves address computation which results in a memory reference, the address is checked against memory bounds in Advast Comparator Unit. Any violation will cause an interrupt. The address is also presented to the associative memory to determine if the word requested is stored locally in STORQ, IDXQ, or PRTQ. If the request word is not in local memory, the address is passed on to the COMM address register (CAR), together with control information telling COMM where to place the contents of this address when it arrives from memory. The index queue (IDXQ) and the program reference table queue (PRTQ) are serviced by COMM on both a "demand" and a "need" basis. Any fetch reference made to Main Memory by the Central Processor that is relative to the base index register or the program reference table is placed in the associative memory by COMM.

The storage queue (STORQ) in ADVAST is continuously monitored by COMM and serviced on a "need" basis. The STORQ requires "store only" service, and COMM stores data from STORQ to Main Memory to keep STORQ available for use by FINST.

The PRTQ is the local storage for the most recent PRT-relative references to Main Memory. PRT provides a local storage for control words used in program jumps and words containing alternate memory address bounds. The execution of all program jumps is controlled by ADVAST. The initializing of the jump control register is accomplished by ADVAST, as is the distribution of the jump control word and the formation of the return control word.
COMM monitors the Stack Extension and controls the execution of store and fetch operations in order to maintain a certain predetermined number of operands for use by FINST. COMM monitors ILA similarly to the way in which it monitors the Stack Extension, but differs in that there is only one-way service required, i.e., information flowing only to the ILA. COMM provides store only service to the STORQ by transferring data to the memory module periodically, thus keeping the STORQ available for use by FINST. COMM provides input data to the TEMQ unit, upon the direction of the ADVAST unit, which in turn, initiates all requests for service. IDXQ and PRTQ units receive input data under control of the COMM unit. IDXQ contains all words that are referenced by any of the index instructions. PRTQ contains the most recent references to program segments and procedures that have been utilized during the execution of the program. If the IDXQ/PRTQ is full at the time, COMM will remove the oldest piece of data in the Queue and return it to Main Memory, or destroy it. Output flow from the COMM unit is via the Communications Address Register (CAR) and the Communications Store Register (CSR). The CAR contains address and control information while the CSR contains data. Prior to transmission of data and address information, parity is provided to each from the Parity Generator.

ADVANCED STATION (ADVAST)

All instructions executed in the Central Processor are handled by ADVAST. Those instructions which are strictly FINST operations are simply decoded by ADVAST and then transferred to FINST. ADVAST also senses all interrupt conditions, responds to specific interrupt situations, and controls the preliminary interrupt processing sequences.

ADVAST is the program processing portion of the Central Processor. All ADVAST operations begin at the Instruction Look Ahead (ILA). ILA is a local memory unit used for buffering instruction words of the current program much in advance of their use. The capacity of ILA is 12 words with 52 bits per word. Since the longest instruction contains four 6-bit syllables, the minimum buffering available is for 24 instructions. With this amount of "look ahead", COMM keeps the ILA sufficiently ahead of actual ADVAST computations to effectively "mask" the time taken in fetching program words.
COMM monitors ILA and services it on a "need" basis, automatically executing fetch operations to maintain a predetermined number of instructions in ILA. Instructions are taken from ILA in sequence and placed in the ADVAST instruction register (AIR). AIR holds the current OPCODE syllable and associated variant and/or address syllables. The operation and variant syllables are decoded by ADVAST control to determine what operations are to be performed, if any, by ADVAST. If no further ADVAST operations are required, the instruction is transferred to FINQ and TEMPQ in the Final Station, where operator and operand processing is completed.

The combination of OPCODE and variant dedetermines if address arithmetic is to be performed by ADVAST and, if so, which base register is to be applied, and what limits are to be employed in the memory bounds check in the ADVAST comparator unit (ACU). If the requested address is stored locally (in ADVAST) and does not require any action by COMM, the associative memory automatically cycles the local queue (PRTQ, IDXQ, or STORQ) containing the desired word, causing this word to appear at the output. If the requested word is to be used by ADVAST, it is available for computation at the queue output. If it is intended for FINST, ADVAST transfers the word to TEMPQ, which is the Final Station's local operand queue. An arithmetically derived address which is not found in local ADVAST memory must be fetched from a Memory Module via the COMM unit. Once COMM is signalled to fetch a word intended for the final station, ADVAST does not have to wait for the fetch to be completed. Instead, ADVAST is free to move onto the next instruction.

Address arithmetic involves the ADVAST adder which has three inputs - the address syllable, a base register, and an index amount - thus enabling one pass addition of the address syllables in the instruction string. Indexing is applied to address computation by means of the ADVAST address register (AAR). The AAR is the accumulator for indexing arithmetic. The local queue may contain up to 24 index words, thus enabling most indexing to be accomplished without reference to Main Memory.

FINAL STATION (FINST)

The Final Station is the portion of the Central Processor which performs arithmetic and logical operations, and all stack and stack test operations. All FINST operations are initiated by instructions taken in sequence from the final queue (FINQ), which receives its instructions from the ADVAST instruction register.

As ADVAST completes its preprocessing of instructions requiring FINST operations, it places the OPCODE in FINQ, and the associated variant syllables or locally stored operand in TEMPQ. If an operand is involved which must come from Main Memory, ADVAST presents COMM with a TEMPQ address where COMM will place the operand when it arrives. (TEMPQ is serviced by COMM on a "demand" basis.)

Instructions are transferred one at a time in FINST from the final queue (FINQ) to the FINST instruction register, which holds the instruction until it is executed by the FINST hardware. This hardware includes a Comparator, which is used for all stack and field testing, and also for logical functions such as "IMP" and "OR", and some field manipulations such as "Clear Field" and "Complement Field". The ONE'S Detector is used for normalization, and conversion from integer to floating point. It is also used in stack and field testing against zero. The adder is used for arithmetic operations only, such as addition, subtraction and division. Shifts and field manipulations are accomplished through the high speed BARREL SWITCH mechanism. Multiplications are performed in the MULTIPLIER. The data which is to be operated on by the FINST hardware is transferred from TEMPQ to the top of the stack (T register) which is the accumulator for FINST. The S register and the stack extension also contain operands.

The results obtained from the hardware operating on the data can be "pushed down" into the stack for temporary storage until needed again as an operand, or else the data can be transferred from the T register to the STORQ in ADVAST, for eventual transfer to Main Memory.

The Stack Extension in FINST is serviced on a "need" basis. COMM monitors its contents and automatically executes store or fetch operations to maintain a certain predetermined number of operands for use by the FINST hardware.

FINST is dependent upon ADVAST and COMM only to the extent that to be operating there must be something in FINQ and TEMPQ. As long as there is a queue of FINST instructions, FINST does not have to halt and wait for ADVAST or COMM. There is a special instruction which halts ADVAST when this instruction appears in the ADVAST Instruction Register. ADVAST does not begin processing again until this instruction reaches the FINST Instruction Register.

Stack Operation

The stack within the FINST consists of a T register (top of stack), an S register (second position of the stack) and a 12-word stack extension. Operands are locally stored within the stack area to the limit of 14 words (52 bits/word). Further operand inserts into the stack are extended beyond this limit to a memory module.

Central Processor instructions pertaining to the stack are normally referenced to the T and S registers. As data inputs are applied to the stack they are stored in successive locations beginning at the T register, progressing to the S register and henceforth into succeeding locations of the stack extension. As the usual computational operations involve two operands, the availability of these operands in the T and S registers provides the means of implementing instruction execution. Should double precision operations be desired, adjacent areas within the stack are used for storing the most and least significant members of the operand. An additional register (P Register) is brought into operation for some of these double precision instructions (e. g. Multiply Double). The P register may be considered as an extension of the T register.

For a basic explanation of the stack concept, and its relation to Polish Notation, refer to Appendix F.

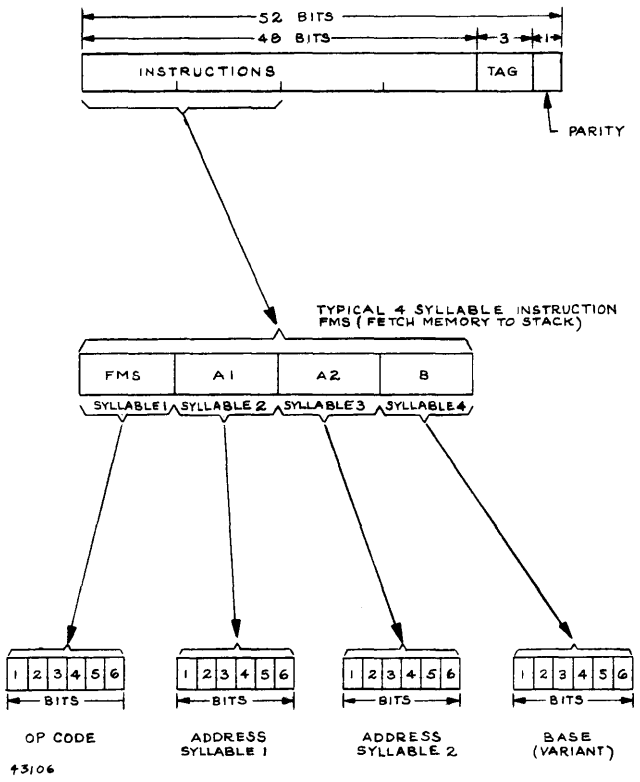


Figure 3-4. Central Processor Module, Instruction Word Format

B8500 Central Processor Instructions

LEAST SIGNIFICANT OCTAL DIGIT

MOST SIGNIFICANT OCTAL DIGIT	x 0	x 1	x 2	x 3	x 4	x 5	x 6	x 7
0 x	STOP	DUP	SSMA	FRS	ICN	FMS	SSM	SJ
1 x	FMSA	SRR	XS	RTS	ESP		JXMT	FMA
2 x	ETB	RND	ITB	SLIT	JSTL	INSD	JFT	FMC
3 x	NORM	INT	IRR	CBB	JSTA	EXTD	FMT	
4 x	ADD	SUB	AND	SSR		X	XM	
5 x	ADDM	SUBM	IMP	ARIT		FAS	CLRF	NOP
6 x	MUL	DIV	ORX	SHF		INS	COMF	FINQ
7 x	COMP	DIVI	OR	IOP	BSR	EXT	FILF	STOP

Operational Characteristics

The instruction list currently consists of 83 OP CODE, ADDRESS and/or VARIANT combinations. A complete set of Central Processor instructions, variant definitions and configurations, and word formats are given in Appendices B, C, and D, respectively. The basic structure of the instructions is shown in Figure 3-4, Central Processor Instruction Word Format.

A complete set of instructions is included in Appendix B, Central Processor Instructions. The basic structure of the instructions is shown in Figure 3-4, Central Processor Instruction Word Format.

Variation of instruction length ranges from a single 6-bit syllable format to the depicted 4 syllable construction. The FMS (Fetch Memory to Stack) instruction constitutes the longest instruction employed in the B8500.

INTERRUPT BIT PROCESSING

A possibility of seventy interrupt conditions exists within the Central Processor. When an interrupt condition occurs, program control of the Central Processor transfers from the normal mode to control-mode-1. The latter is the first of a two level control mode system which services all interrupts. Normal mode consists of that mode of operation having available only those instructions necessary for efficient program execution along with the maximum interrupt routine entrance capability.

The interrupt bit takes precedence over any other tag configuration. With this bit "set" an interrupt routine is initiated upon word entry into the Central Processor.

A flow diagram of the interrupt procedure is shown in Figure 3-5, Central Processor Interrupt Routine Entry. The interrupt sequence begins with the setting of an Interrupt Condition Register (ICRn) which corresponds to a particular interrupt, e.g. (Detected Interrupt Tag, Memory Bounds Violation, etc.). If a corresponding Interrupt Mask Register (IMRn) is set, the Interrupt Jump Register (IRJ) is set and the Central Processor will perform the hardware controlled interrupt logic sequence. This sequence provides for the storing of basic information which is required for program resumption once the interrupt has been serviced. The registers which are stored in the FINST stack are the ADVAST Address Register (AAR), the Program Count Register (PCR), the Base Data Register (BDR) and various control Flip-Flops (CFF). The Base Interrupt Register 1 (BIAR1) which identifies the starting point of the interrupt processing procedure, is transferred to the cleared PCR. Control-mode-1 interrupt processing is performed in this procedure, storing critical registers in an area referred to as the Hot Job Table (HJT). When the procedure progresses to the point which specifies the ICR, the specific procedure corresponding to the initiating interrupt condition is entered and interrupt processing is continued unto completion.

An interrupt procedure at the control-mode-1 level may encounter a transfer to a control-mode-2 interrupt level. The entry into this more restricted interrupt procedure is enabled by conditions such as a parity error, no access to Memory and a Stop. Interrupts pertaining to control-mode-2 result in the execution of another IRJ to the second level interrupt procedure which is defined by the Base Interrupt Address Register 2 (BIAR2). The procedure directs the Central Processor through the sequences pertaining to the particular second level interrupt being serviced and carries the interrupt to completion.

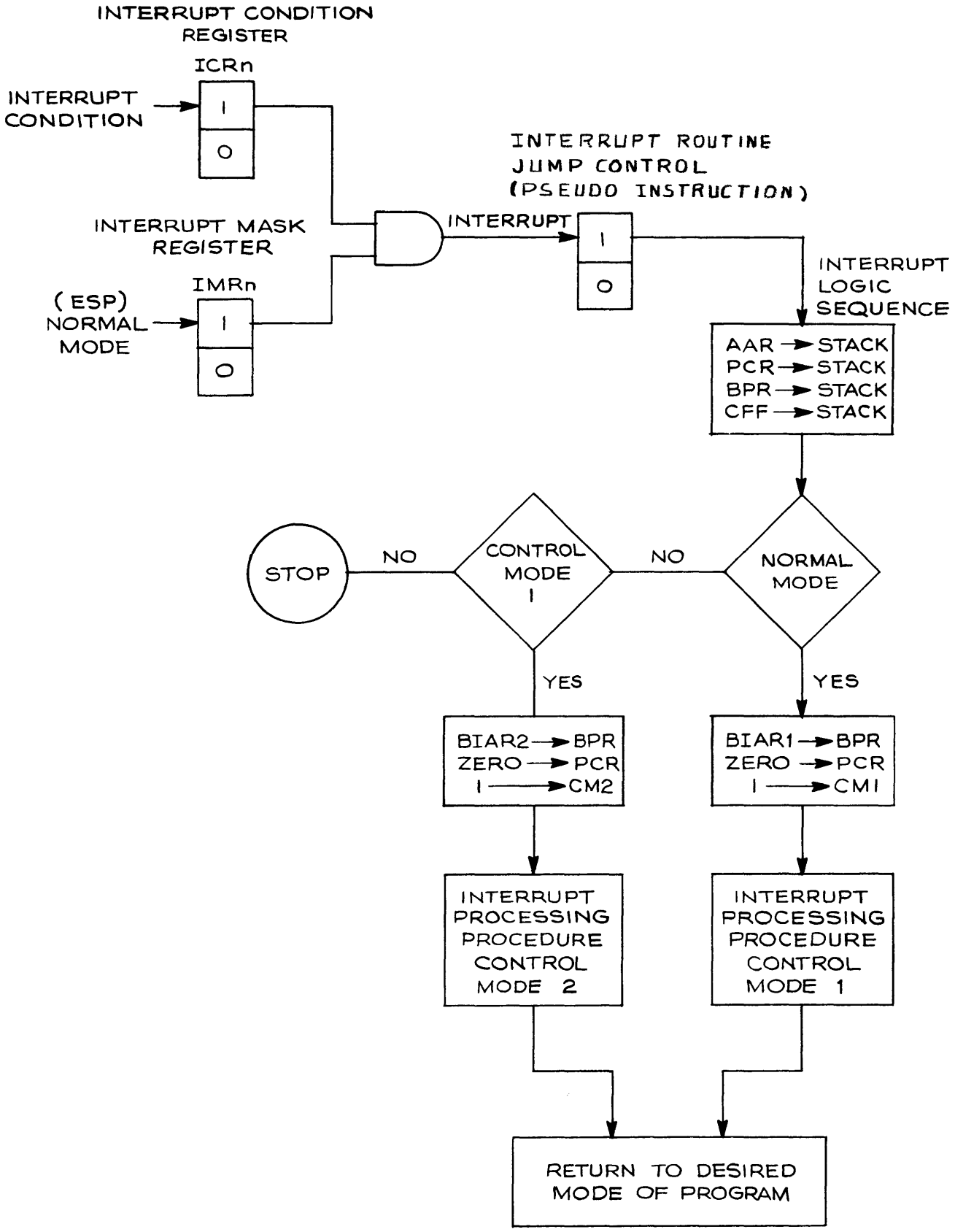


Figure 3-5. Central Processor Module, Interrupt Routine Entry

TABLE 3-2

MEMORY MODULE CHARACTERISTICS

Complete thin film memory

Fifty-two bit words -- 48-bit operands

4-word transfer (208 bits)

Memory fail register

Fetch Operations

- Fetch a single word
- Fetch a single word and modify the tag code
- Fetch four consecutive words
- Fetch the Memory Module Fail Register

Store Operations

- Store a single word
- Store four consecutive words

Cycle time - 500 nanoseconds

Access time - 250 nanoseconds

16,384 words per module

Up to 16 modules in a system

Basic clock - 20 megacycles (supplied from external source to achieve synchronism with other modules)

B8505 MEMORY MODULE

General Description

The Memory Module provides high-speed, random access thin-film storage for the B8500 Data Processing System. A single module has a 16,384 word capacity with a word length of 52 bits. It retains information as directed by the Central Processor or I/O modules in the system. The memory module is physically contained in two cabinets. This unit provides 16 busses for data transfer to as many as 16 other modules. Each buss has 52 parallel lines for input and 52 parallel lines for output. The B8500 System may be expanded to a level of 16 memory modules, thereby providing a rapid access capability of 262,144 words for a full complement of modules. The Memory Module characteristics are summarized in Table 3-2, Memory Module Characteristics.

Functional Description

A functional description of the Memory Module is shown in figure 3-6, B8505 Memory Module Block Diagram. The operation of the Memory Module is started by a request from a module with which it is interfaced. The requesting module transmits a request, a request strobe and data (Control Word). The request (REQ) signal is used for priority control within the Priority and Strobe Logic where request evaluation is performed. The request is accepted by the memory if its channel has the highest priority of any active channels. The request strobe is used to transfer the control and address portions of the Control Word into their respective memory module areas. The request strobe also sets a "busy" signal, which is used to indicate memory status to subsequent requests from other other modules during the completion of the current memory operating cycle. An acknowledge (ACK) signal, indicating the memory module acceptance of the Control Word, is sent to the initial requesting module.

CONTROL WORD

The OPCODE portion of the Control Word (shown below) describes the type of operation (read or write) which is to be performed between the memory and the requesting device.

The OPCODE syllable bits have the following meanings:

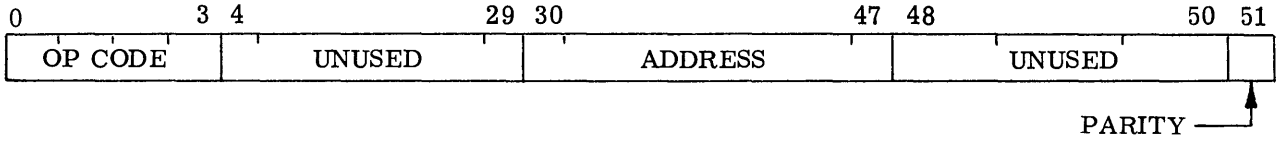
Bit Position	Value 0	Value 1
0	Fetch	Store
1	1-Word	4-Words
2	Normal	Modify Tag
3	Zeros or Normal	Ones or Fail Word

The ADDRESS consists of 18 bits and thus has the range to select the 262,144 words of data, which a full complement of memory modules can provide.

When a 4-word memory operation is specified, Address bits 46 and 47 must be zeros for normal word order to be maintained, i. e., word 0 through 3 of the selected 4-word address. A different word code in the two least significant bits of the Address will result in processing the selected word first followed by the remaining three words in a cyclic fashion.

FETCH OPERATIONS

A fetch OPCODE describes one of four possible operations which will transfer data from the memory module to the requesting device. The operation is sensed in the Timing and Controls section and the address is recorded in the Memory Address Register (MAR). The specified address enables corresponding Memory Switches and Drivers, which select the word location within the thin-film memory stacks. Read Cycle control signals from the Timing and Control Section initiate the read cycle and the Sense Amplifiers receive data read-out information from the thin-film Memory.



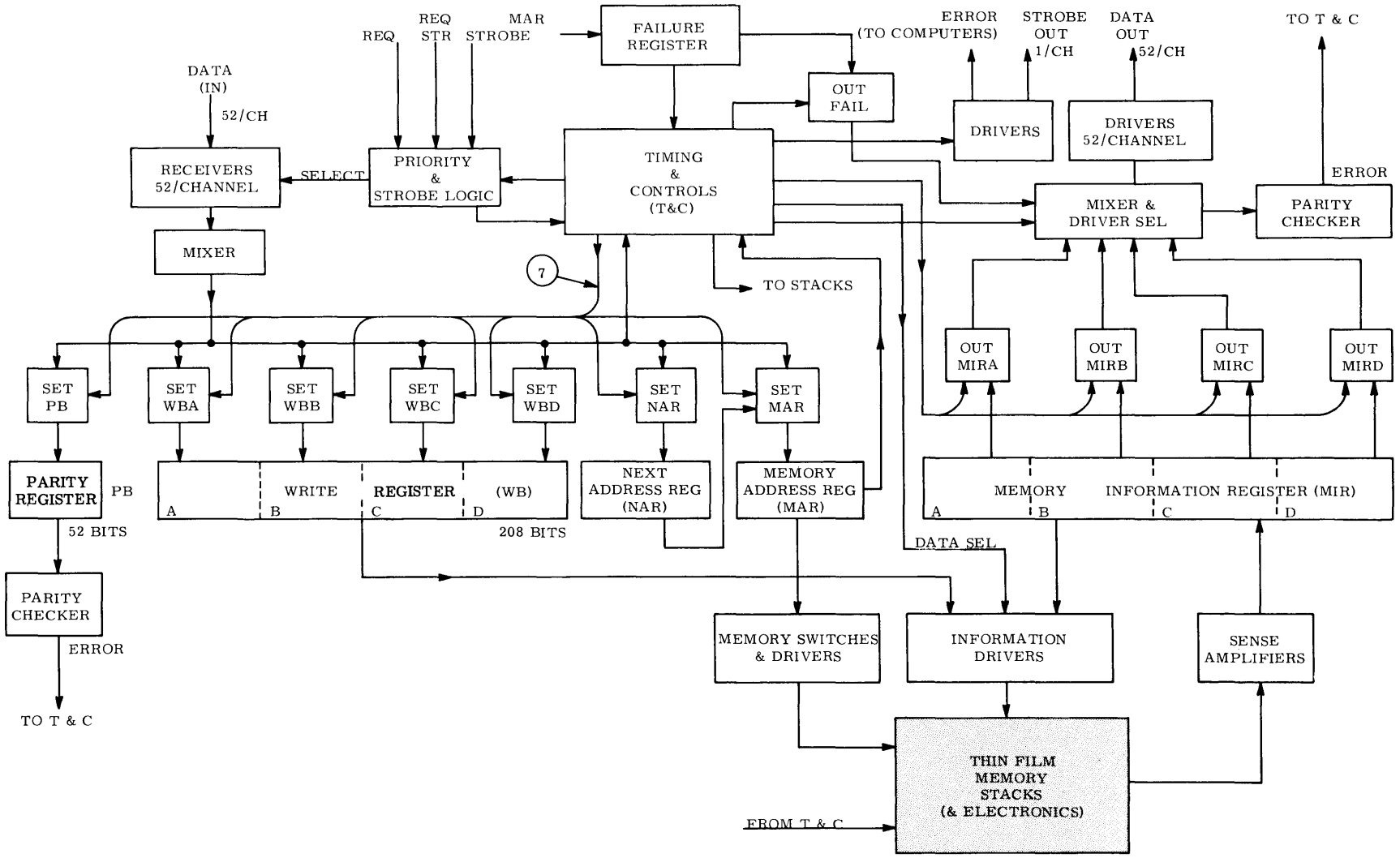


Figure 3-6. Memory Module, Block Diagram

The Information Register stores the four word data which is further selected by the word select logic MIRA through MIRD. The word is transferred to the requesting module via the Mixer and Driver Select logic, the Drivers, and the final output stage within the memory module. A strobe output is also provided at the driver stage for transmittal to the requesting module along with the data. The conclusion of the fetch operation includes the return of the word to the thin-film location which was accessed. This is accomplished by way of the Information Register to Information Drivers path shown in figure 3-6.

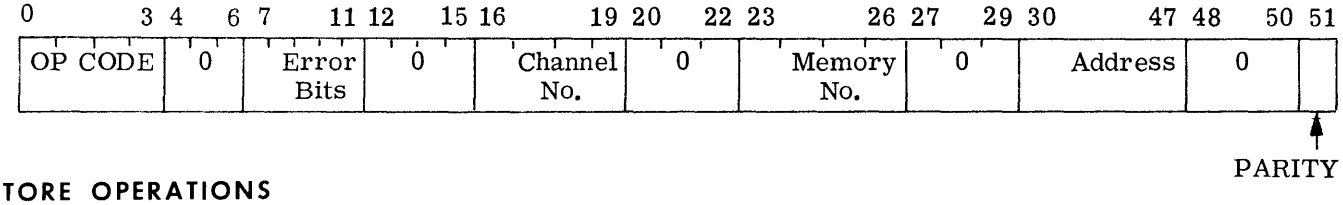
STORE OPERATIONS

A store OPCODE describes one of two possible write operations which will transfer data from the requesting module into the memory module. The operation is either a single-word or a four-word store. The request is sensed by the Priority and Strobe Logic, a request strobe transfers the Control Word operation into the Timing and Controls section and the address into the Address Register. As in the fetch operation, the Memory Switches and Drivers corresponding to the Memory Address specified by the Control Word are selected and the store cycle is initiated by the Timing and Controls section. The data input is sensed by the Receivers and Mixer, transferred under Control Word direction through the Write Controls (WBA through WBD) and stored in the 208-bit Write Register. The data is transferred through the Information Drivers into the thin-film Memory, where the write operation terminates.

