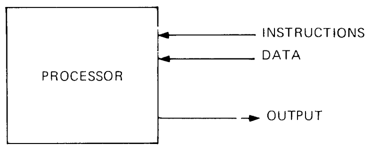
An Introduction To Burroughs B 6800 Systems

CONTENTS

SYSTEM ARCHITECTURE

History of Electronic Computer Systems

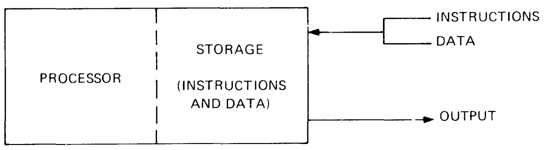
The first electronic computer, the ENIAC (Electronic Numerical Integrator And Calculator), was built in 1944 at the University of Pennsylvania by John Mauchly and J. Presper Eckert. It consisted of a "processor" which accepted a sequence of instructions (the program) and data (operands) from external devices and produced the results of calculations (output) which were recorded on an external device.



This machine was capable of "reading" and executing one instruction at a time.

In 1946, John von Neumann, professor of mathematics at the Princeton Institute for Advanced Study, conceived the stored program computer. In a paper titled "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" he set forth a number of principles which have influenced the design of every computer built since that time. The central concepts which have endured are:

a. Instructions and data are stored together in a homogeneous storage medium, and are indistinguishable;
b. Instruction modification during execution;
c. The program instruction counter, a register which contains the address of the memory location of the next instruction to be executed. The counter is automatically incremented with each instruction fetch.



The first computer to employ von Neumann's ideas was completed at Princeton in 1952.

Nearly all commercially produced computers since that time have been designed according to these principles and are said to possess "von Neumann architecture." Index registers and base registers have been added to the processor to facilitate indirect addressing and program looping and branching.

Von Neumann machines are characteristically well-suited for executing one program at a time. Developments in the early 1960s enabled them to concurrently operate on a number of programs or tasks. This is called multiprogramming. Although the processor still executes only one instruction at a time, several tasks can be stored in its memory at the same time.

In 1961 Burroughs introduced the B 5000, a revolutionary computer system which has permanently altered industry ideas of how a general purpose computer system should be structured. The B 5000 was the first computer system designed to automate both coding and operation.

Coding —	all programming, including system software, was done in higher-level languages; no assemblers were used.
Operation —	system operation was managed by a comprehensive software system called the Master Control Program (MCP).

These accomplishments were made possible by introducing a number of new concepts which were radically different from what had become conventional architecture. Although the B 5000 retained the von Neumann concept of a stored program computer, certain internal structures and indirect addressing mechanisms were utilized to make code distinguishable from data and to eliminate the program instruction counter and the requirement for modifying code during execution. The result was the first multiprogramming, multiprocessing computer system.

This same architecture has been carried forward by Burroughs Corporation through two successive generations of products including the B 6700, B 7700, B 6800 and B 7800 systems. What might be termed technological inertia has apparently prevented other manufacturers from converting from the conventional von Neumann architecture. However, many of the features introduced by the B 5000, such as operating system software, multiprogramming and virtual memory have subsequently been adopted by all other systems. There is one basic difference, however, between Burroughs B 7000/B 6000 series and conventional systems. Conventional systems have to adopt capabilities which are inconsistent with their basic architecture. The B 7000/B 6000 series architecture was designed to provide them.

Characteristics of B 7000/B 6000 Architecture

The architecture of these systems is the result of a composite of structures which are mutually reinforcing. Collectively they yield a number of distinguishing features.

Compiler-oriented hardware — Internal machine operators designed by programmers provide the functions specifically needed for efficient execution of programs written in higher-level languages. The use of a machine language assembler is excluded.

Comprehensive operating system software — The Master Control Program (MCP) provides management of all systems resources and job tasks automatically.

Automatic program segmentation — Higher-level language compilers automatically divide the program object code into variable length segments (as opposed to the fixed-size pages of other systems) based on the logical structure of the program. This assures that the memory requirement for executing the program will be minimized to only that amount necessary to contain the program segment(s) needed at any time during the course of the program execution history. Data arrays are treated similarly.

Descriptors — Individual words of information are used for locating program and data segments either in main memory or secondary memory (on disk). A descriptor is a pointer with three principle components: an address, a word count, and a presence-bit.

The address portion of the descriptor is the absolute address of the location of the first word of the segment. If the presence-bit is "on," the segment is present and the address refers to the main memory location of its beginning word. If the presence-bit is "off," the segment is non-present and the address points to the beginning location on disk.

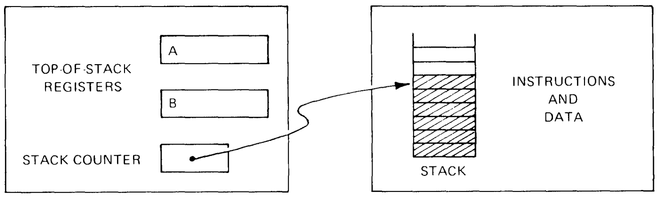
Presence-Bit Mechanism — The presence-bit in a descriptor indicates whether the segment pointed to is present in main memory or must be fetched from disk. If the presence-bit is "on" when the descriptor is referenced, the hardware interprets the address portion of the descriptor to be the absolute address of the main memory location of the first word of the segment, and accesses that location directly. If the presence-bit is "off," a present-bit interrupt occurs which causes the system to automatically fetch the segment from its location on disk pointed to by the address portion of the descriptor and place it in available space in memory. When this occurs, the system also updates the descriptor contents, both to indicate the segment is now present and to define its location. This allows code and data to be located anywhere in memory at any time to facilitate efficient memory management.

The word count portion of the descriptor provides the length of the descriptor in terms of memory words. This permits the segment length to vary with the logical structure of the program. This is in contrast with the rigid constraints imposed by fixed-size pages. Also, the program instruction counter of the conventional systems is, in part, replaced by an indexing mechanism which compares its present value with the word count in the descriptor. A hardware interrupt occurs when they become equal causing the next segment to be referenced.

Stack Mechanism* — The best known feature of B 7000/B 6000 architecture is the "stack." It has become its single, most distinctive characteristic when contrasted with the architecture of conventional systems. The stack is a set of sequential locations in memory that is temporarily assigned to a task or process for the duration of its execution. These locations serve as temporary storage for local variables, descriptors, and information pertaining to the

*The profound nature of the stack and all its implications to system operation are too extensive to cover here. A rigorous treatment of the subject is available in "Computer System Organization" by Elliott I. Organick, Academic Press, 1973, New York and London, Library of Congress Catalog Card Number 72-8834.

task execution. They also serve as an extension to two top-of-stack registers in the processor. The stack mechanism is activated by the use of an address-length register in the processor called the stack counter. This register contains the absolute address of the last referenced location in the stack. The address is automatically incremented by one each time an entry is made into the stack, and decremented each time information is removed. The operation of the stack is therefore last-in first-out. All entries to and exits from the stack are made through the top-of-stack registers.



Through the use of Polish notation, which defines specific rules for the sequence in which operators and operands occur, the last-in, first-out stack mechanism significantly reduces the number of memory accesses necessary to execute a task.

Interrupts — The system control function is invoked through an extensive hierarchy of hardware and software interrupts. This permits asynchronous operation of the various hardware modules. The hardware modules are, in turn, managed by the Master Control Program as a set of resources to be applied against an array of executable tasks upon demand. One of the most important functions of the stack is to store and restore processor state information in a fast and well-ordered manner. This permits the processor to be interrupted whenever it is beneficial to do so, without incurring excessive overhead. For this reason, B 7000/B 6000 architecture is uniquely well-suited for user requirements that are characteristically interrupt driven; on-line, high-volume transaction processing; simple multiprogramming batch; and time sharing environments.

Polish Notation — This method of notation was created to simplify the rules for algebraic evaluation. When applied to algebraic statements, grouping symbols and rules of operator precedence are not required. This allows a statement to be expressed in terms of operators and operands only, with the sequence of their appearance determining the order of execution. Thus the expression

$$(X + Y) \times (U + V/W) + Z$$

becomes

$$XY + VW/U + \times Z +$$

which, when interpreted from left to right, performs each operator in succession on the immediately preceding pair of operands (or results from prior operation). Using this notation in conjunction with the stack mechanism eliminates the need for the use of the fetch and store commands normally required for temporary storage of intermediate results in the evaluation of expressions.

Word Tags — Each word of information stored in memory is appended with a "tag" consisting of three binary digits. The tag uniquely identifies the word contents to the hardware. Words may contain program code, date, or descriptors. In this departure from conventional architecture, instructions are distinguishable from data. This permits the treatment of selected information as being "read-only." Program code, for example, is therefore unmodifiable while data, stored in the same memory medium must be modifiable. Notice that memory protection is provided automatically as a consequence of this construct. The major benefit of unmodifiable code is that multiple user tasks can execute concurrently while sharing a single copy of the code file in memory. This is called re-entrancy.

Zero-Address Instructions — Conventional systems operate on instructions which consist of an operator and one, two, or three addresses which specify the location of operands. B 7000/B 6000 machine instructions contain no addresses. In most cases, operands become available in the top-of-stack registers automatically by virtue of the way in which strings are evaluated. In those cases where a value or an address must be fetched into the stack, an address couple is used. An address couple consists of a number which specifies the lexicographic level (for example a nest level in a procedure or subroutine reference) within the stack, and an index value.

These jointly determine the absolute address of the location in question. The combined storage requirement for both instruction and address couple is less than that needed to store the actual address. This and the absence of addresses appended to operators results in significant program code compaction. Therefore, much less memory is used for storing code and fewer memory accesses are made to store and fetch code during execution. This results in more efficient operation.

Benefits of B 7000/B 6000 Architecture

The economic trade-off between embedding certain well-defined control functions in hardware constructs versus giving the responsibility for system control to software should be apparent. The cost of the former is the one-time expense of implementing it; the cost of the latter is the expense of system overhead every time the control function is executed. The cost in this case continues to accrue for as long as the system is in use. The B 7000/B 6000 concepts stress the former. Other benefits which relate more directly to system operation and the original purpose of the computer system, problem solving, are many. Among these, certain notable ones which were introduced by the Burroughs B 5000, and carried forth to the B 7000/B 6000 series are mentioned as follows:

Virtual Memory — The combined effect of automatic program segmentation, variable-length segments, descriptors, and the presence-bit mechanism. A program (data included) far exceeding the size of physical memory on the system may be written, compiled and executed with no further attention given to it than to the smallest program. The programmer is in no way constrained by memory size, but instead, views the memory to be virtually unlimited.

Re-Entrancy — Program code on a B 7000/B 6000 machine is not modified during execution. The system preserves the content of a code file in memory so that instructions may be fetched by multiple user tasks without conflict.

The stack mechanism and the use of descriptors for referencing the code in memory provide the means by which the code file can be shared among tasks. This reduces the memory requirement when multiple users execute a particular program or language compiler concurrently.

Configuration-Independent Software — The Master Control Program is a code file which is executed along with other programs in the mix. The MCP maintains data files which reflect the presence and status of all the system resources, e.g., processors, memory, peripherals, I/O paths. It also maintains files to monitor all tasks active and waiting. The MCP code file is, therefore, independent of the hardware configuration. Differences in configuration are, therefore, represented by data elements in the MCP's tables. By the same principle, all user programs are hardware-independent. Hence, a configuration can have resources added or removed without any effect to software. No re-compilation is ever required.

Automatic Resource Allocation — The MCP uses its resource tables to manage the use of system resources. By matching the demands of the job stream with resource availability, the MCP controls the operation of the system to assure maximum utilization of system resources, thereby maximizing system throughput and responsiveness.

Process Switching — The use of the stack mechanism to store processor state information when an interrupt occurs provides the means by which productive use of the processor is maximized. There are many reasons for interrupting the processor which relate to system productivity. One example is a task or process interrupting itself because it requires a data file from disk before continuing. Instead of having the processor wait for the completion of the necessary I/O, its current state is stored in the stack and the processor is switched to another process which is waiting for execution. Later, when that I/O is completed, the processor can return to the interrupted process, restore itself to its state at the time of the interrupt, and resume execution at that point. By making the process switching mechanism extremely fast, the MCP can assure maximum utilization of the processor and, therefore, maximize throughput. Conventional systems must execute a complicated and time consuming procedure to switch processes, and hence lose productive time in overhead or in a wait state.

Asynchronous Processes — Through the use of multiple levels of stacks and a mechanism which permits one process to start additional process stacks, the system is able to have asynchronous processes operating concurrently. In this way, when a large volume of similar activities are to be handled, as in the case of transaction processing applications, a multiplicity of processes can be initiated to handle them. For example, when each transaction requires I/O activity, the presence of a number of asynchronous processes guarantees that the processor will always find at least one process available for execution. It will, therefore, have to spend no time in a wait state. The fast process switching mechanism, in this case, maximizes processor productivity. In addition, re-entrant programming allows all the asynchronous processes to be executing out of a common code file, minimizing the requirement for memory space.

The Importance of Architecture

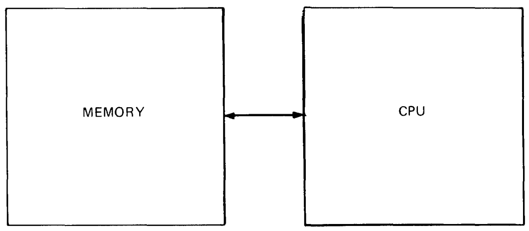
Many economies have been realized in the implementation of B 7000/B 6000 architecture. These translate into greater dollar savings and increased productivity and responsiveness for the user. These systems offer a range of features and capabilities which provide the user with the opportunity to formulate fresh solutions to problems; solutions which relate more directly to the problems themselves rather than to the requirements of the computer system. The B 6800 and B 7800 systems were designed with the guiding philosophy that computers should be tools for their users.

It is crucial to note that during the entire history of computers, hardware costs have gone down and will continue to do so. During the same time, application software development and maintenance costs have been rising, and will continue to do so. In the future, the most significant concern of users of conventional systems as compared to users of Burroughs systems will be the cost of getting the job done.

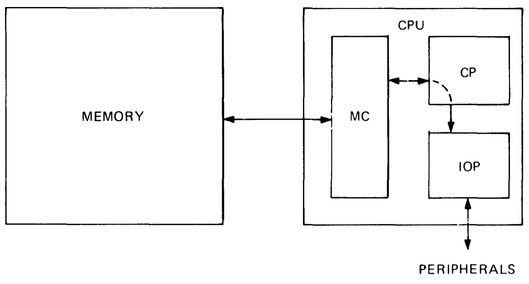
B 6800 HARDWARE

B 6800 System Organization

The basic structure of the B 6800 system consists of a Central Processing Unit (CPU) and Memory.

