File Number 1800-36
Order Number GC26-3718-4

IBM Systems Reference Library

IBM 1800 Multiprogramming Executive Operating System Introduction

This manual is an introduction to the IBM 1800 Multiprogramming Executive (MPX) Operating System. Intended for new and prospective 1800 installation managers, programmers, and operators, it assumes only a knowledge of a few process-control and data-processing terms.

Topics discussed are: what the 1800 system is used for, what some of its capabilities and features are, what one might need from an operating system, what MPX does about these needs, and how MPX is organized.

Fifth Edition (June, 1970)

This is a major revision of, and renders obsolete, Form C26-3718-3 and Technical Newsletters N26-0598 and N26-0602. The entire manual has been rewritten to incorporate changes made to MPX in Versions 2 and 3.

This edition applies to Version 3, Modification 0 of the IBM 1800 Multiprogramming Executive Operating System, and to all subsequent versions and modifications unless otherwise indicated in new editions or Technical Newsletters. Changes may be made to the specifications in this manual at any time; before using this manual in connection with the operation of IBM systems, consult the latest SRL Newsletter, Order Number GN26-1800, for the editions that are applicable and current.

Forms for reader's comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114.

For copies of this or any other IBM publication, see your IBM representative or call your local IBM branch office.

Preface: How to Use this Book

This manual is an introduction to the IBM 1800 Multiprogramming Executive (MPX) Operating System. It is intended for new and potential 1800 installation managers, programmers, and operators; the only prerequisite is knowledge of basic data-processing and process-control terms. The data-processing terms are explained in Introduction to IBM Data Processing Systems, Order Number GC20-1684.

The first chapter, "What the 1800 System Does," discusses kinds of applications in which the 1800 system is used, and the different kinds of work you can expect it to do.

The second chapter, "What MPX Does for You," discusses the programming functions that are carried out by the operating system. It describes needs that you might want the operating system to handle, and what MPX does about these needs.

The third chapter, "How MPX Is Organized," tells how the various parts of the system are put together and where they are located.

As you read this manual, you might want more detailed information about 1800 system physical units and MPX system programs. These books can be used for reference:

1800 System Summary, Order Number GA26-5920, which introduces the physical units that make up an 1800 system.

1800 Functional Characteristics, Order Number GA26-5918, which describes how the physical units work.

MPX Programmer's Guide, Order Number GC26-3720, which discusses MPX system organization and programming techniques.

MPX Planning for Versions 2 and 3, Order Number GC26-3731, which describes the features of MPX added in Versions 2 and 3 of the system.

MPX Subroutine Library, Order Number GC26-3724, which describes each of the system subroutines.

MPX Operating Procedures, Order Number GC26-3725, which tells how to generate, operate, and maintain MPX.

1800 Assembler Language, Order Number GC26-5882, which tells how to write programs in assembler language.

1130/1800 Macro Assembler Programming, Order Number GC26-3733, which tells how to use macro instructions.

1130/1800 Basic FORTRAN IV Language, Order Number GC26-3715, which tells how to write programs in FORTRAN.

Communications Adapter Programming, Order Number GC26-3757, which tells how to write programs to carry out communications with other computers and terminals.

Binary Synchronous Communications--General Information, Order Number GA27-3004, which describes the programming conventions that govern communications between the 1800 and other computers and terminals.

1130/1800 Plotter Subroutines, Order Number GC26-3755, which describes MPX subroutines for controlling the 1627 plotter.

This page intentionally left blank.

Contents

Illustrations

Figures

Tables

This page intentionally left blank.

Chapter 1: What the 1800 System Does

The IBM 1800 Data Acquisition and Control System is a computer system designed specifically for use in real-time applications--applications that require fast response to physical events as they take place. The system accepts analog and digital electrical signals from devices such as thermocouples, pressure and temperature transducers, flowmeters, analytical instruments, and contacts. The response of the system to these signals can take several different forms:

- The system might record data from the signals, operate on it in some way, and make the results available for use. For example, the 1800 system might receive signals from contacts in gas pipelines. It could convert these signals into measurements of the flow in each pipeline and perform computations on the measurements, such as computing the hourly or daily average flow. The system might make the results available to gas company personnel by printing a report or by producing a graph.

- The system might, as a result of signals received, notify an operator that some action is necessary. For example, the 1800 might receive a signal from a thermocouple in a steel-mill furnace. The system could convert the signal to a reading of the temperature of molten steel and compare the reading with prespecified limits. If this comparison showed that the temperature was too low, the 1800 could print a message to an operator telling him to raise the temperature.

- The system might generate signals that cause adjustments to be made without an operator's intervention. For example, an 1800 might receive signals from pulse counters at an intersection of two streets. From these signals, the system might determine that northbound traffic was twice as heavy as westbound traffic. It could then generate signals to controllers that would automatically adjust traffic lights to allow the flow of northbound traffic to increase.

Real-Time Applications

The 1800 is used primarily in three kinds of applications: data acquisition, process control, and manufacturing.

DATA ACQUISITION

Data acquisition is the collection, at its source, of accurate, physically generated data for evaluation and control. The major functions of data-acquisition applications are acquiring sensor-based data as it becomes available and then editing, formatting, reducing, recording, and displaying it. These uses of the 1800 are associated with the monitoring of processes and experiments, the testing of physical materials and structures, and the simulation of processes, structures, and missions.

In a data-acquisition application, the 1800 can acquire data from instruments such as spectrometers, flowmeters, and thermocouples, or from instruments that examine biological specimens such as humans and test animals. The 1800 can also adjust data-acquisition instruments.

For example, an 1800 receiving information from a radio antenna might control the positioning of the antenna.

After the data is acquired, the system reduces it to a manageable form. The system may also improve the usefulness of the collected data, by operating on and analyzing it in various ways or by reducing it on a statistical basis.

After data reduction, the system usually compares the data with previously acquired data or with theoretical models and sends the results to the investigator through graphic output devices.

PROCESS CONTROL

The 1800 in a process-control application collects and analyzes data that describes the behavior of a continuous process, such as the refining of oil. The system can take corrective action either by issuing instructions to an operator or by effecting direct physical changes to the process.

The signals that the 1800 receives from devices attached to the process are called process input. By reading and operating on various items of process input, the 1800 can monitor the status of many parts of the process, such as temperatures, flow rates, and the amount of raw materials being used. Engineering and operational data stored in the system is used to determine what action should be taken to keep the process running properly. These decisions can also be made by control optimization programs, which are programs that make adjustments based on interrelationships of various parts of the process.

The 1800 generates signals that control the valves, switches, and relays that in turn control the process. These signals are called process output.

MANUFACTURING

In a manufacturing application, the 1800 monitors manufacturing operations, tests the quality of products, furnishes production data to higher-level production and inventory control systems, and manages the flow of work between departments. Data in these applications can be collected directly from sensors or entered by individuals into special devices located in the manufacturing plant. These devices are discussed later in this chapter, under "Communications."

Background Processing

The 1800 in a real-time application isn't responding to real-time events all the time. The computer has some free time--perhaps hours at night when the process it controls isn't running; perhaps fractions of seconds between responses to real-time events. The 1800 can use such free time to perform scientific and data-processing tasks, such as processing a payroll and preparing production reports. This work is called background processing, because it's done only when the 1800 has no real-time work to perform. Data for these tasks, together with instructions specifying what the 1800 is to do with the data, constitute data processing input. Data processing input can be entered into the

system from a card reader, a disk drive, a keyboard, a magnetic tape drive, or a paper tape reader. The results of background-processing jobs--data processing output--can be placed on disk or magnetic tape, punched onto cards by a card punch or onto paper tape by a paper tape punch, made into a graph by a plotter, or printed in the form of a message or a report on a printer.

Communications

The 1800 can communicate with individuals and with other computers in various ways.

The 1800/2790 data communication system is a set of input devices that allow communications between the 1800 and individuals throughout a manufacturing plant, a school, a hospital, or another installation. Individuals at these input devices can send the 1800 data recorded in various media--cards, badges, and keyboard entries. The 1800 can respond by turning on lights at some of the input devices and by printing messages on a printer.

The 1800 can communicate with other 1800s and with System/360s by using information stored on disk packs shared by two or more systems. A communications adapter allows the 1800 to communicate over telephone lines with System/360s, 1130 computing systems, other 1800s, and 2770 and 2780 data transmission terminals.

An 1800 in a manufacturing plant might, for example, send statistics on the productivity of several manufacturing machines to a System/360. System/360 could use this information, together with data from other sources, to determine how the overall productivity of the plant might be improved.

Figure 1 pictures an 1800 carrying out real-time tasks, background processing, and communications.

Programming

The 1800 performs the kinds of jobs we've been discussing by executing programs. A program is a sequence of instructions that tell the computer what to do in order to achieve some goal.