CHECKING FUNCTIONS

The Memory Module checks each word received or transmitted for odd parity. Upon detecting incorrect parity the Memory Module interrupts the Processor Modules and retains pertinent information concerning the failure in the Memory Module Fail Register. Additional check capabilities are provided by single pulse operation and automatic checking. A control panel is provided on the Memory Module which contains controls and flip-flop indicators permitting single pulse operation under manual control for maintenance purposes. The Memory Module is permanently connected to the Memory Check Module which has the capability of automatically exercising all of the locations and functions of the Memory Module.

The Memory Module Fail Register records Fail Word data, including a copy of the Control Word for the operation which was being performed when an error was detected. The channel number and memory module number are also provided to show the module interconnection involved. The Fail Word format is shown below.



The error bits are defined as follows:

Bit	Error
7	Parity Error - Control Word
8	Parity Error - Incoming Data
9	Wrong Memory Address - Control Word
10	Parity Error - Outgoing Data
11	Illegal Operation Code - Control Word

Bit No. 51 is used for parity purposes to provide a check on the instruction word. Its purpose is to indicate the overall bit composition of a word and thus provide a means of error detection.

Interface Characteristics

A Memory Module interfaces with the I/O, Central Processor and Memory Check Modules of the B8500. Inter-module communications are accomplished over 16 busses each of which contains 52 parallel lines for input and 52 parallel lines for output.

B8510 INPUT-OUTPUT MODULE

Introduction

The most important feature of the I/O Control Module is its independence from the Central Processor. In effect, the I/O Control Module is a separate processor with its own Local Memory Unit, logic and arithmetic functions, and communication capabilities. Because it provides channels for all data transfers between peripheral devices and the System Memory, the I/O Control Module executes transfers from one peripheral device to another peripheral device independent of direct Central Processor control.

The system operations of the I/O Control Module are programmed separately from the Central Processor Modules. This programming is coded in an assembly language designed solely for the I/O Control Module. Then, the symbolic program is assembled into an object program by means of an assembler program written in ALGOL. Input to the assembler is punched cards. The output consists of a program listing, with relative memory assignments, and a tape that contains the assembled object program for printing and ultimate loading into the B8500.

The simple transfer function between peripheral devices and System Memory is similar to the more conventional descriptor-controlled and buffered I/O in the sense that there is a descriptor generated by the Central Processor for each data transfer executed. However, in data transfers to and from peripheral devices, the I/O Module completely controls the I/O operation from parameters stored in the I/O Module's program data area. The execution of buffered I/O can now be treated by Central Processor programs in terms of tape, card, and print files, rather than by individual tape records, card images, and print lines. This means that I/O processing overlaps the Central Processor for long periods of time, rather than the more conventional technique of overlapping a single record. Using this overlap method reduces the number of interrupts to the Central Processor, thereby reducing the amount of software overhead associated with a Control Program.

The parameters describing the transfer are treated by the Central Processor program as parameters to a procedure that performs the transfer. The procedure call is not to a Central Processor program, but to a program that is, in effect, an extension of the hardware and that is executed completely on an I/O Processing Unit. The return is to the caller in the Central Processor only when the I/O program's function is complete.

Summary

The I/O Control Module has the following characteristics and capabilities:

- Storing and fetching from Memory Modules.
- Up to 512 simplex channels (buffered) per module.
- Independent channel operations controlled by individual descriptors.
- Service time for a channel of 0.6 to 1.8 microseconds.
- Total data handling rate of up to a maximum of 286,000,000 bits per second.
- Local thin film memory of 1024 100-bit words.

A B8500 System may have from 1 to 14 I/O Control Modules.

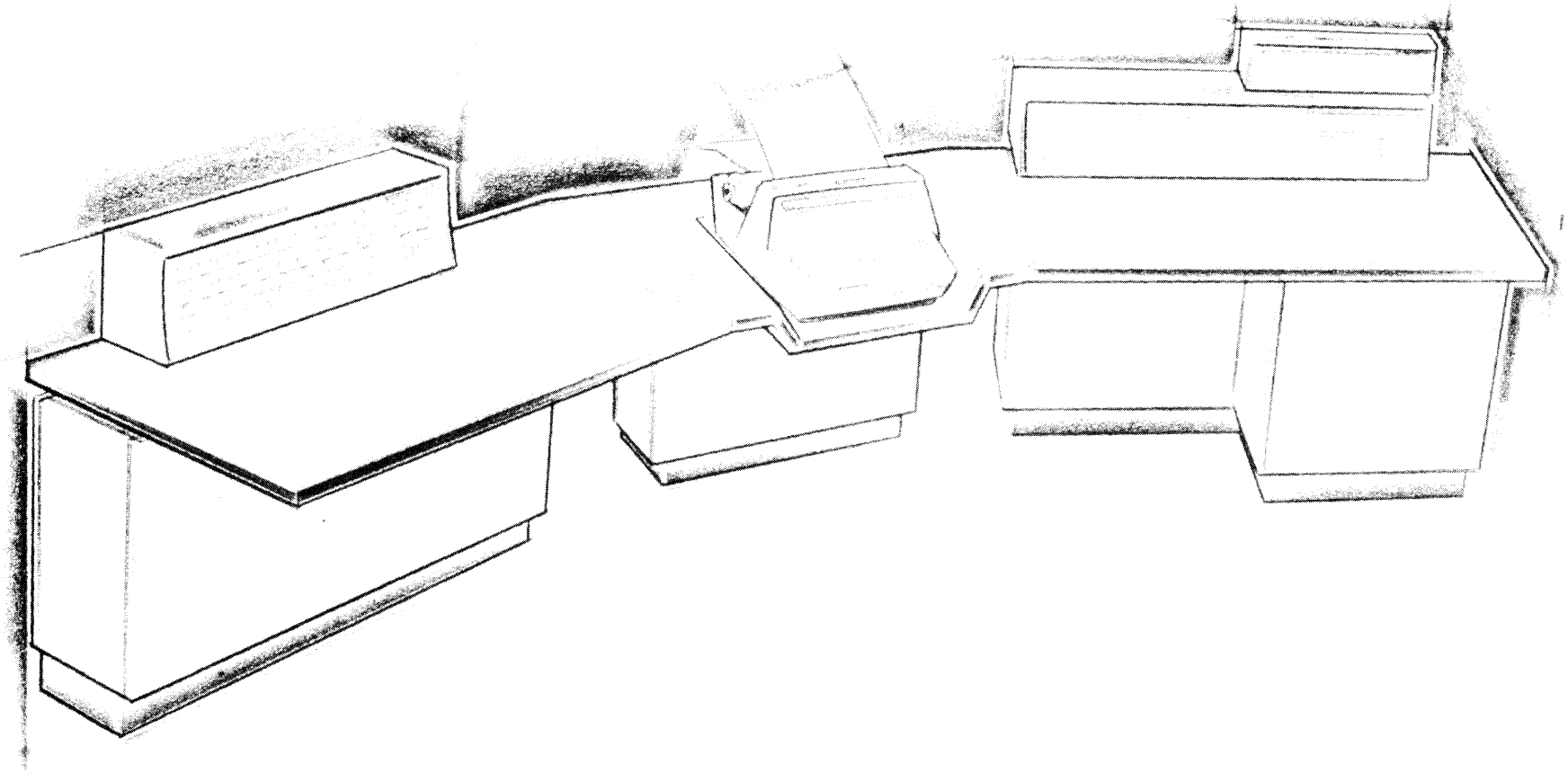


Figure 3-7. B8520 Console

B8520 CONSOLE

General Description

The B8520 Console (Figure 3-7) is the operations center of the B8500 Data Processing System. The console contains control and input/output facilities which provides an operator with a convenient supervisory control center and also provides field engineering the capability of controlling power and testing memory modules. In effect, the console is the communications link between an operator and the B8500 System.
The console is physically divided into three independent cabinet units: the power control unit, the memory checkout unit, and the input/output unit. The units are joined together to form a common desk-type console which permits sit-down operation. The three units can be arranged in configurations varying from a semi-circular arrangement to a straight-line arrangement.

The power control unit contains the necessary controls, indicators, and circuits to provide the power control and sensing functions of the console. The controls and indicators are contained on an upright panel mounted at the rear of the unit. The power control unit is supported in an upright position by the input/output section, which is always placed adjacent to the control unit.

The memory checkout unit contains controls, indicators, and test circuits which facilitate the manual or automatic testing of memory modules in the B8500 System. The unit also contains the system master clock circuitry. All controls and indicators are contained on an upright panel mounted at the rear of the unit's desk top.

The input/output unit consists of a table-type cabinet with a typewriter mounted on its surface. The cabinet is usually placed between the power control unit and the memory checkout unit to link the units into a common desk-top configuration, providing easy access to all panels.

Functional Description

The B8520 Console serves the following three functions:

- Serves as a communications link between an operator and the system
- Provides master timing controls and memory module check facilities
- Serves as a power control and sensing center

Because these three functions are physically implemented by the use of three independent cabinets, the console may be configured to provide one, two, or all three functions. While the power control and input/output functions are optional features which may or may not be required by a user, the memory check function is an imperative requirement.

COMMUNICATIONS LINK

An electric typewriter with both input and output capabilities serves as the direct communications link between the operator and the B8500 System.

MEMORY MODULE CHECK FACILITIES

The memory checkout unit of the console provides the only direct capability of performing comprehensive checks on malfunctioning memory modules. Memory checks can be performed only on memory modules which have been taken "off line". The checks, which can be performed at full memory speed, are divided into the following categories:

a. Stack checks

b. Logic tests

c. Common checks

The memory checkout unit of the console is interfaced with all memory modules in the system by use of one of the sixteen data busses.

POWER CONTROL AND SENSING CENTER

The power control and sensing function is independently implemented by the power control unit. The power control function involves the control of the power supplies in each individual cabinet of the central processing system (processor, memory, I/O, controller, and communications modules).

Power in any cabinet may be turned on or off by either local controls on the cabinet or by a control on the power control section of the console. A separate power control switch is provided for each cabinet. The turning off of power in any cabinet will not affect other cabinets in the system as long as the respective cabinet has been placed "off line".

DISK FILE SYSTEM

General Description

The disk file system used with the B8500 Data Processing System is a fast access mass memory system with a total storage capacity of 200 million 48-bit words. Average access time to any word is 30 milliseconds, and the transfer rates average 8 million bits per second (166,666 48-bit words). This high performance disk file system is a natural evolution from the highly successful commercial disk file system used with the Burroughs B5500 and B200/300 data processing systems. An increase in performance over these systems is achieved by eliminating word and segment interlacing and by reading and writing six data streams simultaneously.

A single disk file system includes, but may not exceed, two disk file controllers, 20 disk file electronics units, and 100 disk file storage modules. The disk file electronics units and storage modules are grouped into a maximum of 20 disk file subsystems within each disk file system. Each of the subsystems includes one electronics unit and a maximum of five storage modules. The six units are arranged side-by-side and interconnected to form a subsystem. Figure 3-8 shows a typical disk file subsystem consisting of one electronics unit (at the left) and two storage modules. The number of storage modules used with each electronics unit is determined by the total storage capacity required for a system. Additional mass storage facilities may be obtained by adding more controllers, electronics units and storage modules.

As shown in Figure 3-9, a group of five storage modules is controlled by each electronics unit, and each electronics unit is cross-connected to two disk file controllers. Each disk file controller serves up to 20 electronics units, and each controller is connected to two I/O modules. Therefore, by using two disk file controllers and two I/O modules, up to 20 electronics units and 100 storage modules are provided with dual paths to the memory modules within the central data processing system. It should be noted that, although only one controller is necessary to control up to 20 electronics units, two controllers are cross-connected to the electronics units to increase the reliability of the system.

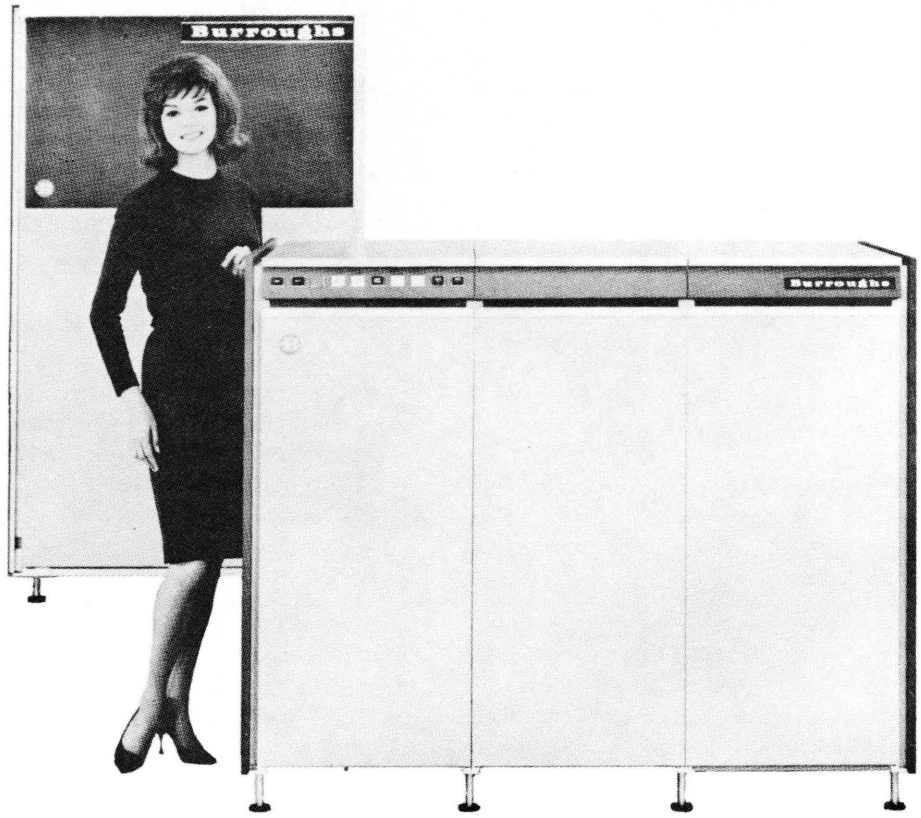


Figure 3-8. Disk File Subsystem

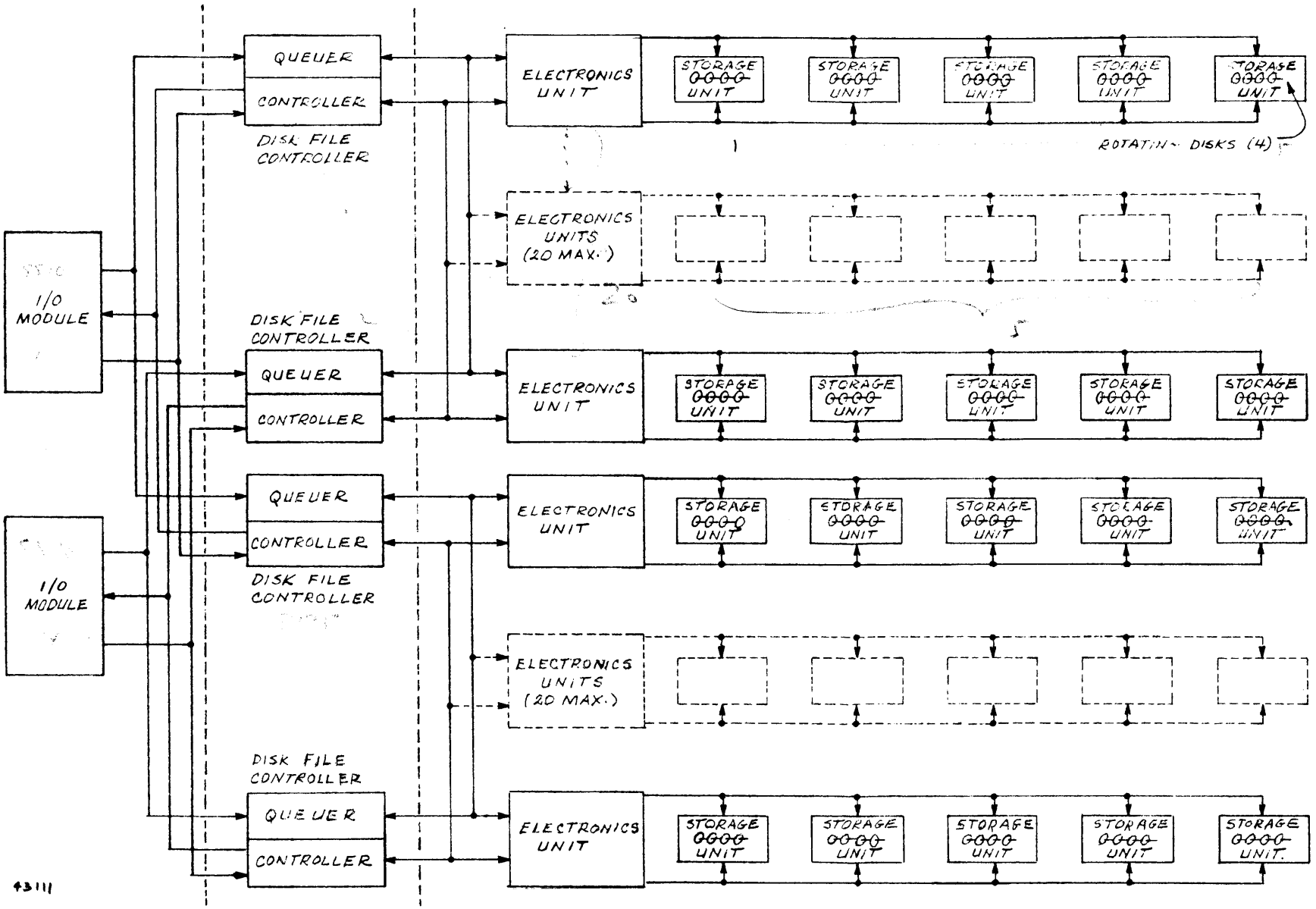


Figure 3-9. Disk File System Configuration

PHYSICAL DESCRIPTION OF ELECTRONICS UNIT

The electronics unit (Figure 3-10) is basically identical to the B471 Disk File Electronics Unit used with the Burroughs B5500 and B200/300 computing systems. The electronics unit contains the necessary components to provide the power, air pressure, motor control, head switching, and gating logic functions for the control of one-to-five storage modules. The electronics unit also contains the logic components for the read and write functions performed by the subsystem. All components are mounted in a cabinet frame which is approximately 53 inches high, 23 inches wide, and 45 inches deep. The total weight of each electronics unit is approximately 450 pounds. The cabinet has removable front and side panels which provide access to a hinged plug-in unit logic rack, a maintenance panel, power supply components, and pneumatic components. Operator's controls and indicators are mounted on the upper face of the cabinet and under a hinged top panel.

PHYSICAL DESCRIPTION OF STORAGE MODULE

The disk file storage module (Figures 3-10 and 3-11) is basically identical to the B475 Disk File Storage Module used with the Burroughs B5500 and B200/300 computing systems. The storage units are modular storage devices which are mounted on casters and housed in garage-type cabinets adjacent to an electronics unit. This modularity feature enhances the expansion capabilities and maintenance of the subsystems.

Each storage module is approximately 53 inches high, 23 inches wide, and 45 inches deep and weighs approximately 575 pounds. When a storage module is wheeled from its cabinet, access can be gained to all basic components. These components include four vertically-mounted disks, disk drive and control mechanisms, pneumatic devices, plug-in units on a hinged rack, and a maintenance panel. Each pair of disks is enclosed in an air-tight cover for purposes of cleanliness.

A total of eight disk faces in each storage module provide for a total storage capacity of 2 million 48-bit alphanumeric words. Each disk face has 50 data tracks comprised of 1250 207-bit data segments. Each track is equipped with a read-write head which is controlled by means of electronic switching.

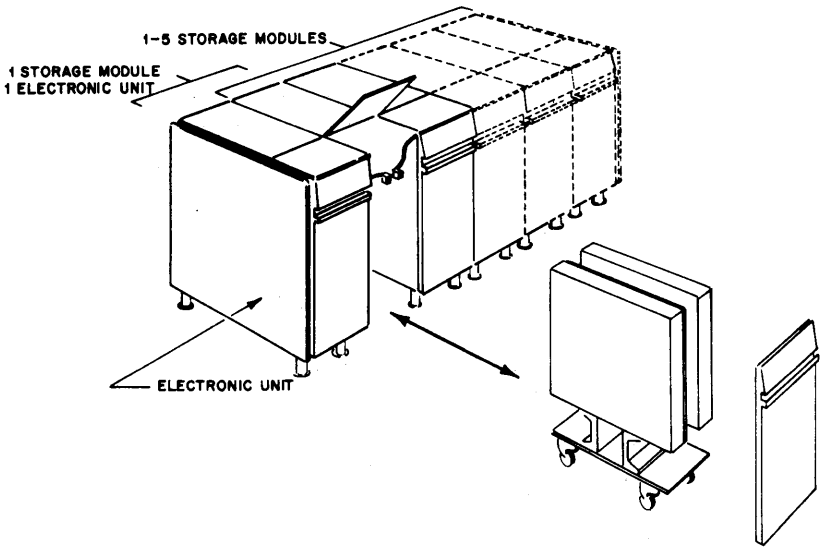


Figure 3-10. Disk File Subsystem Modular Design

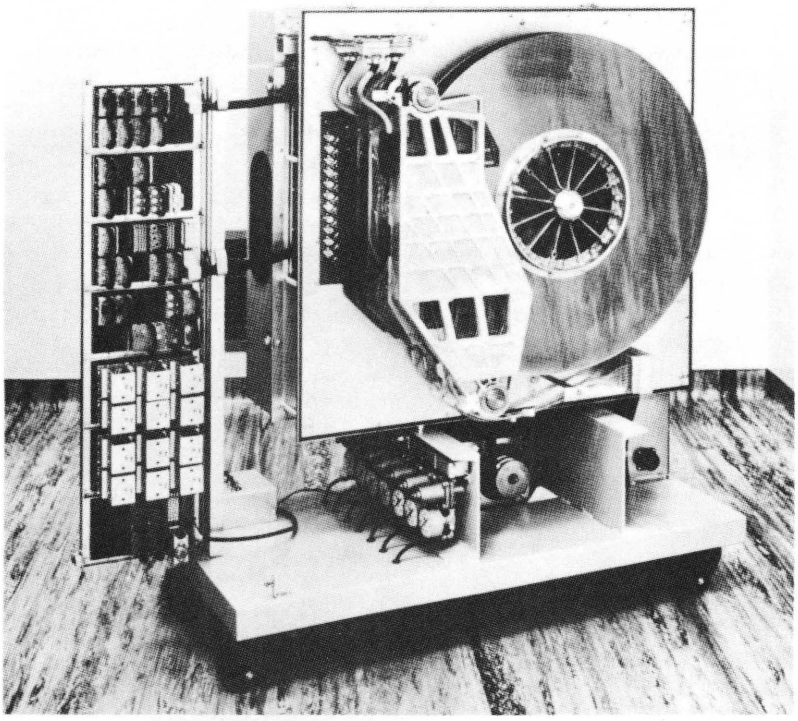


Figure 3-11. Disk File Storage Module

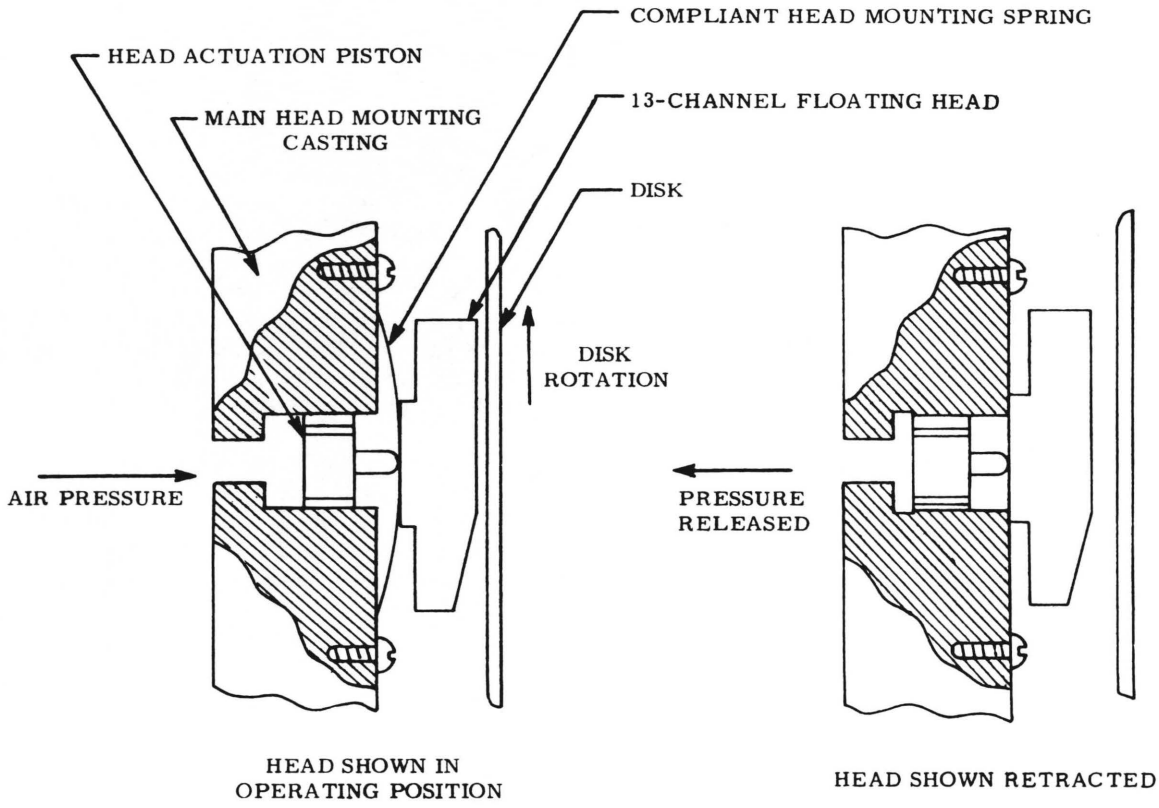


Figure 3-12. Head Mounting and Actuation

The magnetic discs are brass annular rings, hub mounted, 26-1/2 inches in diameter, and 1/8 inch thick. They are plated with an extremely thin magnetic film. One criterion for high density recording is the need for a very thin and uniform magnetic coating. To achieve this, Burroughs has developed several types of magnetic film plating processes which involve both electroplating and electroless plating techniques. Both processes are used in current Burroughs products. The electroplating process was chosen for the disk file. This process plates a magnetic film less than 30 micro-inches thick with a coercivity of about 500 oersteds and a remanence of approximately 6,000 gauss. This plating thickness is less than 1/16 the thickness of typical oxide coated magnetic tape. After a disk is plated, it is dynamically balanced and tested for flaws (before assembly) on automatic flaw testing equipment; all disks are free from flaws in active track areas.