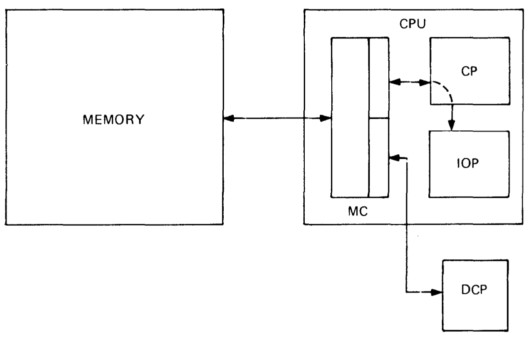


The CPU is comprised of the Central Processor (CP), the Input/Output Processor (IOP) and a Memory Control (MC).

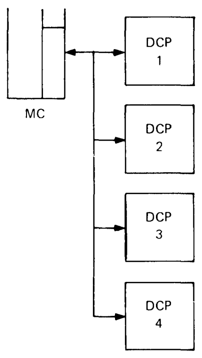


The CP executes program instructions. The IOP transfers data between peripherals and memory. The Memory Control manages the transfer of information between memory and the CPU.

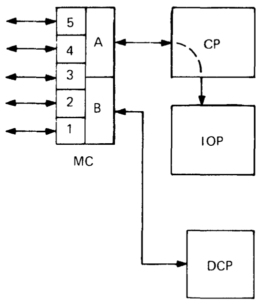
The Memory Control also provides an interface to memory for external devices such as the Data Communications Processor (DCP) and/or the Reader Sorter Processor (RSP).



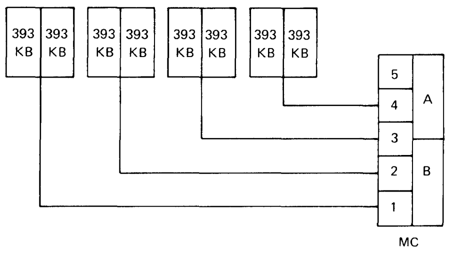
Up to four DCPs or other external devices can be connected to the Memory Control.



The Memory Control provides up to five access paths to memory which are shared by two requestor paths, A and B.

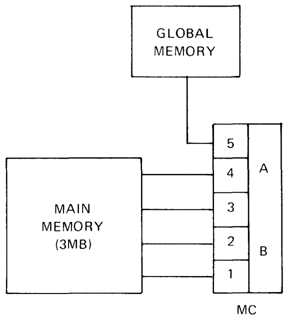


Four of the access paths can be connected to a maximum of eight memory modules; each module contains 393K bytes.

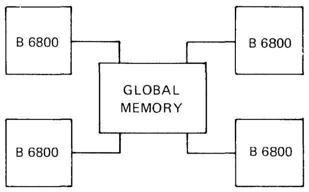


These memory modules can be added to a system as needed to provide a maximum total of three million bytes of memory. This is called Main Memory.

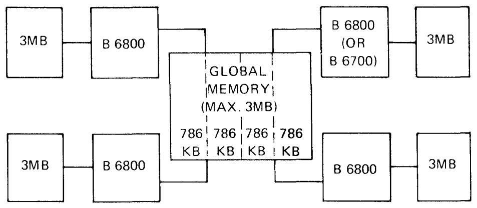
In addition to Main Memory, a second hierarchy of memory has been introduced on the B 6800. It is called Global™ Memory. Global™ Memory can be connected to the fifth access path of the Memory Control.



Global™ Memory forms the basis for a new concept in multiprocessor systems. By utilizing Global™ Memory, up to four B 6800 systems can share a common memory subsystem.



Global™ Memory consists of one to four 786K-byte modules for a maximum total of three million bytes.

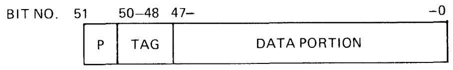


A single-processor system can have up to three million bytes of main memory and three million bytes of Global™ Memory; a dual-processor system can have up to six million bytes of main memory and three million bytes of Global™ Memory. The maximum combined total memory for a 4 x B 6800 multiprocessor system is fifteen million bytes.

Processors	Maximum Main Memory	Maximum Global™ Memory	Total Memory
1	3MB	3MB	6MB
2	6MB	3MB	9MB
3	9MB	3MB	12MB
4	12MB	3MB	15MB

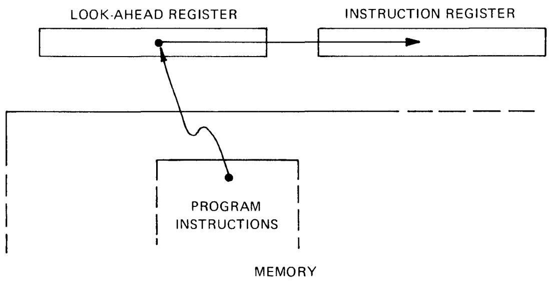
Central Processor Concepts

Architecture — The B 6800 Central Processor is a stack machine. Its instructions are expressed in 8-bit syllables and range from 1 to 12 syllables in length. The CP fetches, stores, and operates on 52-bit words, consisting of 48 data bits, 3 tag bits for control purposes and 1 parity bit.

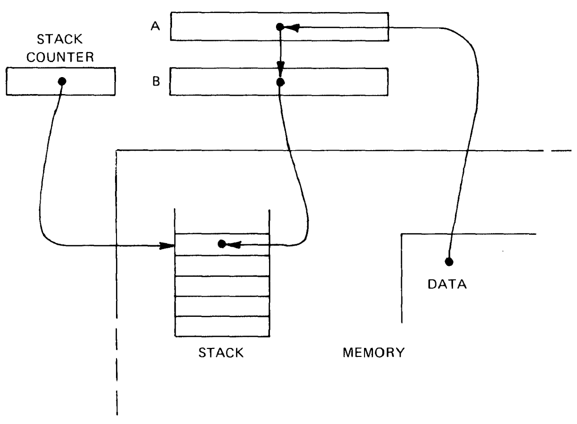


The data portion is used to store program instructions, operands, descriptors and a variety of control words.

Program instructions are placed into an instruction register for execution. Look-ahead logic fetches the next required word of code from memory and places it into a look-ahead register; overlapping instruction fetches with instruction executions. Shifting of the contents of the look-ahead register to the instruction register is done automatically.



All operations in the CP are performed on the contents of two top-of-stack registers, A and B. These registers are capable of handling both single-precision and double-precision operands. Operand fetches from memory are placed into the A register. While this is done, the contents of the A register are shifted automatically into the B register. Similarly, if the B register contains information, its contents are automatically stored into the next word of memory on the top of the "stack." This is called a stack "pushdown."



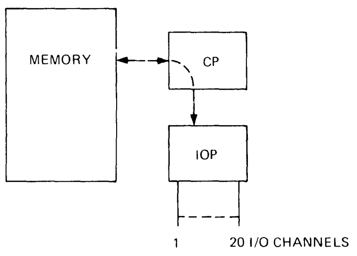
The memory address size is 20 bits, allowing the CP (and IOP) to access an address space of more than six million bytes. A special 20-bit register in the CP called the stack counter register (S register) contains the address of the top word of the stack. When a stack pushdown occurs, the address in the S register is incremented by one, and the contents of the B register are stored in the memory location identified by that address. In this way, the stack grows through sequential locations.

If an operator requiring two operands is to be executed next, and only one top-of-stack register is occupied, the top-of-stack word addressed by the S register is fetched into the B register and the S register contents are decremented by one. Execution of the operator then occurs. This process is called a stack "pop-up." It takes place automatically and is the reverse of a stack pushdown. The pushdown and pop-up are stored and fetched without using store and fetch commands. The stack grows and contracts during the course of the job execution. The utilization of memory as an extension of the top-of-stack registers gives the processor an elastic boundary which always fits the requirements of the currently executing task.

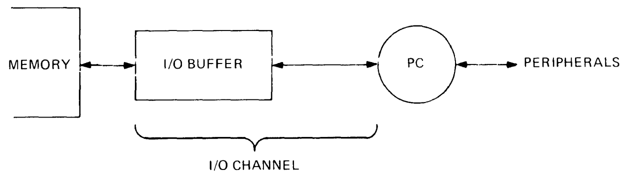
Input/Output Processor Concepts (IOP)

The IOP handles all transfers of information between peripherals and memory. The peripherals include card readers, line printers, magnetic tape units, disk-pack units and head-per-track disk.

The IOP accesses memory on the same path used by the Central Processor. The IOP can coordinate up to twenty concurrent I/O transfers through twenty channels.



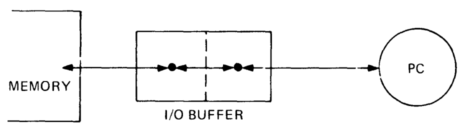
Each I/O channel is buffered; it connects to a Peripheral Control (PC) which manages access to peripheral devices. The PC also is buffered.



Information is transferred between the I/O Buffer and the Peripheral Control along a one-byte or two-byte parallel bus, depending on whether the peripheral is a low-speed (e.g., line printer) or high-speed (e.g., magnetic tape) device. A Peripheral Control is serviced by the I/O Processor at rates up to a maximum 2.2 million bytes per second.

Information is transferred between the I/O Buffer and Memory in full-word accesses (i.e., 6 bytes/access) at the speed of memory.

The I/O Processor may have up to eight I/O Channels each with 256-byte buffers, and 12 I/O Channels each with 512-byte buffers. An I/O Buffer is filled one-half at a time. While one half is being filled, the other half can be emptied.



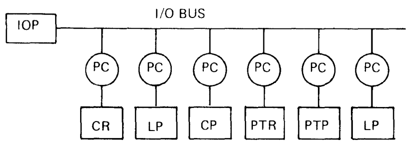
This allows the access to the peripheral to proceed independently of I/O Processor access to memory. It also allows the access to memory to be made in burst mode, thereby reducing memory contention.

All I/O transfers are initiated by the Central Processor through an interrupt to the IOP. The CP interrogates the IOP for an available path to the required peripheral unit. If no path is presently available, the MCP places the I/O request in a queue to await path availability. When the path is available, the CP passes an I/O descriptor to the IOP. The I/O descriptor (six bytes) contains the information necessary to direct the IOP to perform the I/O as required. When completed, the IOP interrupts the CP and passes a result descriptor to it which identifies the I/O and its status.

Peripheral Configurations

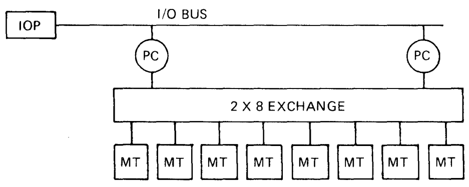
All peripherals are accessed through a Peripheral Control (PC) which provides the interface to the I/O bus, buffering of the peripheral unit (in some cases), and access control to the peripheral unit(s).

Low-speed peripherals (card readers and punches, paper tape readers and punches, printers) each have their own PC.



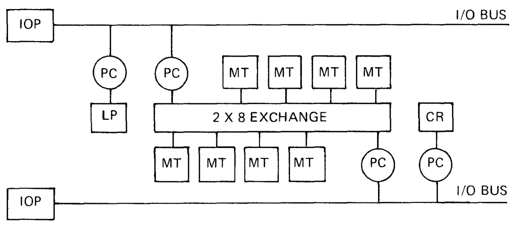
There is one path to each unit in this category. Each peripheral has a unique number, e.g., CR12, LP28. Numbers are 0-255; therefore a maximum of 256 peripheral units may be connected to the IOP.

High-speed peripherals (fixed-head disk, disk pack, magnetic tape) can have multiple paths and use an "exchange" (e.g., a 2 x 8 tape exchange).



In this diagram, I/O operations can be taking place in any two of the eight units simultaneously. If a third I/O is attempted, the IOP would return a "no path" message to the CP and the MCP would queue the operation until a path became available.

In the case of the multiprocessor B 6800, high-speed peripherals can be attached to two or more IOPs via a peripheral exchange.

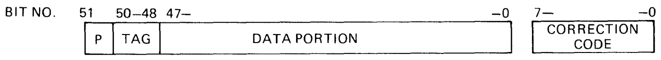


With this arrangement, there is a path to any one of the eight MTs (or other high-speed peripherals) through either of the IOPs. This provides increased throughput as well as path redundancy.

Main Memory Characteristics

The B 6800 Main Memory is word addressable. One word contains 48 binary digits (six bytes) of information, one parity bit and three tag bits which identify the word content (data code for internal security). A word of information is passed to the Central Processor or I/O Processor in one read access.

An 8-bit error correction code is appended to each word in memory. This is used by the Memory Control to correct 1-bit errors if they occur during memory accesses.



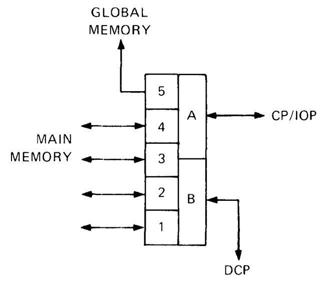
Two-bit errors will be detected and recorded.

Main memory is available in 393K- or 786K-byte increments (modules) which can be added to the system at any time to accommodate increased memory requirements on the part of the user.

Two Main Memory modules can be interleaved in a way that accesses made to sequential locations will alternate between the two modules.

This is done by having odd addresses in one module and even addresses in the other. The result is that memory cycles of the two modules can overlap, allowing the next sequential location to be accessed in one module while the other module is completing its memory cycle for the previous access.

Simultaneous accesses into memory modules on different paths to the Memory Control can be made by the CPU (Central Processor/IOP) and a Data Communications Processor or other external device.



The Memory Control acts as a 2 x 5 exchange, allowing any two of the five memory paths to be active at the same time. Simultaneous requests for the same path will be resolved by the Memory Control.

Global™ Memory Concepts

The B 6800 implementation of Global™ Memory provides three different approaches which demonstrate its organizational versatility. These approaches provide for a multiprocessor mode, a shared resource mode and an independent mode.

MULTIPROCESSOR MODE

A multiprocessor system is controlled by one Master Control Program (MCP) and presents a single system image to users, operators, and programmers. In this mode, the system is analogous to the B 7700/B 6700 multiprocessor systems.

In the multiprocessor mode, the information residing in Global™ Memory is the code and the tables that must be utilized by all processors. Examples of items which must reside in Global™ Memory are: MCP code; unit tables; data base stacks; and buffers for Global data bases. Items which will reside in main memory include: data arrays; user stacks; user code; local I/O buffers; data base stacks; and buffers for local data bases.

Job queue attributes allow users to specify that a job must be run on a specific B 6800 system. This enables installations to bias individual processors to a particular type of work in order to reduce interference between non-compatible applications. Inter-Program Communication (IPC) based programs can run in Global™ Memory so that independent modules can be initiated on separate systems.

SHARED-RESOURCE MODE

Shared-resource B 6800 systems are totally independent. They use Global™ Memory for passing messages from one system to another and as a memory extension for each B 6800 system.

No peripherals are physically shared by systems in this mode. However, peripherals can be switched from one system to another by using the peripheral configuration panel. Files on one system may be logically accessed from other systems via inter-system messages. In this mode, Global™ Memory need only contain message queues which allow a system to request and receive data from another system.