Some programs are unique to a single application. For example, a program that tells the 1800 how to control a valve in a cement-making process would be of no use to someone who uses the 1800 to test the reactions of biological specimens; a program that optimizes gasoline blends would be of no use to someone whose 1800 controls a stacker crane. Each application has its own real-time, accounting, problem-solving, and record-keeping needs, and programmers at each 1800 installation are responsible for writing programs that tell the 1800 how to meet these needs.

Other programs are needed by all or many 1800 applications. Examples of these are:

- A program that tells the 1800 how to read a punched card.
- A program that determines the order in which various programs are to be executed.

The Multiprogramming Executive (MPX) Operating System is a set of programs, sometimes called system programs or control programs, that are needed by all or many applications. By using MPX, you can concentrate on writing the programs that are unique to your application.

In the next chapter there's a discussion of the functions the operating system performs for you.

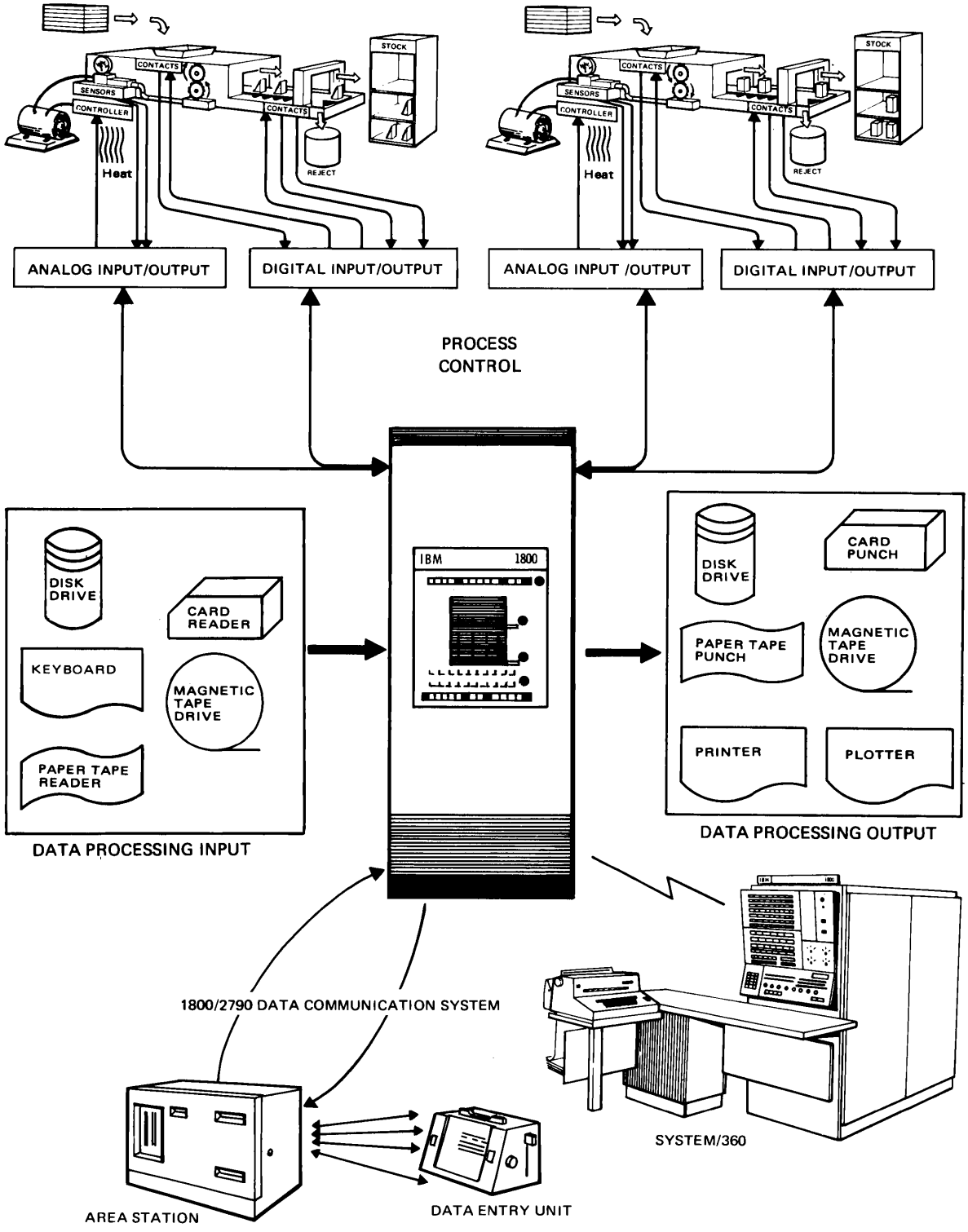


Figure 1. Functions of the 1800 System

Chapter 2: What MPX Does for You

In this chapter, each numbered heading describes something you might want from the operating system. The discussion that follows the heading tells what MPX does about it.

1. The ability to write programs in symbolic languages, store them, and execute them

In order for the 1800 to execute an instruction, the instruction must be in machine language. Machine language consists entirely of zeros and ones. Some of the zeros and ones tell the 1800 what it's to do with data (such as add or subtract), some of them tell it the location of the data it's to operate on, and some of them represent the data itself. Some machine-language statements are shown in Figure 2.

For you to write long programs in machine language, however, would be tedious, time-consuming, and error-prone. MPX includes two language translating programs (or "language translators") to allow you to define solutions and applications in languages that are easier to learn and use than machine language.

LANGUAGE TRANSLATORS

The two language translators are the Macro Assembler and the FORTRAN Compiler. The Macro Assembler translates statements written in the assembler language, and the FORTRAN Compiler translates statements written in the FORTRAN language. The output from both translators is machine-language statements that the 1800 can execute.

Figure 2 shows equivalent FORTRAN, assembler-language, and machine-language statements. Notice that the FORTRAN statement is very much like a mathematical formula. FORTRAN stands for FORmula TRANslating system. The language closely resembles the language of mathematics; it allows engineers and scientists to define problem solutions in a familiar, easy-to-use notation.

The assembler language is more like machine language than FORTRAN is. Each assembler-language statement is translated (or "assembled") by the Macro Assembler into just one machine-language statement. The FORTRAN Compiler, on the other hand, might translate (or "compile") one FORTRAN statement into several machine-language instructions. The nine assembler-language statements shown in Figure 2 all together produce the same results as the single FORTRAN statement shown in the same figure.

The assembler language makes use of symbols, though the symbols are perhaps not so obvious as those used in FORTRAN. For example, in the FORTRAN statement in Figure 2, multiplication is indicated by an asterisk and subtraction by a minus sign. In the assembler-language example, multiplication and subtraction are indicated by the letters M and S.

The assembler language allows you more control over some of the system resources than does FORTRAN. A program written in FORTRAN might require that you write fewer statements than a corresponding program written in

assembler language, but it might also require more computer time and more storage space.

You can decrease the length of your assembler-language programs by defining macro instructions. A macro instruction is an instruction that is written like an assembler-language statement. When the Macro Assembler encounters a macro instruction, it processes a sequence of assembler-language statements that you have specified. You need to specify such a sequence only once. After that, you can specify that the sequence is to be processed by issuing the macro instruction.

The definition and use of a sample macro instruction are shown in Figure 2. Once the macro definition has been stored, issuing the macro instruction would have the same effect as issuing the FORTRAN statement or the sequence of assembler-language statements also shown in Figure 2.

After a program has been written and assembled or compiled, you may want to run (execute) it immediately, store it for future execution, or both.

Programs are executed by a machine unit called a processor-controller. This unit includes a central processing unit (CPU), which carries out arithmetic and logical operations, and main (core) storage, where programs are executed.

There normally isn't room in main storage for all your programs to be there all the time. Most programs are stored in disk storage and then read into main storage when they are to be executed.

MPX allows you to manage storage and execution of your programs in several different ways. You tell the system to store and execute programs by issuing control statements that activate various programs within MPX.

Com	Statement Number	Cont	FORTRAN Statement (columns 7–72)
1	2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
			A = B + C * (D(I) - E) / F

FORTRAN Statement

Label (21–25)	Operation (27–30)	F (32)	T (33)	35–40
	LDX	I	1	SGT1
	LD	L	1	D
	S	L		E
	M	L		C
	SLT			16
	SRT			16
	D	L		F
	A	L		B
	STO	L		A

Assembler-Language Statements

```
01100101110000000000000000111110
11000101000000000000000000111000
10010100000000000000000000111000
10100100000000000000000000100010
0001000010010000
0001100010010000
10101110000000000000000000111010
10000100000000000000000000100000
11010100000000000000000000011110
```

Machine-Language Statements

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operands & Remarks (35–)
	SMAC			
	CALC			SGT, AD, SUB, MULT, DIV, MORE, SPC
	LDX	I	1	SGT
	LD	L	1	AD
	S	L		SUB
	M	L		MULT
	SLT			16
	SRT			16
	D	L		DIV
	A	L		MORE
	STO	L		SPC
	MEND			

Macro Definition

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operands & Remarks (35–)
	CALC			SGT1, D, E, C, F, B, A

Macro Instruction

Figure 2. FORTRAN, Assembler-Language, and Machine-Language Statements

EXECUTIVE PROGRAMS

Some programs stay in main storage at all times. These programs make up what's called the Executive. Some of the MPX control programs are part of the Executive, and when you set up your system--during the process called system generation--you can specify that some programs you have written are to be included in the Executive. Two reasons that you might want to include a program in the Executive are:

- Programs in the Executive are immediately accessible; that is, they can be executed without waiting to be read in from disk storage.