Figure 3-12 shows an outline of the head and gives an indication of its actual shape. The wedge, which gives the head its air-bearing capability, is exaggerated in the drawing and is at an angle of less than a degree with respect to the base of the head.

Figure 3-12 also shows the head mounting arrangement and actuation principle. The head is mounted to a flexible-type flat spring which is rigidly mounted to the main head mount casting. The spring provides the facility for the head to align itself during flying and also provides a force to return the head when the actuator is released. Actuation of the head is accomplished by pushing the head towards the disk with an air-actuated plunger. The actuation force on each head assembly must increase as a function of the distance of the head assembly from the disk shaft center. The boundary layer of air, which provides the lifting force to the head assembly, increases as a function of surface speed, which in turn is a function of radius. The requirements for the different actuation forces are met by using several different air pressure regulators.

Every effort has been made to make the disk file as free from failure as possible, particularly with respect to preventing any contact between the floating heads and the rotating disks. To achieve such a "touch-free" disk file, all components within the floating head system are designed to be fail safe. The disk speed is monitored with a tachometer which prevents the heads from becoming actuated unless the disk is rotating at a safe speed. This same circuit automatically retracts the heads if, for any reason, the disk speed falls below a certain value. Because the heads are air actuated and spring returned, the heads automatically retract in case of failure in air pressure. Power failures of any type automatically release the air pressure, causing the heads to retract.

The final protective device is an "anti-touch" circuit. This circuit applies a signal between the head body and recording medium (disk) and can be adjusted to detect any desired minimum floating gap. If the head comes closer to the disk than this set gap, or if the gap is bridged by a foreign particle, this condition is sensed and the heads are automatically retracted. Provisions have been made for determining which head assembly generated the alarm.

Functional Description

The disk file system provides the B8500 System with a very large storage facility capable of rapidly accessing any record. As described previously, a disk file system is comprised of disk file controllers and the required number of disk file subsystems. The disk file controllers, which are housed in the B8515 Controller Cabinet, facilitate the interfacing and data communications between the electronics units in the disk file subsystems and the I/O modules. Each electronics unit provides for the local control of up to five disk file storage modules, which form the actual storage media of the system. Functional and operational characteristics of the controller, electronics unit, and storage module are provided in the following paragraphs.

DISK FILE CONTROLLER

A disk file controller is used to provide compatibility between the disk file system and the I/O modules. The disk mass memory is composed of a multiplicity of quasi-independent disk sets with a total storage capacity far beyond the address capability of the 18-bit address field of the B8500 instruction words. Therefore, when communicating with the disk file system, the B8500 System must transmit a control word which contains not only the expanded address field necessary to access a specific word or block, but which also contains the coding for the specific function to be carried out. The controller is the instrument for interpreting this control word and consequently controlling the detailed operation of the disk file system.

The more important basic functions of the controller include:

- Managing traffic
- Receiving and storing control words
- Scanning addresses to be serviced
- Selecting optimum address in accordance with disk position
- Decoding starting addresses and updating current addresses as words are read or written
- Assembling and transmitting status fields to indicate the current status of the controller and its devices to the I/O Module
- Buffering for concurrent reading or writing on all 6 data streams of a disk storage module
- Checking and generating parity for words coming to and going from the disk subsystems

The logical capability of the controller allows any one of 200 million 48-bit words to be stored in or read from any one of up to 100 different disk storage modules.

The disk file controller consists of two functional sections: the controller section and the queuer section. In general, the controller section transfers data between a disk and an I/O module, and the queuer section controls the selecting of most accessible disk addresses.

Controller Section

The controller section of the disk file controller consists of four functional units; the interface unit, the address unit, the disk select unit, and the data unit. A block diagram of the controller section and its subunits is shown in Figure 3-13.

INTERFACE UNIT. The interface unit contains communication gates, interface controls, and parity checking and generating circuits. All data, status conditions, and descriptor addresses exchanged between the disks and the I/O module are handled by the communications gates under supervision of the interface controls. Each data path accommodates the standard word format of 48 bits and one parity bit. Two input control lines are used; the start line and the select strobe line. Output control lines consist of the output service request, input service request, status, last byte, and strobe signal lines. When the queuer finds a request within the required comparison limits, it sends the main memory address portion of the request to the communication gates and initiates a request. The interface controls send the input service request signal to the I/O module. When the I/O module responds to the select strobe, the controller places the 18-bit memory module address on the data lines. The I/O accepts the information and with the strobe determines that the 18 bits represent the memory module address which holds the control descriptor. This control descriptor defines the operation carried out on the disk being selected at this time. The I/O module fetches the control descriptor from the memory module, raises the start line to the controller, and waits for the controller to respond by either sending data during a read operation or by requesting data for a write operation. As the interface unit returns the address of the descriptor to the I/O module, the controller address register located in the address input begins a comparison of segment address bits coming from the selected disk.

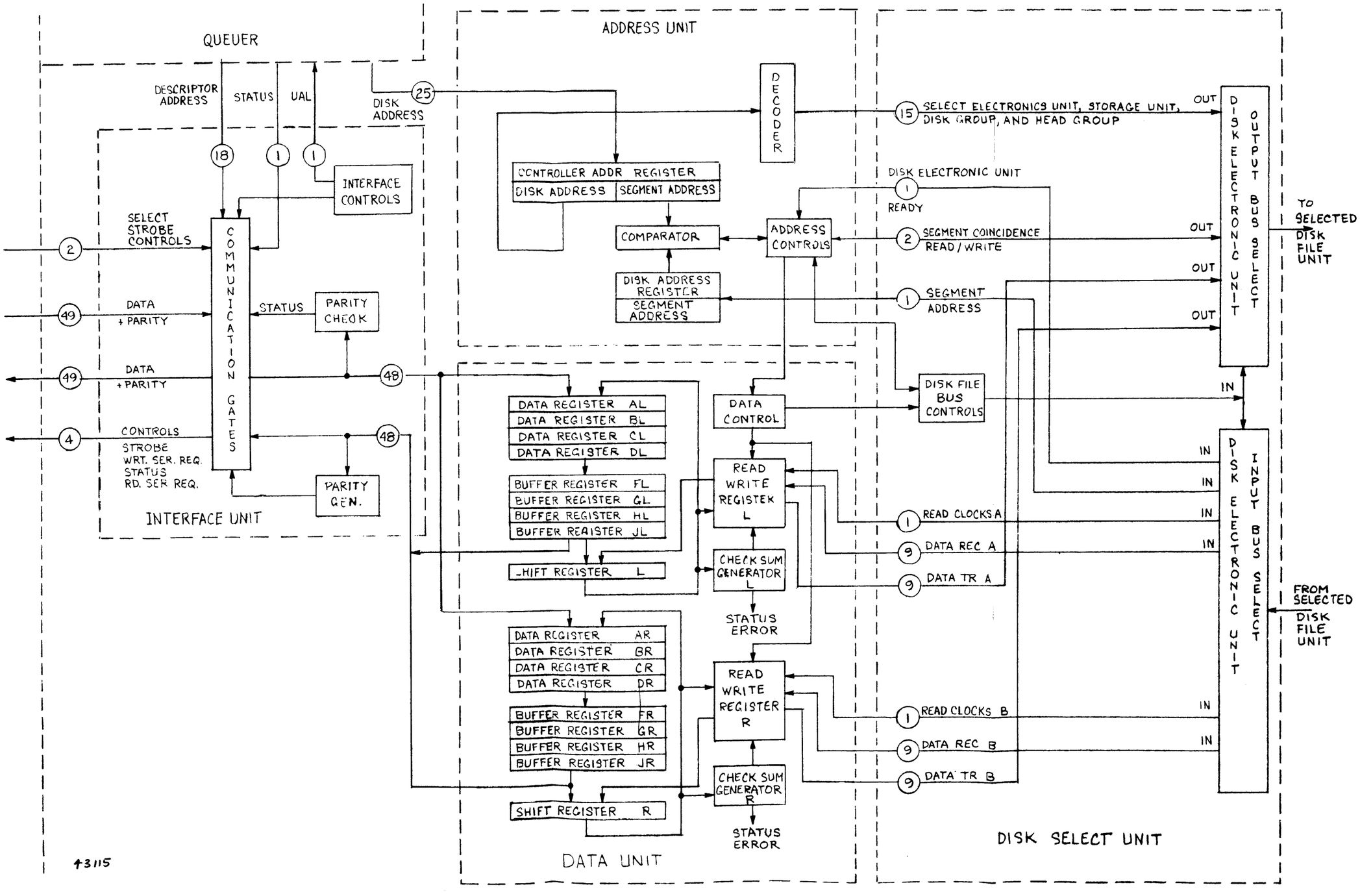


Figure 3-13. Controller Section, Block Diagram

ADDRESS AND DISK SELECT UNITS - The address unit contains the controller address register, disk address register, comparison network, decoding matrix, and address controls. The disk address is decoded to send signals to the selected electronic unit to select a disk storage module, a disk set, a track, a head, and address clocks. The address controls send a signal to the disk select unit which initiates selection of an electronics unit. The disk select unit contains the electronics unit output bus select, the electronics unit input bus select, and the disk file bus controls. When the disk file bus controls select an electronics unit, it allows the remaining selection signals for disk storage module, disk set, and head and address clocks to be sent to the electronics unit. It also allows the data lines and clock lines from the selected electronics unit to enter the controller.

The address clock bits are shifted into the disk address register and compared to the segment address stored in the controller address register. If a write operation is required, the interface controls request a block of 8 words from the I/O module. The interface controls send an output service request pulse. The I/O module responds by fetching the first 4-word block from the memory module, transmitting the block to the controller over the 49 data lines, parallel by bit and serial by word. Once the transfer has started, a word is sent every 100 nanoseconds for a 4-word block, and the select strobe signal accompanies the data. The controller accepts the four 49-bit words at the communication gates and then loads these words into data registers AL, BL, CL, and DL in the data unit.

DATA UNIT. The data unit of the controller consists of eight data storage registers, eight buffer registers, two shift registers, two checksum registers, and data controls.

During a write operation, the first group of four 49-bit words enters through the communications gates and is loaded into data registers AL-DL, and the second group of four words is loaded into data registers AR-DR. When the buffer registers are empty, the contents of data registers AL, BL, CL, and DL are transferred into buffer registers FL, GL, HL, and JL, respectively. The contents of data registers AR-DR are similarly transferred into buffer registers FR-JR. The words stored in each group of buffer registers are successively loaded into an associated shift register from which they are transferred (nine bits at a time) to a read-write register. The data control is required to wait until the address control (in the controller address unit) has received a comparison signal which indicates that the address compares with the selected starting segment address on the disk. The clock signals received from the selected disk are utilized by the data control to send 18 bits of data every character time to the disk. The data (nine bits from each read-write register) are simultaneously written on three zones on one disk face and three zones on another disk face. As each word passes through the shift register, a check sum generator begins to accumulate check bits to be stored on the disk at the end of a block of four words. The transfer rate to the disk is approximately eight words every 48 microseconds (one word approximately every 6 microseconds). As soon as the contents of the data registers have been emptied into the buffer register, the interface control unit sends the output service request pulse to request another eight-word block of data. The buffer registers buffer enough data words to satisfy the worst-case demands for the servicing of I/O modules.

When the final block of eight words is transmitted from the I/O module, the "last byte" signal is sent to the controller to indicate that the operation has terminated. The controller responds once more to notify the I/O module of a good transfer, or sends a status signal which signifies the existence of an error condition. After the controller has written the final eight words of data on the selected disk, the controller becomes available to the queuer and waits for the next job request.

The read operation is handled in a manner similar to the manner described for the write operation. The queuer, upon finding the controller available, sends another request. The controller then returns the 18-bit memory module address to the I/O module and begins selecting the proper disk. When the disk address agrees with the starting segment address on the disk, the read-write registers (assuming a read operation) begin to accept data simultaneously from two disk faces at an average word rate of approximately 6 microseconds. Each read-write register accepts nine bits of data, in parallel, from each disk face approximately 2 microseconds after address coincidence has occurred. Four 9-bit data words are accumulated in the shift register and are then loaded as a 48-bit word into the data registers. After each four-word group has passed through the shift register, the check sum generator accumulates three check bits which are then compared with the check bits read from the disk.

When the buffer registers are empty, the information stored in the data registers is transferred to the associated buffer registers. As soon as the data are available in the buffer registers, the interface control unit transmits the input service request pulse to inform the I/O module of the availability of eight words to be transferred. The I/O module responds by accepting the first four-word block and then the second four-word block. The transfer rate is 100 nanoseconds per word for each four-word block.

If a check sum error is discovered, it is reported with the appropriate status code by the interface controls. The status reports may include parity errors from the I/O modules, check sum errors from the disk, "disk file lockout", or "disk storage unit not available". Some of the status conditions may cause termination of the present operation.

Queuer Section

One function of the disk file controller is the accumulation of disk transactions so that they may be ordered in a manner that keeps the data channels operating as efficiently as possible. This function is accomplished by the queuer section (Figure 3-14), which stores the disk transactions in a memory job stack and compares them with current disk addresses as they come from the selected disk storage modules. When a disk address comes within the accepted time bracket of any current disk address, that particular control word is chosen as the next transaction to be conducted. A block of data may be transferred within approximately 1.5 milliseconds after the preceding data transaction has been terminated.

A request for access is received through the queuer input gates and is stored in the memory job stack under supervision of the input controls. The request is then read into the queuer stack register (QSR), and parity is checked. If parity is wrong, the request is ignored, and a status line to the I/O modules is energized. If the parity is correct, a portion of the 48 bits assigned to the control word is decoded to determine which type of operation is to be performed. If the operation code defines a queue/write or queue/read operation, the 48 bits are placed at the top of the stack under control of the queuer address register (QAR) and the top-of-stack register (TSR). If the operation code defines a bypass queue operation, the QSR holds the 48-bit word until the controller becomes available; then the job is transferred to the controller.

When the controller becomes available, the queuer stack controls begin to sequence through each job word. The operations include the reading of the stack into the QSR, shifting the 25 disk address bits to the controller address register, and then converting the disk address into the proper address format. After the converting operation is accomplished, a request is made for a disk electronics unit and a storage module. When the request is granted, the address data and address clocks from the selected storage module are read and compared with the contents of the address register. If the address of the selected disk is within acceptable bounds, the job is transferred to the controller section. The 18 bits of the job word that contains the main memory address of the job descriptor are returned to the I/O module through the controller-to-I/O interface system.

If a word command is required, the controller requests the first "byte" consisting of eight 48-bit words. If a read operation is required, the controller waits until it has accumulated the first eight words from the disk and then requests an input operation to transfer the "byte" to the I/O module.

While the controller is sequencing the flow of data to and from the selected disk file module, the queuer stack controls place the job at the top of the stack in the location left vacant by the job in process (to effectively compress the stack). If the queuer is not full, it samples the I/O start line and, when the line is high, the queuer accepts more job words through the interface system.

The queuer also has the capability of transferring the contents of the queuer memory stack to the I/O module upon command. When the operation code defines a "dump queuer" command, this command becomes the next operation after the controller is free. The queuer memory stack data is transferred to the controller data registers and is returned to the I/O module as if it were data transferred through the interface between the I/O and controller. The "dump queuer" command provides the program with the flexibility of checking the job stack periodically.

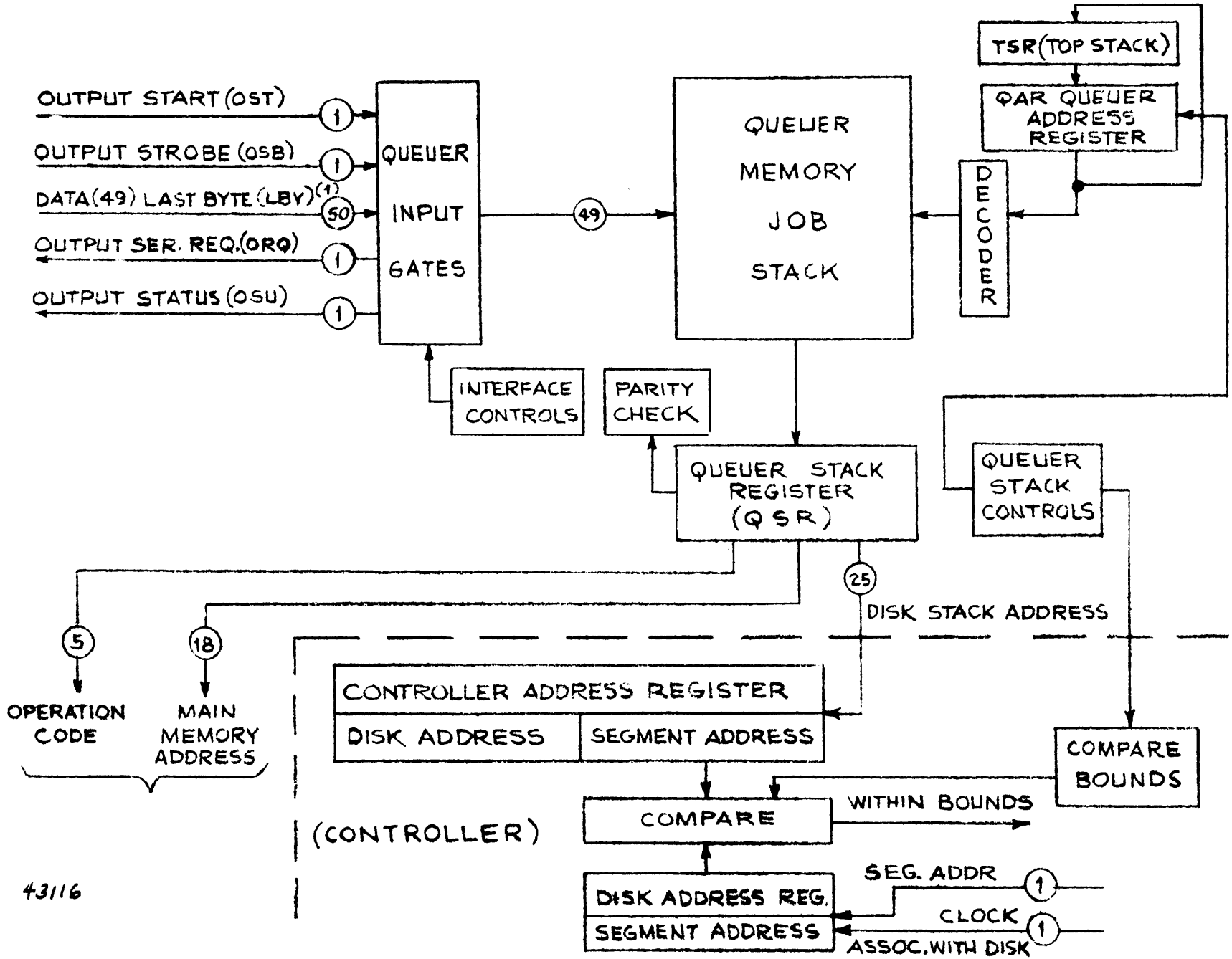


Figure 3-14. Queuer Section, Block Diagram

Interface Characteristics

Figure 3-15 shows the basic interfacing and information flow between the I/O module, a disk file controller, and an electronics unit. The illustration indicates the data flow handled by a channel serving the controller section of the disk file controller and the requests for access to the disks treated by a channel serving the queuer section. The controller interface handles all data, descriptor addresses, and status conditions exchanged between the disks and the I/O module. There are separate data paths into and out of the controller; each data path handles 49 data bits in parallel. This follows the standard format of 48 data bits and one odd parity check bit. Additional inputs to the controller consist of three non-data lines: the start line which informs the controller of an impending input, the select strobe line which provides the timing for the input, and the "last byte" line which signals the controller on the last output data transfer. The output from the controller consists of four control lines:

(a) A strobe line which informs the I/O module of the arrival of data.

(b) An output service request line.

(c) An input service request line.

(d) A status line which separates data from status messages on the 49 lines from the controller to the I/O module. The status line is made true whenever status is transmitted on the data line.

Both input and output word rates at the controller are determined by the transfer rate of the disk. This rate is approximately eight words every 48 microseconds, or an average of one word every 6 microseconds. This rate is maintained for a maximum of 10,000 words per revolution of one disk.

The queuer section of the disk file controller receives requests for disk access from the I/O module. These requests consist of a disk address of no more than 25 bits, control bit(s) for read or write operation, and a memory module address of 18 bits to reference the location of the descriptor to be used to control the transfer of data. These bits are sent to the queuer section over a 49-line data bus (48 data bits and one parity bit). The control signals from the I/O module consist of the start line, which informs the queuer that requests are waiting to be sent to the queuer, the strobe signal which synchronizes the 49 data lines, and the "last byte" signal notifying the controller of the last data transfer. When the I/O module has a request waiting to be sent to the queuer, it raises the start line to the queuer. When the queuer is neither full nor busy, it acknowledges the request by sending the I/O module a pulse over the service request line, which sets a flag in the I/O module associated with this channel. The I/O module then responds by fetching the next request from a memory module and by transmitting it to the queuer over the 49 data lines along with the strobe signal. The request to the queuer from the I/O module contains:

(a) The disk address.

(b) The control bit(s) for read/write.

(c) The memory module address.

The disk address is a binary number, the most significant bits selecting the electronics unit and storage module, and the least significant bits selecting the starting address of the segment on a disk within a storage module.

Operational Characteristics

STORAGE FACILITIES

The disk file storage modules are the basic storage media for the data in the mass storage system. Each storage module (there may be up to 100 associated with one controller) can store 2 million words. The primary functions of the storage modules are to:

- Provide the basic rotating storage media for the memory (four double-faced disks).
- Contain read/write amplifiers for storage tracks.
- Contain read amplifiers for clock and address information.

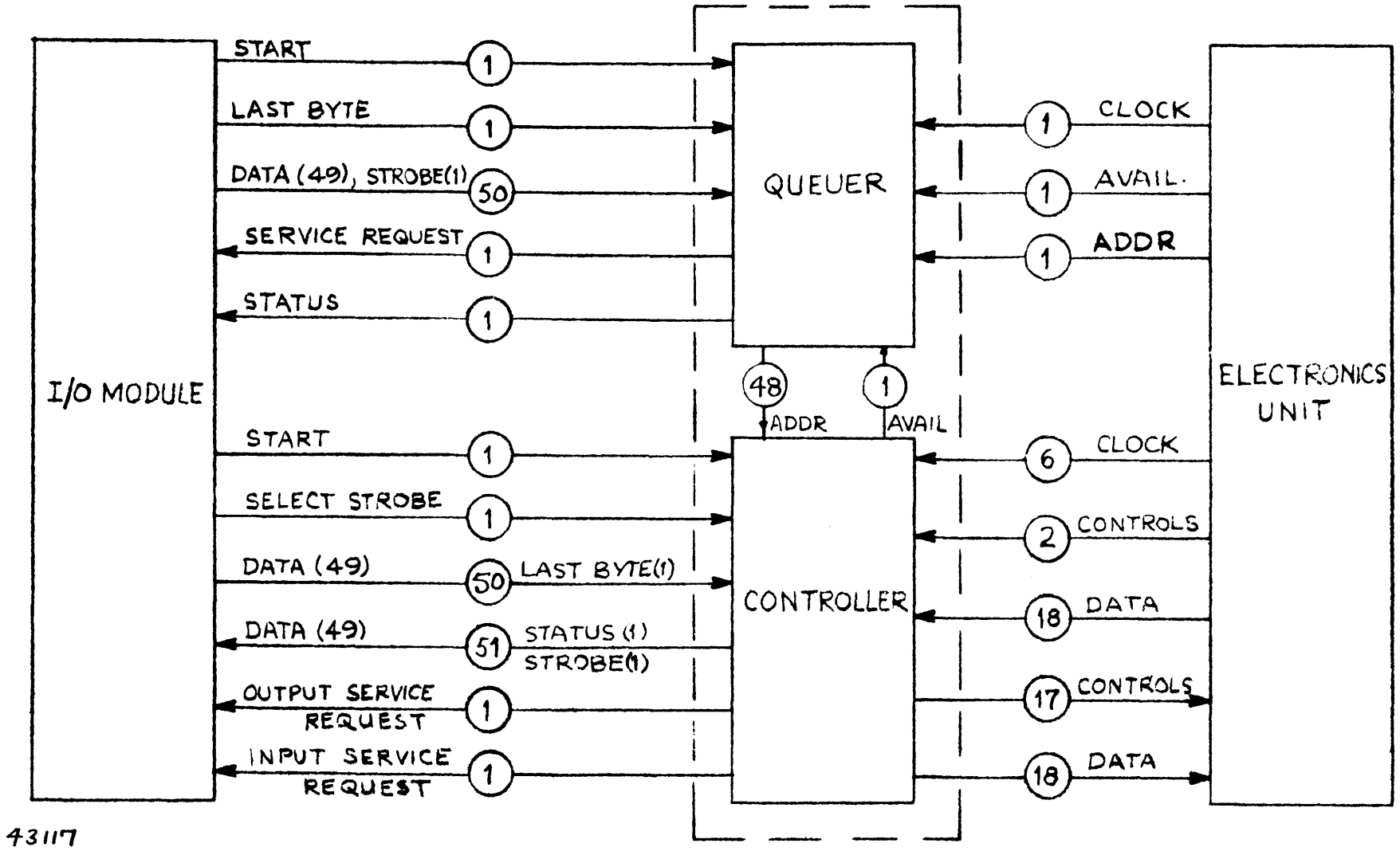


Figure 3-15. Disk File System, Interface Diagram

DISK ALLOCATION

The face of each disk contains 50 tracks; each such track contains 1250 207-bit segments. Each segment of 207 bits provides for four 48-bit words, 3 check bits, 3 clock bits, and 6 space bits distributed as follows:

ZONE	BITS	CLOCK FREQUENCY
1	46	1.0 megacycle
2	69	1.5 megacycles
3	92	2.0 megacycles

One primary bit clock per disk surface (and the associated circuitry) is required to read simultaneously from the two disk surfaces involved in a transfer. Each of the two primary clocks is taken serially from the disk file electronics unit.