Each B 6800 system in the shared resource mode has its own MCP. Each MCP initiates special processes which control the message queues in Global™ Memory. Systems may communicate with each other in this manner. Programs may access files on other systems by opening a file which has an attribute. The attribute indicates on which system the file resides. The MCP will establish communication with the system that owns the file and set up message queues in Global™ Memory for the transfer of data. When a program accesses a record, the MCP will communicate the request to the system that can access the information. The system controlling the file will perform the I/O into a buffer or message area in Global™ Memory. Upon completion of the I/O, the MCP will be notified and it will link the buffer to the requesting program which can then move the information to a work area.

Additional facilities are available while operating in shared-resource mode. These facilities allow: initiation of tasks on other processors; communication between processes in different systems via queues that are accessible by more than one system; and determination of the status of other systems.

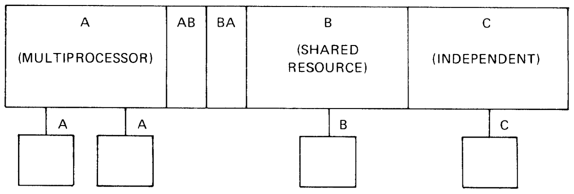
INDEPENDENT MODE

The independent systems approach permits completely autonomous systems to be easily configured with varying amounts of memory, without departing from the simplicity of separate systems.

In independent mode each system has its own copy of the MCP and its own piece of Global™ Memory. Sharing of files and peripherals is prohibited in this mode.

Independent mode is particularly useful when it is essential to isolate operating environments.

It is also possible to run in Global™ Memory with all three modes active at once. The diagram depicts four processor systems running two processors in the multiprocessor mode, one running in shared-resource mode and one running in independent mode.



Global™ Memory clearly provides a degree of flexibility in system configuration unavailable with other architectures.

Global™ Memory Characteristics

The Global™ Memory Subsystem consists of a Global™ Memory Control (GMC) and Global™ Memory Modules (GMM). The Global™ Memory Control has the following characteristics:

- Permits up to four processors to access a common memory area and controls the read/write access by each processor to user-defined logical partitions of memory;
- Has up to four asynchronous requestor ports which provide connection for up to four B 6800/B 6700 systems;
- Has an average access time of 900 nanoseconds;
- Has up to four memory port adapters which provide for the connection of up to four memory modules which are identical in size, with either two- or four-way interleaving.

SYSTEM SOFTWARE

Basic Concepts

The B 6800 systems are programmed exclusively in higher-level languages. No assembler language is provided. Users may therefore concentrate on the problem and its solution with few if any, hardware-related constraints. This provides several benefits:

- Effectiveness; problems are solved in minimum time.
- Efficiency; maximum use is made of the hardware.
- Problem Solving; by concentrating on the problem at the problem level, the data processing department becomes more effective.

The hardware/software system is designed for maximum job throughput. Solutions to problems therefore make efficient use of the hardware. This is made possible by software- and hardware-supported functions.

- Multiprogramming
- Multiprocessing
- Virtual Memory
- Global™ Memory

The exclusive use of higher-level languages requires fast compilers. All compilers (not only B 6800) work on the stack principle when compiling higher-level language statements. The B 6800 compilers rely on hardware rather than software to perform stack manipulations. Hence they provide unusually high compilation speeds for machines of this class. Typical figures are 7,000 card-images/minute for compiling programs.

All System Software is written in ALGOL or a superset of ALGOL:

- ALGOL-60 with extension for I/O, bits and characters. All compilers are written in ALGOL.
- The operating system is written in ESPOL (Executive Systems Problem-Oriented Language), a high-level ALGOL-like language.

Object code files are stored on fixed disk or disk pack and fetched into memory to be executed. There are no "object decks" except the special case of the "loader deck" for system initialization. Object code cannot be modified so the same problem can be executing two or more times concurrently without risk of change to the object code. Object code is protected by hardware from modification by use of tag bits.

Operating System Concepts

The operating system is called the MCP (Master Control Program). It is compiled in ESPOL and the code file resides on fixed disk or disk pack.

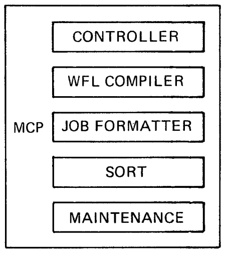
Conventional systems usually require a special operating system for each configuration. Thus the operating system is configuration dependent.

The B 6800 MCP interrogates the system hardware to determine the configuration. The MCP then maintains tables of information which reflect the current status of each hardware resource, updating its tables with each change of status. This is done for mainframe resources and peripherals. Consequently, no system generation is ever required. The MCP automatically adapts to changes in the environment.

The MCP consists of many routines which handle such things as initiation of all I/O operations, memory management and resource allocation. Only a small part of the code (and the tables) to handle these and other functions resides in memory; the rest resides in the MCP code file on fixed disk or disk pack, and is fetched into memory only as needed.

There are no fixed partitions in memory. Each program will occupy only as much memory as it needs for efficient execution, constrained by other concurrent demands on memory. Logical I/O routines are called by a user's program from the MCP code file when required. The resident part of the MCP will, itself, call in routines from the MCP code file when needed (e.g., to read a tape label when a tape is mounted on a drive). Therefore, parts of the MCP will be executed by users' programs while other parts will be operating independently.

The MCP code file contains five modules:



The CONTROLLER is the interface between the operator and the outside world. The WFL (Work Flow Language) COMPILER accepts jobs and creates job streams in ready-to-run format on disk. The JOB-FORMATTER is a print module used to print job output. The SORT is a disk/tape/memory sort, bound into the MCP. MAINTENANCE is an on-line testing package for engineers. The MCP module in the diagram represents all housekeeping and utility routines such as memory management.

SYSTEM OPERATION

Operator Communications

The console operator has a large repertoire of English-type commands to control and determine the status of the system, peripheral devices, and program execution. In addition, there is a comprehensive set of commands that are directive or informative. All system commands are easily recalled or referenced mnemonics.

For example:
"SP" (Show Print Queue)
By entering the SP message on the supervisor console, a list of all jobs in the automatic output queue waiting for a line printer will be displayed.

"PC" (Print Configuration)
A "PC" entry will display the system configuration while a "UA" (Unit Available) will cause a particular peripheral unit, previously made unavailable, to become available to the system.

By use of the "ADM" system message the console operator has the ability to set-up the console in "AUTOMATIC DISPLAY MODE." The operator may logically divide the console screen into areas on which will be displayed a variety of information reflecting the current status of the system at regular intervals.

For example:

ADM (ACTIVE 9, WAITING 5, COMPLETED 6, MESSAGES) DELAY 10

The above entry will display on the supervisor console **active** tasks (jobs) on the first nine lines, jobs or tasks **waiting** to be run will be displayed on the next five lines and the last six **completed** tasks will be displayed on the next six lines, with the entire screen updated every 10 seconds. The remainder of the screen will display system messages to the operator. If the information to display exceeds the screen space allocated, the excess will be displayed after a brief delay.

The Automatic Display feature can be started, stopped or changed at any time to suit the needs of the installation. In addition, multiple pages, each with varying formats, can be displayed each with its own time delay before refreshing the screen with the next page.

There is an "event" option to refresh the screen only when the status of one of the requested items changes but no sooner than a specified delay value.

For example:

ADM EVENT MESSAGE, PER MT DELAY 5

This entry will update the screen when either a new message is generated by the system or the status of the magnetic tape subsystem changes, with a minimum delay of 5 seconds before the screen is updated.

The console operator may also interrogate many other system features such as memory usage, peripheral status, and current job structure and then either display them in the ADM or call out the information on request.

Note: In the case of two or more console screens, any screen may display any choice of information (including "NONE").

In the Work Flow Language there is a feature that provides a block of instructions for the operator. At any stage in the life of the program during execution, the operator may interrogate the job for instructions as what to do next.

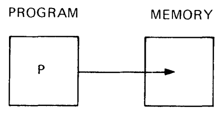
For Example:

"IF NO FILE MSG APPEARS, ABORT THE JOB"

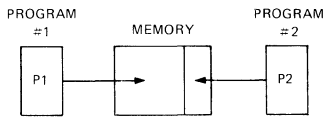
This feature can be used in conjunction with "JOB RUN SHEETS" or may eliminate the need for run sheets altogether.

Multiprogramming/Virtual Memory

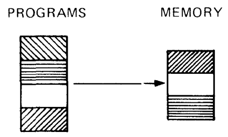
Virtual memory and multiprogramming are incorporated as an inherent part of the design of the B 6800 system. In systems without these features, programs have to be written to fit within the limits of available main memory.



In other systems with slightly more sophistication a partitioned approach is used:

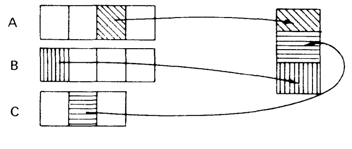


The partitioned approach can be improved by dividing the program into a number of equal-length "pages" so that the number of pages necessary at any given time during its execution will fit into the partition allotted to that program:



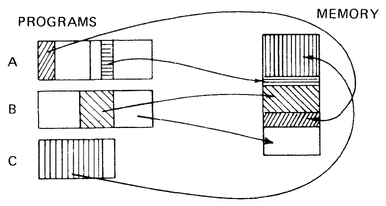
In the preceding example, the program divided into four pages may have any three of the pages in memory at one time. The missing page is stored on another media such as disk pack.

Paging can be used to increase the number of programs in the mix (jobs in main memory at the same time). For example, if programs A, B, and C all use only one page at a time, they can all be active (in main memory and running) at the same time:



The preceding approach makes better use of the system resources, at the expense of an increase in total execution time for each program. The additional execution time is due to the system overhead incurred in overlaying pages.

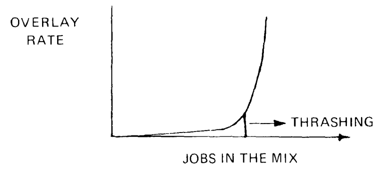
The B 6800 MCP dispenses with partitioning and fixed-length paging. Since the logical structure of programs cannot be efficiently constrained to a fixed page size, code is divided into variable-length "segments." Each segment is allocated space in main memory wherever space large enough can be found.



In the case of the B 6800, an executing program or task has "descriptors" in its working area (stack) pointing to each segment of code or data. Each descriptor is marked according to whether the segment is present in memory or absent (stored on disk) by setting a "presence-bit" in the descriptor. If sufficient space is not available in memory for the segment, space is made available by overlaying into space already in use. Data segments are written back to disk and then marked absent in the appropriate descriptor. Code and read-only data cannot be modified, so these segments are not written to disk. Only the descriptor is changed.

On partitioned memory systems, the "memory requirements" of a program can be easily determined; the program is the size of the partition. Similarly, the program will use exactly that amount of memory (some memory will be wasted). On paged systems the memory used by a running program will be the number of page locations allocated to that program. With the B 6800 virtual memory, the actual memory used by a running program is determined by the total demands of all the running programs in the system. With only one program in the mix, the program will use as much memory as is needed (it can logically use more than the real memory available on the system; some segments will always be on disk). With additional programs in the mix, each program may be constrained automatically to use less memory. Hence, the "overlay rate" may increase and each program's execution time may increase slightly. By this method, it is possible to maximize the use of the system resources, and consequently, the total system throughput. Because program segmentation and overlay are handled automatically by the system without any effort on the part of the programmer, this is said to be a "virtual memory" system. It permits both program and data to exceed many times the physical memory size.

If the system is allowed to continue indiscriminately adding jobs to the mix, the overlay rate will increase to produce unacceptable overhead. At some point as the number of jobs in the mix increases the overlay rate begins to increase rapidly.



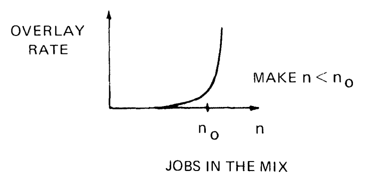
When a rapid increase in overlaying occurs and more time is spent overlaying than executing jobs, the system is said to be "thrashing" or "overlay bound." Thrashing occurs if the memory available for the mix is so restricted that an insufficient number of segments are present in memory at a given time. Each segment request results in an overlay call.

There are three approaches to avoid thrashing:

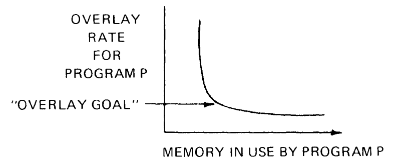
- Restrict the number of jobs in the mix by scheduling;
- Design programs to use a smaller amount of memory at any one time, i.e., have a smaller "working set," and do not pass control from one segment to another and back again unnecessarily (this occurs with paging systems);
- Add additional main memory.

On the B 6800 system, one can use the MIXLIMIT message to restrict the number of jobs in the mix. The MCP will then schedule jobs as long as the available memory is greater than the memory estimated (by the compiler at compilation time) to run the program.

The B 6800 MCP will suspend the lowest priority job in the mix in order to accommodate a job whose working set expands during its execution, i.e., its demands on memory increase. This action will be taken only if the available memory becomes less than a specified amount. This mechanism brings the number of jobs, n, below the thrashing point, n_o:



To make more memory available, the user can either add more memory to the system or deliberately overlay memory before it is requested. The desirable overlay rate will depend on the mix and is ideally at a rate just low enough to avoid thrashing.



Treating the above diagram as applicable to the whole system (an assumption only), overlayable memory for each program is deliberately overlayed to reach the "overlay goal." This overlay goal is a run-time parameter specified by the operator. Also, because the shape of the curve is different for different programs, the overlay goal can be set for individual programs.

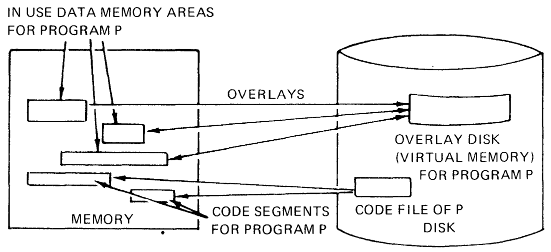
Note: The job-suspension and overlay goal methods of control are optional. Without either method, thrashing can be avoided by scheduling jobs and by regulating the MIXLIMIT to the job queues.

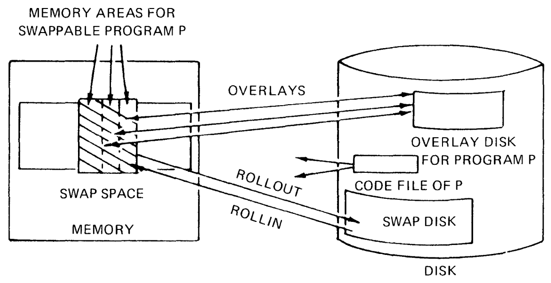
Swapping Technique

Some programs, especially in a data communications environment, are "burst-oriented." An example is a conversational program, which may take 30 seconds to wait for a reply to be typed into a remote terminal. During that time it is not necessary for any part of the program to be in memory (including save memory, such as stacks and I/O buffers). It is, therefore, rolled out to disk, and then rolled in again only when required, i.e., when the data communications I/O operation is completed. Rolling in again may require the rolling out of another program; hence, the term "swap-mode" program.