- Programs in the Executive are accessible to all other programs. For example, say that ten different programs stored on disk all contain some sequence of statements that carries out some function. You could make that sequence into a subprogram, or subroutine. If you placed that subroutine in the Executive, it wouldn't be necessary for each of the ten programs to contain its own copy of the subroutine. By issuing one or two statements, each of the ten programs could specify that the subroutine was to be executed.

There are some drawbacks to placing programs in the Executive. One is that you can't alter the Executive without stopping the system and doing another system generation, and another is that the bigger the Executive, the smaller the area available for execution of other programs. You'll probably use other means of storage and execution for most of your programs. These other means are discussed in the next section.

CORELOADS

If a program isn't part of the Executive, then before it can be executed, it must be made part of a coreload. A coreload is an executable program or program portion. It contains machine-language instructions produced by the Macro Assembler or the FORTRAN Compiler, together with control information necessary to execute these instructions.

Coreloads are built by an MPX program called the Builder. By issuing a control statement, you can tell the Builder to make your assembled or compiled program into a coreload. You can build several different kinds of coreloads:

- You can build a coreload that is to be stored on disk and executed repeatedly. An MPX program called the Disk Management Program stores the coreload on disk when you issue the appropriate control statement. Later you can use another statement to read the coreload into main storage. The reading is done by a program called Program Sequence Control (PSC). The coreload remains in main storage until its execution is complete and the space it occupies is needed for execution of another program.

- You can build a coreload that is to be executed but not stored for reuse. Such a coreload might perform mathematical calculations that need to be performed only once. You can use a control statement to cause the Builder to build such a coreload and transfer it to main storage, where it remains until its execution is complete and the space it occupies is needed for execution of another program.

- You can build what's called a SPAR (special area) coreload. A SPAR coreload is different from the coreloads discussed above in that the space occupied by an ordinary coreload can be used for another

coreload as soon as execution of the first coreload is complete. A SPAR coreload remains in main storage until you specify in a control statement that the space it occupies can be used for another coreload. Subroutines in SPAR coreloads can be accessed just as fast as subroutines in the Fxecutive. The difference is that you can change the contents of a SPAR coreload without a new system generation, but changing the contents of the Executive requires a regeneration of the system.

2. Fast response to real-time events

Response to events as they take place is of paramount importance in data-acquisition and process-control applications. For example, an 1800 might count the particles from a radioactive source that collide with a target. Many thousands of these collisions might occur every second, and the 1800 must record all of them in order that the data acquired be usable.

Many features of MPX contribute to fast response to real-time events.

ACCESSIBILITY OF PROGRAMS

One feature that has already been discussed is that some programs can reside permanently in main storage, either in the Executive or in a SPAR coreload. The 1800 can execute such a program without spending time transferring the program from disk storage to main storage.

Another feature is that coreloads are stored on disk in machine language. As soon as a coreload is copied from disk into main storage, it's ready to be executed--no additional time need be spent to make it ready for execution.

MULTIPROGRAMMING

Multiprogramming is a technique that contributes to many of the features of MPX--among them, fast response to real-time events. Multiprogramming allows many programs to be in main storage at the same time and to share the use of system resources.

To effect multiprogramming, you must divide main storage into these sections when you generate your MPX system:

1. The Executive, which, as we've already said, includes many of the MPX control programs and can include some of your programs.

2. Up to 24 partitions where your programs are executed. You can define two kinds of partitions:

   - Partitions for execution of coreloads that serve your real-time needs, including SPAR coreloads.

   - VCORE, a partition where your background-processing coreloads can be executed. The Macro Assembler, the FORTRAN Compiler, and some other MPX programs are also executed in VCORE.

Every real-time MPX system must include the Executive and at least one partition. The number and size of other partitions depend on your needs.

Multiprogramming allows programs to be executed faster--and thus provides faster response to real-time events--by letting different programs use different resources of the system at the same time.

Any program, at any given time, requires only a fraction of the total resources of the system. For example, while a program is using the central processing unit to do addition or subtraction, it isn't using a printer; while a program is using a disk drive, it isn't using the CPU. Under MPX, different programs can use the CPU, printers, and disk drives, as well as other input/output devices at the same time, thus shortening the execution times of some or all of the programs.

Let's look at an example of the advantages of multiprogramming. Suppose programs A, B, and C are being executed in a system without multiprogramming. Program A reads some information from disk, operates on it, and prints out a report. Program B performs some computation, prints a message, performs some more computation, and writes the result on disk. Program C performs some computation, reads some information from disk, performs some more computation, and writes the result on disk.

Figure 3 illustrates the sequence in which various operations would be performed when the three programs are executed one after the other. Notice that while any one part of the system, such as a disk drive, is being used, the other parts, such as the CPU, are idle.

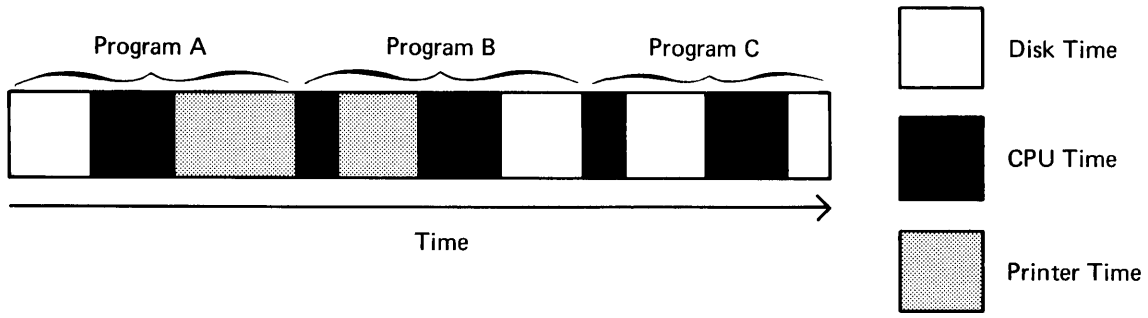


Figure 3. Sequential Execution of Programs

Figure 4 shows the sequence in which the same functions might be carried out under MPX. Note that the three resources involved (CPU, disk drive, printer) are in some cases all used at the same time. Note also that Program A is completed just as soon, and Programs B and C are completed sooner, than they were in the system without multiprogramming.

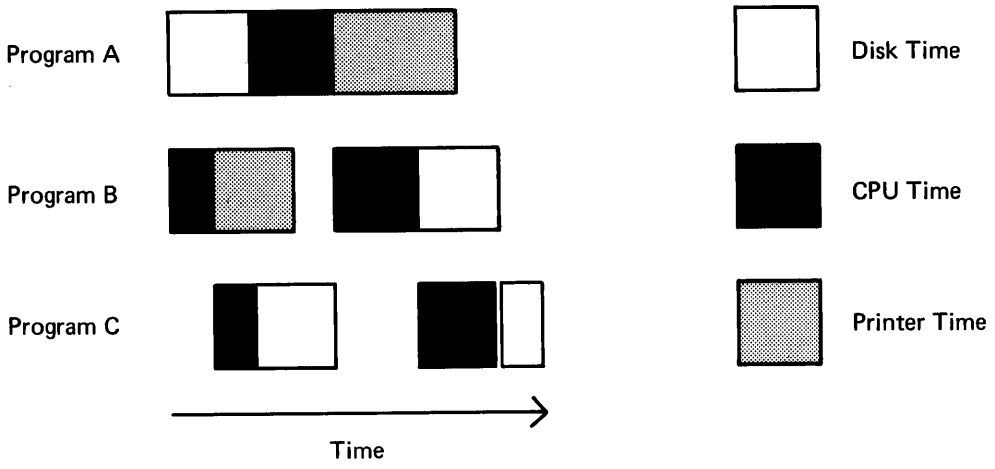


Figure 4. Concurrent Execution of Programs

INTERRUPTS

In order that MPX respond to a real-time event, it must have some way of knowing that the event has taken place. To accomplish this, each 1800 can be connected so that physical events cause interrupts. An interrupt is the recognition of an event that alters the sequence of program execution by causing execution of a specific program. When you set up your system, you specify which program is to be executed in response to each interrupt. The program is said to service the interrupt.