The data bit clocks and dead zones are organized such that during a data transfer operation, the respective spacer and data bits of the three zones of one disk face are within one "character time" of the respective spaces and data bits of the three zones of the other disk faces. Because data is transferred to or from two disk faces at a time, eight words (four on each disk face) are located by an address consisting of four binary-coded decimal (BCD) digits. The storage modules are organized such that disk faces 1 and 8, 2 and 7, 3 and 6, or 4 and 5 can be selected as four separate disk sets, the addressing is accomplished as follows:

DISK SET	DISK FACES	ADDRESSES
1	1 and 8	0000 to 1249
2	2 and 7	2000 to 3249
3	3 and 6	4000 to 5249
4	4 and 5	6000 to 7249

The combination of 5,000 addressable segments per track per storage module and 50 tracks per disk face allows 250,000 addressable segments per storage module. Since each segment address defines the location of 8 words, each storage module can store 2,000,000 words.

PARALLEL READING OR WRITING

The reorganization of the standard Burroughs B475 disk storage modules to achieve improved performance for the B8500 System has been accomplished with a minimum of modifications. The basic rotational mechanism, head assemblies, and wiring of the storage module within the disk enclosure remain essentially unchanged. The changes necessary are primarily in the external electronics, and in the number of leads brought through the disk enclosure bulkhead board.

In the standard serial disk file, a word is written or read serially by bit from the disk. This characteristic is still retained in the parallel disk system. However, instead of limiting the reading or writing capability to only a single "stream" of data from a single disk face at any given time, the parallel disk system permits simultaneous reading or writing three streams on each of two disk faces. Therefore, a total of six streams of data are transferred in parallel to or from a given storage module.

MAGNETIC TAPE SYSTEM

General Description

The magnetic tape system is another element in the hierarchy of storage facilities available to the B8500 System. A magnetic tape system consists of the maximum of two magnetic tape controllers and eight magnetic tape units. Each reel of magnetic tape used with the magnetic tape units has a 2.88 million 48-bit word capacity. The average data transfer rate possible is 9,000 words per second. Seven or nine-track tape formats may be implemented in the magnetic tape systems. One of the magnetic tape units used with the B8500 System is the B425 (Figure 3-16), which uses a seven-track format.

The magnetic tape unit is an electromechanical device which, under program control, is capable of reading, writing, backspacing, and erasing magnetic tape. Tape loading and unloading operations are performed manually at each magnetic tape unit. The magnetic tape unit can accommodate 10-1/2 inch reels containing up to 2400 feet of 1/2-inch wide Mylar tape. The tape is driven under a dual-gap read-write head at a speed of 90 inches per second. The data packing density is 800 bits per inch.

The magnetic tape controller, which is housed in the cabinet of the B8515 Controller, is a half-duplex device which permits alternate transmission of data in both directions between a magnetic tape unit and an I/O module. Each magnetic tape controller is capable of controlling a maximum of eight magnetic tape units. However, two controllers are cross-connected to the eight magnetic tape units to allow the simultaneous operation of two magnetic tape units within the same system, and to increase the reliability of the system.

Functional Description

MAGNETIC TAPE CONTROLLER

All functions of the magnetic tape system such as reading, writing, backspacing, erasing, advancing, and rewinding are carried out under control of the magnetic tape controller. Each magnetic tape controller can command the selected magnetic tape unit to respond to the following eleven different instructions for tape transport operations:

(a) Write one record containing up to 16,380 words as specified by the descriptor.

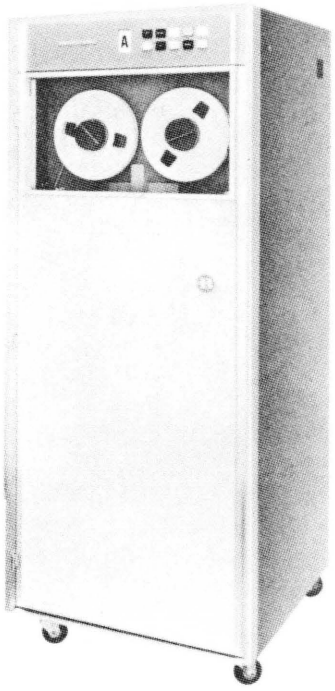


Figure 3-16. B425 Magnetic Tape Unit

(b) Write an end-of-file record. (This is a minimum length record containing an end-of-file character.)

(c) Erase the number of words specified by the descriptor.

(d) Rewind to the load-point marker.

(e) Rewind to the load-point marker, and force the specified tape transport into a lock-out mode.

(f) Read one record in the forward direction, or read the number of words specified by the descriptor. (The magnetic tape controller terminates on end-of-record, or the I/O module terminates on word count equals zero, whichever occurs first. However, the tape unit only stops between records.)

(g) Read, in the reverse direction, one record or the number of words specified by the descriptor. (The stipulations of the preceding instruction apply except that the tape is moving backwards.)

(h) Advance one record, but do not transfer data to the I/O nor check parity.

(i) Advance to the end of the end-of-file record.

(j) Backspace one record, but do not transfer data to the I/O nor check parity.

(k) Backspace to the end-of-file record.

Figure 3-17 is a block diagram of the magnetic tape controller. Descriptions of the subunits shown in Figure 3-17 are given in the following paragraphs.

Line Drivers and Receivers

The line drivers used in the magnetic tape controller are of the standard signal buffer type used in the components of the central processing system. The line receivers are of the standard AND gate type.

MTU Select Register and Parity Logic

The first word that the magnetic tape controller (MTC) receives from the I/O module contains the 3-bit magnetic tape unit (MTU) selection code. These three bits are channeled into the priority logic circuits which check a selected MTU to determine if it is already in use or in the process of being selected by an alternate controller. The logic circuitry then decodes the three bits into eight discrete outputs, one of which selects the magnetic tape unit to be used in impending operations.

Operation Register and Decoder

The first word received by the MTC contains a 4-bit operation code which is channeled into the operation register. The operation decoder then selects one of the eleven possible operations to be performed by the magnetic tape unit.

Level Converters and Select Gates

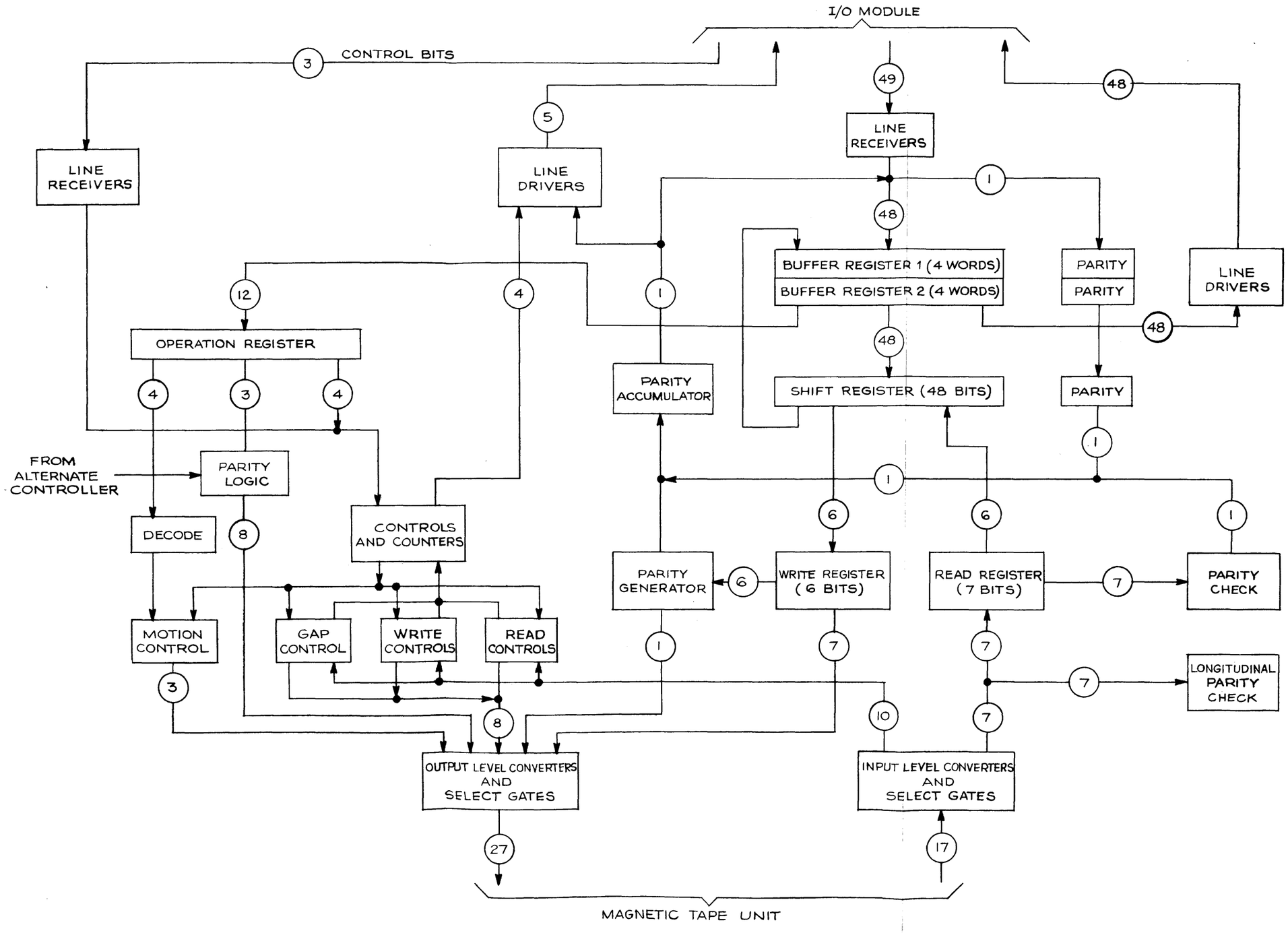
All information transferred between the MTC and the MTU passes through level converters and select gates. The output level converters convert MTC logic levels into levels compatible with MTU signals. The input level converters convert signals coming from the MTU to levels compatible with the MTC levels.

Data Handling Registers

BUFFER REGISTER 1. Buffer register 1 receives four words of data from either the I/O module or the shift register. When buffer register 1 is fully loaded, the data are then automatically transferred to buffer register 2, and buffer register 1 is then ready to accept four more words of data.

BUFFER REGISTER 2. Buffer register 2 receives four words of data from buffer register 1 and then transfers the data, one word at a time, to either the shift register or the I/O module.

SHIFT REGISTER. During a write operation, the shift register receives a 48-bit word from buffer register 2. The word is moved through the register six bits at a time and then transferred to the write register. During a read operation, data is transferred from the MTU through the read register six bits at a time until the shift register is full. The word in the shift register is then transferred to buffer register 1.



43119

Figure 3-17. Magnetic Tape Controller, Block Diagram

WRITE REGISTER. The write register receives a 6-bit character from the shift register. Parity is generated on these bits and a 7-bit character is sent to the MTU.

READ REGISTER. The read register receives a 7-bit character from the MTU. Parity is checked on these bits and then the parity bit is stripped from the character. The six remaining bits of data are then transferred to the shift register.

Character Parity

During a write operation, the shift register sends 6-bit characters, one character at a time, to the write register. Odd or even parity (dependent on whether the alpha or binary mode is selected) is then generated. During a read operation, the MTU transfers 7-bit characters, one character at a time, to the read register. Odd or even parity is then checked.

Word Parity

Word parity is generated during both read and write operations. The parity is checked during a write operation by accumulating character parity and then comparing accumulated parity with parity sent from the I/O module. Parity is generated during a read operation by accumulating character parity and generating word parity after the receipt of eight characters. Parity is checked on the Control Word in a manner similar to the manner of checking parity during a write operation.

Controls and Counters

The combinations of controls and counters initiate and regulate all transfers of data and control signals between the I/O module and the MTC, and between the MTC and the MTU.

The I/O module then energizes the select strobe signal to strobe the Control Word into the MTC and updates its descriptor by resetting the control-word bit.

With the Control Word in its registers, the MTC determines if the selected tape unit is unoccupied by sensing if its appropriate ready bit is true. If a write operation is to be performed, the write ready level is only true if the file protection ring is on the tape reel. If the MTU is ready for data transfers, the operation specified by the I/O descriptor is started by energizing the appropriate motion controls: forward, reverse, or rewind.

WRITE OPERATION

If a write operation is specified by the descriptor, the MTC enables the write service request line to the I/O module, signifying that the MTC is ready to receive four data words for storing on tape. When a data word is on its output data lines, the I/O module energizes its select strobe signal to enable the data transfer to the MTC. This operation continues until four 48-bit words (plus parity bits) have been transferred. During error-free operation, the MTC has no further communication with the I/O module until more data is required.

When all data words are in the buffer registers and moved down to the shift register, the MTC transfers the data, six bits at a time, to the selected MTU, and a parity bit is generated. When the write control lines from the MTC are energized, the character is written on tape, and the write control signals are de-energized. The write process continues until the "last byte" signal is sensed. At this time, a three-character gap is erased, and the longitudinal parity character is written. After a read-after-write check is complete the MTC then energizes the stop motion control signals and then sends a service request to the I/O module before sending a status indicator to the I/O module. The MTC monitors its write operations by utilizing the read-after-write capabilities of the magnetic tape units.

READ OPERATION

If a read operation is specified by the I/O descriptor, the MTC samples the seven lines (data and parity) from the MTU by using the read clock, checks the lateral parity of the characters, and accumulates the longitudinal parity (so that a longitudinal redundancy check can be made at the end-of-record). A new character is strobed into the MTC with each read clock signal. As soon as it assembles a 48-bit data word, the MTC transfers the word to the buffer register. After four words have been transferred to the buffer register, the MTC initiates a read service request to the I/O module. If the I/O module is ready to accept a four-word data byte, it responds to the MTC by sending a select signal to the MTC.

When the final character is sampled by the MTC, it terminates the tape reading process and energizes the stop motion control. If the I/O module reads data which is less than a full record, the start level will be lowered when the final four-word "byte" is received. The MTC will continue operation to the end-of-record but the only status sent to the I/O module will be the status detected and reported while the start line was high. If a lateral parity error occurs, the MTC will (1) stop sending data until the end-of-record and then send the error status, or (2) a question mark (?) will be inserted in place of the character containing the bad parity and the error status will be reported at the end-of-record. (No character which contains bad parity will be passed on to the I/O module.)

METHODS OF TERMINATIONS

An I/O module to MTC data transfer operation may be terminated by one of two methods; by lowering the start line level from the I/O module, or by raising the status line level from the MTC and energizing the read service request line. The I/O module always terminates an operation when its word count becomes 0. If, however, the MTC senses an end-of-record (with or without parity error) or an "end of file", it prepares to send the coded reason for termination over the data lines to the I/O module.

Interface Characteristics

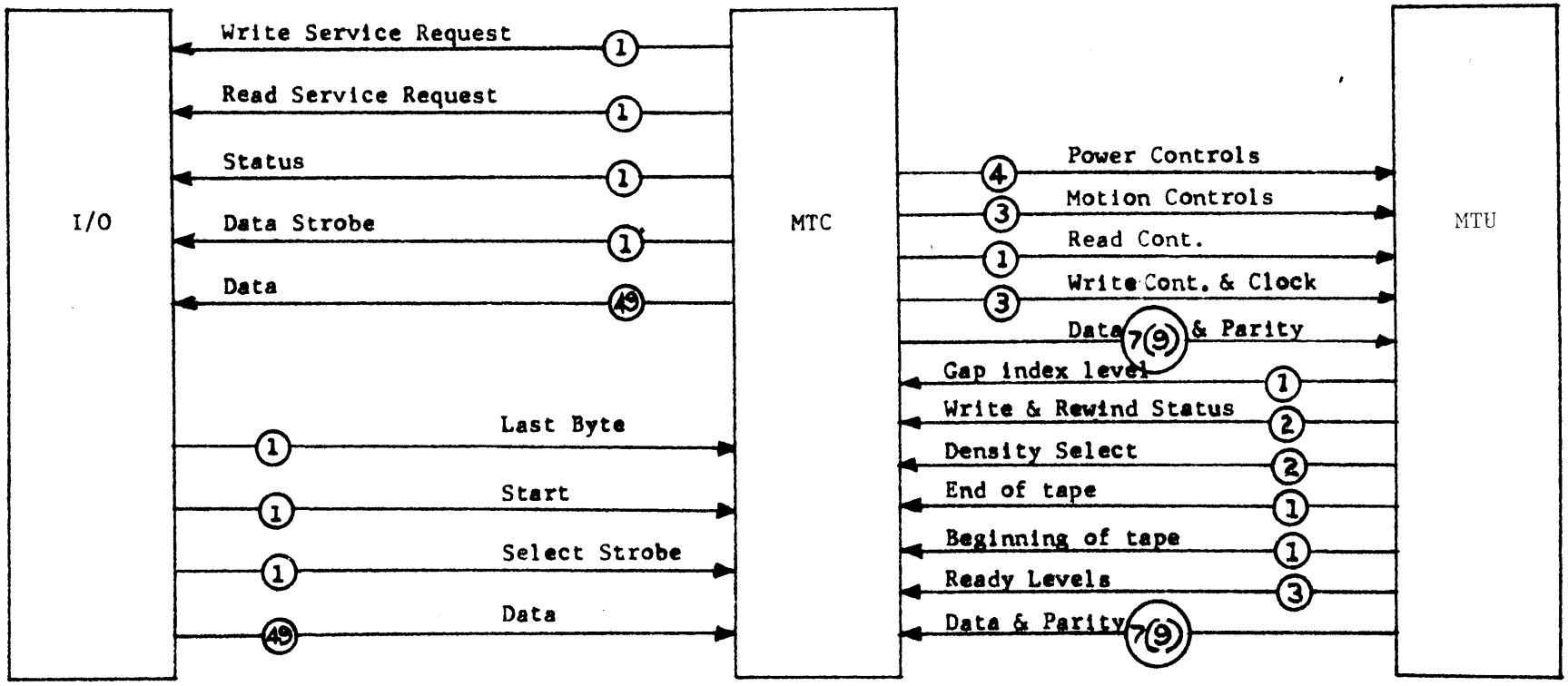
Figure 3-18 shows the interface configuration of the magnetic tape subsystem.

Each magnetic tape controller (MTC) has its own descriptor address in the descriptor area of the I/O module. When a new descriptor is initiated, the control-word bit of the descriptor is set, and the I/O raises its start line to the selected MTC. The MTC, if available, responds with a write service request to the I/O module. The I/O module services this request by placing a Control Word on what is normally its output data lines. The format of the Control Word is shown in Table 3-3.

TABLE 3-3

FORMAT OF CONTROL WORD FOR MAGNETIC TAPE OPERATIONS

Bits	Function
2-5	Specifies which operation is to be performed
6-8	Selects one of the eight magnetic tape units
10 and 11	Selects one of the three recording densities or specifies operator selection of density
14	Specifies ALPHA or BINARY character mode
17	Specifies whether or not to terminate operation upon detection of character parity error. (When operation is not terminated because of a parity error, a question mark (?) is substituted for the character indicating the error.)
21	Specifies the internal or external type of format



F3120

Figure 3-18. Magnetic Tape System, Interface Diagram

Operational Characteristics

Operational characteristics of the magnetic tape unit are summarized below:

- Processes 10-1/2-inch reels containing 2400 feet of one-half inch wide tape
- Reads and writes tape at 90 ips
- Recording density is 200, 556, or 800 frames per inch (as required)
- Data capacity is approximately 22. 1 million alphanumeric characters per reel
- Alphanumeric transfer rate is 72, 000 characters per second
- Start time is 5 milliseconds
- Start/stop time is 10 milliseconds
- Utilizes a dual-gap head with nonreturn-to-zero recording
- High-speed rewind at over 320 inches per second
- Quick-action reel locks facilitate reel mounting and removal
- File rings provide automatic file protection
- Automatic tape positioning and use of a permanent leader simplify and speed loading
- Compatible with other magnetic tape systems recording 200, 256, or 800 frames per inch on half-inch tape

In general, the magnetic tape unit reads or writes data in either binary or single-frame alphanumeric format. The tape format may be made compatible with IBM Model 729-IV magnetic tapes. The recorded format consists of seven longitudinally recorded parallel tracks with each character recorded as a lateral set of bits, one bit on each track. One track contains bits which provide a parity check for each character. Automatic checking in the write mode of operation is made possible by utilizing a dual-gap read/write head.

CARD READERS

General Description

Two models of card readers are available for use with the B8500 Data Processing System, the B124 and the B129. The B124 Card Reader reads cards at an 800 card-per-minute rate and the B129 Card Reader reads cards at a 1400 card-per-minute rate. Because both card readers are identical in physical appearance and operation, only the B129 Card Reader (see Figure 3-19) is discussed in detail.

The B129 Card Reader is contained in a single, free-standing cabinet which is approximately 50 inches high, 48 inches long, and 29 inches deep. The total weight of the card reader is approximately 920 pounds; rubber casters on the cabinet facilitate positioning the card reader at any location. The card reader is a self-contained unit which contains its own power supplies and logic and control circuitry.

The hopper and stacker of the card reader each have the capacity of approximately 4000 cards of standard thickness. Vibrators are provided in the floors of the hopper and stacker to help reduce friction and to eliminate the need for joggling the cards. Both the hopper and stacker have adjustable rails to permit the feeding of either 51, 60, 66, or 80 column cards. Card stock of post card thickness can be read. However, cards of different lengths or thicknesses cannot be intermixed during any one run. Cards punched column binary can be read by depressing the LOAD button on the control panel. The reader is designed for heavy-duty operation, with high production, and a small amount of maintenance. Card jams are extremely rare.

A removable front cover permits access to a logic gate, a blower motor, and a cooling fan. A hinged top cover permits access to the card transport mechanisms to facilitate the cleaning of the card path. Operating controls and indicators are located on a panel at the top of the cabinet, and a maintenance panel is located on the logic gate.

Figure 3-19. B129 Card Reader

Functional Description

The B129 Card Reader is a high-speed, electro-mechanical device capable of reading punched cards at a rate of 1400 cards-per-minute. Under system operation, the card reader feeds one card at a time on command from the card reader controller in the B8515 Controller Module.

A card is fed in a vertical position, and read serially, one column at a time. The reading of a card is accomplished photoelectrically by photodiodes (solar cells). Each character from the card is checked to determine if it is valid; then the standard card code is translated into the six-bit binary-coded-decimal character and is supplied to the input-output channel a character at a time. Invalid characters are sensed and replaced by the "?", and an error indication is supplied to the processing system. Binary punched cards (six bits) are read column by column. Because there are 12 bit positions in a card column, two binary characters occupy each card column.

The card reader constantly monitors for card jams in the feed and read areas. If a jam occurs, the condition is immediately sensed, and card feeding is stopped.

Interface Description

The interfacing of the card readers with the Central Data Processing System is accomplished by the use of a Card Reader Controller. The Card Reader Controller, which is housed in the B8515 Controller Module, contains the necessary circuitry to provide compatibility and buffering between the card readers and the I/O modules. A description of the interfacing and information flow between peripheral devices and the Central Processor is provided later in this manual.

CARD PUNCHES

General Description

Two types of card punches are available for use with the B8500 Data Processing System; the B303 card Punch and the B304 Card Punch. Both card punches feed, punch, check, and stack 80-column cards. The B304 can punch both standard and post card thicknesses. The B303 punches cards at a maximum rate of 100 cards per minute, and the B304 punches cards at a maximum rate of 300 cards per minute.

Figure 3-20. B303 Card Punch

PHYSICAL DESCRIPTION OF B303 CARD PUNCH

The B303 Card Punch (Figure 3-20) is contained in a free-standing cabinet which is approximately 53 inches high, 44-1/2 inches wide, and 28 inches deep. The total weight of the card punch is approximately 910 pounds; rubber casters facilitate positioning the card punch at any location. The card punch is a self-contained unit which contains its own power supplies, logic and control circuitry.

The cabinet is divided into two sections; the upper section, which contains electromechanical components, and the lower section, which contains the power supply, logic, and control components. Removable covers permit access to internal components and subassemblies through both the front and rear of the cabinet. Operating controls and indicators are located on a panel at the top front of the cabinet in front of the stacker mechanism. The hopper and the stacker in the card punch will each hold up to 800 cards. Cards may be added to the hopper or removed from the stacker while the unit is operating.

PHYSICAL DESCRIPTION OF B304 CARD PUNCH

The B304 Card Punch (Figure 3-21) is contained in a free-standing cabinet which is approximately 47 inches high, 73 inches long, and 27-1/2 inches deep. The total weight of the card punch is approximately 1283 pounds; the cabinet is supported by four adjustable feet. The card punch is a self-contained unit which contains its own power supplies, logic and control circuitry.

Cards are loaded into a hopper located to the left of the control panel. The hopper is comprised of a main hopper and an auxiliary hopper. The main hopper (the lower of the two) has a maximum capacity of 450 cards. The auxiliary hopper has a maximum capacity of 3000 cards and is used to maintain a constant level of cards in the main hopper. Cards are placed into the hoppers face down, "12" edge feeding first.