To implement swap-mode efficiently, some or all of the memory area for one swappable program must be in one area called SWAPSPACE.

This place is then written to a fixed place on disk (called SWAPDISK, this is not virtual memory nor OVERLAYDISK), in one physical I/O operation. The program can use virtual memory but obtains main memory only within the allocated SWAPSPACE. The size of SWAPSPACE is specified by each particular installation.


IN USE DATA MEMORY AREAS FOR PROGRAM P
OVERLAYS
OVERLAY DISK (VIRTUAL MEMORY) FOR PROGRAM P
MEMORY
CODE SEGMENTS FOR PROGRAM P
CODE FILE OF P
DISK



Swappable jobs (or tasks) can be so designated in the job deck (Work Flow Language Statements) and in the queue to which the job is assigned. The primary application of this feature is in time sharing.

Batching Technique

One area of possible inefficiency is the start-up and end-of-job overhead for short (execution time) programs. The combination of start-up and end-of-job could easily exceed the execution time. By combining consecutively run programs into a single program, the overhead is more acceptable.

The automatic re-entrant feature of the B 6800 system reduces the start-up and end-of-job times for batch processing by requiring only one copy of the program in memory. For example, when more than one FORTRAN program is being compiled in the same MIX, only one copy of the FORTRAN compiler will be in memory, thus saving not only start-up and end-of-job time, but also memory.

System Log

The B 6800 MCP has the facility for maintaining a "JOBLOG" for each job executed. In addition, a "SUMLOG" (SUMMARYLOG) may also be maintained. All information pertaining to the execution of jobs such as times, files opened and closed, messages displayed, etc., may be written in either or both logs.

JOBLOG information may be obtained by entering one of the following messages via the operator console:

- "HARDCOPY" intercepts messages to and from the console and writes a copy to a disk file. This file may be obtained upon request.
- "REMOTESPO" (Remote Supervisory Print Out) can be specified for one or more data communications terminals where operator information can be received and entered. This can also be a CRT device.
- A user may also designate his own hard copy SPO (Supervisory Print Out) programs rather than use the above Burroughs supplied programs.

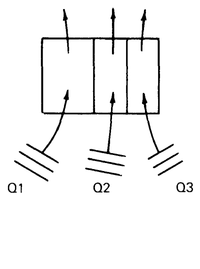
The SUMLOG is used primarily for program and job debugging. The log is especially useful for the display of messages if incorrect results are obtained. It also displays the abort point in the event of program failure. The SUMLOG may be printed in time order or selected types of entry (e.g., messages) using the Burroughs loganalysis program.

The SUMLOG is used for billing and accounting as well. Job times can be extracted and bills produced by using the Burroughs furnished logger program. Details of the SUMLOG file are made available so that customized billing/accounting programs may be written.

Job and Task Control

CONVENTIONAL PARTITIONING

Using this method, the physical machine is divided into fixed partitions. Each partition has only one "stream" or queue for jobs and the jobs are constrained to use only those resources allocated to that particular partition. The multiprogramming factor is usually the number of partitions.

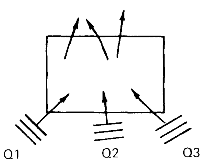


B 6800 APPROACH

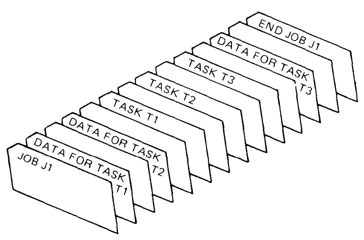
Jobs are batched into "job queues" or streams depending on their logical characteristics or class of service required, e.g.,

- Small jobs, fast turnaround
- Larger jobs, slower turnaround
- Very large jobs, overnight turnaround
- "Cafeteria" type service, almost instant turnaround
- Jobs with no operator interference
- Jobs with operator interference, e.g., tape handling, cards, etc.

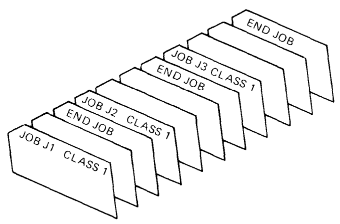
Jobs from these queues will be released into the system and the necessary resources allocated by the MCP on demand.



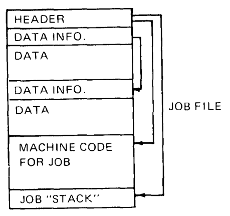
A job is a collection of related tasks and the data for those tasks.



A queue is a collection of jobs of the same class.



The Work Flow Language (WFL) compiler will compile a job and then create a job file on disk containing machine instructions as to what tasks are to run in which order, and the action to take on errors.



IMPLEMENTATION

With the B 6800, we can ignore all classes of service and submit all jobs to the system in one (default) queue. Alternately, a typical work-flow management strategy would be:

— Queue for small jobs, high MIXLIMIT, fast turnaround, low limit on resources, high charge for jobs using this queue.

— Queue for large jobs, low MIXLIMIT, slow turnaround, no limit on resources, lower charge rate for jobs using this queue.

— The MCP will allocate resources so that (for example) one large job and several small jobs can be sharing the system, but a second large job will have to wait for the first large job to complete, hence a slower turnaround.

— MIXLIMITs may be changed dynamically to meet changing demands or fluctuations in the work flow of the installation.

QUEUE PRIORITIES

Associated with each queue can be a default priority (0-99, with 99 running the fastest) and a priority limit. If the job priority is not specified, a default priority will be assigned. If the job priority is greater than the appropriate queue limit priority, that job will not be inserted in any queue. Jobs are linked into the queue in priority order, within one priority in first-come first-served order. The operator can rearrange the order of jobs in a queue if required. For example, consider a queue with default priority = 70, limit priority = 80, and jobs arriving in this order (where J(X) means job name J with priority X, J means job name J with no declared priority):

J1(75), J2, J3, J4(72), J5(81), J6, J7(78), J8(68).

The order of jobs in the queue will be:

J7(78), J1(75), J4(72), J2(70), J3(70), J6(70), J8(68).

J5 will not be placed in this queue since its priority limit is higher than the queue priority limit.

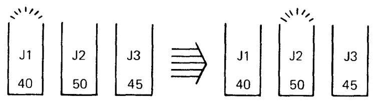
OTHER QUEUE ATTRIBUTES

Job queues can be created or deleted at run-time but are normally set up as installation standards. The following is a list of attributes that may be set up as installation standards:

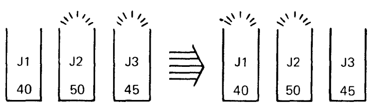
- PRIORITY;
- MIXLIMIT;
- IOTIME, Limit for each job;
- PROCESSTIME, Limit for each job;
- LINES, Limit on number printed for each job;
- CARDS, Limit on number punched for each job;
- FAMILY, Disk or disk-pack family name alternates;
- TURNAROUND, An approximate time interval before the next job in that queue is selected to run;
- SUBSPACES, Memory allocation alternate.

Multiprogramming has been implemented by both hardware and software. The MCP keeps tables of the current "mix." One job in the mix is denoted by a "stack" or chunk of memory serving as a working-area for that job. If a processor is doing data manipulation with one stack, it will keep the stack number (assigned by software) in its registers. If control needs to be relinquished (e.g., the program requests an I/O operation), the processor will execute a "movestack" instruction to the job waiting in the "ready queue" with the highest priority. For example, take jobs J1(40), J2(50), J3(45) — all ready to run. A processor is executing in the stack for J1. J1 requires an I/O operation for which it must wait, and relinquishes control.

The processor will then turn to the stack for J2. When the I/O operation for J1 is complete, J1 is put in the "ready queue" but will not get a processor until job J3 has a turn.



The "ready" queue is based upon priority which demands that higher priority jobs be processed before lower priority jobs in the queue.



Work Flow Language (WFL)

The Work Flow Language is a higher-level language in which jobs are written. It is designed especially to control jobs and the flow of tasks within jobs. There are no macros or "catalogued procedures" to be found in WFL. Rather, one writes subroutines and uses them in the job deck. WFL statements are English-like; they can be preceded by declarations of common card files and subroutines. The job deck is compiled by the WFL compiler to create a code file on disk (the "job file") together with embedded card data.

OVERALL STRUCTURE

The first card of a WFL deck has an invalid EBCDIC card-code (conventionally a 1-2-3 multi-punch) in column one. The remainder of the first and all following cards is free-format, with statements separated by semi-colons. (An invalid punch in column one acts as a semi-colon.) An invalid punch is flagged by the hardware as a question mark, thereby providing easy recognition of control cards.

Let's look at an example: run two programs PI and P2; P1 has card input INP, P2 has card input CRDS (the invalid character is represented by a "?"):

```
? JOB FRED;
BEGIN % ANYTHING AFTER A "%" IS A
COMMENT
RUN P1;
DATA INP
  {data deck}
? % END OF FILE FOR INP
RUN P2; DATA CRDS
  {data deck}
? END JOB
```

Global card decks are accessible to all tasks within the job (e.g., programs P3 and P4 both read a card deck called CARD):

```
? JOB FRED;
BEGIN
DATA CARD
  {global card deck}
? RUN P3;
RUN P4;
? END JOB
```

Task identifiers are used to control job flow. In the following example, the task identifier is T:

```
? JOB FRED TREE/STRUCTURED/TITLE;
BEGIN
RUN DOIT  [T];
IF T IS ABORTED THEN RUN CLEANIT;
? END JOB
? JOB FRED;
BEGIN
RUN DOIT   [T];
IF T IS ABORTED THEN
BEGIN
  WAIT ("CAN I RUN CLEANIT?," OK);
  RUN CLEANIT;
END;
? END JOB
```

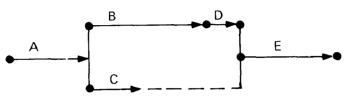
The task identifier is established by the execution (or attempted execution) of the task "DOIT."

Note that the wait statement contains a message for the system operator. The job will be suspended until the operator enters an "OK" message. Alternatively the operator may enter a "DS" (discontinue processing) in which case the task and job will be aborted.

WFL provides for error recovery from program faults (e.g., divide-by-zero, irrecoverable I/O error):

```
? JOB FRED;
BEGIN
ON FAULT,
  BEGIN
    DISPLAY "FAULT OCCURRED.";
    RUN CLEANIT;
    GO XIT;
  END;
RUN DOIT;
XIT;
? END JOB
```

One of the most significant features of WFL is the ability to easily handle asynchronous processing. Asynchronous processing can be depicted as follows:

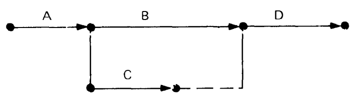


Upon completion of task A, asynchronous tasks B and C are spawned. Here is how the WFL would be written to handle such a sequence:

```
? JOB FRED;
BEGIN
  RUN A;
  PROCESS C  [TSK];
  RUN B;
  RUN D;
  WAIT (TSK);
  RUN E;
? END JOB
```

Notice that we have used a new verb, PROCESS. PROCESS and RUN are equivalent in that both cause execution to occur. The RUN verb, however, constrains the system to wait until the completion of the task; the PROCESS verb does not.

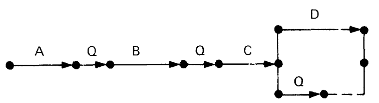
WFL also provides for error recovery from halt/load restarts.



```
? JOB FRED;
BEGIN;
ON RESTART, GO TO L;
RUN A;
PROCESS C  [ASYNCT];
RUN B;
WAIT (ASYNCT);
L;
  IF FILE X/Y IS PRESENT THEN RUN D;
? END JOB
```

In this case the presence of the file X/Y implies that the job was executing task D when the halt/load occurred.

As was mentioned earlier, WFL provides a subroutine capability. Let's examine how this might be used.



In this job stream task Q is to be run three separate times. Here is the WFL for this stream:

```
? JOB FRED;
BEGIN
SUBROUTINE Q;
  BEGIN
    IF FILE X/Y ISNT PRESENT THEN
ABORT;
    RUN CHECK/PROF;
  END;
RUN A;
Q;
RUN B;
Q;
RUN C;
PROCESS Q;
RUN D;
? END JOB
```

Note that one calls a subroutine merely by mentioning its name.

One-task jobs have a simpler syntax. Instead of:

```
? JOB FRED;
BEGIN
RUN PROG/FIVE;
? END JOB
```

it could be:

```
? RUN PROG/FIVE;
? END
```

or even:

```
? RUN PROG/FIVE; END
```

Furthermore, if commands are entered at the operator's console, one can drop the "?," the ";" and the "END."

```
RUN PROG/FIVE
```

PROGRAM EXECUTION

There are many optional control cards for specifying task attributes, for example:

VALUE —	The program can pick this up;
OPTION —	Various bits, e.g., for program dumps;
PRIORITY —	May be set from 0-99 unless incompatible with job queue;
PROCESSTIME —	Establishes a limit for CPU time, e.g., abort if program gets into interminable loop;
IOTIME —	Establishes a limit for I/O channel activity.

Example:

```
RUN THISPROGRAM;
VALUE = 52;
PRINTLIMIT = 4000;
OPTION = FAULT, DSED;
PROCESSTIME = 60;
```

In this case the value "52" is passed to the program; the lines printed will be limited to 4000; on a fault of any type, the job will be discontinued; and processor time will be limited to 60 seconds.

Control cards for specifying file attributes are called "file cards." For example, programs P1 and P2 are to read the same global card deck called CARD, but P2 expects the card file to be called DRAC:

```
? JOB FRED;
BEGIN
DATA CARD
  {global card deck}
?
RUN P1;
RUN P2; FILE DRAC (TITLE = CARD);
? END JOB
```

Note that multiple attributes can be expressed in a file statement. Other file attributes available include:

```
FILE F (TITLE = JIM);
FILE F = JIM;
FILE F (TITLE = (USERB) JIM);
FILE F (KIND = TAPE, TITLE = JIM);
FILE F (KIND = TAPE, BLOCKSIZE = 300);
FILE F (KIND = PACK,
            PACKNAME = MYPACK);
FILE F (KIND = PACK, CYLINDERMODE);
FILE F (CYLINDERMODE, SINGLEPACK);
FILE F (PROTECTED, KIND = PETAPE);
FILE F (KIND = TAPE, UNITNO = 186);
FILE F (KIND = TAPE, SERIALNO = 2001);
FILE PRNT (KIND = BACKUP TAPE,
FORMMESSAGE = "2-PART");
```

Here is one more example:

```
? RUN PROG/22;
VALUE = 1;
FILE F = BLOGGS;
? END
```

File Storage Control

Files normally reside on head-per-track disk, disk packs or magnetic tape. At any one time a limited number of disk packs or tapes will be mounted and their contents will be available to the system.