When the interrupt occurs, a physical indicator is set. The 1800 uses the indicator to determine the source of the interrupt. Then MPX executes the program that has been specified to service that interrupt.

Interrupts can, as we've said, be caused by real-time events. For example, if an 1800 were controlling the testing of resistors as they were manufactured, an interrupt might be generated every time the testing mechanism finished testing a good resistor. A different interrupt would be generated when the resistor being tested was found to be defective. Different programs would be executed to service the two interrupts. In an actual process, the two programs might cause the resistors to be moved to different locations.

An interrupt is also generated each time an input/output device finishes an operation. Programs within MPX are executed in response to those interrupts. Interrupts can also be generated by programs. For example, in the resistor-testing application described above, an interrupt was generated when a defective resistor was encountered. The program executed in response to that interrupt might move the defective resistor to a specified spot, update some stored production data, and generate another interrupt. The program servicing this second interrupt could begin the testing of another resistor.

MPX can distinguish up to 760 different interrupts. The actual use of interrupts at your installation depends on your needs.

3. The ability to specify relative importance to different interrupts

In real-time applications, it is essential that some interrupts be given a higher priority than others. For example, say your 1800 is controlling a paper-making process. Assume that an interrupt is generated because the amount of rag being combined with wood pulp is not correct for the grade of paper being produced. A program to correct the rag input begins execution. Now say something goes wrong with the machine that dries the paper, so that if corrective action is not taken quickly a fire would result. You would want an interrupt caused by this event to have priority over the first interrupt, even though the first interrupt had not been completely serviced.

INTERRUPT LEVELS

The 1800 allows you to assign priorities to your interrupts by means of a system of interrupt levels. Each interrupt, whether generated by an external device, an input/output device, or a program, is assigned to an interrupt level from 0 (highest priority) to 23 (lowest priority). Up to 16 interrupts can be assigned to the same level. Each of the 16 can be caused by up to 16 different events.

If a program is being executed as a result of a level 5 interrupt, and a level 2 interrupt occurs, execution of the level 5 program is suspended until the level 2 interrupt has been serviced. After the level 2 interrupt is serviced, execution of the level 5 program can be resumed just as if it had not been suspended.

THE BASIC LEVEL

There's another level that's lower than any of the interrupt levels. This lowest level is called the basic level. Background processing is done on the basic level, and you can specify that other programs be executed on the basic level, too. Programs are executed on the basic level only when there are no interrupts waiting for servicing on interrupt levels.

INTERRUPT LEVEL OPERATION

Figure 5 shows the sequence of execution of four programs: a background job and programs servicing interrupts on levels 9, 10, and 11. In the figure, the level 11 interrupt occurs first, the level 10 interrupt second, and the level 9 interrupt third.

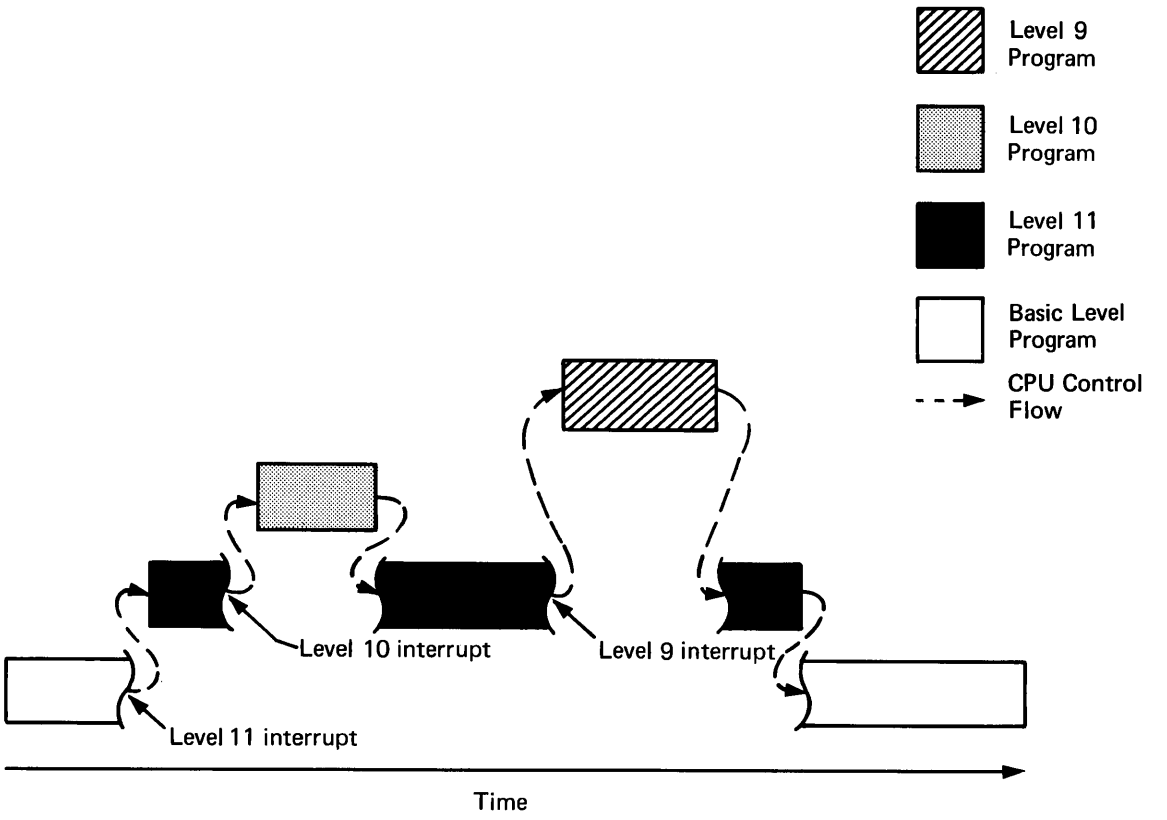


Figure 5. Execution of Programs on Interrupt Levels

Notice that the level 11 program is interrupted twice, once by a level 10 interrupt and once by a level 9 interrupt. The level 10 program is executed without interruption, because no interrupt on a higher level occurs during its execution. The background-processing job is suspended when the first interrupt occurs; its execution is not resumed until there are no interrupts waiting to be serviced.

If a program is being executed as a result of an interrupt on a level, more interrupts on that same level may occur before execution of the program is complete. For this reason, MPX maintains a queue (waiting list) of coreloads to be executed on each interrupt level.

Execution priorities can also, at your option, be established within a level. For example, say a program is executing as a result of a level 2 interrupt. During its execution, two more level 2 interrupts occur; the one that occurs second is specified to have a higher priority than the first. First, the program already being executed will complete its execution. Then the two interrupts will be serviced. The one that occurred second will be serviced first, because it has the higher priority.

An MPX control program called Master Interrupt Control (MIC) analyzes each interrupt as it is recognized and arranges for it to be serviced, by passing control to a program that's already in main storage, placing a coreload in the queue for the appropriate interrupt level, or copying the contents of VCORE onto disk and placing an interrupt-servicing coreload in VCORE.

Another MPX control program, Program Sequence Control (PSC) handles scheduling of programs that are requested by sources other than interrupts. For example, an instruction in one program may be a request that another program be executed. PSC would handle scheduling of this second program.

4. The ability to alter the way your interrupts are serviced without stopping the system

In real-time applications, it is often necessary that interrupt-servicing programs have a very short access time. Access time (the time between an interrupt and the beginning of execution of the program) for a program is shortest if that program is already in main storage. We've discussed before the two ways of keeping programs in main storage. One is by making them part of the Executive; the Executive, however, can't be altered unless the system is stopped and regenerated. The second way to keep a program in main storage is to make it part of a SPAR coreload.

When you generate your system, you can specify, if you want, that some of your coreloads are to be SPAR coreloads. SPAR coreloads that you specify are loaded into their partitions by the MPX control programs LSPCL and LSPAR. Each SPAR coreload then remains in main storage until you call LSPCL or LSPAR to replace it or the MPX control program CSPAR (clear SPAR) to release the partition for execution of another coreload. By using one or more SPAR coreloads, you can ensure that interrupt-servicing programs have minimum access time while still allowing yourself to change these programs without stopping the system.

5. The ability to do accounting, problem-solving, and record-keeping jobs as well as real-time tasks

Computer time not needed for real-time tasks can, as discussed under "Background Processing," be used for other kinds of work. These include accounting, problem-solving, and record-keeping jobs. Execution of background jobs must be carried out in VCORE on the basic level. You can't assign priorities to background jobs; they're executed one after the other in the order they're entered into the 1800.

Execution of background jobs is managed by an MPX control program called the Batch-Processing Monitor Supervisor (SUP). SUP is part of a group of programs that make up the Batch-Processing Monitor (BPMON). BPMON contains several programs that have already been mentioned: the Macro Assembler, the FORTRAN Compiler, the Builder, and the Disk Management Program. All these programs are executed as background jobs.