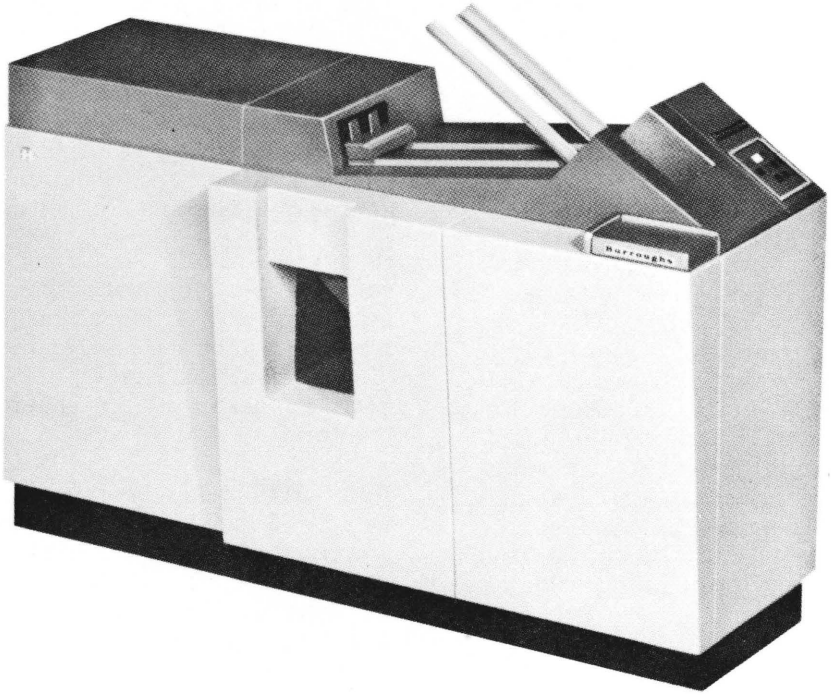


Figure 3-21. B304 Card Punch

Output cards are sorted into one of three stackers; a main stacker, an auxiliary stacker, and an error stacker. The main stacker is located along the top cover of the cabinet and is used during normal operations. The auxiliary stacker is located at the left side of the cabinet and contains cards "out-sorted" there under program control. The error stacker, located to the right of the auxiliary stacker and near the front of the card punch, contains cards detected to be in error by the card punch read check station. Access to the auxiliary and error stackers is gained through the opening in the left side cover of the cabinet.

Removable covers permit access through the sides of the cabinet to the electromechanical mechanisms, power supplies, and logic circuitry. Operating controls and indicators are located on a panel on the front of the cabinet, just to the right of the hoppers.

Functional Description

The card punches are electromechanical devices which provide alphanumeric, 80-column card outputs for the B8500 Data Processing System. The cards are automatically punched, under control of the processing system, one card at a time. Cards may be gang-punched, however, under local control.

Because the card punches do not have the capability of storing all of the information necessary to punch a card, all the information for one card is first stored in the controller and then transferred to the punch one row at a time. Cards are punched one row at a time beginning with row 12 on a card, by utilization of an electronic row buffer. The information is punched in standard Hollerith card code. Timing pulses, used to control the transfer of the information, are generated by the card punches and sent to the processing system. After a card has been completely punched, a hole count is made by the card read station of the punch in order to verify that the correct information has been punched.

The B303 Card Punch utilizes an immediate access clutch, whereas the B304 Card Punch utilizes a six-point clutch.

Interface Description

The interfacing of the card punches with the Central Data Processing System is accomplished by the use of a Card Punch Controller. The Card Punch Controller, which is housed in the B8515 Controller Module, contains the circuitry to provide compatibility and buffering between the card punches and the I/O Modules. A description of the interfacing and information flow between peripheral devices and the processing system is provided later in this manual.

LINE PRINTER

General Description

The B329 Line Printer (Figure 3-22) is a drum type printer capable of printing 37 alphanumeric characters at a maximum rate of 1040 lines per minute, or 64 characters (37 alphanumeric and 27 special characters) at a maximum rate of 700 lines per minute. Any one of the 64 characters can be printed in each of 132 printing positions.

The B329 Line Printer is contained in a two-section cabinet which is approximately 55 inches high, 76 inches wide, and 30-1/2 inches deep. The total weight of the unit is approximately 1750 pounds; rubber casters are provided on the cabinet to aid in the positioning of the line printer. The line printer is a self-contained unit which contains its own power supplies, operating controls and indicators, a buffer memory, and the necessary logic and control circuitry.

Removable panels permit access to the interior components of the line printer through the front, rear, and side of the cabinet. Hinged top and front covers on the printer facilitate the handling of paper and carriage tape loading operations and permit access to the mechanical sections of the printer.

Printing is done on continuous forms which may be from 5 inches to 20 inches in width, including the marginal punched strips. Maximum length of each form section can be 22 inches (at six lines per inch) or 16-1/2 inches (eight lines per inch). Forms are placed in the bottom of the cabinet and transported by pin-feed tractors through the unit to a stacker at the rear of the unit. As many as five carbon copies plus an original may be printed on each form.

Functional Description

The B329 is a high-speed electromechanical device which can print 700 (64 characters) or 1040 (37 characters) single-spaced lines per minute. The line printer is fully buffered through a self-contained 132-character buffer memory. The graphics have a horizontal spacing of 10 characters per inch. Vertical spacing is either six or eight lines to the inch and is controlled by the operator. All formating, editing, form skipping, and spacing are done under program control, via the line printer controller.

Information is transferred from the processing system through the line printer controller into the printer buffer memory. The information is transferred in parallel by bits and serially by character. A line is printed only after the last character is loaded into the buffer memory. A command level from the controller then transfers the format instructions (space or skip) and initiates the action to print the contents of the memory buffer.

The paper carriage is controlled by the coordinated action of a clutch and two brakes, which are actuated by the format instructions received from the controller. The vertical spacing format may be a single-space, double-space, or a skip to any one of eleven pre-determined positions punched on a carriage control format tape used with the line printer. A twelfth position punched on the tape is used exclusively as an end-of-page indicator.

As each print command is being executed, the buffer memory can be filled by the controller during the paper motion involved with the preceding command. The controller and the line printer are interlocked during and after the buffer loading operation until the print command and formating levels are received from the controller. These levels are received only after the paper motion has stopped for the preceding operation; then the controller is released from the printer.

Interface Description

The interfacing of the line printer with the Central Data Processing System is accomplished by the use of a Line Printer Controller. The Line Printer Controller, which is housed in the B8515 Controller Module, contains the circuitry to provide compatibility and buffering between the line printer and the I/O modules. A description of the interfacing and information flow between peripheral devices and the Central Data Processing System is provided later in this manual.

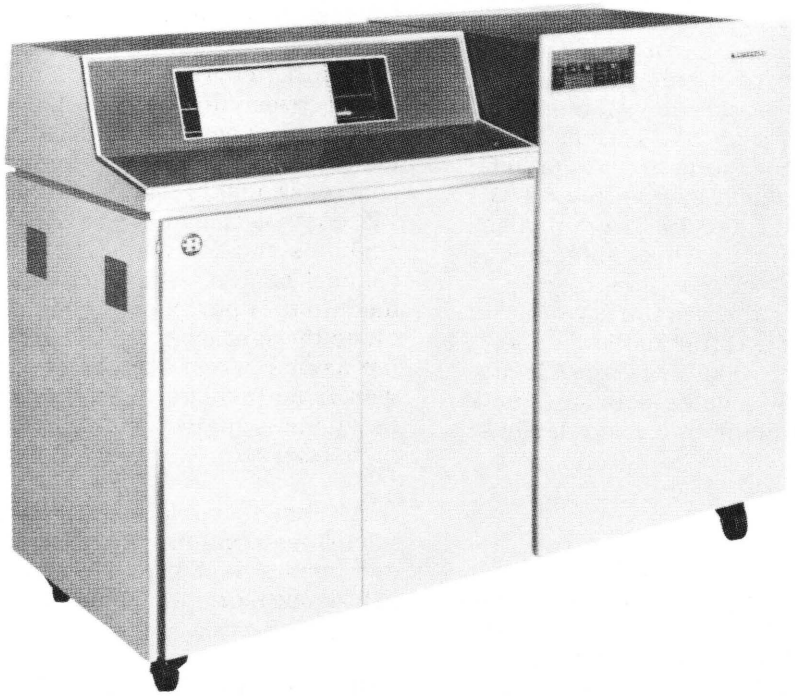


Figure 3-22. B329 Line Printer

EQUIPMENT INTERFACING

General Description

Various quantities of central processor modules, memory modules, I/O modules, controllers, and peripheral devices can be integrated into configurations to meet the requirements of specific applications of the B8500 System. As shown in Figure 1-2, the processor, I/O modules, and the console are interfaced with the memory modules by data transfer busses. Each bus handles 52 parallel input data lines, 52 parallel output data lines, and the necessary number of control lines. A maximum of 16 busses may be used to interface a system; 15 to interface the memory modules with the processor and I/O modules, and one to interface the memory modules with the memory check module in the B8500 Console.

All interfacing is accomplished through the use of units within the various modules. Figure 3-23 shows the interfacing between the units, the configuration of the units, and the signal flow in the system. The units contain the necessary fetch, store, and address registers, logic circuits, timing and control circuits, drivers, and receivers to coordinate the transfer of data between the modules.

Each I/O module has the capability of interfacing with 512 simplex (one-way) peripheral devices over 512 individual control channels (256 input and 256 output) as well as 64 data transfer busses (32 input and 32 output). Peripheral equipment controllers (contained in common B8515 Controller Modules) are used to provide the proper interfacing of data and control signals between the I/O modules and the various peripheral devices. These controllers supply the proper amount of data buffering to ensure that no data is lost while an I/O module is servicing other channels. Communications modules are used specifically to interface I/O modules with communications lines (such as teletype lines) to minimize the possibility of a single component failure disabling more than a single communication line. The interface circuitry in the communications modules provides for compatibility between the communication devices and the I/O modules.

Information Flow

Information flow between units in the B8500 System involves three phases; input, processing, and output. The main thin-film memory (in the memory modules) is a high-speed random access, temporary storage device for information to be handled during each of the three phases. During the input phase, information is transferred from the input peripheral devices to the main memory by way of the peripheral controllers, I/O Modules, and Disc Sub-system. During the processing phase, the information in main memory is transferred to and manipulated by a processor, upon completion of which the data is returned to main memory. During the output phase, the information is taken out of main memory and handled as output information, traveling the reverse route of the input data flow.

COMMUNICATIONS BETWEEN PERIPHERAL EQUIPMENT AND I/O MODULES

Communications between peripheral equipment and the I/O modules is handled by peripheral equipment controllers, communications modules, and the data service units in the I/O modules. The data service units contain the necessary registers, buffers, and control circuits to manipulate input/output information of various byte sizes and speeds. Each of the 64 data busses has the capability of handling, in parallel, a byte size of 51 data bits plus one parity bit. The byte size of data handled is determined by the type of peripheral device used. For example, if an I/O module is communicating with 8-to-100 word-per-minute Teletype lines, a 6-bit byte size would be sufficient to ensure that each of the individual Teletype lines is serviced without the loss of data.

The peripheral equipment controllers contain the necessary drivers, receivers, registers, parity check and generating logic, buffers, converters, and timing control circuits to buffer and transfer data between devices and I/O modules. Communications between I/O modules and peripheral equipment controllers is accomplished by the use of 512 control channels and 64 data transfer busses. The 512 control channels are divided into 256 input channels and 256 output channels. The output channel control signals transmitted from an I/O module to a peripheral equipment controller include the last byte signal, the start signal, and the device select strobe. Output control signals sent from peripheral equipment controllers to the devices include status and service request signals.

Input control signals sent from I/O modules to peripheral equipment controllers include the start and input select signals, and a service signal is sent back from peripheral equipment controllers to the I/O modules.

Data word transfers on each of the 32 output busses include 48 data bits and one parity bit. Data transfers on each input bus include 48 bits, one parity bit, and a status strobe.

COMMUNICATIONS BETWEEN I/O MODULES AND CENTRAL PROCESSORS

As shown in Figure 1-4, communications between processors and I/O modules are handled by the communications units in each processor, and by the processing unit in each I/O module. The communications unit in a processor contains the necessary drivers and receivers to drive signal lines to the I/O modules and to receive and standardize signals from the I/O modules. The I/O processing units contain registers and control circuits which control the input/output communications with a processor. Each set of data transferred from a processor to an I/O module includes 18 bits for the job stack address, one bit for the job stack flag, and one strobe bit. Each set of data transferred from an I/O module to a processor module includes the I/O-complete interrupt, parity error interrupt, and the memory access interrupt bits.

COMMUNICATIONS BETWEEN PROCESSORS OR I/O MODULES AND MEMORY MODULES

Communications between the Processors or I/O modules and the memory modules is handled by the communications unit in each processor, I/O, and memory module. The communications unit in the processor and I/O modules contains:

a. Fetch and store registers which buffer information transferred to and from memory modules.

b. Address registers which buffer the memory module addresses.

c. Parity check and generating logic circuits which check parity of data received from memory modules and generate parity bits for data sent to memory modules.

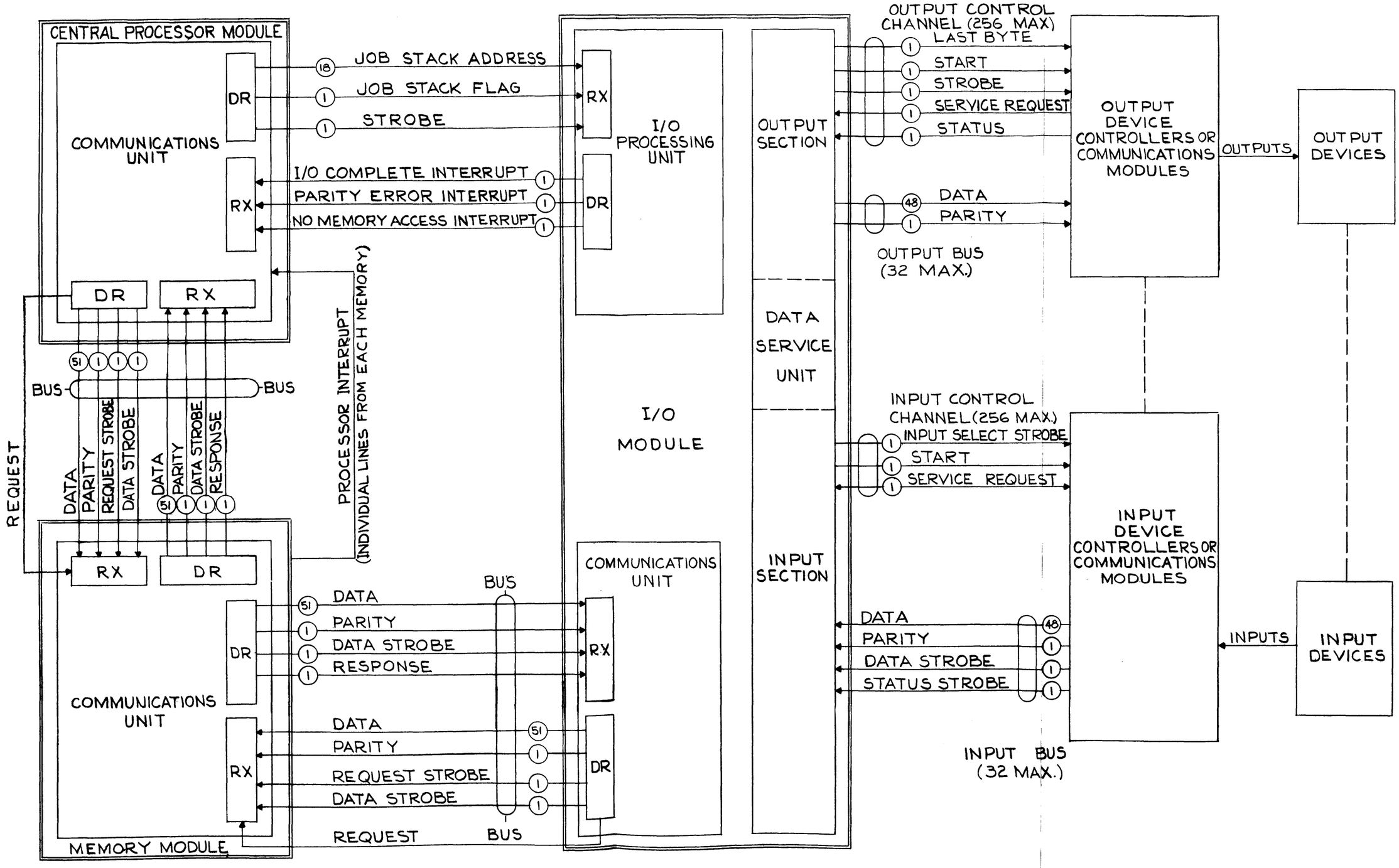
d. Timing and control circuits which control sequential operations required by the communications processes.

e. Drivers which drive signal lines to the memory modules.

f. Receivers which receive and standardize signals received from memory modules.

The communications unit in the memory modules contain the necessary drivers and receivers to handle data received from and transmitted to the processor and I/O modules. Fifteen data transfer busses are provided in the system to interconnect the processor and I/O modules with the memory modules. Each bus has 52 parallel input data lines, 52 parallel output data lines, and the necessary number of control lines.

Data transferred (through each bus) from a processor or I/O module to each memory module include 51 data bits, one parity bit, one request strobe, and one data strobe. Data transferred from the memory modules to a processor or I/O module include 51 data bits, one parity bit, one data strobe, one response signal, and one interrupt signal. In addition, a request signal is sent (over an individual line) from each I/O or processor module to each of the memory modules, and a processor interrupt signal is sent from each memory module to each processor module.



43125

Figure 3-23. System Interface and Information Flow Diagram

APPENDIX A

B8500 CHARACTER SET

B8500			PUNCHED CARD COMBINATION
INTERNAL BA 8421	OCTAL	PRINTER GRAPHICS	
00 0000	00	blank	None
00 0001	01	A	12, 1
00 0010	02	B	12, 2
00 0011	03	C	12, 3
00 0100	04	D	12, 4
00 0101	05	E	12, 5
00 0110	06	F	12, 6
00 0111	07	G	12, 7
00 1000	10	H	12, 8
00 1001	11	I	12, 9
00 1010	12	≥	12, 8, 2
00 1011	13	. (period)	12, 8, 3
00 1100	14	<	12, 8, 4
00 1101	15	(	12, 8, 5
00 1110	16	+	12, 8, 6
00 1111	17	← (tape mark)	12, 8, 7
01 0000	20	&	12
01 0001	21	J	11, 1
01 0010	22	K	11, 2
01 0011	23	L	11, 3
01 0100	24	M	11, 4
01 0101	25	N	11, 5
01 0110	26	O	11, 6
01 0111	27	P	11, 7
01 1000	30	Q	11, 8
01 1001	31	R	11, 9
01 1010	32	x (multiply)	11, 8, 2
01 1011	33	$	11, 8, 3
01 1100	34	*	11, 8, 4
01 1101	35	)	11, 8, 5
01 1110	36	; (semi-colon)	11, 8, 6
01 1111	37	≤	11, 8, 7
10 0000	40	- (minus)	11
10 0001	41	/	0, 1
10 0010	42	S	0, 2
10 0011	43	T	0, 3
10 0100	44	U	0, 4
10 0101	45	V	0, 5
10 0110	46	W	0, 6
10 0111	47	X	0, 7
10 1000	50	Y	0, 8
10 1001	51	Z	0, 9
10 1010	52	≠	12, 11
10 1011	53	, (comma)	0, 8, 3
10 1100	54	%	0, 8, 4
10 1101	55	]	0, 8, 5
10 1110	56	>	0, 8, 6
10 1111	57	? (See Note 1.)	0, 8, 7
11 0000	60	0	0
11 0001	61	1	1
11 0010	62	2	2
11 0011	63	3	3
11 0100	64	4	4
11 0101	65	5	5
11 0110	66	6	6
11 0111	67	7	7
11 1000	70	8	8
11 1001	71	9	9
11 1010	72	: (colon)	8, 2
11 1011	73	#	8, 3
11 1100	74	@	8, 4
11 1101	75	[	8, 5
11 1110	76	=	8, 6
11 1111	77	" (quotes)	8, 7

Tape Mark - Used on B8500 as EOF record.

NOTE 1. All other punched card combinations are "mapped" into this character.

APPENDIX B

CENTRAL PROCESSOR INSTRUCTIONS

OPERATOR SYLLABLE 1		ADDITIONAL SYLLABLES	MEANING
MNEUMONIC	OCTAL	2nd 3rd 4th	
ADD	40		Add (Floating)
ADDM	50		Add Integer Magnitude
ARIT	53	Vd	Double Precision Arithmetic Operations
AND	42		Logical AND
BSR	74	A	Load Barrel Shift Register
CBB	33	Vc	Convert Binary Coded Decimal to Binary
CLRF	56	L, S	Clear Field
COMF	66	L, S	Complement Field
COMP	70		Complement
DIV	61		Divide
DIVI	71		Divide for Integer Quotient
DUP	01		Duplicate Top of Stack
ESP	14	I	Enter Executive and Scheduling Program
ETB	20		Extract tag bits
EXT	75	L, S	Extract Field
EXTD	35	L2, S, L1	Extract Field Double
FAS	55	L1, L2	Fetch Address Register to Stack
FINQ	67		Final Queue Empty
FILF	76	L, S	Fill Field
FMA	17	A1, A2, B	Fetch Memory to Address Register
FMC	27	A1, A2, B	Fetch Memory Conditionally
FMS	05	A1, A2, B	Fetch Memory to Stack
FMSA	10		Fetch Memory to Stack Absolute
FMT	36	A1, A2, B	Fetch Modify Tag
FRS	03	R	Fetch Register to Stack
ICN	04	N	Interrupt Computer N
IMP	52		Implication
INS	65	L, S	Insert Field
INSD	25	L2, S, L1	Insert Field Double
INT	31		Integerize
IOP	73	T	Initiate Input Output Program
IRR	32		Interrupt Routine Return
ITB	22		Insert tag bits
JFT	26	L, S, Vt	Jump on Field Test
JSTA	34	Vt	Jump on Stack Test Arithmetic
JSTL	24	Vt	Jump on Stack Test Logical
JXMT	16	A1, A2, Vi	Jump on Index Modify and Test
MUL	60		Multiply
NORM	30		Normalize
NOP	57		No operation
OR	72		Logical OR
ORX	62		Exclusive OR
RND	21		Round
RTS	13	Vs	Rearrange Top of Stack
SHF	63	V4	Shift

OPERATOR SYLLABLE 1		ADDITIONAL SYLLABLES	MEANING
MNEUMONIC	OCTAL	2nd 3rd 4th	
SJ	07	A1, A2, V_j	Set up Jump
SLIT	23	LIT	Short Literal
SRR	11		Sub Routine Return
SSM	06	A1, A2, B	Store Stack to Memory
SSMA	02		Store Stack to Memory Absolute
SSR	43	R	Store Stack to Register
STOP	77, 00		Stop Processor
SUB	41		Subtract
SUBM	51		Subtract Integer Magnitude
X	45	A1, A2	Index
XM	46	A1, A2	Index and Modify Index
XS	12		Index by Top of Stack

*Address and Variant Syllables

A - Shift Amount

A_1, A_2 - Relative Address

B - Base Register Specifier

I - Relative Address (for ESP Instruction)

L - Length of Field

L_1, L_2 - Literal

LIT - Short Literal

N - Computer Module Number

R - Register Specifier

S - Start of Field

T - I/O Module Number

V4 - Shift Variant

Vc - Conversion Variant

Vd - Double Length Arithmetic Variant

Vi - Index Modifier Variant

Vj - Jump Variant

Vs - Stack Manipulation Variant

Vt - Jump Test Variant

*For further definition, see Appendix C.

FETCH AND STORE INSTRUCTIONS

05 - FMS A_1, A_2, B -Fetch Memory to Stack

The stack is pushed down once. The contents of the memory location specified by the sum of AAR + A1A2 + a base register, designated by B, is placed in the top of the stack. At the completion of the operation the AAR is cleared if specified by the B syllable.

10 - FMSA -Fetch Memory to Stack Absolute

The stack is pushed down once. The contents of the memory location specified by the AAR is placed in the top of the stack. At the completion of the operation the AAR is cleared.

06 - SSM A_1, A_2, B -Store Stack to Memory

The top of the stack is stored into the memory location specified by the sum of AAR + A1A2 + a base register, designated by B. The stack is stepped up once and the AAR is cleared if specified by the B syllable.

02 - SSMA -Store Stack to Memory Absolute

The top of the stack is stored into the memory location specified by the AAR. The stack is stepped up once and the AAR is cleared.

03 - FRS R -Fetch Register to Stack

The stack is pushed down once. The contents of the register specified by the sum of the AAR + R is placed in the top of the stack. At the completion of the operation the AAR is cleared.

43 - SSR R -Store Stack to Register

The top of the stack is placed in a register specified by the sum of AAR + R. The stack is stepped up once and the AAR is cleared.

LITERALS

55 - FAS L_1, L_2 -Fetch Address Register to Stack

The stack is pushed down once. The sum of the AAR + L1L2 is placed in the 18 least significant data bit positions of the top of the stack. The 30 most significant data bits of the top of the stack are filled with zeroes. At the completion of the operation the AAR is cleared.

23 - SLIT LIT -Short Literal to Stack

The stack is pushed down once. The six bits contained in the LIT syllable are placed in the 6 least significant bit positions of the top of the stack. The 42 most significant bit positions of the top of the stack are filled with zeroes.

INDEX INSTRUCTIONS

45 - X A_1, A_2 - Index

The index field of the word specified by A1A2, relative to the BXR, is added to the AAR. The sum is placed in the AAR.