File control therefore consists of:

— Allocating files to disk packs
— Allocating files to tapes
— Allocating files to head-per-track disk units
— A means of copying files from one media to another (library maintenance)

To implement file control, the "family" concept is used. A "family" is an "aggregation of mass storage." A family may be assigned to a job queue so that only that family need be resident for jobs from that queue. When the queue structure is changed, the packs (for example) can be changed.

VOLUME FAMILIES

With disk packs the family consists of the base pack and continuation packs, bearing the same name; for head-per-track disk, the family is all storage control units of the same label ("disk"); for tape, the family is all physical reels in one multi-reel file. As an example let us consider disk packs. Each disk pack is assigned a name:

serial no. 1; FRED; base pack
serial no. 2; FRED; continuation pack; base pack = 1
serial no. 3; FRED; continuation pack; base pack = 1
serial no. 4; JIM; base pack
serial no. 5; PACK; base pack
serial no. 6; PACK; continuation pack; base pack = 5

In this example, there are three families:

FRED (serial numbers 1, 2, 3)
JIM (serial number 4)
PACK (serial numbers 5, 6)

Disk packs 5 and 6 are called "system resource packs" with a "packname" of PACK.

Thus, files can reside on a disk family (e.g., FAMILY = DISK), a disk-pack family (e.g., FAMILY = PACK, or FAMILY = FRED), or a tape family (e.g., FAMILY = TPE217, where TPE217 is the name in the tape label).

New disk packs can be mounted on the disk-pack drives and labelled to put them in the required disk-pack family by means by a software input message. The label is physically written on the first sectors of the disk pack.

MAGNETIC TAPE FILES

In the following diagrams "*" represents a tape mark. When the MCP encounters a tape mark on reading the tape it forces end-of-file action in itself or the application program.

The format of a single file, single-reel tape file named FRED would appear as follows:

```
VOL1 . . . (zero for ID)
HDR1 . . . FRED
HDR2
*
data
*
EOF1
EOF2
*
*
```

When loading tape, the MCP positions the tape after the first tape mark, after verifying the labels. Note: Burroughs uses standard USASI labels.

The format of a multifile, single-reel tape file containing files called FRED/ONE and FRED/TWO would be:

```
VOL1 . . . FRED
HDR1 . . . ONE
HDR2
*
data
*
EOF1 . . . ONE
EOF2
*
HDR1 . . . TWO
HDR2
*
data
*
EOF1 . . . TWO
EOF2
*
*
```

The MCP reads the label and positions the tape after the first tape mark. If the user opens a tape file called FRED/ONE, the MCP will search the peripheral tables to find a tape unit containing a volume called FRED. That unit is then assigned to the program for the duration of the program. If the user opens a tape file called FRED/TWO, the MCP will find the volume FRED, search up the tape for file FRED/TWO and position the tape after the 4th tape mark (in this case) ready for the program to read the first data record.

The format of a single-file, multi-reel tape file called JIM would be:

```
VOL1                    VOL1
HDR1 . . . JIM, reel = 1 HDR1 . . . JIM, reel = 2
HDR2                    HDR2
*                       *
data                    data
*                       *
EOV1                    EOF1
*                       EOF2
*                       *
                        *
```

Both reels are members of the family called JIM. If a program opens a tape file called JIM specifying reel 2, ("family index" = 2), only the second reel need be mounted.

New tapes may be given a serial number and then become a "scratch" tape. The VOL1 tape label has a pattern recognized by the MCP. By purging a "labelled" tape, the tape also becomes a scratch tape.

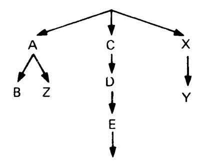
DISK FILE NAMING CONVENTIONS

Files can reside in different families and have the same name, e.g.,

```
FRED ON DISK
FRED ON PACK
FRED ON MYPACK
```

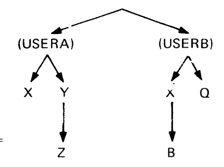
The first file is located on head-per-track disk; the second on a system resource pack; the third on a disk-pack family named MYPACK. Duplicate file names within a family are not allowed. The file name and family uniquely identify a file; more than one file of the same name, even if different generations, is not allowed. Names do not have to be a single identifier; up to 14 identifiers can be used, each containing up to 17 characters. This gives a "tree structure."

A/B
A/Z
C/D/E
X/Y



In this case, all the files in the "directory" A (viz., A/B and A/Z) can be referred to as A/=. A private directory of files can belong to a user (defined as a usercode). The usercode is given in parentheses. For example, if a user USERA owns files X and Y/Z, and a user USERB owns files X/B and Q, the set of files would be:

(USERA) X
(USERA) Y/Z
(USERB) X/B
(USERB) Q
OR (USERA) = AND(USERB) =



Library Maintenance

For backup purposes, files or groups (directories) of files on disk can be copied to tape, creating a "library tape." The library tape is a standard multi-file tape (possibly also multi-reel) with the first file being a tape directory containing the names of the files on the library tape. Actual tape file names are standardized by the MCP. Library maintenance is not automatic but is initiated whenever convenient for backup purposes.

Example:

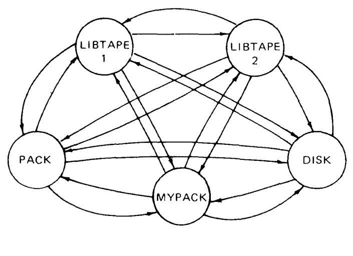
```
COPY A/ = FROM DISK TO TPE217
```

The Library tape format is:

```
VOL1 . . . TPE217
HDR1 . . . FILE000
HDR2
*
tape directory with list A/B, A/Z
*
EOF1 . . . FILE000
EOF2
*
HDR1 . . . FILE001
HDR2
*
data of file A/B
*
EOF1 . . . FILE001
EOF2
*
HDR1 . . . FILE002
HDR
*
data of file A/Z
*
EOF1 . . . FILE002
EOF2
*
*
```

The "COPY A/Z FROM TPE217 TO DISK" will cause the MCP to search the tape directory, find TPE217/FILE002 and then copy it to disk into a file called A/Z.

Note: The library maintenance function will copy files to and from any library tape family, disk-pack family or head-per-track disk family:



The volume library is a compile-time MCP option to create a catalog and volume library. If used, disk, disk pack or tape volumes will be entered in a library (disk file maintained by MCP), even if the tape, etc., is not mounted. One can add all scratch tapes to a family called "scratch"; then if a cataloged scratch tape is mounted and used as an output tape, that tape is deleted from the scratch volume and entered in a new volume family. A utility (LISTVOLUMELIB) will print this volume library; a subset of the volume library is a tape library.

If the file cataloging option is set, files can be resident in the catalog but not physically resident in the system (not mounted). For instance, one can COPY & BACKUP files from head-per-track disk to a library tape (or disk pack) then remove the files on head-per-track disk. If a program subsequently tries to open that file, a "NO FILE" message will be displayed with the serial number of the library tape (or disk pack) on which the file can be found. Similarly, if the file first exists on a library tape (e.g., from another installation) one may COPY & CATALOG the file from the library tape to head-per-track disk (or disk pack).

The normal mode of operation on a B 6800 system is to spool print files to temporary backup storage. This procedure releases memory and other system resources, giving greater utilization of the total system. The MCP automatically prints all spooled files as the printer becomes available. Upon printing, files are purged. Spooled files are printed in sequence either by mix-number or smallest first, selectable by an operator option.

Access Control

Control of access is an MCP function. A user provides a "usercode" and optionally a "password." Usercodes are created by a utility called SYSTEM/MAKEUSER which itself is run under a "privileged usercode."

MAKEUSER builds a mini-data base called USERDATA file. To start, one may run MAKEUSER without a privileged usercode if the USERDATA file is absent. Having made a privileged usercode one can then run MAKEUSER and disable the MU (MAKEUSER) and PU (Privileged User) messages.

Each job deck may have a usercode associated with it. This can be optionally enforceable. The user must supply the password if there is one, and may change the password to prohibit access by others using his usercode.

Disk file security can be applied to all files, whether in a usercode directory or not, but is most useful in a usercode environment. The owner of the file (or a privileged user) may make the file read-only, write-only, read-write-allowed, or secured. Additionally, the owner can make the file "private" (access only by a job running under the owner's usercode), "CLASSA" (access by any other usercode, subject to read/write restrictions) or "CLASSB" (guarded by another, private file called the "guardfile"). For CLASSB security, a separate guardfile lists valid usercodes and the mode in which the access is allowed.

Examples:

a. Stop access to the ESPOL compiler:
SECURITY SYSTEM/ESPOL CLASSA SECURED

b. User USERA wishes the file F to be read by USERB, read and written by USERC, but access denied to all others:

SECURITY (USERA) F CLASSB G
Where G is a guardfile stating
DEFAULT = NO
USERCODE USERB = RO, USERC = RW

If a job supplies an invalid usercode or password, the WFL compiler will not compile the job deck — it will give a syntax error and hence the job cannot run.

Note: The use of usercodes is entirely optional, an installation management decision.

System Initialization

There are three distinct types of initialization available on the B 6800. They are the Cold Start, the Cool Start and the Halt/Load. Each has a special function.

The Cold Start is used when the system is first installed. It assumes that nothing is on the system. The Cold Start requires:

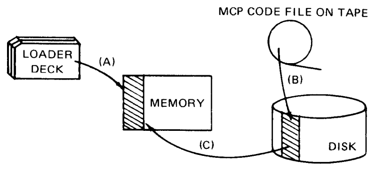
MCP object file on a library tape,
System loader object deck.

By setting the "card load select" button on the console and then pressing the "load" button, the hardware will execute a hardwired series of bootstrap instructions to read a card into memory and then transfer control to the machine coded instructions. This loaded bootstrap program reads in an object deck (about 50 cards produced by the ESPOL compiler) and then transfers control to this program. This program, known as the utility loader (UTILLOADER), reads a parameter card containing a tape name and a file name. UTILLOADER then searches the system to find which units are tape units and reads the tape labels. On finding a matching file name, it then determines if it is a library tape and if so, searches the directory on the tape. Finding the designated file, it loads the program LOADER into memory and transfers control to the program. (UTILLOADER also can load other ESPOL programs used for testing and maintenance.)

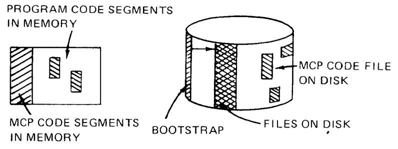
The LOADER program can perform many functions, the most important of which is loading the MCP from disk, tape, or disk pack. Its functions are specified by parameter cards.

After loading the MCP, the LOADER program initializes a disk directory and loads the primary initialized portion of the MCP into memory. Then it transfers control to this new program, i.e., the MCP. The MCP then begins its own initialization.

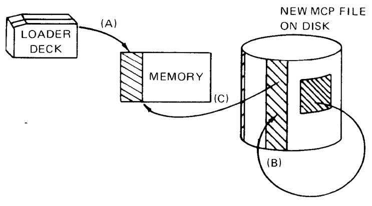
A Cold Start forces loss of any files on the disk family where the MCP is loaded.



The Cool Start operation on the B 6800 reloads the MCP code file on disk but leaves the remaining files intact. Bootstrap information, including the address of the MCP code file, is contained in the lowest address disk segment.



If the MCP code file becomes corrupted or overwritten, the same loader deck can reload the MCP from tape to disk, and if necessary, move the location of the code file on disk. A Cool Start can load a new MCP code file from disk by specifying the name of the file without disturbing the disk directory.



A Cool Start from disk can be done through the MCP by a CM (change MCP) message. In both cases, the MCP performs the initialization routine. No disk files are lost during this procedure; only MCP reinitialization of memory tables occurs. It is often used just to change MCPs.

The Halt/Load is the most common initialization, accomplished by:

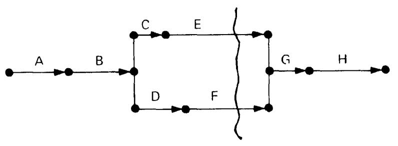
a. Pressing the HALT button on the console.
b. Pressing the LOAD button, with the "card load select" button off.

The hardware reads the bootstrap segment at the lowest disk address, fetches the current MCP code file and the MCP reinitializes its tables. This process takes only seconds to perform. All of Main Memory is tested in the process, and any modules failing the test are deleted by the MCP from the configuration.

Recovery Aspects

HALT/LOAD RECOVERY

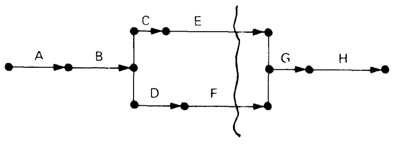
All tasks are aborted during a Halt/Load condition and memory is reinitialized. Jobs and job queues reside on disk and therefore are still present. By default the job is restarted at the last point where no task was running. Let us examine a job/task structure which is interrupted by a Halt/Load.



If a Halt/Load occurred when tasks E and F were running, the job would be restarted at the point when B had finished. A single-task job would be restarted.

A task can determine if it has been restarted and take appropriate action. WFL features allow this automatic restart to be overridden, e.g., to "start again from beginning" or "go to the end," or "reload files."

An MCP run-time option, AUTORECOVERY, when reset will set the MIXLIMIT of all queues to 0, and inhibit automatic printing of spooled files and restart of data communications.



Normally a task will be restarted from the beginning. Checkpoints may be taken, however, with checkpoint files residing on disk or disk pack. If a checkpoint file exists for a task, the task may be restarted from that checkpoint. Checkpoint files can either accumulate (many files) or can overwrite (purge) the previous file.

An output file (tape or disk) created by a task is normally lost if a Halt/Load occurs during the task execution. The program can make such files "protected." For an output tape file, a tape-mark is written before rewinding after the Halt/Load, so end-of-file can be found.

COLD START RECOVERY

The major implication of a Cold Start is the loss of the disk file directory. Prior to a Cold Start, the following steps should be performed:

- Copy off files to a library tape from the Halt/Load unit;
- Cold Start or remove files;
- Copy back files from the library tape.

Where fixed locations for files are desired, the installation can use Installation Allocated Disk (IAD). IAD allows users to reserve particular physical locations of disk. The MCP will not allocate any space in these areas.

Standard system software can be used to construct a file "header" giving a pointer to the IAD area. All disk files have "headers" which are normally constructed by the MCP. This IAD header becomes part of the normal file directory so that the file can be read, written, or copied to another storage medium.

IAD areas can be allocated/deallocated at run time by RESERVE/RETURN software function, specifying the physical location. For RESERVE, all files in the required area are copied to available disk elsewhere.

I/O RECOVERY

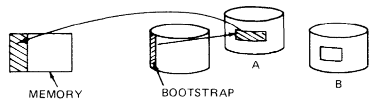
The MCP normally handles all I/O errors asynchronously. It will initiate a separate stack (process) to handle retries, thus freeing itself to continue handling normal events. Tape parity retry involves backing the tape and retrying a total of eighteen times; head-per-track disk retries occur ten times; disk pack is retried by the controller three times — if all three are unsuccessful, the MCP will re-issue the I/O operation up to ten times to the controller. In all cases, the program is informed only if all retries are unsuccessful.