You can also build real-time coreloads to execute on the basic level. If you do this, you can assign BPMON a priority within the basic level. Then a program called Time-Sharing Control (TSC) schedules the use of VCORE. TSC allocates use of VCORE to BPMON if there are no coreloads of a higher priority than BPMON waiting to be executed on the basic level.

6. The ability to keep the system busy and to maximize the work done

Several features of MPX contribute to the system's efficient use of time.

One of these is the overlap of processing and input/output operations. This overlap keeps the CPU and the input/output devices operating simultaneously.

Another of these features is the queueing of coreloads for execution on each level. Queueing allows the proper coreload to be read into a partition as soon as the one already there has completed execution.

The system is also kept busy by the automatic transfer of control to BPMON when there is no process work to be done.

7. A method by which programs can communicate with each other

You may want to make data generated by some of your programs available for use by other programs. For example, you might have one program that records the number of items produced by various manufacturing machines, and a second program that prints a report on the productivity of your manufacturing activities. You would want the numbers recorded by the first program to be available to the second program.

MPX would allow such programs to communicate with each other by means of main-storage areas called COMMON areas.

Any program being executed in a partition can define an area called COMMON within that partition. It can leave data in COMMON for future use by another program being executed in that partition. You have to be careful when you're using this kind of COMMON, because a COMMON established by a coreload isn't protected from other coreloads that are executed in the same partition. For example, say coreload TIME defines a COMMON in its partition, and stores data there for subsequent use by coreload PLUS. Between the execution of TIME and PLUS, coreload FILL is executed in the same partition, but FILL doesn't define a COMMON at all. FILL could use the COMMON that TIME defined for any purpose at all, and could destroy the data stored there by TIME. By the time PLUS was executed, the data stored by TIME would no longer be in COMMON.

A second kind of COMMON area is called INSKEL COMMON. INSKEL COMMON is part of the Executive, so it's in main storage at all times. Any program you write, no matter where it's executed, can store data in INSKEL COMMON and can read data that other programs have stored there.

You define the sizes of all the COMMONs, so they can be as large or as small as your needs dictate.

8. The ability to use the input /output devices easily

INPUT/OUTPUT CONTROL SUBROUTINES

MPX includes subroutines that make it possible for you to use all the input/output (I/O) devices shown in Figure 6. The minimum and maximum number of each device supported is shown in Tables 1 and 2. For each of the devices shown, there are one or more input/output control subroutines that manage the transfer of data between the device and main storage.

Any program can initiate such a transfer by issuing a call to the appropriate input/output control subroutine. A call consists of several

instructions in which you specify the location of the data to be transferred, where it's to be sent, how much data is to be transferred, and what's to be done when the transfer is complete. For example, a call to the input/output control subroutine for the card read punch might specify that the contents of one card in the card read punch are to be read into a specified location in main storage. A call to the input/output control subroutine for the 2311 disk storage drive might specify that 29 characters beginning at a certain location in main storage are to be copied onto a disk pack, beginning at a certain location on the disk pack.

The following MPX input/output control subroutines control data-processing I/O devices:

- BULKN--handles output to and input from the 1810 disk storage unit and sections of 2311 disk storage drives that are treated like 1810s.
- CARDN--handles the reading and punching of cards on the 1442 card read punch.
- FILEN--handles input to and output from the 2311 disk storage drive.
- MAGT--handles input to and output from the 2401 and 2402 magnetic tape units.
- PAPTN--handles input from the 1054 paper tape reader and output to the 1055 paper tape punch.
- PLOTX--handles output to the 1627 plotter.
- PRNTN--handles printing and line control on the 1443 printer.
- TYPEN, WRTYN--handles input from the 1816 printer keyboard and output to the 1816 and the 1053 printer. (TYPEN and WRTYN are two names for the same subroutine.)

The following input/output control subroutines control process I/O devices:

- AIRN--handles input of data to main storage from a random set of analog input points.
- AISN, AISQN--handles input of data to main storage from a sequential set of analog input points. (AISN and AISQN are two names for the same subroutine.)
- DAOP--handles output to digital and analog output devices.
- DIEXP--handles reading of a group of digital input points and storing of the values in a special format.
- DINP--handles reading and checking of several groups of digital input points.

The following input/output control subroutines control communications I/O devices:

- BSCIO--handles input from and output to the computers and terminals attached to the 1800 by the communications adapter.
- 2790 Input/Output Control Subroutine--handles input from the 2791 and 2793 area stations, 2795 and 2796 data entry units, and 1035 badge readers, and output to the 2791 area stations and 1053 printers attached to area stations.

To run MPX, you must have at least one 1442 card read punch, one 1053 printer, and one disk drive (2311 or 1810 Model A). If you don't have 2311s, then the 2790 system requires an 1810 Model A2 or B2. The 1053 printer and the disk drive are used by some of the MPX system programs, so the input/output control subroutines for those devices must be in the Executive. You can put CARDN in the Executive or a coreload.

Of the input/output control subroutines for other devices, you need to include only those for devices you are using in your system. Any of these other input/output control subroutines can be put in either the Executive or a coreload.

Table 1. Minimum and Maximum Machine Configurations

Device	Number in Maximum System	Number in Minimum System
2311 Disk Storage Drive	8	1 or 1(Model A1 or B1)
1810 Disk Storage Unit	1(Model A3 or B3)	
1442 Card Read Punch	2	1
1816 Printer Keyboard	8 (including up to two 1816s)	1
1053 Printer		
1443 Printer	1	0
1627 Plotter	1	0
1054 Paper Tape Reader	1	0
1055 Paper Tape Punch	1	0
2401/2402 Magnetic Tape Unit	2	0
Communications Line Adapter	8	0
2790 Loop Adapter	2	0

Table 2. Maximum 2790 Loop Configuration

Device	Number in Maximum Loop
2791 Area Station or 2793 Area Station	100
2795 Data Entry Unit	3 per 2791 Model 1 or 2793 area station; maximum of 1024
2796 Data Entry Unit	
1035 Badge Reader	300 (3 per 2791 Model 1 area station)
1053 Printer	100 (1 per 2791 Model 1 or 2793 area station)
User-Supplied Input Device	100 (1 per 2791 Model 1 area station)

DISK UTILIZATION PROGRAMS

The Disk Pack Initialization Program (DPIP), the BOM Disk Write Addresses Program (BDWAP), and the Disk Management Program (DMP) are programs that give you further assistance in the use of disk storage.

DPIP allows you to test a 1316 disk pack (used with 2311 disk storage drives) for defects and initialize it for use. BDWAP initializes a 2315 disk cartridge (used with 1810 disk storage units). By issuing control statements that activate parts of DMP, you can carry out such functions as storing programs and data on disk, deleting programs and data previously stored, copying disk packs and cartridges, and reserving disk space without actually storing anything.

9. Accessibility of subroutines that are used by many programs

Some subroutines may be used as part of the execution of many different programs. Storing a copy of such a subroutine with each coreload that uses it would require extensive disk space. The partitions where the coreloads are executed would also have to be large enough to accommodate the subroutine along with the rest of the coreload. You can save space by placing such subroutines in the Executive. Then they can be accessed by a program executing in any partition.

10. The ability to interrupt a subroutine, use it, and then complete the interrupted execution properly

If a program is being executed on an interrupt level, and a higher-level interrupt occurs, execution of the program is suspended until the higher-level interrupt is serviced. However, a problem arises when a subroutine in the Executive is interrupted in this way and the program servicing the higher-level interrupt calls the same subroutine.

Let's look at an example. Say you have two communications adapters, one of which generates level 1 interrupts and the other of which generates level 2 interrupts. The input/output control subroutine BSCIO services all interrupts for both communications adapters. Say BSCIO is transmitting or receiving data with the level 2 communications adapter when an interrupt from the level 1 communications adapter occurs. The first execution of BSCIO is suspended, and another execution of BSCIO is begun to service the level 1 interrupt. If BSCIO were a conventional subroutine, the partial results of the first execution would be lost. The communications that had taken place during the first execution would have to be repeated.

To prevent this kind of problem, MPX subroutines can be written to be reentrant. When execution of a reentrant subroutine on a given interrupt level is interrupted, all the results accumulated so far are saved in an area called the level work area, and execution can later be resumed at the point where it was interrupted. There's a level work area for each interrupt level in the system.

All MPX system subroutines that may be called from more than one level are available in reentrant form. This means they can be called from different interrupt levels with no loss of partial results. Many of the system subroutines are available in both reentrant and nonreentrant forms. The reentrant forms generally require less main storage, but require more execution time than the nonreentrant forms. You can choose the form that better meets your needs.

11. The ability to perform some tasks on a timed basis

Assume that your 1800 is controlling the packaging of peanut butter. You might want the 1800 to weigh a finished jar of peanut butter every two seconds, check the flow of salt into the peanut butter every two minutes, and print an inventory report every two hours.