46 - XM A_1, A_2 -Index and Modify Index

The index field of the word contained in the memory location specified by A1A2 relative to the BXR is added to the AAR, with the sum being placed in the AAR. The index field is incremented or decremented by the increment amount field and stored.

12 - XS -Index by Stack

The 18 least significant data bits of the top of the stack are added to the AAR. The result is placed in the AAR, and the stack is stepped up once.

INDIRECT ADDRESSING

17 - FMA A_1, A_2, B -Fetch Memory to Address Register

Test the tag bits of the word specified by AAR + A1A2 + the base register designated by B.

NOTE: The following tag responses are made only if the B syllable designates the PRT as the base, or the AB flag is set. Otherwise, any tag configuration will be treated as a No-Operation.

If the tag bits indicate an indirect address, fetch the contents of the location specified by the address field in the tagged word and test the tag bits.

If alternate bounds word is indicated (by the tag bits), place the contents of the address field into the Alternate Bounds Lower register (ABL) and the contents of the limit field into the Alternate Bounds Upper register (ABU). The displace field + ABL is placed in the AAR.

If a jump is indicated, jump to the program segment and instruction specified by the contents of the tagged word.

If a No-Operation is indicated, the 18 least significant bits of the tagged word is placed in the AAR.

27 - FMC A_1, A_2, B -Fetch Memory Conditionally

The FMC instruction differs from the FMA instruction in that if a No-Operation or AB tag is found, the stack is pushed down and the contents of the memory location specified by the tagged word, or ABL + displace, is placed in the top of the stack. At the completion of the operation the AAR is cleared if specified by B.

JUMP INSTRUCTIONS

There are two types of jumps, direct jumps and indirect jumps.

The direct jump transfers control to an instruction within the same program segment, relative to the base program register. The direct jump can be executed without performing a test (unconditional jump), e.g., GO to Label; or after a test (conditional jump), e.g., IF A = B then GO to Label.

There are four types of indirect jumps, all can be conditional or unconditional. An indirect jump transfers control to the segment, word, and syllable specified by the contents of a Jump Control Word (JCW). The Jump Control Word specifies the type of indirect jump and is located relative to the PRT base register.

Type 1 - Segment Jump - Jumps to any word and syllable of another program segment.

Type 2 - Intra-Segment Return Jump - Jumps to any word and syllable in the current program segment and stores a Return Control Word (RCW) in the location specified by the BXR.

Type 3 - Segment Return Jump - Jumps to any word and syllable of another program segment, storing a Return Control Word in the location specified by the BXR.

Type 4 - Procedure Jump - Jumps to any word and any syllable of a new procedure, using jump control words from both the callers PRT and the called PRT. Return information is stored in the called PRT and the location specified by the BXR.

07 - SJ A1, A2, V_j -Set Up Jump

The Set Up Jump instruction loads the Jump Control Register (JCR) with the sum of AAR + A1A2 and the contents of the V_j syllable. The V_j syllable specifies the type of jump, conditional or unconditional, direct or indirect, and the syllable that is being jumped to. If an unconditional direct or indirect jump is specified, enter the jump sequence. If conditional, a test is performed to determine if a jump sequence is to be executed. If direct. the address contained in the JCR is relative to the BPR. If indirect, the address contained in the JCR is relative to the PRT and the jump sequence is controlled by a Jump Control Word.

34 - JSTA V_t -Jump on Stack Test Arithmetic

The top of the stack is algebraically compared with either zero or the second in the stack, in the manner specified by the test variant (V_t) syllable. The stack is adjusted as specified by the V_t syllable.

If test is true enter jump sequence as specified by the contents of JCR.

24 - JSTL V_t -Jump on Stack Test Logical

The top of the stack is logically compared with zero or the second in stack, in the manner specified by the contents of the test variant (V_t) syllable. The stack is adjusted as specified by the V_t syllable. If the test is true, enter the jump sequence specified by the contents of JCR (direct or indirect sequence).

26 - JFT L, S, V_t -Jump on Field Test

A field of the top of the stack, starting at bit position S, and for a length of L, is compared, in the manner specified by V_t, with a like field of the second in the stack, or zero. The stack is adjusted as specified by V_t.

If the comparison is true enter the jump sequence.

17 - JXMT A1, A2, V_i -Jump on Index Modify and Test

The index field contained in the location specified by BXR + A1A2 + AAR is incremented or decremented by the contents of the increment field or top of the stack and compared with the contents of the limit field, in the manner specified by the index variant (V_i).

If the comparison is true enter the jump sequence. The stack is adjusted as specified by V_i.

11 - SRR -Subroutine Return

The subroutine return instruction transfers control to the instruction following the jump that caused the entrance to the subroutine.

STACK MANIPULATING INSTRUCTIONS

1 - DUP -Duplicate Top of Stack

The stack is pushed down once, and the contents of the top of the stack are copied into the second position in the stack.

13 - RTS V_s -Rearrange Top of Stack

The stack variant (V_s) allows eight manipulations within the stack, as follows:

0-V_s = REV -Reverse

The contents of the top of the stack is exchanged with the contents of the second in the stack.

6-Vs = REVD -Reverse Double

The contents of the first two locations (T and S) in the stack are exchanged with the third and fourth locations in the stack.

4-Vs = DUPD -Duplicate Double

The contents of the first two locations (T and S) are duplicated in the third and fourth positions in the stack.

3-Vs = SWP -Swap T and P

The contents of the T and P registers are exchanged.

2-Vs = RTP -Replace T with P

The contents of the T register are replaced with the contents of the P register.

1-Vs = RPT -Replace P with T

The contents of the P register are replaced with the contents of the T register.

7-Vs = SSU -Step Stack Up

The stack is stepped up once.

5-Vs = SSD -Step Stack Down

The stack is pushed down once and the top of the stack is set to zeroes.

ARITHMETIC INSTRUCTIONS

40 - ADD -Add

The top of the stack is algebraically added to the second stack position and the sum is placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

41 - SUB -Subtract

The top of the stack is algebraically subtracted from the second stack position and the difference is placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

60 - MUL -Multiply

The second stack position is multiplied by the top of the stack. The product is placed in the top of the stack and P register. The stack is stepped up once (with the exception of the top of the stack).

61 - DIV -Divide

The second stack position is divided by the top of the stack. The quotient is placed in the top of the stack with the remainder in the P register. The stack is stepped up once (with the exception of the top of the stack).

53 - ARIT V_d -Double Precision Arithmetic Operations

The variant (V_d) specifies four double precision arithmetic operations, as follows:

0-V_d = ADDD -Add Double

The double precision operand contained in the first and second positions of the stack is algebraically added to the double precision operand contained in the third and fourth positions of the stack. The sum is placed in first and second positions (T and S) of the stack. The stack is stepped up twice (with the exception of the top two stack positions).

1-V_d = SUBD -Subtract Double

The double precision operand contained in the first and second stack positions is algebraically subtracted from the double precision operand contained in the third and fourth positions of the stack. The difference is placed in the first and second stack positions. The stack is stepped up twice (with the exception of the top of the stack and the second stack position).

$2\text{-}V_d$ = MULD -Multiply Double

The double precision operand in the first and the second stack positions is algebraically multiplied by the double precision operand in the third and fourth stack positions. The product is placed in the first and second stack positions. The stack is stepped up twice (with the exception of the top of the stack and the second stack position).

$3\text{-}V_d$ = DIVD -Divide Double

The double precision operand contained in the third and fourth positions is divided by the double precision operand contained in the first and second stack positions. The quotient is placed in the first and second positions of the stack. The stack is stepped up twice (with the exception of the top of the stack and the second stack position).

50 - ADDM -Add Integer

The entire contents of the top of the stack (T) is added to the entire contents of the second stack position (S). The sum is placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

51 - SUBM -Subtract Integer

The entire contents of the top of the stack (T) is subtracted from the entire contents of the second stack position (S). The difference is placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

71 - DIVI -Divide for Integer Quotient

The dividend contained in the second stack position is algebraically divided by the divisor contained in the top of the stack. The integer quotient is placed in the top of the stack. The remainder is placed in the P register. The stack is stepped up once (with the exception of the top of the stack).

21 - RND -Round

The most significant bit of the magnitude field in the P register is added to the least significant magnitude position of the top of the stack. The sum is placed in the top of the stack.

30 - NORM -Normalize

The mantissa of the top of the stack is shifted left until the most significant bit of the mantissa is a one. The exponent field is adjusted by the amount of shift.

31 - INT -Integerize

The integer part of the magnitude field of the floating point number is right justified and the exponent field is set equal to zero.

The result remains in T. If the integer part is greater than 2^{35}-1, an error condition bit is set. If the floating point number is less than one, T is set equal to zero, and an error condition bit is set.

32 - CBB V_c -BCD Conversion

The conversion variant (V_c) specifies a signed or unsigned conversion from BCD to binary, or from binary to BCD, on the value contained in the top of the stack. The V_c specifies if the BCD code is 4 bits per decimal digit or 6 bits per decimal digit.

74 - BSR A -Load Barrel Shift Register

The amount of shift specified by A is added to the contents of the AAR and the sum is placed in the Barrel Shift Register (BSR). At the completion of the operation the AAR is cleared.

63 - SHF V_b -Shift

The shift instruction shifts a value in the manner specified by the shift variant (V_b). The amount of shift is taken from either the top of the stack or the BSR. The BSR contains the amount the top of the stack is to be shifted, or the top of the stack contains the amount the second in the stack is to be shifted.

LOGICAL INSTRUCTIONS

42 - AND -Logical AND

The operand contained in the top of the stack is ANDed with the operand contained in the second position of the stack. The result is placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

72 - OR -Logical OR

Logically OR the contents of the top of the stack with the contents of the second stack position. The results are placed in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

62 - ORX -Exclusive OR

Perform an exclusive OR on the entire contents of the top of the stack with the second stack position. Place the result in the top of the stack. The stack is stepped up once (with the exception of the top of the stack).

52 - IMP -Implication

The implication instruction sets a bit in the top of stack (T) true if the corresponding bit of second in stack (S) is false, or if the bits of T and S are both true. The result of the implication appears in T. The stack is stepped up once (with the exception of the top of the stack).

70 - COMP -Complement

A one's complement is performed on the contents of the top of the stack.

FIELD INSTRUCTIONS

75 - EXT L, S -Extract Field

The contents of the AAR is added to the contents of the L and S variants. Place the modified L and S syllables in Temporary Queue (TEMQ). Extract a field of length L, starting at bit position S, and right justify the field in the top of the stack. The AAR is reset to zero at the completion of the operation.

65 - INS L, S -Insert Field

The AAR is added to the contents of the L and S variants. The modified L and S syllables are placed in TEMQ. The right justified field, of length L, in the top of the stack is inserted into a field of the second stack position, starting at the bit position specified by (S) for a length of L. The stack is stepped up once and the AAR is reset to zero.

66 - COMF L, S -Complement Field

The AAR is added to the contents of the L and S variants. The result is placed in TEMQ. A one's complement is performed on a field of length L, starting at bit position S of the top of the stack. The results are placed in the top of the stack and the AAR is reset to zero.

76 - FILF -Fill Field

The AAR is added to the contents of the L and S variants. The modified L and S syllables specify a field of length L starting at bit position S of the top of the stack that is to be filled with one's. The result is left in the top of the stack and the AAR is reset to zero.

56 - CLRF L, S -Clear Field

The AAR is added to the contents of the L and S variants. The modified contents of the L and S syllables specify the field of length L, starting at bit position S of the top of the stack, that is reset to zero (cleared). The result is left in the top of the stack and the AAR is reset to zero.

35 - EXTD L_2, S, L_1 -Extract Field Double

The AAR is added to the contents of the L_2 and S variants. The modified contents of L_2 and S, along with L_1, specify a field of length L_1 L_2, starting at bit position S of the double precision operand contained in the top two positions of the stack, that is to be right justified to bit position 96. The AAR is reset to zero.

25 - INSD L2. S, L1 -Insert Field Double

The AAR is added to the contents of the L2 and S variants. The modified contents of the L2 and S syllables, along with L1, specify a field of length L1 L2, starting at bit position S of the second double precision operand contained in the stack, into which the least significant L1 L2 bits of the first double position operand are to be inserted. The AAR is reset to zero.

CONTROL INSTRUCTIONS

57 - NOP -No Operation

The NOP instruction indicates that the next instruction in sequence is to be executed.

14 - ESP I -Enter Executive and Scheduling Program

The ESP instruction is a programmatic entry into control mode. The ESP instruction causes the execution of the instruction specified by the address syllable I, which is applied relative to BIAR1.

The BPR, PCR, AAR, and Control Flip-Flops are placed in the stack. The contents of BIAR1 are placed in the BPR. The address syllable I is placed in the PCR. The instruction contained in locations BPR + PCR is placed in Instruction Look Ahead (ILA).

00, 77 - STOP -Stop Processor

The STOP interrupts the normal mode of processing the computer if its associated mask bit is set, causing entry to control mode 1. The stop is ignored if the mask bit is not set. While operating in Control Mode 1, the execution of a STOP instruction, regardless of mask setting, will transfer the processor to Control Mode 2. A STOP in Control Mode 2 halts the processor regardless of mask setting.

04 - ICN N -Interrupt Computer N

This instruction is used to set a condition bit in the computer specified by the variant syllable N. The condition bit will be set in the specified computer, regardless of the computer's mask configuration, but will not actually be interrupted unless that condition bit is unmasked and the computer is in normal mode. If the computer is in control mode 1 (CM1) it will ignore the interrupt regardless of the mask setting. If in control mode 2 (CM2) and running, the interrupt will still be ignored. If the computer is in CM2 and stopped, the interrupt will cause a restart. The variant syllable in indexable.

73 - IOP T -Initiate Input/Output

The IOP instruction has two variations specified by the special syllable T. The first of the two variants transmits the 18 least significant bits of the top of the stack to the Job Stack Address Register (JSAR) in the I/O module specified in the syllable T. The instruction also sets the Job Stack Flat (JSF) in the I/O, which initiates the I/O module on the jobs in the newly specified job stack. The second variation does not involve transmitting a new job stack address. It simply sets the JSF of the specified I/O module. This variation re-initiates the I/O on the existing Job Stack at the current address in the JSAR. The variant syllable is indexable.

20 - ETB -Extract Tag Bits

This instruction extracts the three tag bits from the word at the top of the stack and places them in the three least significant data bit positions at the top of the stack.

22 - ITB -Insert Tag Bits

The three least significant data bits in the top of the stack are inserted into the tag area of the second stack position. The stack is stepped up once.

32 - IRR -Interrupt Routine Return

The IRR instruction is used to restore the information necessary to resume execution of a normal mode program after processing an interrupt. Attempted execution of IRR while in normal mode is an interrupt condition. The IRR instruction returns the processor to normal mode from either level of control mode.

36 - FMT A1, A2, B -Fetch Modify Tag

This is an instruction used to accomplish a memory "lock out". The sequence executes a fetch of the contents of the address specified by the syllables AAR + A_1A_2 + a base specified by B. At the time of the fetch from memory the two most significant tag bits are set to "ones". When the word reaches the central processor the tag bits are investigated (by additional instructions) and if the tags are "zeros" then "lock out" is accomplished. If the tags are already "ones" this indicates that another program has accomplished the lock out. This instruction provides program means of controlling memory access. A variation specified by B allows FMT to be used for "unlock" (set tags to zero).

67 - FINQ -Final Queue Halt

This instruction is used to halt FINST and cause FINST to notify ADVAST when the instruction appears at the top of Final Queue. ADVAST halts immediately after inserting the FINQ op code in the Final Queue and waits for FINST response, at which time ADVAST will proceed.

APPENDIX C

CENTRAL PROCESSOR INSTRUCTION

SYLLABLES VARIANT DEFINITIONS AND CONFIGURATIONS

SHIFT AMOUNT VARIANT (A) - Specifies the shift amount relative to the AAR which is to be placed in the shift amount register by the Load Barrel Shift register instruction, BSR.

XXXXXX Shift amount. All amounts are legal but logical shift amounts greater than 48 are treated as 48. Arithmetic shifts greater than 35 are treated as 35.

RELATIVE ADDRESSING VARIANT (A_1, A_2) - The A_1 A_2 syllables are six bit syllables which are combined to produce a twelve bit relative address.

BASE VARIANT (B) - Specifies the base register to be used in address arithmetic involved in the following instructions: FMS, SSM, FMA, FMC, and FMT.

CONFIGURATIONS
012345

XXX000 No base register
XXX001 Base Program Register
XXX010 Base Index Register
XXX011 Base Data Register
XXX100 Program Reference Table
XXX101, XXX110, XXX111 Illegal Configuration
XX0XXX Reset AAR at conclusion of instruction
XX1XXX Do not reset AAR
X0XXXX - Reset tag bits
X1XXXX FMT only - set tag bits
0XXXXX - Regular Fetch
1XXXXX FMS only - Fetch Memory Fail Register

ESP VARIANT (I) - The I syllable is a six bit syllable used in the ESP instruction as a relative address which is added to the BIAR1.

FIELD LENGTH VARIANT (L) - Specifies the length of the field for all field instructions.
XXXXXX All field lengths are legal but single length fields of greater than 48 will be treated as 48. Double length fields of greater than 96 will be treated as 96.

LITERAL VARIANT (L_1, L_2) - The L_1, and L_2 syllables are six bit syllables used as a twelve bit literal with the FAS instruction.

SHORT LITERAL VARIANT (LIT) - specifies a six bit literal character.

COMPUTER NUMBER VARIANT (N) - Specifies the particular Central Processor to be interrupted by the instruction, Interrupt Computer N, ICN.

XX0000	Illegal Configuration
XXXXXX	Specifies Computer to be interrupted, most significant two bits are ignored.

REGISTER VARIANT (R) - Specifies the particular register to be addressed by either the Fetch Register to Stack Instruction, FRS, or the Store Stack to Register Instruction, SSR.

00XXXX	Illegal Configurations
010000	Stack Extension Pointer, SEP
010001	Base Index Register, BXR
010010	Base Program Register, BPR
010011	Base Data Register, BDR
010100	Normal Lower Bounds Register, NLBR
010101	Normal Upper Bounds Register, NUBR
010110	Alternate Lower Bounds Register, ALBR
010111	Alternate Upper Bounds Register, AUBR
*011000	Purge Computer Stack
011001	Stack Lower Bounds Register, SLBR
011010	Stack Upper Bounds Register, SUBR
011011	Program Reference Table Base, PRT
011100	Program Reference Table Limit, PRTL
011111	Computer Number Register, CNR
100000	Barrel Shift Register, BSR
100001	Program Count Register, PCR
100010	Jump Control Register, JCR
110001	Base Interrupt Address Register 1, BIAR1
110010	Base Interrupt Address Register 2, BIAR2
110011	Processor Fail Register, PFR
110100	Advast Memory Address Register, AMAR
110101	Incremental Time Clock, ITC
111110	Interrupt Mask Register, IMR
111111	Interrupt Condition Register, ICR

All other configurations are illegal.

X indicates either binary digit, 0 or 1.

* Illegal for SSR

FIELD START VARIANT (S) - Specifies the starting position of fields for all field instructions.

XXXXXX All starting positions are legal but the normal range of starting positions is 0 through 47.

I/O UNIT NUMBER VARIANT (T) - Specifies the particular I/O to be addressed by the Initiate I/O Program instruction, IOP.

XX0000	Illegal Configuration
XXXXXX	Specifies I/O number, most significant two bits are ignored
XX1111	Illegal Configuration
X0XXXX	Set I/O Program Flags only.
X1XXXX	Transmit New Job Stack Address and Set I/O Program Flag

BARREL VARIANT (V_B) - Specifies the particular type of shift to be performed by the Shift instruction, SHF.

X0XXXX	Shift amount is in BSR, shift T
X1XXXX	Shift amount is in T, shift S
XX0XXX	Single length, shift T
XX1XXX	Double length, shift T & P
XXX000	Right - End Off - Arithmetic
XXX001	Right - End Off - Logical
XXX010	Right - End Around - Arithmetic
XXX011	Right - End Around - Logical
XXX100	Left - End Off - Arithmetic
XXX101	Left - End Off - Logical
XXX110	Left - End Around - Arithmetic
XXX111	Left - End Around - Logical

CONVERT VARIANT (V_C) - Specifies the particular type of conversion to be performed in the BCD conversion instruction, CBB.

XXX000	4 Bit B.C.D. to Binary
XXX001	Illegal Configuration
XXX010	6 Bit B.C.D. to Binary, unsigned
XXX011	6 Bit B.C.D. to Binary, signed
XXX100	Binary to 4 Bit B.C.D.
XXX101	Illegal Configuration
XXX110	Binary to 6 Bit B.C.D., unsigned
XXX111	Binary to 6 Bit B.C.D., signed

DOUBLE LENGTH ARITHMETIC VARIANT (V_D) - Specifies which of four Double Length Arithmetic operations is to be performed by the Arithmetic Instruction, ARIT.

XXXX00	Add Double
XXXX01	Subtract Double
XXXX10	Multiply Double
XXXX11	Divide Double

INDEX VARIANT (V_I) - Specifies the particular modification and test to be performed in the Jump on Index Modify Test instruction, JXMT.

00XXXX	No Modification
01XXXX	Add Index increment to Index value
10XXXX	Add T (31, 18) to Index value
11XXXX	Subtract T (31, 18) from Index value
XX0XXX	Compare Index value with T (31, 18)
XX1XXX	Compare Index value with Index limit
XXX000	Test is unconditionally false
XXX001	Jump on Index value greater
XXX010	Jump on Index value equal
XXX011	Jump on Index value greater or equal
XXX100	Jump on Index value less
XXX101	Jump on Index value unequal
XXX110	Jump on Index value less or equal
XXX111	Test is unconditionally true

JUMP VARIANT (V_J) - Specifies what type of jump is being prepared by the Set Up Jump Instruction, SJ.

X0XXXX	Unconditional Set Up Jump
X1XXXX	Conditional Set Up Jump
XX0XXX	Type of Jump is indirect
XX1XXX	Type of Jump is direct
XXX000	
XXX001	
XXX010	
XXX011	Specifies first syllable to be executed after a direct jump
XXX100	
XXX101	
XXX110	
XXX111	

STACK VARIANT (V_S) - Specifies particular variations of the Rearrange Top of Stack instruction, RTS.

XXX000	Reverse T and S
XXX001	Replace P with T
XXX010	Replace T with P
XXX011	Swap T with P
XXX100	Duplicate double length operand
XXX101	Step Stack down (Reset T)
XXX110	Reverse double length operand
XXX111	Step Stack up

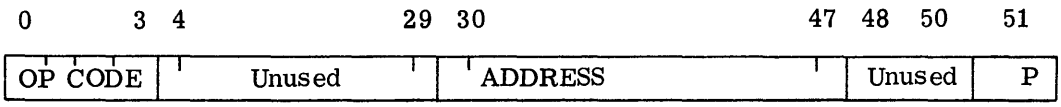
TEST VARIANT (V_T) - Specifies type of test to be performed by Jump on Stack Test Arithmetic, JSTA: Jump on Stack Test Logical, JSTL; or Jump on Field Test, JFT. Also specifies stack manipulations at conclusion of the instruction.