When a program does not provide for I/O errors and the condition arises, the task will be aborted with an error indicating the I/O statement which failed. A program can recover from errors by using, for example, a "declarative section" (in COBOL).

Bad segments of disk can be removed from the MCP available disk table by the XD message. The XD message creates a file called BADDISK of the bad segments, preventing further use until an engineer can investigate the problem.

The MCP will maintain duplicate files if requested by the user (e.g., if file FRED is to be duplicated, the MCP will keep the files FRED/01 and FRED/02). The program deals strictly with a file named FRED. If an I/O error occurs on one copy, the MCP will access the other copy.

Critical system files can be duplicated, e.g., the MCP code file, the disk file directory (a list of headers) and the disk file access structure. As an example, we may store the MCP code file duplicates on separate families, say disk packs A and B.



During normal operation the system uses the MCP code file on A. Code moves from A into memory and the bootstrap for a Halt/Load points to A. If disk pack A fails, however, the bootstrap is altered to Halt/Load to disk pack B; the system automatically uses the MCP code file on B. Using the same principle, one can duplicate the directory and the access structure on the same families.

DATA MANAGEMENT

Overview

A typical program may be required to do numerous READs, WRITEs, SEEKs, OPENs, CLOSEs, STARTs, REWRITEs, and DELETEs on a file of information. The program, in fact, performs none of these physical I/O operations. Rather, the program makes requests of the MCP to perform each of these functions. The MCP will assign the "logical file" as seen in the program, to the "physical file" existing on the head-per-track disk, disk pack or tape.

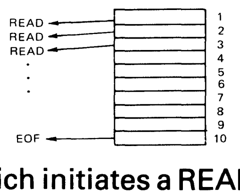
The MCP will automatically allocate space on random access devices, and support buffering and blocking. It provides full support for files which are organized sequentially (as in a tape environment); for files which are organized in a random manner (records are accessed by their relative position within the file); and for files organized with an index (records are accessed by the value of a key within the record).

The system also supports a comprehensive Data Management System (DMS II), which provides multiple access methods and data relationships.

MCP Logical I/O Facilities

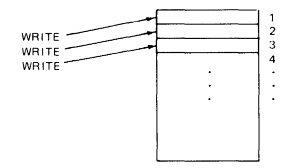
READs, WRITEs, and SEEKs can be performed with or without an actual key or record key.

When a READ is performed without an actual key or record key, the first READ will position the file at the first record. Subsequent READs get the next physical records. When the end-of-file is encountered the MCP will notify the program and the AT END branch is taken.

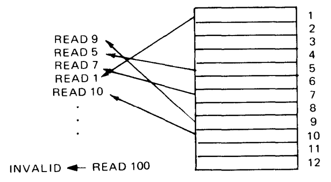


A program which initiates a READ is suspended by the MCP until the logical I/O (transfer of one logical record from the buffer to the user program record area) is complete.

Similarly, a WRITE without an actual key or a record key will write the file serially (sequentially).



READs performed with an actual key or a record key will retrieve the record specified by the actual key or the record key. When the key specifies a record which does not exist in the file, an INVALID KEY condition results.



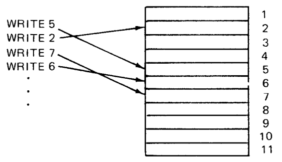
A READ will wait for the completion of physical I/O (i.e., an implied SEEK) when required. Since multiple buffers are permitted, the MCP will attempt to make the record available from a buffer. Suppose a file has five buffers and records 1, 30, 90, 50, 15 are accessed by the first five READs. If the sixth READ accesses any one of these five records then no physical I/O takes place.

A SEEK or START will fetch the logical record but allow processing to continue. A subsequent READ will actually get the record into the program-area (this technique is used to speed up the program by overlapping I/O and processing):

WITHOUT SEEKS	WITH SEEKS
READ (Record 1)	READ (Record 1)
WAIT	SEEK
PROCESS (Record 1)	PROCESS (Record 1)
READ (Record 2)	READ (Record 2)
WAIT	SEEK (Record 3)
PROCESS (Record 2)	PROCESS (Record 2)
READ (Record 3)	READ (Record 3)
WAIT	SEEK (Record 4)
PROCESS (Record 3)	PROCESS (Record 3)
READ (Record 4)	READ (Record 4)
WAIT	SEEK (Record 5)
PROCESS (Record 4)	PROCESS (Record 4)
READ (Record 5)	READ (Record 5)
WAIT	SEEK (Record 6)
PROCESS (Record 5)	PROCESS (Record 5)

Notice that the SEEKs permit far smoother execution. Without them the program must be suspended during each READ; with them, processing continues.

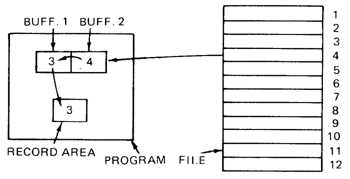
A WRITE with actual key or record key will write to the required record.



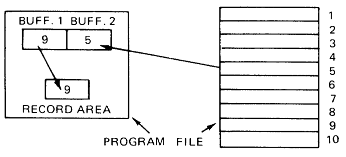
Note that the file would contain seven records but records 1, 3 and 4 would not be valid. A subsequent serial read would get end-of-file action trying to read record 8 but would have read all seven records.

BUFFERING

Buffering is a technique used to increase throughput; the default is 2 buffers (1 alternate area).

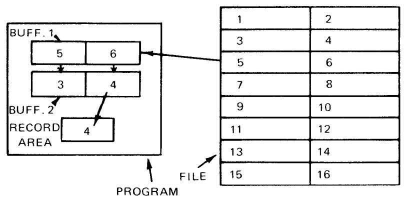


By using buffering, three serial READs would provide records 3 and 4 in the buffers. Buffering results in "physical I/O" (file-to-buffer) taking place before "logical I/O" (buffer-to-record area). This is especially useful when doing random SEEKs. A SEEK gets the record into the buffer, and the READ gets the record from the buffer into the record area. For instance SEEK 9, READ 9, SEEK 5 would produce the following results:



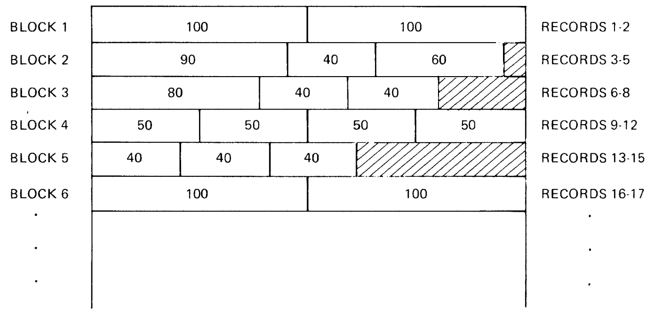
BLOCKING

Blocking is a technique used to decrease physical I/O time; each physical I/O reads a block of logical records. This is especially useful for READing or WRITEing serial (sequential) files.



In this example, each pair of serial READs involves only one physical I/O. The BLOCK in this case contains two records.

Variable-length records are especially useful for tape. For example, records 40 to 100 characters long with a maximum block length of 200 characters would appear on the tape as follows:



There are various ways of specifying length of record:

- By using the first 4 bytes in the record in character format;
- By using the first 2 bytes in the record in binary value;
- By externally specifying via "file attributes" elsewhere in the record.

Evolution of Data Base Approach

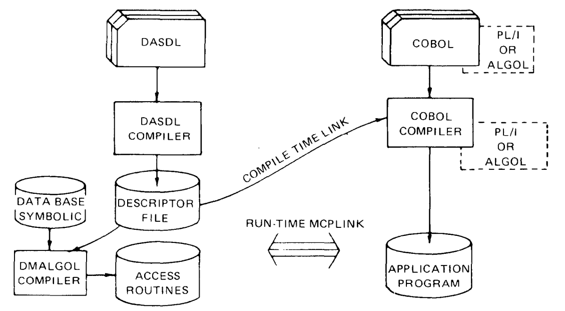
Data Management has been an integral part of data processing from the time of the first commercially available computers. Data Management at first was punched card management, as all files were stored on the 80-column card. As technology progressed, Data Management gradually shifted from punch orientation to magnetic tape and finally to disk. By the late 1960s, Data Management systems were available to handle very large, randomly accessible files on an application-by-application basis.

Today's Data Management Systems manage data bases which are independent of applications. They not only provide a variety of ways to link information but also automatically maintain sets and subsets of the information. The effect has been to make application programming easier.

Data Base Software (DMS II)

DMS II is one of the most sophisticated Data Management Systems available today. It is noted for its high performance and ease of use.

In DMS II, specifications of file structures and data layouts are defined using DASDL (Data And Structure Definition Language). Here is an example of how a simple (single-file) data base would be specified. Manipulation of the data base is accomplished by providing extensions to the "host language" (COBOL, PL/1, and ALGOL).



FILE ORGANIZATION AND ACCESS METHODS

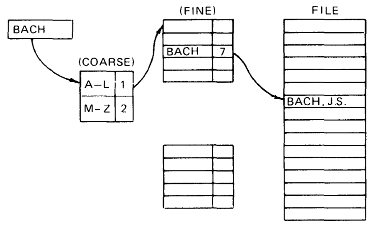
Serial File — In a serial file, data is read from beginning to end. Information (symbolic keys) can be maintained in order (ordered list) or in no particular order (unordered list). For unordered lists, a sequential search is made to locate data. For an ordered list, a binary search or other search method could be used to locate data.

Direct File — In a direct file, the data records contain an actual key or record key. The key states the physical location in the file in which the record is to be placed (the symbolic key is the actual key or record key, i.e., the logical record number in the file).

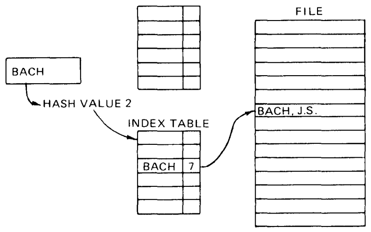
RECORD 1	1-BACH
RECORD 2	2-BEETHOVEN
RECORD 3	3-LIGETI
RECORD 4	4-HANDEL
RECORD 5	5-MAHLER

Random File — In a random file, a symbolic key (e.g., a surname or social security number) is converted via an algorithm to a numerical value which serves as an address for the record in question.

Index Sequential File — In an index sequential file, the symbolic key is searched for in a set of index tables. There is a coarse table to get to the required fine table. Symbolic keys are stored sequentially in the fine tables. Fine table entries point to the actual record.



Index Random File — In an index random file the symbolic key is converted to an index table number. The table is searched to find the location of the record associated with the symbolic keys.

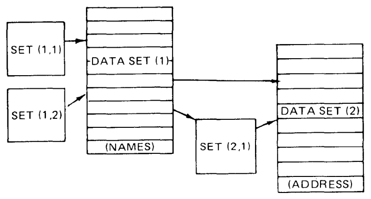


DATA SETS AND SETS

DMS II supports both data sets and sets. A data set is the collection of data that is stored and is to be accessed.

A set is a file containing the access tables. There may be several sets relating to one data set for different access methods; each set is one physical file.

There may be links from data sets to sets or other data sets. Embedded sets (chains) and embedded data sets may also be used.



Data is stored in a data set and in a set, according to allocation principles discussed in the next Section.

HOST LANGUAGE FEATURES

The COBOL, ALGOL and PL/1 languages for the B 6800 have been extended to allow data base access. Some examples of the COBOL syntax are:

FIND FIRST
FIND NEXT
FIND LAST
FIND PRIOR
FIND KEY = "HASTINGS"
CREATE
STORE
DELETE
ASSIGN — to put link to another set
LOCK/FREE — to avoid contention if about to update a record in a multiprogramming or transaction-oriented environment.

(The FIND verb may specify which set, i.e., which access structure, to use.)

AUDIT AND RECOVERY

DMS II provides extensive audit and recovery capabilities. Auditing is achieved by requesting the system to audit the data base and specifically to maintain an audit trail; recovery is achieved through the invocation of declared restart information (called the RESTART DATA SET) and the audit trail. Both facilities are instrumented through the DASDL language; both facilities are entirely optional.

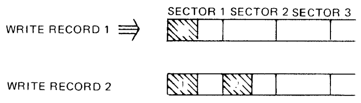
The audit trail consists of a trail of changes to the data base data files and index tables, along with various control records. The audit is written to tape, disk, or disk pack. The audit trail is used as the input to the various forms of data base recovery.

The RESTART DATA SET holds program restart information. During the recovery process, the recovery routines store program restart information (based on the contents of the audit trail) into the RESTART DATA SET. Since the RESTART DATA SET is part of the data base, user programs can conveniently retrieve sufficient information from this data set to reinitialize themselves after completion of the recovery process.

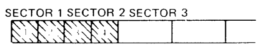
MCP Disk Allocation

On the B 6800 system, head-per-track disk and disk pack are physically divided into 180-byte sectors. The hardware performs READs and WRITEs a segment at a time.

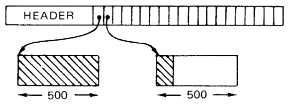
Example: An unblocked file with 90-byte records wastes 50% of storage.



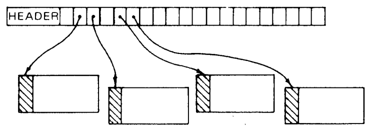
but a blocked file with 90-byte records, 2 records/block uses all storage.



Under program control, the MCP will divide the physical file into "rows" or "areas." For example, a file of 10,000 records may be divided into 20 rows of 500 records each. The "file header" (maintained on disk by the MCP and pointed to by the disk file directory), contains a pointer to each allocated row of the file.



If 600 records are written serially, one row will be completely filled and the second row will contain 100 out of a possible 500 records as follows:



Note: Space has been reserved on disk or disk pack for only 1,000 of the requested 10,000 records. The file may be expanded at a later time to the full 10,000 records. If required, the MCP will automatically provide space for additional records beyond the 10,000.

Space is allocated for a row of a file only when access is attempted for a record within that row. Until that time, the space remains available for use.

Each row of a file is a separate area of storage. By default the MCP will spread these rows among all members of the family. For example, in the case of a disk-pack file with a Base Pack and two Continuation Packs, rows will be assigned as follows:

Records 1-500, row 1, family index 1 (BP)
Records 501-1,000, row 2, family index 2 (CP1)
Records 1,001-1,500, row 3, family index 3 (CP2)
Records 1,501-2,000, row 4, family index 1 (BP),
Etc.

To access record 1,200 (row 3) on Continuation Pack 2, one needs to mount BP and CP2 only, because the file header is stored on the BP. This can be overridden by the "singlepack" file attribute, which causes all areas to be allocated on one disk pack.

Within an individual storage unit (single pack, or one subsystem of head-per-track disk), areas are allocated in the smallest available space large enough to contain the request. Available space tables are maintained by the MCP for each disk family.

The actual locations of files are not known or required by the user.

Disk-pack files can be allocated in "cylinder mode." In this case each file row is contained within a cylinder boundary. If row size $\leq$ cylinder size, then the whole row can be read without any arm movement.