Timers built into the 1800 system and into MPX allow you to do things on a timed basis. A timer is a device that generates an interrupt every time a specified interval of time elapses. Three physical timers are built into the 1800 hardware. Two of the physical timers are available for your use; MPX uses the third to keep track of the time of day and

the programmed timers. A programmed timer is a location in main storage. A value is placed in this location and subsequently altered at regular intervals until it reaches zero, at which time an interrupt is generated. When you generate your system, you can include up to 30 programmed timers.

All 32 timers available to you can be specified to generate interrupts ("time out") at different time intervals. In the example above, you could set three timers to time out at two-second, two-minute, and two-hour intervals.

Timer operations are controlled by an MPX program called Interval Timer Control (ITC). ITC can also check to make sure that expected interrupts from I/O devices are received. If such an interrupt is not received (a "no response" situation), ITC prints a message and sets an indicator that prevents the device from being used until the problem is corrected.

12. The ability to do conversions and arithmetic and functional calculations

Part of MPX is a library of subroutines that are often needed by user programs. Some of these subroutines change the representation of data from one form to another. Others compute functions of variables such as sines, logarithms, and square roots. Still others perform arithmetic operations on data in various formats.

These subroutines reside on disk and must be made part of a coreload or included in the Executive before they can be executed. The Builder can make any of these subroutines part of any coreload.

13. The ability to communicate with other computers

The communications adapter (CA), which permits the 1800 to communicate with other computers, was mentioned under "Communications" in Chapter 1. All communications using the CA are done in accordance with a set of ground rules defined as binary synchronous communications (BSC). BSC requires that data being transmitted be formatted in particular ways, and that it contain certain defined control information.

MPX includes an input/output control subroutine and several additional subroutines in support of communications. You must set up data to be transmitted in accordance with BSC specifications, but the MPX control programs handle all the actual transmission and receipt of data.

The 1800 can also communicate with other 1800s and with System/360s by sharing information stored on disk packs. If your system includes 2311 disk storage drives, then you can create sets of data that two systems can share.

14. The ability to communicate with the 1800 from remote devices

The 1800/2790 data communication system was described briefly under "Communications" in Chapter 1. This system allows the 1800 to exchange information with operators at special input/output devices.

MPX allows you to determine the nature of these exchanges by defining transactions. A transaction is a set of steps in which an operator may send information to the 1800, the 1800 may send information to him, and the 1800 may store the information it receives. Following the collection and storing of a predetermined amount of data, you will probably want to execute another program which will analyze the collected data. MPX allows you to do this by automatically setting an interrupt. You must specify the program to be executed and the amount of data to be collected before the interrupt occurs. Thus the data collected from the remote stations can be processed in any way you choose.

For example, you might set up a transaction that allows individuals throughout a plant to order supplies from a central warehouse. An interrupt might occur after 100 orders had been received. The program servicing the interrupt might print a list of the orders and punch cards to be attached to the supplies before they are delivered.

After you define your transactions and set up your system, MPX handles all transfer of data between the 1800 and the remote input/output devices.

15. The ability to develop your own programming language or some instructions for the use of people at your installation

You'll probably discover that there are some sequences of assembler-language instructions that you use over and over again. The Macro Assembler allows you to condense each such sequence into a single instruction called a macro instruction. When the Macro Assembler encounters a macro instruction, it processes a prespecified sequence of assembler-language statements.

In some cases, it may be possible for you to use macro instructions to create a programming language tailored to the needs of your application. By writing several macro instructions with descriptive names, such as ORDER, TEST, or FLOW, you might greatly simplify the programming effort at your installation.

16. System recovery from errors when necessary; timely assistance in recovery from other errors

In a real-time application, it is important that an error--either a programming mistake or a machine malfunction--affect system operation as little as possible. MPX contains error-handling procedures designed to keep the system running whenever possible.

Many programming errors are detected before a program is actually executed. For example, the FORTRAN Compiler detects errors in the syntax of FORTRAN statements; DMP will detect an error if you try to store a program in an area that is too small to hold the program. These pre-execution errors are detected by the Macro Assembler, the FORTRAN Compiler, the Builder, SUP, and DMP. Each of these programs causes an error message or code to be printed whenever it encounters an error. You can then use the error messages and codes to correct your program.

Some programming errors don't come to light until a program is executed. A program might, for example, try to write in an area of disk storage that has been reserved for some other use. Errors that prevent successful completion of an I/O operation are detected by a program

called Error Alert Control (EAC). EAC also responds to machine malfunctions. If overall system operation is threatened by the error, EAC notifies you with a message and carries out recovery procedures. It may restart the program that was executing when the error occurred; it may reload the entire system (Executive and SPARs) from disk; or it may terminate execution of the program that was executing when the error occurred. If the error doesn't affect overall system operation, EAC notifies you with a message. If the error has prevented successful completion of an input/output operation, a code is stored at a specified main-storage location. This code indicates the nature of the error.

When you build a coreload, you specify the kinds of recovery procedures to be carried out. Retrying an unsuccessful operation is often sufficient, because an error may be due to a temporary condition. You also have the options available to the system: terminate execution of the coreload, restart the coreload, or reload the system.

Various other features of MPX help you in isolating errors. You can have the system print (or "dump") the contents of an area of disk or main storage. The system can also trace a set of main-storage locations, recording all changes to the values in those locations. Tables and logs of many of the errors that occur, whether or not the errors are corrected, are maintained. Diagnostic programs available to the IBM customer engineer allow him to diagnose malfunctions of hardware units without taking them off the system.

17. Protection of programs and data areas

Various parts of MPX help you protect your programs and data from programming errors and from interference by unauthorized users.

You may want to protect an area of main storage so that no program can write in it. An MPX subroutine (STORP) allows you to do this. Areas used by the system are automatically protected in this way. The first and last two words of each partition are also protected in this way, to keep a program from overlapping the border of the partition in which it is being executed.

Programs are protected from errors in other programs by the recovery techniques described under heading 16. An error that causes execution of one program to be terminated or restarted doesn't affect the execution of other programs.

When you're using the Disk Management Program to reserve space on a 1316 disk pack, delete information from a 1316, or copy a 1316, you can use a system of label checking provided by MPX to make sure you're operating on the correct disk pack before carrying out the operation.

Various checks are built into the programming support for the 1800/2790 system to prevent operators from entering incorrect or unauthorized data. When you set up your transactions, you can specify that data entered is to be checked in various ways; for example, you may require that a data entry contain only numeric characters. You can also specify that the system is to check the identity of the person entering the data, in any of several ways, before accepting it.

The 1800 may have communications adapters set up so that it is possible to establish a connection with the 1800 by dialing a telephone. In this case, a system of identification sequences allows the 1800 to check the identity of another computer before carrying out any exchange of data.

18. The ability to leave out parts of the system that you don't need

Most 1800 installations don't require all the resources of MPX. You might not, for example, plan to do any communications with other computers; you might not plan to use a plotter; or you might not use any SPAR coreloads. So that you can include only the resources you require, the operating system consists of a set of independent programs. These programs can be arranged and linked together in many combinations to form operating systems tailored to the needs of different applications. Thus you don't have to pay, in main storage or execution time, for functions you're not interested in.

Your operating system isn't arranged before delivery. Instead, you receive copies, on disk, of all the programs you can include in your MPX system. During the system generation process, you construct the kind of operating system you need.

19. The ability to grow without disruption

As your needs change, you might want to add some of the optional features of the 1800 system. For example, you might want the 1800 to start controlling two processes instead of one. You might want to add magnetic tape units to your system, or you might want to establish a communications system with other computers.

It's important that system growth not be disruptive. MPX is designed so that new features can be added with a minimum of lost time and reprogramming.

One reason MPX can do this is the construction of the system, which was discussed under heading 18. The system is a set of independent programs. You can add one or replace one in a new system generation without changing the operation of the rest of the system.

Another way the operating system helps ensure growth without disruption is by adherence to standard methods of programming. These standards are used by all the MPX programs, and you can use them in your programs. These include data formats, ways of linking programs, and ways of communicating between programs. These standards help ensure that your programs are compatible with all the MPX programs, no matter what system configuration you choose in the future. They're described in the MPX Programmer's Guide, Order Number GC26-3720.

Chapter 3: How MPX is Organized

You can define two kinds of MPX systems: one is a real-time system, which can respond to real-time events and, if you wish, can also do background processing. The other is a batch-processing system, which has no real-time capabilities. A batch-processing system executes programs one after the other as they are entered into the 1800. It can do only the kinds of work that a real-time system handles as background processing.

A real-time system is controlled by a set of programs called the Executive, as discussed in Chapters 1 and 2. A batch-processing system is controlled by a set of programs called the Basic Operating Monitor (BOM). Either the Executive or BOM, whichever is controlling the system, must be in main storage at all times.