000XXX	Compare T with zero, step stack up once
010XXX	Compare T with zero and do not step stack
100XXX	Compare T with zero, put test result in T (48, 1)
110XXX	Compare T with zero, step stack down, put test result in T (48, 1)
001XXX	Compare T with S, step stack up twice
011XXX	Compare T with S, do not step stack
101XXX	Compare T with S, step stack up, put test result in T (48, 1)
111XXX	Compare T with S, step stack down, put test result in T (48, 1)
XXX000	Test is unconditionally false
XXX001	Test for $S > T$ or $T >$ zero
XXX010	Test for $S = T$ or $T =$ zero
XXX011	Test for $S \geq T$ or $T \geq$ zero
XXX100	Test for $S < T$ or $T <$ zero
XXX101	Test for $S \neq T$ or $T \neq$ zero
XXX110	Test for $S \leq T$ or $T \leq$ zero
XXX111	Test is unconditionally true

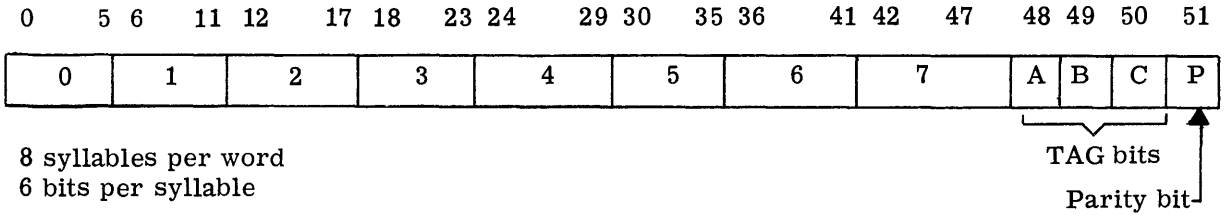
APPENDIX D

CENTRAL PROCESSOR WORD FORMATS

MEMORY ADDRESS

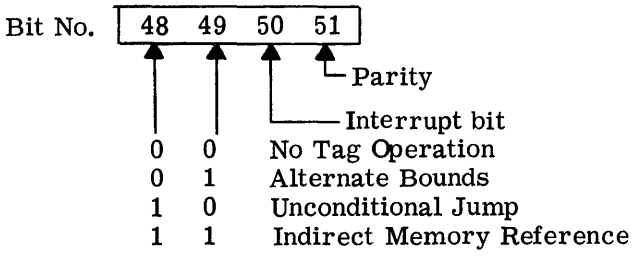


INSTRUCTION FORMAT

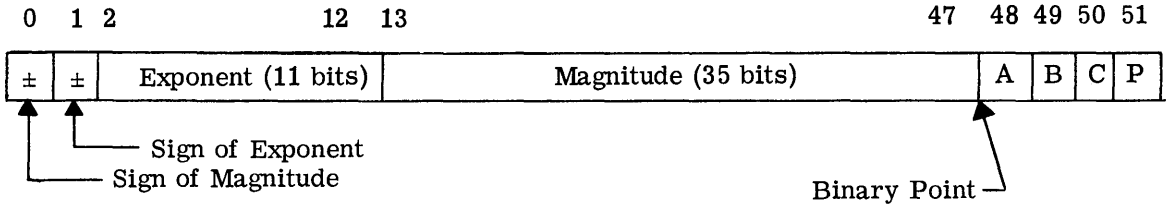


8 syllables per word
6 bits per syllable

TAG bits have the following meanings:

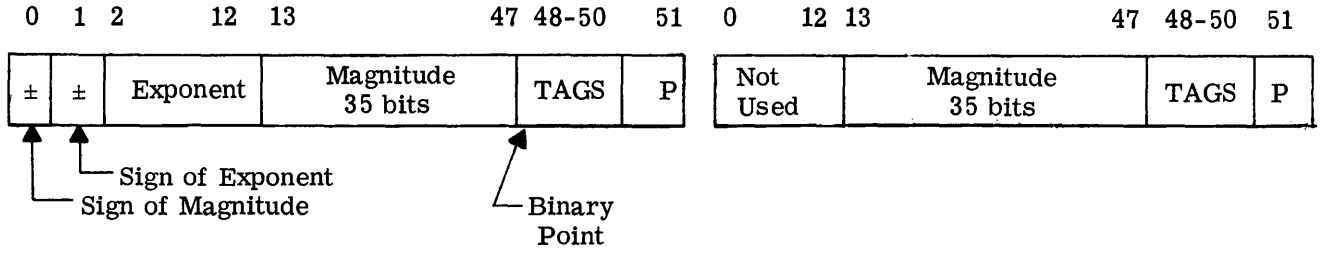


FLOATING POINT BINARY

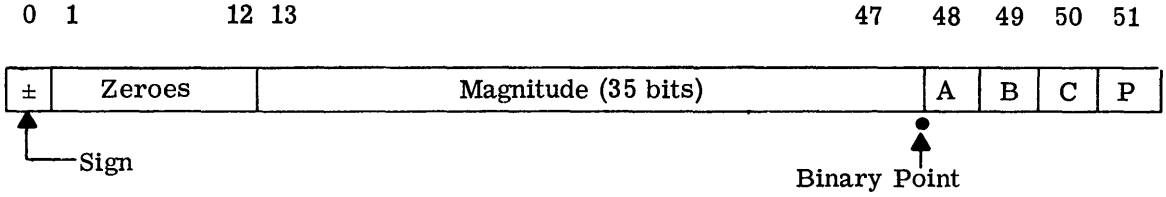


0 = positive
1 = negative

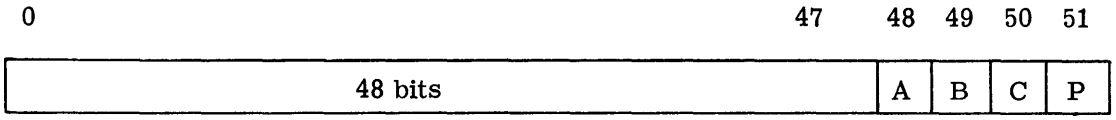
FLOATING POINT (Double precision)



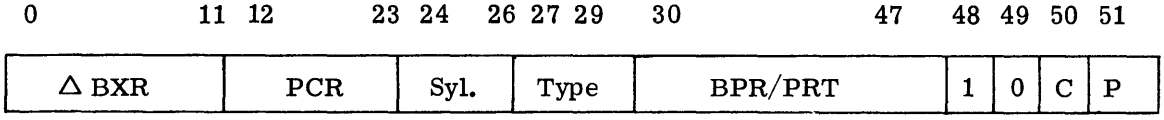
INTEGER BINARY



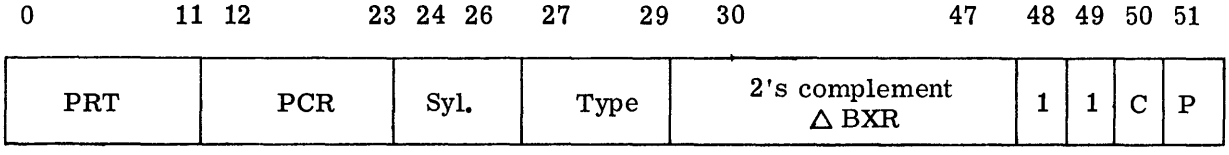
LOGICAL WORD



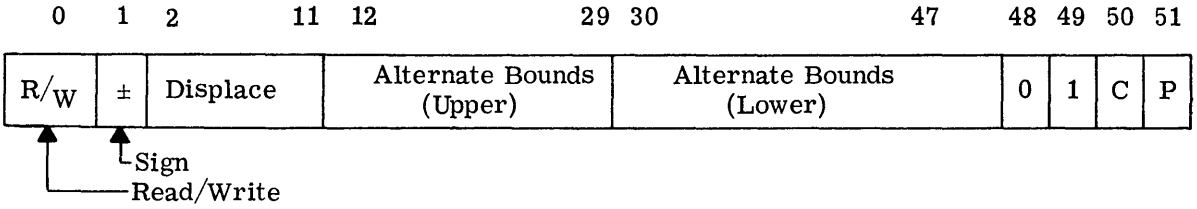
JUMP CONTROL WORD



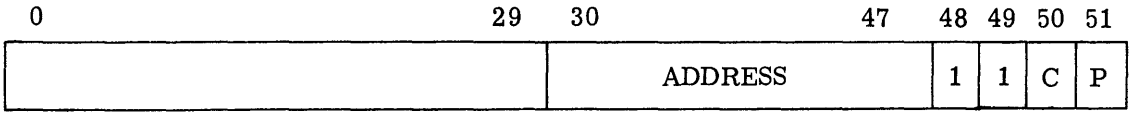
RETURN CONTROL WORD



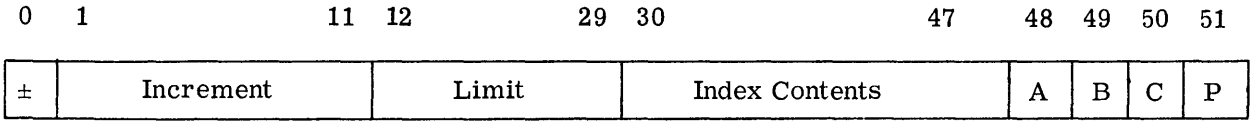
ALTERNATE BOUNDS WORD



INDIRECT ADDRESS WORD



INDEX WORD



APPENDIX E

ABBREVIATIONS AND ACRONYMS

A_1 Address Syllable 1

A_2 Address Syllable 2

A1A2
Specifies 2 Syllable, 12 Bit Relative Address

AAR
ADVAST Address Register

AAU
ADVAST Adder Unit

AB
Alternate Bounds Tag Bit

ABL
Alternate Bounds Lower

ABLR
Alternate Bounds Lower Register

ABS
ADVAST Barrel Switch

ABU
Alternate Bounds Upper

ABUR
Alternate Bounds Upper Register

ACR
ADVAST Communications Register

ACUGO
Address Comparator Unit GO

ACU
ADVAST Comparator Unit

ADD
Algebraic Add

ADDD
Add Double

ADDM
Add Integer Magnitude

ADDR
Address Field

ADVAST
Advance Station

AID
ADVAST Instruction Decoder

AIR
ADVAST Instruction Register

ALGOL
ALGOrithmic Language

AMAR
Associative Memory Address Register

AND
Logical And

AOR
A Operand Register

ARIT
Double Precision Arithmetic Operation

ASLP
ADVAST Storage Queue Load Pointer

ASM
Associative Memory

ASMA
Associative Memory Address Register

AT
ADVAST Time

AUX
Transfer P Register to Top of Stack

B
Base Register Specifier

000 - No base
001 - BPR
010 - BXR
011 - BDR
100 - PRT

BCD
Binary Coded Decimal

BDR
Base Data Register

BEP
Bottom of Computer Stack Extension

BIAR1
Base Interrupt Address Register 1

BIAR2
Base Interrupt Address Register 2

BOR
B Operand Register

BPR
Base Program Register

BSR
Barrel Shift Register or as an instruction Load Barrel Shift Register

BXR
Base Index Register

CAR
Communications Address Register

CAT
Channel Assignment Table

CBB
Conversion of Binary and BCD

CFR
Communications Fetch Register

CHBR
Channel Base Register

CHORE
Chain Of Runs Executive

CLRF
Clear Field

COBOL
COmmon Business Oriented Language

COMF
Complement Field

COML
Logical Complement

COMM
Communications Unit of the Central Processor Module

COMP
Complement

COMT
Two's Complement

CPM
Central Processor Module

CRC
Card Reader Controller

C/S
Collector/Scheduler

C/S Q Collector/Scheduler Queue

CSR
Communications Store Register

DEL
Delay

DFC
Disc File Controller

DJEZ
Decrement and Jump Equal to Zero

DJNZ
Decrement and Jump Not Equal to Zero

DIV
Divide

DIVD
Divide Double

DIVI
Integer Divide

DIVM
Divide Mantissa

DUP
Duplicate Top of Stack

DUPD
Duplicate Double Length Words

EAR
Effective Address Register

EOB
End of Block

EOF
End of File

EOM
End of Message

EOR
End of Record

EOT
End of Transmission

ESP
Executive and Scheduling Program or as an instruction Enter Executive and Scheduling Program

ETB
Extract Tag Bits

EXT
Extract

EXTD
Extract Double

f
Included Field

$\bar{f}$
Excluded Field

F
Two Instruction Syllables Designating the Included Field

FAS
Fetch Address Register to Stack

FILF
Fill Field

FINQ
Final Station Instruction Queue or as an instruction Final Queue Empty

FINST
Final Station

FLIP
Final Station Load Pointer

FMA
Fetch Memory to Address Register

FMC
Fetch Memory Conditionally

FMS
Fetch Memory to Stack

FMSA
Fetch Memory to Stack Absolute

FMT
Fetch Modify Tags

FORTRAN
FORmula TRANslator Language

FR
Function Register

FRE
Function Register Extension

FRS
Fetch Register to Stack

GO
Start Channel Data Transfer Designated in AOR

H
H Register

HAT
H Register to A Register Transfer

I
Indirect Tag Bit

IBA
Instruction Base Address

IC
Instruction Counter

I/C
Interpreter/Controller

ICN
Interrupt Computer N

ICR
Interrupt Condition Register

IDIOT
IDeal Interface OuT

IDXL
Index Limit

IDXQ
Index Queue

IDXV
Index Value or Contents

ILA
Instruction Look Ahead

IM_N
Interrupt Mask Bit N

IMP
Implication

IMR
Interrupt Mask Register

INS
Insert

INSD
Insert Double

INT
Intergerize

I/O
Input/Output Module

IOP
Initiate Input/Output Program (IOP)

IRR
Interrupt Routine Return

ISAR
Interrupt Stack Address Register

ISLR
Interrupt Stack Limit Register

ITB
Insert Tag Bits

J
Jump Tag Bit

JAT_n
Jump to ADVAST Time n

JCT
Jump Control Register

JCW
Jump Control Word

JEZ
Jump if Equal Zero

JFT
Jump Field Test

JNZ
Jump if Not Equal Zero

JPHS
Jump to Phase Special

JSAR
Job Stack Address Register

JSF
Job Stack Flag

JSTA
Jump Stack Test Arithmetic

JSTL
Jump Stack Test Logical

JSW
Job State Word

JSWT
Job State Word Table

JUMP
Jump Unconditional

JXMT
Jump on Index Modify and Test

JXnN
Jump when Index n Not Equal to Zero

JXnZ
Jump when Index n Equals Zero

L
Number of Bits in a Field

L1
Literal Syllable 1

L2
Literal Syllable 2

LIL2
Specifies 2 Syllable 12 Bit Literal

LAP
Look-Ahead Pointer Register

LCS
Left Circular Shift

LDX
Load Index Register

LIT
6 Bit Literal

LM
Local Memory

LRC
Longitudinal Redundancy Character

LS
Left Shift

LTB
Load Tag Bits

MEM
Memory Module

MFR
Memory Failure Register

MM
Memory Module (Main Memory)

MTC
Magnetic Tape Controller

MUL
Multiply

MULD
Multiply Double

NBL
Normal Bounds Lower

NBLR
Normal Bounds Lower Register

NBU
Normal Bounds Upper

NBUR
Normal Bounds Upper Register

NOP
No Operation

NORM
Normalize

OP
Instruction Operation Code

OR
Logical Or

ORX
Exclusive Or

P
Extension of Top of Stack Register or
Instruction Base Designation Syllable

PBA
Parameter Base Address

PCR
Program Count Register

PFR
Processor Failure Register

PRT
Program Reference Table or Program
Reference Table Base Register

PRTL
Program Reference Table Limit Register

PRTQ
Program Reference Table Queue

R
Register Specifier

RAAR
Reset ADVAST Address Register

RCS
Right Circular Shift

RCW
Return Control Word

REL
Release Processor

REQN
Request Access

REV
Reverse T and S

REVD
Reverse double length words

RND
Round

RPF
Reset Program Flag

RPT
Replace P with T

RS
Right Shift

RTB
Register Transfer Bus

RTP
Replace T with P

RTS
Rearrange Top of Stack

RW
Read Only Bit

S
Second in Stack Register or as part of an instruction Starting Bit Number of a Field

SBLR
Stack Bounds Lower Register

SBUR
Stack Bounds Upper Register

SCC
Shift Control Counters

SCD
Syllable Count Decode

SEP
Stack Extension Pointer Register

SFE
Skip on Field Equal

SFG
Skip on Field Greater

SFL
Skip on Field Less

SFU
Skip on Field Unequal

SHF
Shift

SIC_N
Set Interrupt Condition "N"

SIRJ
Set Interrupt Jump

SIS
Store in Interrupt Stack

SJ
Set Up Jump

SLIT Lit
Short Literal to Stack

SPF
Set Program Flag

SRJ
Subroutine Jump

SRR
Subroutine return

SSC
Syllable Shift Counter

SSD
Step Stack Down

SSM
Store Stack to Memory

SSMA
Store Stack to Memory Absolute

SSR
Store Stack to Register

↑ STACK
Step Stack Up

↓ STACK
Push Stack Down

↑ STACK-1
Step Stack Up without effecting T

↑↑ STACK
Step Stack Up Twice

STB
Store Tag Bits

STEP
Step Stack Up

STOP
Stop Processor

STOR
Store

STORQ
Storage Queue

STX
Store Index Register

SUBD
Subtract

SUBD
Subtract Double

SUBM
Subtract Integer Magnitude

SURE Q_T
Set Unit Request for Temp Q

SWP
Switch T and auxiliary (P) register

SWR
State Word Register

T
Top of Stack Register

TEMPQ
Temporary Queue

TEP
Top of Computer Stack Extension Pointer

TFT
True-False Toggle

TLP
Temporary Queue Load Pointer

TOOL
The Only LOgical Language

TSC
Transfer State Word to Channel

TSR
Temporary Storage Register

TTY
Teletype Controller

VAR1
Variant Syllable Bit One

X
Index

(X)
Contents of Location Contained in X

X [S, L]
If $X = X_1X_2 --X_n$, then X [S, L] = $X_SX_{S+1}X_{S+2} ---X_{S+L-1}$

X [Y]
Contents of Field Y in X

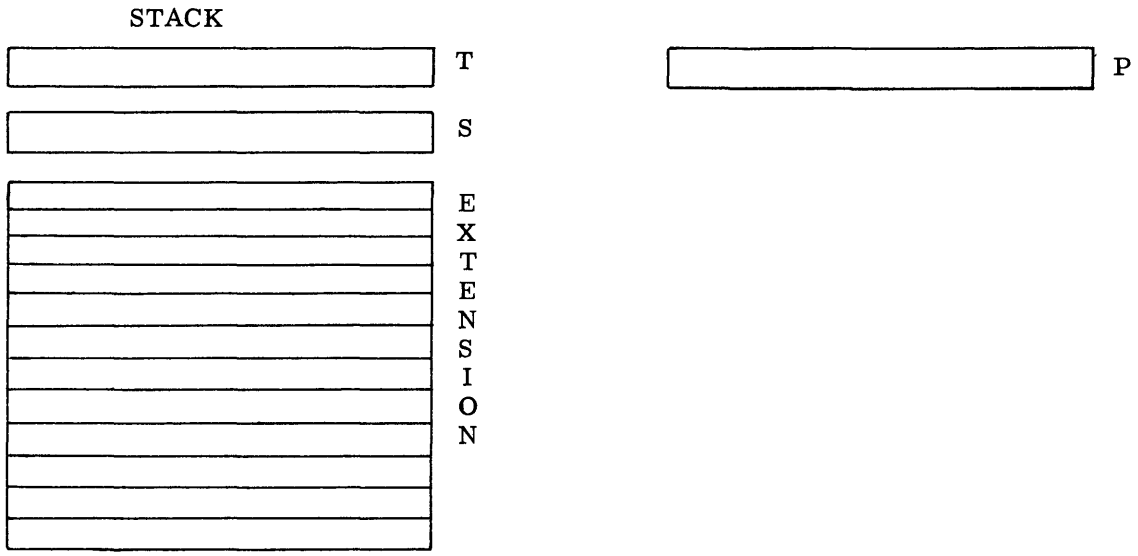
XM
Index and Modify Index

XS
Index by Top of Stack

APPENDIX F

POLISH NOTATION AND THE STACK CONCEPT

B8500 arithmetic operations are implemented in the final station (FINST) of the central processor. Operators are presented to the hardware from the final instruction register (FIR) and operands are presented from the stack. The stack is a conceptually contiguous series of locations used to store words for processing. The T register (top of the stack or A register) and the S register (second in the stack or B register) are 'hard registers' used in hardware manipulation. There is, in addition, a hardware stack extension of twelve words. Further stack capacity is provided by linking the stack to main memory for large capacity operand storage. As each third (4 words) of the extension is filled it is transferred to sequential locations of main memory. The process is reversed for accessing the operands. A separate P register (or C register) is provided for those operations which require a third register. The stack appears as follows:



The stack has upper and lower bounds registers to limit the number of words contained within the stack. The stack may be manipulated by hardware instructions to alter its contents either by pushing the stack down (effectually moving each word previously contained down to a location one position deeper than its previous location) or by stepping the stack up (moving each word previously contained up to a location one position higher than its previous location). Double push down and step up is also utilized where applicable. Thus as operands are presented to the stack at either T or S, they are stored sequentially one on top of the other. Likewise, as they are called to use by the hardware, the stack presents them sequentially, the most recent operand first. Attempting to use more operands than were previously put into the stack will result in a stack bounds violation and an interrupt condition will be set.

The use of the stack in such a manner as to have the two operands for an arithmetic operation (or one new operand and the result of the previous operation) present in the T and S registers at the time the operator is in the FIR gives rise to the use of Polish Notation. Basically, Polish Notation is a method for writing expressions without a need for bracket characters to delimit the scope of an operator. For instance, in common algebraic notation we might say:

$$X = D\ (M + N) / T + P$$

In extended ALGOL this would be represented as:

$$X \leftarrow (D * (M + N))/(T + P)$$

In Polish Notation the statement would be written:

$$D\ M\ N + *\ T\ P + /\ X \leftarrow$$

which in words means:

Fig. 1) Obtain the value of D.

Fig. 2) Obtain the value of M.

Fig. 3) Obtain the value of N.

Fig. 4) Add the two most recently obtained operands (i.e., M and N).

Fig. 5) Multiply the two most recently obtained operands (i.e., (M + N) and D).

Fig. 6) Obtain the value of T.

Fig. 7) Obtain the value of P.

Fig. 8) Add the two most recently obtained operands (i.e., T and P).

Fig. 9) Divide the next to last obtained operand by the most recently obtained operand (i.e., (D * (M + N)) by (T + P)).

Fig. 10) Assign X the value of the most recently obtained operand (i.e., (D * (M + N))/(T + P)).

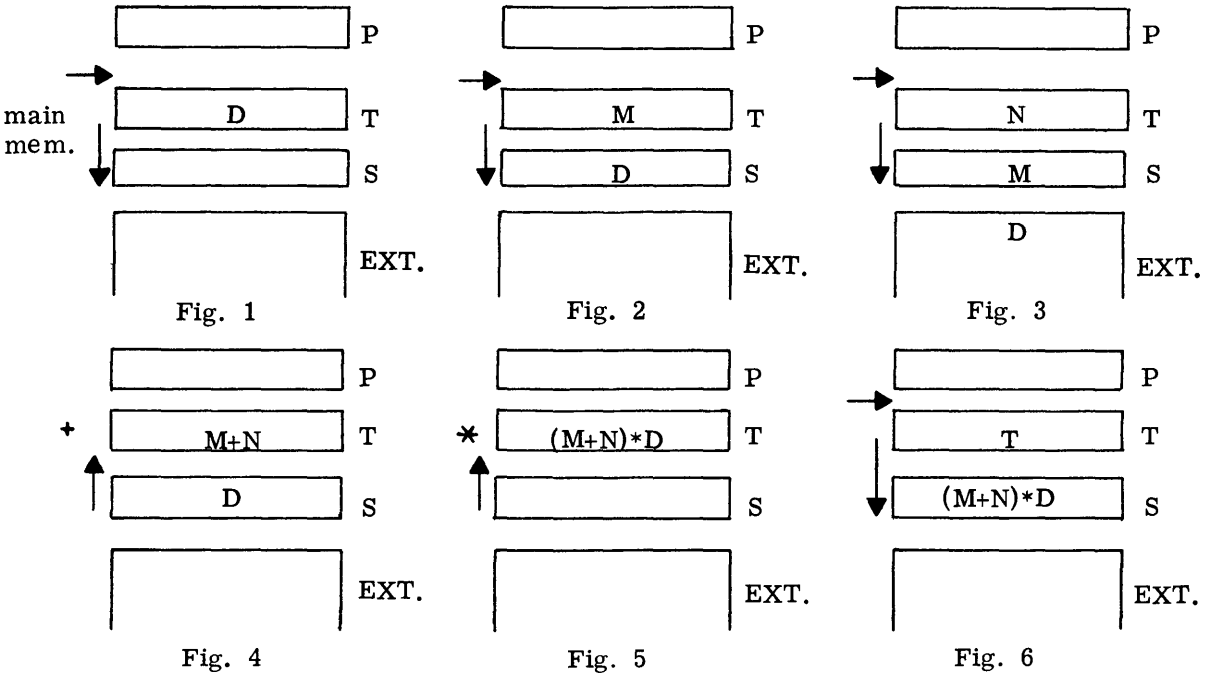


Fig. 1 Fig. 2 Fig. 3

Fig. 4 Fig. 5 Fig. 6

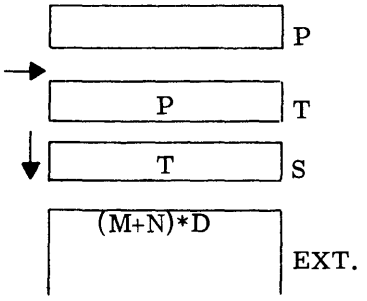


Fig. 7

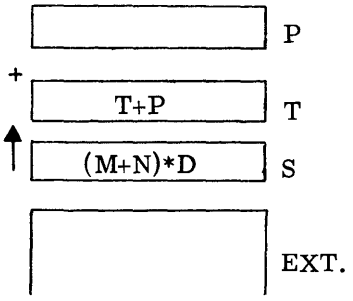


Fig. 8

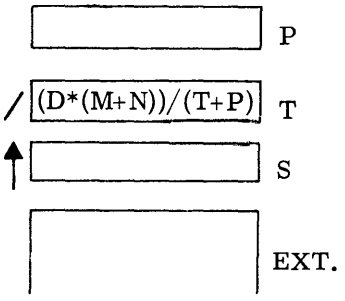


Fig. 9

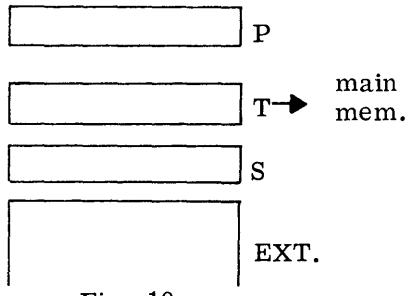


Fig. 10

The following rules are used to interpret Polish Notation:

1) The Polish "string" is read from left to right.

2) Operands are obtained (i.e., read) until the last operand has occurred or until an arithmetic operation occurs.

3) When an arithmetic operation occurs, the two most recently obtained operands are operated upon as though the operation had occurred following the second from the last operand and preceding the most recent operand.

4) When an arithmetic operation is performed on two operands, the result is one operand (i.e., it becomes the most recent operand).

5) The above sequence occurs until an address variable appears. An address variable is a notation followed by a left arrow. This will cause the value of the recently obtained operand to be assigned to the address variable.

APPENDIX G

GLOSSARY

(Some terms defined are characteristic to the industry, but others are significant only with respect to the B8500 System. For a more complete glossary of industry terminology it is recommended that reference be made to glossaries published in such trade journals as ACM Communications and Computers and Automation.)

ACCESS, RANDOM
Access to storage under conditions in which the next position from which information is to be obtained is relatively independent of the previous position or access.

ACKNOWLEDGE
Send a meaningful signal in answer to a request or query.

ADDER
A device capable of forming the sum of two or more quantities.

ADDRESS
A label, such as an integer or other set of characters, which identifies a memory location or storage device.

ADDRESS, ABSOLUTE
The label assigned to a specific storage location by the designers of a machine.

ADDRESS, BASE
The label identifying the first word in a data area or routine. The base address is added to the relative address to obtain the absolute address.

ADDRESS, RELATIVE
A positional indicator to identify a location in a storage area with respect to the origin or base of that area.

ALGOL
(for ALGOrithmic Language) an international problem oriented language designed for the concise, efficient expression of arithmetic and logical processes, and the control of these processes.

ALGORITHM
A statement of the steps to be followed in the solution of a problem.

ALLOCATE
To assign storage locations to the main routines and subroutines, thereby fixing the absolute values of any symbolic addresses.

ALPHANUMERIC
Contraction of alphabetic-numerical; a system including letters, digits, and special symbols.