DATA COMMUNICATIONS

Survey of Data Communications Uses

There are a variety of reasons why data communications may be required on computer systems today. These include:

Data Collection — For gathering information which was formerly key-punched. This information can be batched or input intermittently. Preliminary editing is often done at terminals.

Message Switching — The text of a message is routed through the central system from one terminal to another. Note that the central system must resolve contention between input from and output to the same terminal (e.g., if terminal is sending information, then the output message must be queued).

Transaction Processing — A transaction from a terminal is received, the text of the message is analyzed, the data base is accessed and/or updated, a response message is composed, and the response is sent to the originating terminal.

Time Sharing — Originally used with teletype-like terminals but most recently with increasingly intelligent terminals. In the time sharing environments, source programs are created using a text-editing facility and compiled using interpretive or interactive compilers.

Remove Job Entry (RJE) — Jobs entered on a smaller remote computer are executed on a larger host computer.

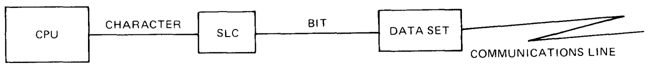
Response times are important and vary with the application. Very fast response with file updating is called "real time;" changes are made quickly enough to affect the environment.

Data Communications Hardware

CONVENTIONAL APPROACHES

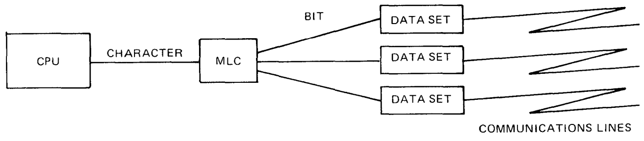
Depending upon the sophistication of the system, there are three conventional approaches to data communications hardware implementation.

1. Single-Line Controls (SLC) which connect to the mainframe. They take bits from one line (telephone, TELEX®, etc.) and pass on a character to the mainframe (CPU or I/O function).



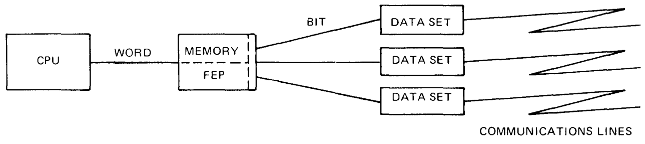
The CPU must take time to process each character in this arrangement.

2. Multi-Line Controls (MLC) which connect to the mainframe. The MLC must have logic to identify the characters from different lines, sort out start/stop bits, generate parity bits, etc. The CPU sorts out the characters into words.



There are usually a limited number of lines into a MLC (normally 64).

3. Front-End Processors (FEP) which assemble characters into words, are programmable (usually in assembler), and have their own memory.



B 6800 Data Communications Hardware

In the flow of information from a terminal to the central system, the Data Comm Subsystem components handle progressively larger units of information. When information is sent from a terminal to the central system, the terminal transmits discrete bits representing a character.

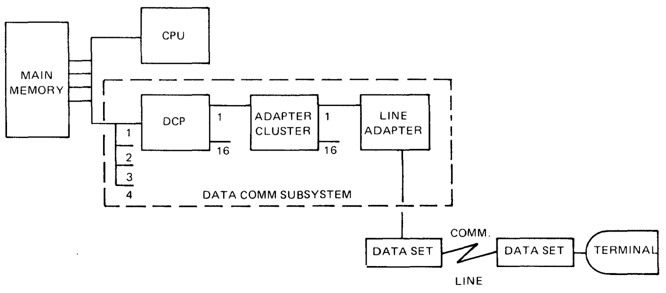
For phone-line transmission, the bits are fed into a data set (modem), transformed into a modulated bit-serial stream, and then transmitted over a telephone line. The modulated bit stream is then received by another data set where it is re-transformed into the original discrete bit pattern and connected to the Adapter Cluster through an appropriate Line Adapter. For direct-connect stations, the line is connected to the Line Adapter.

The Line Adapter matches the electrical characteristics of the line or data set to those of the Adapter Cluster.

The Adapter Cluster accumulates a single character, notifies the Data Communications Processor (DCP) that the character is available, and transmits the character to the DCP upon request.

The DCP examines the control characters associated with each transmission to relate the transmission text to its proper message area in main memory. It accumulates the characters of the text into words and dispatches each word into the appropriate Main Memory area. The DCP may also check parity and perform character translation, for example, from ASCII to EBCDIC codes. The reverse is done for outgoing messages.

The DCP can also interface directly with the CPU for control instructions. The DCP READs and WRITEs information to the Main Memory via the Memory Control.

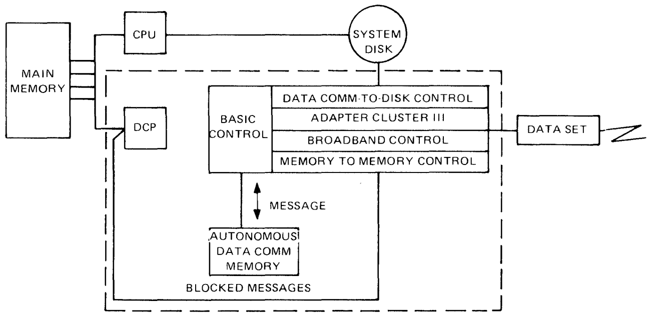


Each Single-Processor B 6800 can handle up to 4 DCPs. Each DCP can accommodate up to 16 Adapter Clusters, to each of which may be attached a maximum of 16 Line Adapters. Thus, a Single-Processor B 6800 could have as many as 1,024 lines attached, with multiple terminals on each line.

B 6800 AUTONOMOUS DATA COMMUNICATIONS HARDWARE

The B 6800 can be configured to allow data communications to operate with varying degrees of autonomy from the main frame. This enhanced configuration includes Front-End Controls, a large dedicated (393K-byte) Data Comm Memory, and direct access to a 23 ms head-per-track disk subsystem shared with the main system.

The autonomous feature accepts a bit stream from the line adapter, performs message-level accumulation (multiple words as contrasted with single word), translation, editing, and transmits the complete message block to Main Memory and system disk when appropriate.



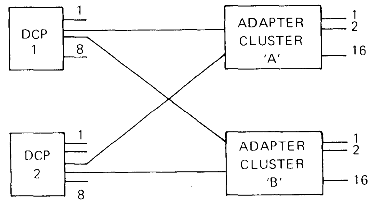
Autonomous data comm supports a capability to:

a. Inform currently active stations of a "down" main system;
b. Inform new dial-up stations of a "down" main system;
c. Suspend data comm in an orderly fashion during a "down" main system situation;
d. Preserve the integrity of messages in progress at the time of system failure;
e. Resume normal data comm operations at a subsequent Halt/Load;
f. Time-stamp incoming messages by the DCP;
g. Handle message tanking;
h. Audit incoming messages;
i. Continue output of previously tanked messages on disk throughout the duration of the failure;
j. Continue to accept input from stations which, in normal operation, are designated as being audited and/or in data collection (file) mode and which have no requirement for interaction with the main system.

DATA COMMUNICATIONS

The Adapter Clusters and Basic Controls may be cross-coupled to two DCPs. Each DCP is directed to service only a designated subset of the clusters connected to it. In the event one DCP should fail, the other DCP can be programmatically directed to service up to 16 clusters. This allows one DCP to back-up the other in the event of a malfunction.

A cluster mask in each DCP, configured by DCP software, regulates control of up to 16 adapter clusters.

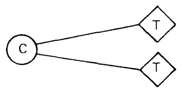


NETWORK CONFIGURATIONS

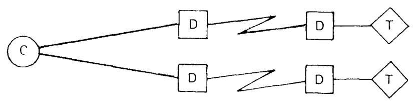
The following diagrams represent a sampling of the possible network configurations. In the diagrams:

C = Central Computer
T = Terminal
D = Data Set

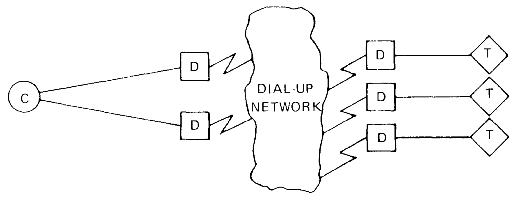
Direct connect lines. No data sets.



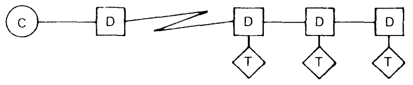
Leased Line. Using data sets but not using the dial-up line-switching system.



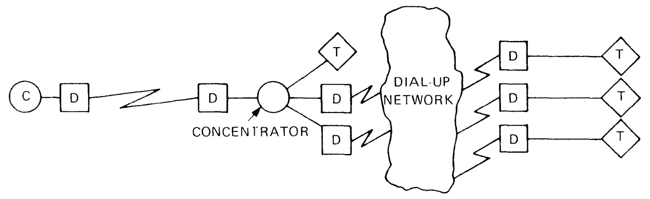
Switched network. Using data sets and the dial-up telephone systems.



Multi-drop lines. Connecting multiple terminals to one physical line. Each terminal has a unique address.



Line concentrator.



The line from the central computer to the concentrator could be leased or switched. Lines from the concentrator to terminals could be direct-connect or switched lines (e.g., local telephone only).

When multiple terminals are dropped on a line, the system asks if the terminal has a message in its buffer (terminal is POLLED). The terminal responds with an ACK or NAK, or sends a message. The system asks if the terminal is in a state to receive a message (terminal is SELECTED). The terminal responds with an ACK or NAK. These or similar conversations are called "line disciplines."

PROTOCOLS

The B 6800 data communications system supports every major protocol including:

Asynch;
Synch;
Bi-Synch;
BDLC/SDLC.

Data Comm Software Overview

Data communications software for the B 6800 consists of three functionally and physically separate entities:

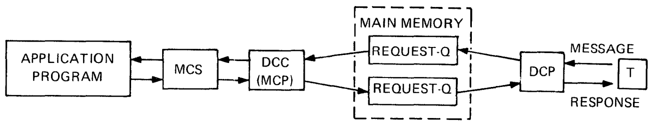
— NDL (Network Definition Language) to program the DCP for physical line handling.

— MCS (Message Control System) to handle internal distribution of messages.

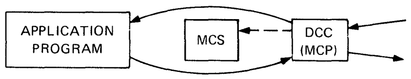
— The application program to take action based on the actual message text.

MESSAGE FLOW

Messages are handled in characters by the DCP. The DCP assembles characters into words and adds a "message header" for the terminal originating the message. The DCP queues messages into main memory. The DCC (Data Communications Controller, a part of the MCP) examines the messages in the queue. It inserts a pointer to the message in the appropriate queue for the MCS servicing that terminal. The MCS analyzes the header and may decide to pass the message on to an application program. Here is a diagram of what goes on:



There are times when it may not be necessary to have the MCS handle the message. In these cases the MCS is bypassed. Such MCSs are termed "non-participating."



Network Definition Language (NDL)

Network Definition Language for the B 6800 is a descriptive language used by data communications programmers to specify the characteristics of a network. When compiled the resultant program —

- provides DCP code for the various line disciplines required for different types of terminals (contention devices, polled devices, slow poll, fast poll, etc.);
- describes the physical network: the lines into the computer; how they are connected; what the line characteristics are; (direct connect, data set connect, speed, auto dial-out, etc.); what stations are connected to each line.

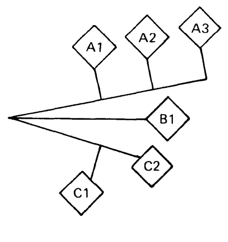
Note the difference between a logical "station" and a physical "terminal." Application programs and MCSs tend to refer to stations while DCP/NDL talks to terminals. Each station declared in NDL has a physical address associated with it. It also has a description of its physical connection, an assigned address, and a logical address (Logical Station Number). These are all assigned by the NDL compiler.

NDL programs are organized into sections —

- the CONTROL section allocates the use of a logical line to the stations assigned to that line.
- the REQUEST section contains control instructions for each type of terminal (e.g., read-request-for-TD 800, write-request-for-TD 800, poll-request-for-TD 800, select-request-for-TD 800). It tells the DCP how to deal with special characters (e.g., SOH, start-of-header; ETX, end-of-text), how to check parity, and what actions to take.
- the MODEM section defines data set types for each type of line (e.g., synchronous/asynchronous, transmit-delay, etc.).
- the TERMINAL section defines terminal types (e.g., buffer size, screen/no screen, odd/even parity, ICTDELAY (inter-character-transmit delay). It also defines what data set is associated with the terminals.
- the STATION section defines each of the "ports" into the data comm system (e.g., station name, terminal type, input/output enabled, name of controlling MCS, control character) for messages from this station to its MCS.
- the LINE section defines each line, physical location (DCP/cluster/adapter) and lists each station on the line.
- the DCP section lists the DCPs and the amount of local memory for each.
- the FILE section associates station(s) with file names for use by application programs.

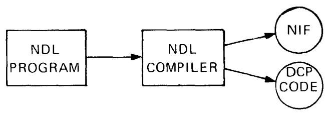
  Example:

  FILE F1 = A1, A2, A3, B1, C1, C2.
  FILE F2 = A3, B1, C2.
  FILE F3 = C1.



(Stations A3, B1, and C2 constitute the "family" of file F2.)

The NDL compiler produces the DCP code file and a Network Information File (NIF).



Several sets of NIF/DCPCODE can exist in the disk file directory under different prefixes (e.g., A/NIF, A/DCPCODE, B/NIF, B/DCPCODE) but only one set can be operational at a time. These can be exchanged without affecting main memory or any non-data comm programs.

The Message Control System (MCS)

MCSs are designed to provide many aspects of control. Among these are the ability to:

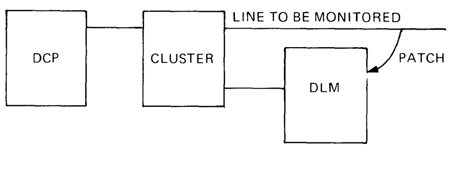
- Control the stations attached to it (as declared in NDL);
- Make a station ready/not ready to communicate;
- Accept input from a station;
- Assign or deny a station access to a file;
- Switch messages;
- Reconfigure the network in event of failures.

Burroughs supplies several MCSs for the B 6800 to handle many common data comm environments.

CANDE (Command AND Edit) is the time sharing MCS. It includes the ability to create disk files, edit them, compile programs and execute them.

RJE (Remote Job Entry) provides the ability to submit input files (e.g., card decks) to the central system and receive printed output from the central system for remotely originated jobs.

DIAGNOSTICMCS is provided for trouble-shooting data comm lines/models or DCPs. It is used especially in conjunction with a hardware device called a DLM (Data Line Monitor).



GEMCOS (GEneralized Message COntrol System) is a program product designed to rapidly provide users with a Message Control System (MCS) that is tailored to meet the specific requirements of a given installation. The MCS is transaction oriented, and provides users with the flexibility to meet a board range of throughput and processing requirements. GEMCOS anticipates and provides for a dynamic operating environment. Flexibility and efficiency are the key words for this product. A major consideration from the user viewpoint is that the GEMCOS product is also relatively simple to learn, install, and interface.