You receive BOM on disk as part of your original MPX system. If you're defining a real-time system, you use BOM to supervise the building of the Executive. Many of the programs from BOM are incorporated into the Executive.

How BOM is Organized

The layout of BOM is shown in Figure 7. As you can see from the figure, BOM begins at address 0 in main storage. It is divided into two main parts, the Executive I/O and the BOM program set.

EXECUTIVE I/O

These are the main parts of the Executive I/O part of BOM:

1. Fixed Area--an area that MPX system programs use to communicate with each other.

2. Trap Area--an area used to preserve, or "trap," the contents of a main-storage area for use in error diagnosis.

3. INSKEL COMMON--an area used for communications among your programs.

4. Device tables--a table of information about each input/output device that's part of the system.

5. Input/output control subroutines and shared subroutines--Input/output control subroutines for the devices that MPX itself uses (the 1053 printer and the disk drives), and subroutines that are shared by all coreloads and system programs. This area can also contain control subroutines for other devices, if you choose to put them here.

6. Error Alert Control (EAC, described in Chapter 2).

7. BOM utility programs--Programs that carry out reloads, dumps, and traces (described in Chapter 2).

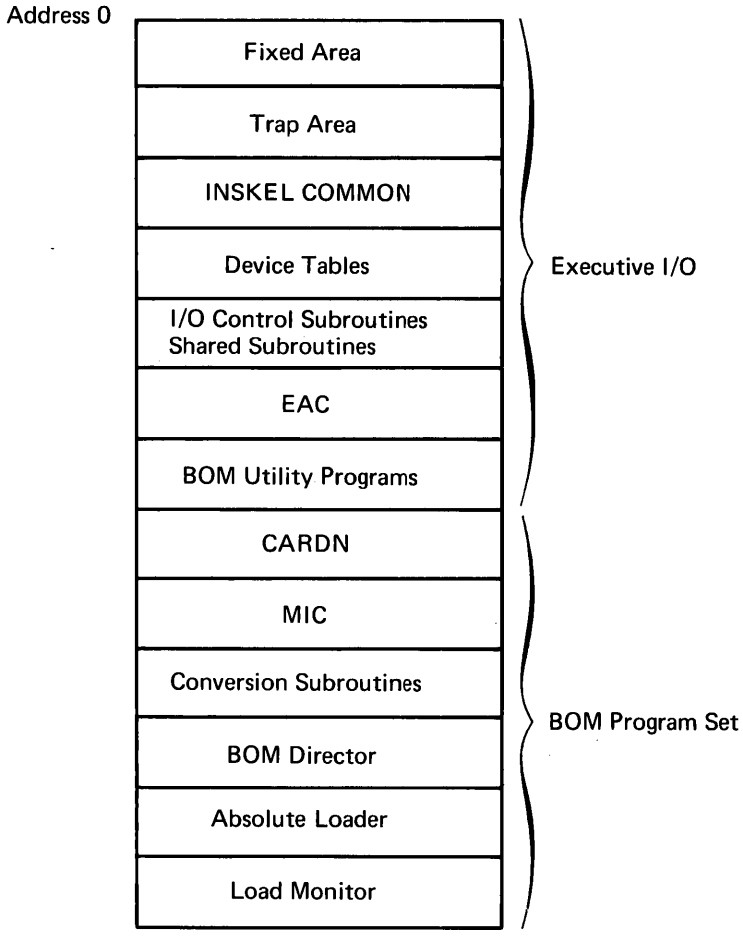


Figure 7. The Basic Operating Monitor

BOM PROGRAM SET

The BOM program set consists of the following programs:

1. CARDN--the input/output control subroutine for the 1442 card read punch.

2. Master Interrupt Control (MIC, discussed in Chapter 2).

3. Conversion subroutines--programs that convert between card and main-storage representations of data.

4. BOM Director--a program that initiates BOM and directs the operation of the absolute loader and the load monitor.

5. Absolute Loader--a program that allows you to load machine-language programs from cards into main storage for execution, and to store programs and data on disk.

6. Load Monitor--a program that initializes a batch-processing MPX system for execution.

How the Executive is Organized

During system generation, while your system is under control of BOM, you can build an Executive to supervise operation of a real-time system. The layout of the Executive is shown in Figure 8.

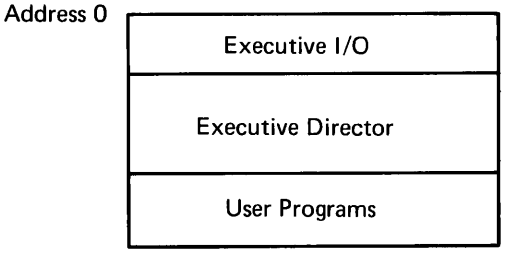


Figure 8. The Executive

The Executive I/O area is identical to its BOM counterpart.

The Executive Director contains four programs that were discussed in Chapter 2: MIC, ITC, PSC, and TSC. It also contains the level work areas (also described in Chapter 2) and various system tables.

After the Executive Director can come any of your programs that you have elected to include in the Executive.

How Main Storage is Organized

In a batch-processing MPX system, main storage is divided into two parts, BOM and VCORE. BOM contains the control programs that must be in main storage at all times, and VCORE is the partition where all other programs are executed, including programs that you write, and some of the MPX programs, such as the FORTRAN Compiler.

In a real-time MPX system, main storage is divided into three parts:

- The Executive, which contains the control programs that must be in main storage at all times.

- Some number of partitions (from 0 to 23) where programs are executed in response to interrupts.

- VCORE, the background partition.

Figure 9 shows main-storage layouts for real-time and batch-processing MPX systems.

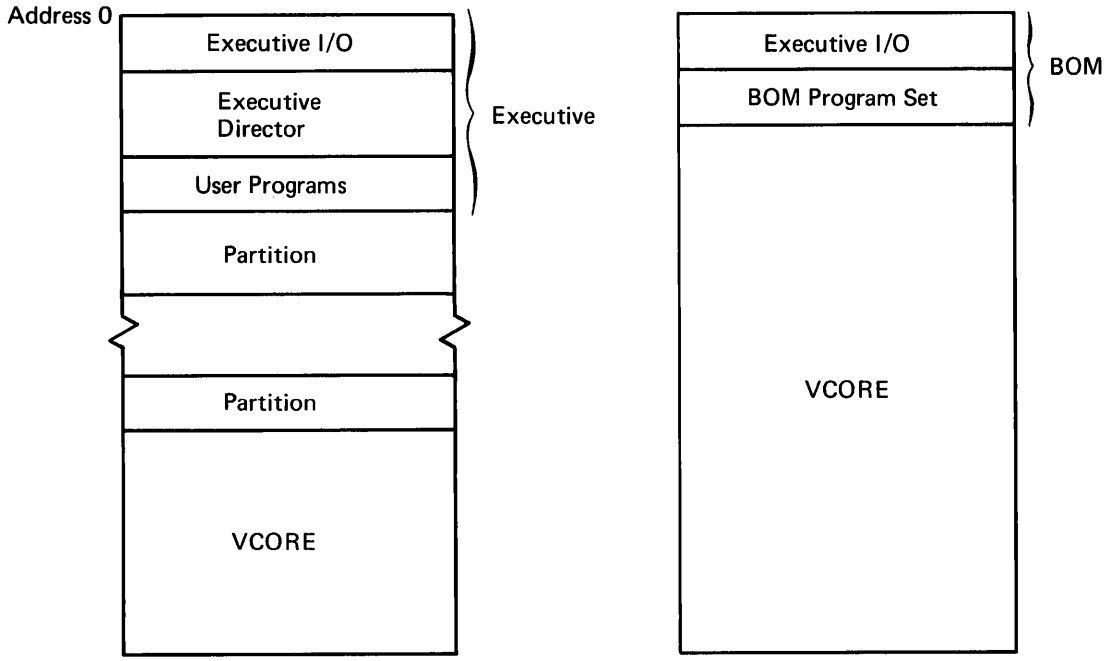


Figure 9. Real-Time and Batch-Processing MPX Systems

How the Batch-Processing Monitor is Organized

The Batch-Processing Monitor (BPMON) contains the MPX programs that are executed as background jobs. The Batch-Processing Monitor Supervisor controls the transition between different background jobs, including programs you write and the other programs that are part of BPMON.

The BPMON programs are shown in Figure 10. Their functions were discussed in Chapter 2. All the BPMON programs reside on disk, and are read into VCORE when they are to be executed.