ARGUMENT
The quantity or quantities submitted to a function, e.g. X as in the intrinsic function: SIN(X), or the known reference factor necessary to find the desired item in a table or array. Sometimes referred to as a "key" as in "search key."

ARRAY
An ordered arrangement of items of information.

ASYNCHRONOUS PROCESSING
A method of computer system processing in which operations are taken up in response to signals indicating completion of predecessor operations instead of master clock signals.

AUDIT TRAIL
A file maintained by the system upon which are recorded the changes made to specified file(s) during the period of time since the last complete copy of the specified file(s) was removed from the system. The audit trail may be used for reconstruction of file destroyed by program error or system malfunction.

AUTOMATIC PROGRAMMING
Technique which employs the computer itself to translate programming from a form that is easy for a human being to produce and understand into a form suitable for use by a computer.

BASE
A number, the powers of which are assigned as the unit value of columns in a number system; also called radix or place value. For example, 2 is the base in binary representation, 8 is the base in octal notation, and 10 is the base in decimal notation.

BATCH PROCESSING
The method of presenting to a system sequentially several unrelated informations upon which the same logical procedures are to be followed.

BINARY
Involving a choice or a condition in which there are but two alternatives. For example, the binary number system uses the base two and contains only two symbols, zero and one.

BIT
Contraction of binary digit. Usually represents the status of one flip-flop, either off or on (0 or 1).

BLOCK
(1) A group of computer words or records considered or transported as a unit, an item, or a message.

(2) In flow charts, an assembly of symbols with each symbol representing a logical unit of a program.

BOOLEAN
Refers to a system dealing with truth values, operating upon logical conditions rather than numbers.

BOOLEAN ALGEBRA
A system of algebra dealing with logical values as variables and having basic operators such as "and," "or," "not," etc.

BOOLEAN VARIABLES
An operand in a Boolean algebra expression. A Boolean variable may have the value of "true" or "false," commonly represented in computers by one and zero respectively.

BUFFER
Any reserved area within a device which stores information temporarily during data transfers. A facility linked to: (1) an input device in which information is assembled from external storage and stored ready for transfer to internal storage; or (2) an output device into which information is transmitted from internal storage and held for transfer to external storage. Since computation continues while transfers between buffer storage and external devices take place, buffers are used to compensate for differences in the speed of the various components of the system so that the system can operate as an integrated unit.

BUSS
The interconnecting line(s) between devices which carry signals or communications between them.

CALL
(1) A set of characters or bits which demand an action to take place or some item of information; for example subroutine call, ESP call, descriptor call.

(2) Transfer control from a main routine to a subroutine or macro.

CELL
(1) In thin film memory one bit of storage.

(2) A 1000 Å thick 30 X 80 mil nickel-iron alloy deposit within a glass substrate.

CHANNEL
A path along which information may flow.

CHARACTER
One of a set of elementary symbols which may be arranged in ordered groups to express information; these symbols may include the decimal digits 0 through 9, the letters A through Z, punctuation symbols, special input and output symbols, and any other symbols which a computer may accept.

CHARACTERISTIC
The exponent portion of a floating-point number. (See Floating-Point Representation.)

CLEAR
Reset to zero.

COBOL
A COmmon Business Oriented Language designed for expressing problems of data manipulation and processing in English narrative form.

COLD JOB TABLE
An array containing information necessary for initialization of a user program which has been introduced to the system for scheduling to run.

COLLECTOR
A routine within the Executive Scheduling Program which has the function of calling all files necessary for a particular programs operation. It submits requests to the operator(s) and physically verifies that all requested files are present.

COMPILE
Reduce a source program written in a non-machine language to produce a machine language routine to solve the problem defined by the source program.

COMPILER
A translator program which reduces a problem-oriented language into the machine language of a particular computer.

COMPLEMENT
(1) TRUE COMPLEMENT
The quantity which when added to a given number yields the base of that number, e.g. the true or 10's complement of $6_{\overline{10}}$ is 4.

(2) BASE-1 COMPLEMENT
The quantity which when added to a given number yields one less than the base of that number, e.g. the base-1 or 9's complement of $3_{\overline{10}}$ is 6.

COMPUTER
A machine which can calculate or perform reasonable arithmetic and logical operations and transform the results of these operations into a usable form.

COMPUTER STACK
The storage always available in the central processor, consisting of the T and S registers and a twelve operand micro-logic memory, for a total capacity of 14 operands.

COMPUTING SYSTEM
A group of interconnected equipments which as a unit perform reasonable arithmetic and logical operations and transform the results of these operations into usable form. The individual devices and equipments of the system are highly specialized to optimize the efficiency with which their particular functions within the system are performed.

CONCATENATING
Linking together by forming a chain or series, a series or order of things depending on each other.

CONFIGURATION
Relative arrangement of various components of the system.

CONSOLE
The unit of a computing system which provides communication between the computer and the operator. The B8500 console contains indicators for displaying the status of the system, a means for manual intervention or operation of the system, and a means of testing the memory modules.

CONTROL MODE
Condition of operation wherein instructions that can be performed in normal state are augmented by additional control operations. Some of the ESP routines are written to operate in control mode.

DATA ARRAY
Any ordered set of data, such as the information on a card, a tape record, a print line, the contents of a working area, etc.

DEADLINE
A type of job or CHORE in which a time for completion has been set and scheduling and priority assignment are dynamic in regard to meeting this goal.

DEBUG
To isolate and correct the mistakes in a program or the components of a computing system.

DESCRIPTOR
A computer word used specifically to define characteristics of a program element. For example, descriptors are used for describing a data record, a segment of a program, or an input-output operation.

DIAGNOSTIC ROUTINE
Routine designed to detect and locate either a malfunction of the system or a mistake in programming.

DISK
A type of relatively high speed storage device upon which words of information are magnetically recorded on concentric tracks of a circular plane.

DOUBLE PRECISION
A quantity having twice as many digits as are normally carried in a specific computer word. Often called double length.

DUMMY FILE
A dummy file is a file that a run will call for in its normal processing, but will receive an End of File indication on the first read from that file. In the B8500 this mechanism allows chores to be written to drive programs which have optional files.

DUPLEX, FULL
A line or buss capable of handling communication in both directions simultaneously.

DUPLEX, HALF
A line or buss capable of handling communication for a simplex device or in one direction at a time for a duplex device.

EDIT
The act of arranging information from input-output devices. This may involve the selection of pertinent data, the insertion of symbols such as page numbers and check-protection characters, and standard processes such as zero suppression.

EXECUTIVE SCHEDULING PROGRAM (ESP)
A computer program to control the operation of the system. It is designed to minimize the amount of intervention required of the human operator. ESP performs the following functions: schedules programs to be processed; initiates segments of programs; controls all input-output operations to ensure efficient utilization of each system component; allocates memory dynamically; issues instructions to the human operators and verifies that their actions were correct, etc.

EXPONENT
A number may be divided into an exponent and a mantissa. The exponent positions radix point location relative to the mantissa. In the B8500 the exponent is expressed as an 11 bit field and a sign.

EXTENSION MODE
The mode of stack operation which extends the computer stack to a memory stack area.

EXTERNAL STORAGE
Storage facilities removable from the computer itself but holding information in a form acceptable to the computer (magnetic tape, punched card, etc.).

FIELD
A set of one or more bits, digits or characters which is treated as a unit of information.

FILE
A collection of records; an organized collection of information directed toward some purpose. (The records in a file may or may not be sequentially filed according to a key contained in each record.) B8500 files may be of the following types: program, random, serial, and dummy.

FIXED-POINT REPRESENTATION
An arithmetic notation in which all numeric quantities are expressed by the same number of digits with the decimal point (for base 10) or octal point (for base 8) assumed in a fixed location in each number. Alignment of numbers with different assumed locations of the points must be performed by the program before an arithmetic operation such as addition can be performed.

FLIP-FLOP
A bi-stable device which may assume a given stable state depending upon the pulses of one or more input points and which has one or more output points. The device is capable of storing a bit of information, controlling gates, etc.

FLOATING-POINT REPRESENTATION
An arithmetic notation in which all numeric quantities have an associated indication of the decimal point location (base 10) or octal point location (base 8). Automatic alignment of numbers and calculation of the location of the point can be provided in arithmetic on floating-point numbers. A floating-point number consists of two parts: a 35 bit real value with sign called the mantissa; and a signed number called the characteristic (or exponent) which indicates the number of places to the right or left that the actual binary point is from the assumed binary point in the mantissa.

FORMAT
The predetermined arrangement of characters, fields, lines, page numbers, punctuation marks, etc. in input, output, or working storage records.

FORTRAN
A FORmula TRANslator language for writing problem oriented statements to be compiled and executed on a computing system.

FRACTIONAL
The portion of a number which is to the right of the decimal, octal, or binary point.

FRAME
In thin-film memory forty substrates in an 8 by 5 array, providing 256 words of 104 bits each. Eight frames comprise a side.

GATE
An electronic circuit with two or more inputs and one output, with the property that a pulse goes out on the output line if and only if some specified combination of pulses occurs on the input lines.

HARDWARE
The mechanical, magnetic, electrical, and electronic devices from which a computer system is constructed.

HARDWARE INDEPENDENT PROGRAMMING
Property of the B8500 to accept changes in system configuration and adjust programs accordingly to yield maximum utilization of all modules without reprogramming or recompilation of programs.

HOT JOB TABLE
An array containing the information necessary for initialization or reinitialization of a program which has been introduced to the system and is scheduled to run or has been suspended by the Executive Scheduling Program and will be resumed.

HOUSEKEEPING
Operations not directly concerned with the objective of a program; e.g., packing or rearranging data, subroutine linkages, etc.

INACTIVE FILE LIST
Directory of all files known to the system. (See SYSTEM DIRECTORY.)

INDEX
Increment or decrement to a base address.

INDICATOR
A light, usually on the operator's console, that is turned on to indicate a particular condition occurring in the computing system.

INDIRECT ADDRESS
An address which identifies a memory location containing an address. The contents of the memory location is the address of the desired information or may also be an indirect address.

INPUT-OUTPUT (I/O)
(1) Information introduced to or produced by the system.

(2) The Input-Output Processor Module which handles formatting reception and delivery of the above information to and from the peripheral device controllers and main memory.

INTEGRAL
The whole number portion of either a decimal or octal number. Refers to all the digits to the left of the decimal.

INTERFACE
The between device lines of communication. That which two or more devices have in common for the purpose of communication.

INTERRUPT
A signal generated as a result of a detected error condition, or service request. Provides the Executive Scheduling Program with the facility to maintain control of all system functions.

ITERATION
A single execution of repetitive program steps or a loop.

JUMP
An operation which may alter the execution sequence of a program. Normally instructions are executed in sequence; a jump operation causes a termination of the sequence and directs the Processor to a specified location for the next instruction. A conditional jump operation is a jump operation which takes place only if a specific condition exists in the Processor. Usually the condition is a result of a test or comparison operation. If the specific condition does not exist, a jump is not executed and sequential execution of instructions continues.

KEY
One or more digits or characters used to identify an item of information.

KEYBOARD
The portion of the supervisory printer via which the operator can communicate to the system.

LANGUAGE, MACHINE
Information recorded in a form which a computer can handle. The coded operations that control information and addresses employed within the processor to express a program.

LIBRARY
Collection of fully tested standard programs and subroutines for repeated use by, or incorporation into, other programs.

LINK
To provide a means by which physically noncontiguous data or program areas may be sequentially accessed.

LITERAL
An element in a program which is itself a quantity or alphanumeric constant to be used by the program rather than being an address of the quantity or constant.

LOCAL MODE
The mode of stack operation which limits the stack to the computer stack.

LOCATION
A storage position in a storage device distinguished by a unique address.

LOCK
Set a bit which when tested will indicate the condition and preclude any operation on the locked data.

LOG
Summary of scheduling, timing, program runs, etc. maintained by the Executive Scheduling Program.

LOOP
A coding technique whereby a group of instructions is repeated with instruction modification and/or with modification of the data being operated upon. It is a series of instructions, the last of which directs the computer to start again at the first instruction of the series.

MACRO
A subroutine of general utility; a group of instructions written to fulfill a certain purpose which may be called any number of times within one or more programs.

MAGNETIC DISK
A rotating disk with a magnetizable surface on which information may be stored as a pattern of polarized spots along any one of a number of concentric circular recording tracks.

MANTISSA
Significant bits of a floating-point number (35 binary bits in a single-precision number of the B8500). (See Floating-Point Representation.)

MAPPING
The technique of placing conceptually contiguous information in physically non-contiguous locations while providing a table and a method of linking the so formed portions of information.

MASK
A word in memory or a register which indicates which parts of another word are to be operated upon.

MASTER CLOCK
The device which controls the basic timing cycle of the computer. B8500 master clock frequency is 20 megacycles.

MEGACYCLE/SEC.
A million cycles per second. The basic pulse rate of the B8500 is 20 megacycles/second.

MEMORY EXCHANGE
Electronic switching logic which controls information flow among Memory Modules and the Processors or Input/Output Modules.

MEMORY MODULE
Two thin-film stacks each of which provide 4,096 words of 104 bits. As the two stacks of a memory module are simultaneously accessed the effective read or write may be to one of 4,096 words of 208 bits. To the processor, considering 52 bit words, the effective capacity is 16,384 words.

MEMORY STACK
(1) The storage locations made available by ESP to extend the computer stack in main memory, the capacity of which is limited by the stack bounds assigned by ESP.

(2) In a memory module the thin-film stack. (See THIN-FILM STACK)

MICROSECOND
One millionth of a second (0.000001 sec. or 1μs).

MILLISECOND
One thousandth of a second.

MODE, WORD
The method of operation of the B8500 in which the basic unit of information is a word composed of 48 bits plus three tag bits and one for parity.

MODULARITY
The property of a system resulting from the construction or assembly of the system from logical subunits (modules). In the B8500 this property provides the capability of constructing a system with the proper number of each type of module to match varying processing requirements efficiently and to maximize the utilization of each module.

MODULE
A logical subunit that may be easily detached from, or included with, the whole system. Processor, Memory, Input-Output, Magnetic Tape Units, and Storage Disks are typical modules of the B8500 System.

MODULUS (MODULO)
The number of distinct integers in a finite system of numbers. For example, in a modulo 5 system, the numbers are 0, 1, 2, 3, and 4. In this system larger numbers are expressed by dividing them by the modulus until a remainder less than the modulus is obtained. For example, 19 is 4 in the modulo 5 system. If a counter is "Modulo 5"; when it is set at 4, an increment of 1 will result in a setting of 0.

MULTIPLEX
The technique of sending several signals simultaneously over the same line.

MULTIPROCESSING
Processing several programs or program segments concurrently on a "time-share" basis. The Processor is only active on one program at any one time while operations such as input-output may be performed in parallel on several programs. The Processor is directed to switch back and forth among programs under the control of the EXECUTIVE SCHEDULING PROGRAM.

NESTING
Enclosing one program element of a particular type, such as a subroutine, within another of the same type.

NORMALIZE
To adjust the exponent and mantissa of a floating-point result so that the mantissa lies in the prescribed standard (normal) range; standardized.

NORMAL MODE
The condition of operation wherein the instructions are restricted to the conventional aspects of computation (adding, subtracting, information transfer, etc.). The detection of an exceptional condition (interrupt) that occurs while in this mode suspends operation in this mode and ESP processing begins in control mode.

OBJECT PROGRAM
A set of machine-language instructions for the solution of a specified problem, obtained as the end result of the compilation process (see Compiler, Source Language).

OCTADE
A group of 3 bits used to represent one octal digit. There are 16 octades in one B8500 word.

OCTAL
A number system based on powers of 8 rather than 10 as in the decimal system. Includes only the digits 0, 1, 2, 3, 4, 5, 6, and 7.

OPERAND
Any of the quantities entering into an operation. An operand is typically a number for arithmetic operations. For comparison operations, an operand may be an alphanumeric field.

OPERATORS
Symbols that denote a fixed, predefined set of operations to be performed in a specified sequence. There are a number of classes of operators in the B8500: for example, the arithmetic operators are: +, -, * or x , /, ** or xx; the relational operators are: =, $\neq$, >, $\geq$, <, $\leq$.

OVERFLOW
In arithmetic operations, the generation of a quantity beyond the capacity of the register or location which is to receive the result; over capacity; the information contained in an item of information which is in excess of a given amount.

OVERLAY
A technique for bringing routines into high speed memory from some other form of storage during processing, so that several routines will occupy the same storage location at different times; used when the total memory requirements for a program exceed the available high speed memory.

PACK
To combine several brief or minor items of information into one machine item or word by utilizing different sets of digits for the specifications of each brief or minor item.

PARALLEL OPERATION
Flow of data through the system or any part of it, using two or more communication lines or channels simultaneously.

PARALLEL PROCESSING
Processing more than one program at a time on a parallel basis, where more than one Processor is active at a time (distinguished from Multiprocessing where only one Processor is active on one program at a time).

PARAMETER
In a subroutine, a quantity which may be given different values when the subroutine is used in different parts of one main routine but which usually remains unchanged throughout any one such use. To use a subroutine successfully in many different programs requires that the subroutine be adaptable by changing its parameters.

PARITY CHECK
A summation check in which the binary digits, in a character or word, are added (modulo 2) and the sum checked against a single, previously computed parity digit. A B8500 word (52 bits) must contain odd parity (an odd number of Binary one's).

PERIPHERAL EQUIPMENT
Any of the several devices, primarily used to communicate with a system, not considered a part of the main processing and control system. On the B8500, the peripheral equipment includes Magnetic Tape Units, Disks, Line Printers, Card Readers, Card Punches, Communications Devices, Keyboard, Message Printer, Paper Tape Readers and Punches, Graphic and Video Displays.

POINTER
A register or storage location assigned to contain the address of a changing location.

POLISH NOTATION
A method of writing logical and arithmetic expressions without the need for parentheses, originated by the Polish logician J. Lukasiewicz. For example: Normal algebraic notation
(Z + Y) x (A - B) in Polish notation:
ZY + AB- x

POLLING
A technique of querying Input terminal devices. In the case where more than one device is attached to a line or buss, only one may be active at a time. The processor queries each device on a line in sequence. If the device is idle, it sends an answering signal and the polling sequence is continued. If the device is ready, e.g. tape mounted and under read head, the polling signal will act as a start signal and the device will be active on the line until completion of operation at which time polling will be continued with the next device.

PRECISION
The degree of exactness with which a quantity is stated. For example, the number 2.783 is precise to four digits, but does not necessarily have four digits of accuracy.

PRESENCE BIT
A single flag bit appearing in descriptors to indicate whether or not the information to which reference is made by the descriptor is in thin-film main memory at this time.

PRIORITY
A class assigned to a program or program segment to specify the relative processing demand. The priorities of all programs to be run are taken into consideration by the Executive Scheduling Program in arriving at a schedule.

PROGRAM (Noun)
A plan for the solution of a problem. A B8500 program may be a statement of the problem in ALGOL, COBOL, FORTRAN or the translated, segment object (compilation result) program.

PROGRAM (Verb)
To plan a computation or process from the original statement of the problem to the delivery of the results, including the integration of the operation of the resulting program into an existing system (for conventional computers). For the B8500: A system analysis and statement of the problem in source language.

PROGRAM REFERENCE TABLE (PRT)
An area in memory for the storage of descriptors for each external object referenced by a compilation. Permits programs to be independent of the actual memory locations occupied by data and parts of the program. Thus programs and data can be placed into any available memory areas without modification to the program.

PRT LINE
An entry within the program reference table.

QUEUE
Locations in a high speed local memory provided specifically for the storage of instructions and operands to which rapid access is desired.

RANDOM ACCESS
See Access, Random.

REAL TIME
Solving problems as they occur so that results can be used to guide the continuing operation.

REAL VARIABLE
A variable over the rational and irrational classes of numbers. In ALGOL a real variable is a floating-point number as distinct from an integer variable which is an integer.

RECORD
A group of fields maintained together as an item.

RECURSIVE
Having the characteristic of occurring within itself. The recursive occurrence could itself be recursive. For example: In ALGOL a Procedure which contains a Procedure Statement in its body calling for the activation of itself.

REENTRANT
A conceptual term referring to the independence and self-initializing capability of a program or program segment. The coding and data areas are organized in such a fashion that the execution by two or more processors at the same time will be independent of one another.

REGISTER
The hardware for storing one or more computer words or for maintaining internal system control.

RELOCATABILITY
A facility whereby programs or data may be located any place in memory at different times without requiring modification to the program. In the B8500, segments of the program and all data are independently relocatable with no loss in efficiency.

RESET
To return a device, bit or word to zero or its initial condition.

RESTORE
To return a cycle index, a variable address, or other computer word to its initial or a preselected value.

RETURN
An operator in a subroutine which transfers control to the next instruction in the original routine of the program which caused entry to the subroutine.

RETURN POINT
The instruction in the program segment to which control is transferred after the completion of a subroutine or an intercession by the Executive Scheduling Program.

ROUTINE
A set of coded instructions arranged in proper sequence to direct the computer to perform a desired operation or series of operations.

RUN
One performance of a program on a computer.

SCHEDULING
Designation of time and sequence of projected operations. One of the functions of the Executive Scheduling Program.

SEGMENT (Verb)
To divide a program into an integral number of parts, each of which performs some part of the total program and is capable of being completely stored in internal memory.

SERIAL
Processed one after the other in a single facility or single piece of equipment; sequential.

SERIAL TRANSFER
A system of data transfer in which elements of information are transferred in succession over a single line.

SET
To return a device, bit or word to one or to the "on" state.

SIDE
In thin-film memory eight frames. A side contains 2048 words of 104 bits. Two sides comprise a thin-film stack.

SIMULTANEITY
Concurrent communication between various units of a system at the same instant.

SLEEP TABLE
An array of information describing the exact processor status at the time of suspension of an Executive Scheduling process.

SOFTWARE
Programs, routines, and procedures which combined with hardware are a computer system (the Executive Scheduling Program, compilers, etc.).

SOURCE LANGUAGE
The language used by the programmer to state the definition of a problem. ALGOL, COBOL and FORTRAN are examples of source languages. Source languages are closely related to the type of problem being stated. Source language should not be confused with machine language. A program is written in source language by the programmer. This source program is then translated to the object program (in machine language) by a compiler program. (See Object Program, Compiler.)

STACK
(1) The total storage of operands which are automatically shifted toward or away from the T register, in response to the operand demands of the instruction string. (See Computer Stack, Memory Stack (1), Extension Mode, Local Mode.) A Stack as used in the B8500 operates on the "last-in first-out" principle, that is, the last item of information placed in the stack will be the first item of information used when information is required from the Stack. Operators perform their operations on information at the top of the stack. (See Operators.)

(2) In a memory module the thin-film stack. (See THIN-FILM STACK.)

STORAGE
Any device into which information can be copied, which holds this information, and from which the information can be obtained at a later time.

STORAGE ALLOCATION
Assignment of specific memory addresses to individual program elements (done automatically in the B8500) at object running time by Executive Scheduling Program.

SUBROUTINE
The set of instructions necessary to carry out a defined operation; a subunit of a program.

SUBROUTINE CALL
A set of characters or lists which initiate a subroutine and contain the parameters or identification of the parameters required by the subroutine.

SUBSTRATE
In thin-film memory a 70x43x0.2 mm glass plate containing 768 cells in a 24 by 32 array. Forty substrates make a frame.

SYLLABLE
A portion of a word. In the B8500, 6 bits.

SYNTAX
Connected system or order of symbol arrangement, the rules or grammar of a language.

SYSTEM
An assembly of components united by some form of regulated interaction to form an organized whole.

SYSTEM DIRECTORY
List of all files currently active or physically present on the system. (See INACTIVE FILE LIST.)

TAG BITS
Three bits which are used to indicate the type of word, i.e. a descriptor or control word.

TERMINAL
A device most remote from the central processor in a computing system configuration. That which is in direct communication with persons or equipment external to the system.

THIN-FILM STACK
In memory two sides (16 frames) providing 4,096 words of 104 bits. Two stacks comprise a memory module.

THROUGH-PUT
The total productive work capability of a system.

TIME-SHARING
Interruption of the operation of the main program in a computer by subsidiary or unrelated calculations, with the object of making economic use of computer speed when this is disproportionate to input-output speeds. Also, time-sharing refers to the ability of a module to refer to memory when the Memory Module is not being referenced by some other module. Since some modules make infrequent references to memory during their operation, by interleaving the references several modules can appear to be having access to a Memory Module simultaneously.

TRANSFER
To copy, exchange, read, record, store, transmit, transport, or write data; to change control.

TRANSLATE
To produce a statement in one language equivalent in meaning to a statement in a different language.

VERIFY
To check a data transfer or transcription, especially those involving manual processes such as keypunching.

WORD
A set of characters or binary digits which occupies one storage location and is treated by the computer as a unit and transferred as such. A word in the B8500 may contain 8 alphanumeric characters, a binary value in fixed or floating-point notation, two to eight instructions, literal syllables, or a program or data descriptor.

B8500
SYSTEM REFERENCE MANUAL
FORM BJ-8 OF APRIL, 1966

CHECK TYPE OF SUGGESTION:

☐ ADDITION ☐ DELETION ☐ REVISION ☐ ERROR

AFFECTING:

PARAGRAPH _____ ON PAGE NO. _____

CUT ALONG THIS LINE

SUBMITTED BY
NAME _____ LOCATION _____

YOUR COMMENTS ON IMPROVING THIS PUBLICATION ARE GRATEFULLY ACKNOWLEDGED.

BURROUGHS CORPORATION

STAPLE
THIRD

FOLD DOWN SECOND FOLD DOWN

Postage Will be Paid by Addressee

No Postage Stamp Necessary If Mailed in the United States

BUSINESS REPLY MAIL
First Class Permit No. 73, Paoli, Pa.

Burroughs Corporation
Defense, Space and Special Systems Group

Paoli, Pa., 19301

Attn: B8500 Program
Department 4419

CUT ALONG THIS LINE

FOLD UP FIRST FOLD UP