Highlights of Burroughs Generalized Message Control System

- Transaction Control Language
- Access Control
- Message Routing
- Message Formatting
- Message Paging
- User Program Reentrancy
- Recovery, Including Synchronization with Data Base Rollback
- Network Control
- Administrative Message Switching
- Retransmission of Output, Upon Request

An MCS provides the data communications user with a viable interface between the Data Communications Processor (DCP) and the application programs that are to process transactions associated with remote terminals. The DCP is pictured as the heart of the data communications system in that it controls line disciplines and physical network I/O. The MCS is envisioned as the brain of the data communications system in that it provides the intelligence necessary for objective decision making regarding the disposition of messages once they have been assembled. The MCS and DCPs work together to provide overall control of the data communications system. As a result, a user's application programs can be designed, and in fact implemented, independent of the network/terminal environment in which they will operate. Burroughs B 7800/B 6800 GEMCOS accommodates change in order to protect user investment in application programs.

GEMCOS MAIN FEATURES

Transaction Control Language

The various environments under which the MCS operates dictated that Burroughs develop a simple method which would allow users to describe requirements unique to their specific environment. The MCS must then adapt to meet those requirements. Both the requirements and the environment are expected to be dynamic. The method chosen to allow this description is the Transaction Control Language (TCL).

The user utilizes TCL to provide information such as message routing criteria, and certain MCS options selected. In addition, TCL is used to describe: access-control requirements; message formatting and paging criteria; dynamic load control criteria for user programs; the types of recovery selected for user programs.

The language is of free-form structure, utilizing key words to describe both the environment and the requirements of the data communications user. The result of a TCL compilation is the generation of a set of customized tables. The MCS code which interprets those tables is modular in structure. The GEMCOS design is totally compatible with the virtual memory concept.

As a by-product, the TCL compiler optionally provides users with a hard-copy record of their data communications system description.

Access Control

Access Control is optionally available on a station-by-station basis. If Access Control is declared to be in effect for a given station, the legitimate users of that station can be specified. Additionally, a valid sign-on procedure at the station is required to gain access to the system. Specific limitations can be described as to which transactions are to be allowed for the particular user signing on at the station. Additionally, a mechanism is provided which allows a user-written program to participate in evaluation of sign-ons prior to access being granted.

Routing

Many types of message routing are provided by the MCS. Messages can be routed from station to station(s), station to user program, user program to station(s), user program to broadcast lists, Batch Transaction Processor to user programs, and user program to Batch Transaction Processor. In instances where a message is to be routed to a station and the station is out of service, a chain of alternate stations can be specified in TCL.

Station-to-station(s) messages are considered to be Administrative Messages which flow through the system, are audited and delivered to the specified destination stations.

Routing for station-to-user program transactions is normally determined by examination of the message key (defined in TCL) which is contained within the message. An option also exists for a station to attach itself to a user program. When a station is attached to a user program, message keys are not examined by the MCS.

User programs can address output to specific stations on a station-by-station basis. User programs can also address output to Areas. A label is associated with a list of stations in TCL. This list is considered to be an "Area." The output message will be broadcast to all stations listed for the Area. A rotary list can also be associated with an Output Area. The output message will be delivered to the station within a rotary list that has the shortest queue at request time. The Area feature allows users to make final determination of output distribution after programs have been written. This means that user programs can be written without knowledge of the network with which they will interact. Association of Area labels and lists occurs at TCL generation time.

A Batch Transaction Processor is a special type of user-written program which can read a tape, disk, or card file and route transactions to user programs via the MCS queues. A user program processing a transaction in this mode is then expected to issue a response to the Batch Transaction Processor to indicate completion. This mechanism allows for concurrent batch/on-line updating of the same data fields within a data base and still allows synchronized data base recovery. In addition, this mechanism provides the ability to more closely control the batch-transaction throughput rate.

Formatting

Formatting of messages, independent of user programs, is provided as an option via a special subset of the Transaction Control Language. The Formatting feature of GEMCOS can provide the application programmer with true device independence. A user writing an application is not required to know hardware control codes or buffer capacity for the various terminals on the network. Rather, the application programmer can deal with data strings at the point of MCS interface. A user independently describes in TCL the formatting to be performed by the MCS on those data strings.

The MCS retrieves format descriptions, based upon both the message-identification key and the device class of the station involved and applies the format to the data. This allows tremendous flexibility at the stations, transparent to the application programmer. For example, the application program can tell the MCS to output a given message to two stations on the same network. If those two stations are described as being in different device classes, the resulting output can be drastically different based upon two different format descriptions provided in TCL. Different formats imply that such data field characteristics as length, sequence, form information, and whether certain fields are required or optional, can vary from station to station. Conversely, two different stations can supply input for the same type of transaction under different formats, and the data will arrive at the application program in a standard format. In fact, in cases where each input field from a terminal is delimited by a unique key, the sequence of data entry can vary from transaction to transaction at the same station, even though the same type of transaction is being repetitively performed. Such activity will be transparent to the application program under the Formatting feature of GEMCOS.

Some of the other capabilities provided under Formatting include:

- Forms retrieval
- Format modification without compilation of or interruption to application programs.
- Paging of both input and output, including forms.
- Numeric field verification.
- Variable-length fields, with zero/space fill and right/left justification.

The Formatting feature of GEMCOS provides the user with the ability to have flexibility out on the network, and at the same time device independence at the application program level.

Paging

Paging means that a string of characters constituting a single logical message must be physically segmented because the buffer capacity of the terminal involved has been exceeded. In effect, the MCS will act as an extension to that terminal's buffer capacity. Multiple types of terminals, each having a different buffer capacity, can be utilized for the same multipage transaction. The physical segmentation for each type will vary based upon buffer capacity. This variation is transparent to the application program involved.

Input, Display, and Update paging are provided:

- Input paging allows a terminal operator to enter a logical transaction into the system in a series of transmissions. That operator has the option to review pages and make corrections prior to releasing the transaction for processing.
- Display paging allows an application program to direct an output message to a terminal, where the message exceeds that terminal's buffer size. The MCS will deliver the first page automatically. Succeeding output transmissions are based upon terminal operator request. Browsing is permitted in both the forward and backward directions. The operator may terminate or return to the start of the transaction at any time.
- Update paging is a combination of Display and Input paging. An application program provides data to the terminal operator in the form of an output message. The terminal operator is permitted to review an output page and either change the unprotected data or leave it the same. The operator is then normally expected to return the page to the MCS. The MCS will then provide the operator with another page to work on, until the operator indicates that the transaction is to be released for processing. Browsing is permitted in both directions.

Reentrancy

Parallel processing of a wide variety of transactions is considered to be the normal mode throughout the total system. Parallel processing by a single application program of multiple transactions is readily available through the Reentrancy feature of GEMCOS, and can be totally transparent to the application programmer. Reentrancy implies that at any point in time multiple executions of the same program code can be in process for separate transactions. In addition, physical executions of any program can be concurrent on multiple-processor systems. Only one copy of the machine code is present in reentrant mode.

Based upon user specification, the MCS will dynamically initiate new executions of user programs as dictated by run-time transaction loads. Conversely, as volumes drop, the MCS will initiate termination procedures for user programs. The purpose of the Reentrancy feature is to provide the user with the ability to dynamically adapt to changing network demands, in order that response times can be minimized through parallel processing.

Recovery

The B 7800/B 6800 GEMCOS program product provides a range of recovery capabilities within the Transaction Control Language. The user has the flexibility to analyze application-oriented needs and then, on a program-by-program basis, select the recovery options required.

Recovery is invoked when some failure has occured which interrupts normal processing of transactions and which requires corrective action. The reestablishment of normal processing is considered to be the result of the recovery process.

It is desirable to free user programmers from recovery concerns to the maximum extent possible, since recovery solutions are often extremely complex. Burroughs accepts responsibility for playing a major role in assisting the user during the actual recovery process. The goal of GEMCOS is to make the recovery process transparent to application programs. In return, the user is expected to follow several straightforward programming conventions for normal processing. These conventions vary slightly, based upon the level of recovery required.

The MCS will automatically generate an audit trail of input/output transaction images to disk or disk pack. Generation of this audit trail is transparent to the application program.

The audit trail provides protection for message queues. It is the MCSs responsibility to reestablish transaction queues during the recovery process.

Recovery capabilities range from a rather conventional checkpoint/restart technique to an automatic transaction-queue, application program, and data base rollback-and-synchronization scheme. Once again, the emphasis is on providing users with the flexibility to easily adapt in order to effectively meet a broad range of on-line data processing requirements. B 7800/B 6800 GEMCOS is continuous-processing oriented.

Network Control

In order to dynamically affect certain levels of control over the data communications processing environment, the user has the capability to identify certain Network Control Stations. Such stations are privileged in that they may participate in various network management-oriented transactions. The MCS notifies these stations of various exception conditions as they occur. Network Control Stations are permitted to make various inquiries to the MCS. In addition, these stations may dynamically alter a subset of the Transaction Control Language features. Included within the scope of a Network Control Station is the ability to notify the MCS that the on-line network is being physically reconfigured. The system takes appropriate action such that only those stations experiencing reconfiguration need be temporarily out of service.

ADDITIONAL GEMCOS FEATURES

- A variety of statistical information concerning stations, programs, and the MCS is available to the user in a real time mode. In addition to real time accessable statistics, a set of files is provided containing input and output messages. Using these files the user can obtain statistics concerning peak-load conditions, network utilization and response times. This allows fine tuning of a system following initial implementation.
- Retransmission of output upon request. This feature can be useful in the event of paper jam or tears at a terminal.
- The system as a whole detects, diagnoses, and recovers from a variety of error situations.
- A Line Analyzer software module is contained within the MCS. This module is designed to work in conjunction with a hardware component to troubleshoot various terminal/line/data set/adapter-oriented problems.
- Network Control Stations can monitor transmissions to/from stations on the network transparent to the network.
- The capability to pass user programs fixed data relating to input stations and/or the particular transaction type is provided.

- User programs can be described as PERMANENT or TEMPORARY. This description determines whether a program will normally terminate when there are no input transactions queued for it for some user-defined period of time. Describing a program as PERMANENT does not mean that the code segments of that program will permanently reside in main memory.
- An interface is provided for generalized, user-written routines.
- Administrative Message Switching is permitted between stations on the network.
- Routing headers for computer-to-computer transmission are provided for.
- User programs must be written in higher-level languages. Assembler code is not used on the B 7800/B 6800. The MCS is capable of interfacing combinations of COBOL, PL/1, and ALGOL.
- A special GEMCOS subsystem assists users in analyzing their basic system design prior to implementation. It allows transactions to actually be transmitted across lines, flow through the MCS to user programs, and result in data base access and response generation. The user can measure actual data movement capacity under varying structural environments.

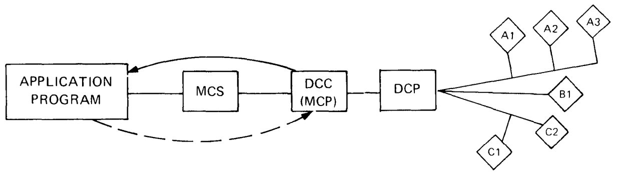
Application Program Considerations

Since the communications software is totally contained within the three functionally separate modules (NDL, DCC, MCS), the application program is not concerned with communications. To wit:

- A non-data comm program reads a logical record from a card reader and writes a logical record to a line printer;
- A non-data comm program reads a logical record from a disk file and writes a logical record back to a disk file;
- A non-data comm program reads a logical record from a disk file with an actual key and writes back to the disk file with a different actual key;
- A data comm program uses a "remote file" like a random access disk file;
- The application does a "WRITE" to a file, giving the logical record. The DCC/DCP does a physical write including start bits, stop bits, etc., using information as to which physical terminal is attached to the file. All this is masked from the application program;
- The application does a "READ" from a file to get the logical record (i.e., message text). The DCC/DCP does a physical read from the terminal attached to the file. Again, this is masked from the application program;
- For a multi-station file, as declared in NDL, the application can write to any station in the file.

Let's take an example of FILE F2 with a family of stations = A3, B1, C2. The "relative station number" (RSN) for these stations in this file is A3:1, B1:2, C2:3. The application can write to one particular station by specifying RSN in the actual key. By specifying RSN = 0, the application can do a "broadcast write" to all stations in the file.

When the application does a read from the file, it may wish to know which station submitted the message. The application can find out by interrogating the LASTSTATION file attribute (maintained by the DCC):



Write to file F2, RSN = 3, msg. goes to C2 only.
Write to file F2, RSN = 0, msgs. go to A3, B1, C2 only.

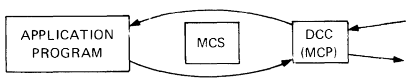
The application can find other information about the file via file attributes (and about the station via file attributes specifying RSN), e.g.,

FAMILY —	Can read or add new station to file family
TITLE —	Station name
WIDTH —	Of buffer in the physical terminal
TIMELIMIT —	Can set this on a read, so the application can take action if no input is received within a certain time
SCREEN —	Denotes whether or not the terminal is a screen device

The MCS is informed when a remote file is opened by an application program. At that point it can either assign the station(s) (to the file) as declared in NDL or it can deny the assignment. In this event the application would take end-of-file action.

The MCS may intercept each message and pass it on (e.g., for logging or for a trail for recovery), or it may relinquish any responsibility for it.

If a file assignment is allowed and the MCS decides to be non-participating, messages then bypass the MCS. The MCS will become involved only when end-of-file action is taken by a convention established between the application program and the terminal users, or the application closes the file.

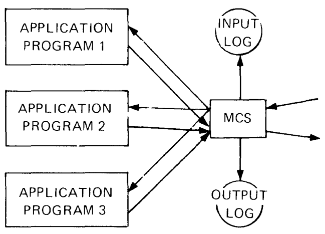


All error messages for a given station go to its controlling MCS, as do control messages (e.g., messages with the first character = NDL-supplied "control character"). For example, if CONTROL CHARACTER = "?", then the MCS can specify several messages to itself. For example, "?WRU" (who are you) might give an identifying response, while all other messages transmitted from the terminal go to an assigned file (e.g., terminal input).

For example, a sample terminal session might appear as follows:

```
ONE (TERMINAL INPUT)
TWO (APPLICATION RESPONSE)
?WRU (TERMINAL INPUT)
#I AM CANDE (output) (MCS RESPONSE)
THREE (TERMINAL INPUT)
END (convention for end-of-file)
```

- Note that a station can be attached to only one controlling MCS at a time.
- Also note that with a non-participating MCS, a station can be attached to many output files but to only one input file (otherwise, to which file would input be sent?). The MCS can also, for multiple output files, postpone assignment or could request the terminal operator to accept output or not.
- The MCS can also have multiple applications reading the same file (station).
- When no file assignment exists, all messages go to the MCS.



In a typical B 6800 one is likely to find several MCSs running simultaneously.

Burroughs