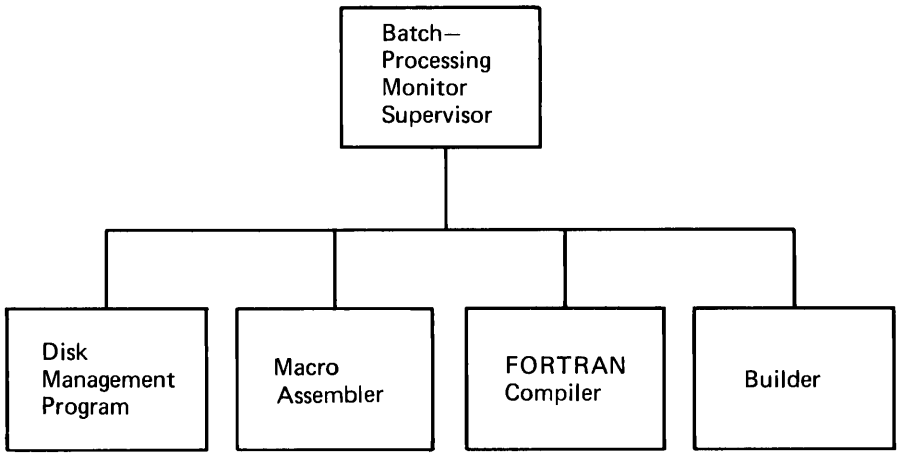


Figure 10. The Batch-Processing Monitor

How the System Residence Disk is Organized

Copies of all programs defined for an MPX system, together with storage areas and tables used for special purposes, must be kept together at all times on part of a disk. This disk is called the system residence disk.

The layout of the system residence disk area is shown in Figure 11.

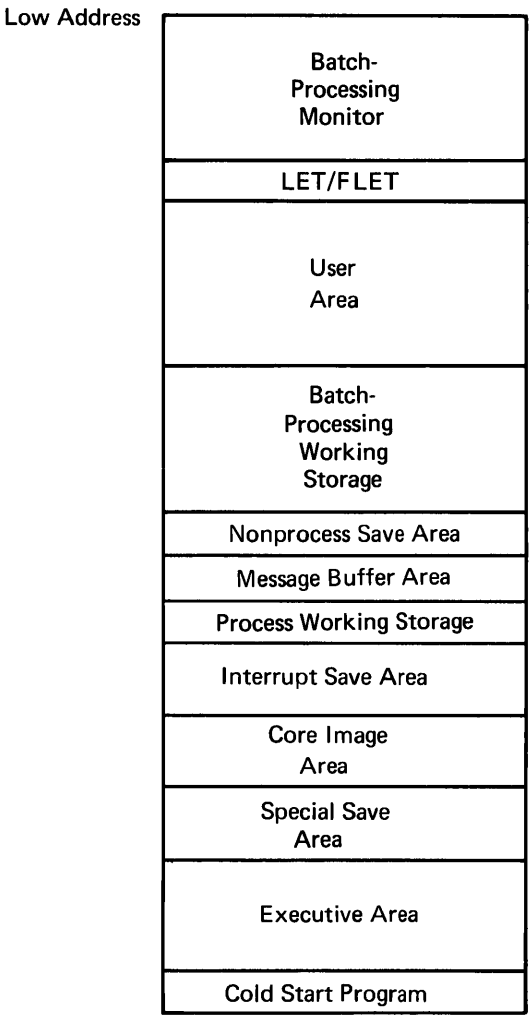


Figure 11. MPX System Residence

The parts of the system residence area for a real-time system are:

1. The Batch-Processing Monitor (already discussed in Chapters 2 and 3).

2. LET/FLET--the Location Equivalence Table and the Fixed Location Equivalence Table; tables used to keep track of all the programs and subroutines stored on a disk.

3. User Area--programs that can be executed in different places in main storage and that can be stored in different places on disk. These include the MPX subroutine library and your programs that have not yet been built into coreloads.

4. Batch-Processing Working Storage--an area used for temporary storage of data during the execution of background-processing or batch-processing jobs.

5. Nonprocess Save Area--an area used to save the contents of VCORE when a background program is suspended so that a coreload with a higher priority than that of BPMON can be executed in VCORE on the basic level.

6. Message Buffer Area--an area used to hold messages before they are printed.

7. Process Working Storage--an area used for temporary data storage during the execution of real-time programs.

8. Interrupt Save Area--an area that you can use to save the contents of VCORE when an interrupt to be serviced in VCORE on the level of the interrupt occurs.

9. Core-Image Area--programs that can be executed in only one place in main storage, and that are assigned to one fixed disk location. They can be accessed faster than relocatable programs, because their permanent address can be kept in the calling program.

10. Special Save Area--an area used when a CALL SPECL statement is used. For information on CALL SPECL, see the MPX Programmer's Guide, Order Number GC26-3720.

11. A copy of the Executive.

12. The Cold Start Program, which loads the Executive into main storage and turns control over to it.

How a Coreload is Organized

The organization of an MPX coreload is shown in Figure 12. The area labeled COMMON is an area that can be used by all coreloads executing in the same partition to communicate with each other. Restrictions on the use of this kind of COMMON were discussed under heading 7 in Chapter 2.

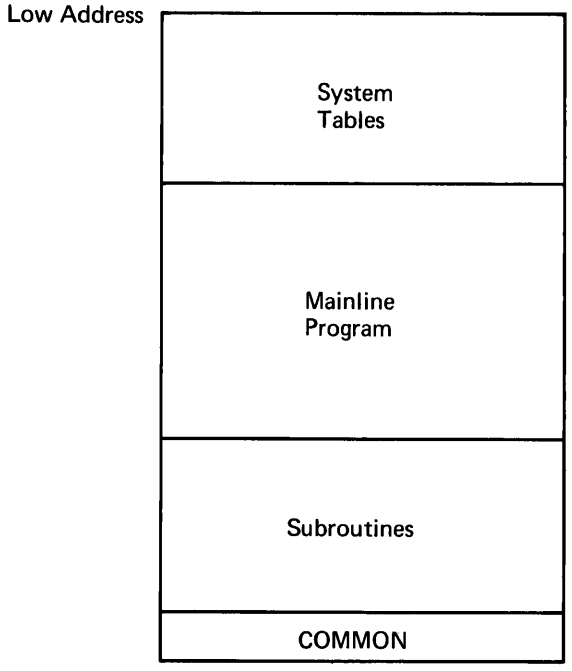


Figure 12. A Coreload

This page intentionally left blank.

IBM / **Technical Newsletter**

File Number 1800-36 (MPX Version 3)

Re: Order Number GC26-3718-4

This Newsletter Number GN26-0615

Date October 30, 1970

Previous Newsletter Numbers None

IBM 1800 MULTIPROGRAMMING EXECUTIVE OPERATING SYSTEM INTRODUCTION

This technical newsletter provides replacement pages for the IBM 1800 Multiprogramming Executive Operating System Introduction, Order Number GC26-3718-4. Pages to be inserted and/or removed are listed below.

Front Cover	17,18
v,vi	21,22
7,8	34.1-34.8
11,12	

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

This technical newsletter adds to the manual an appendix that contains general information about the pulse count and external alarm features for the 1800/2790 Data Communication System. These features of the 2790 system have been announced but not yet released.

Minor corrections and additions to the manual have also been made.

File this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications, Dept. D78, San Jose, Calif. 95114

PRINTED IN U.S.A.

File Number 1800-36
Order Number GC26-3718-4

IBM Systems Reference Library

IBM 1800 Multiprogramming Executive Operating System Introduction

This manual is an introduction to the IBM 1800 Multiprogramming Executive (MPX) Operating System. Intended for new and prospective 1800 installation managers, programmers, and operators, it assumes only a knowledge of a few process-control and data-processing terms.

Topics discussed are: what the 1800 system is used for, what some of its capabilities and features are, what one might need from an operating system, what MPX does about these needs, and how MPX is organized.

An appendix describes the general characteristics of the pulse count and external alarm features for the 1800/2790 Data Communication System. It is assumed that the reader has a general understanding of the 2790 system as described in the 1800/2790 Data Communication System Programming manual, Order Number GC26-3732. The features described in this appendix have been announced but not yet released.

Fifth Edition (June, 1970)

This is a major revision of, and renders obsolete, Form C26-3718-3 and Technical Newsletters N26-0598 and N26-0602. The entire manual has been rewritten to incorporate changes made to MPX in Versions 2 and 3.

This edition applies to Version 3, Modification 0 of the IBM 1800 Multiprogramming Executive Operating System, and to all subsequent versions and modifications unless otherwise indicated in new editions or Technical Newsletters. Changes may be made to the specifications in this manual at any time; before using this manual in connection with the operation of IBM systems, consult the latest SRL Newsletter, Order Number GN26-1800, for the editions that are applicable and current.

Forms for reader's comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114.

For copies of this or any other IBM publication, see your IBM representative or call your local IBM branch office.

Contents

assembler language, but it might also require more computer time and more storage space.

You can decrease the lengtn of your assembler-language programs by defining macro instructions. A macro instruction is an instruction that is written like an assembler-language statement. When the Macro Assembler encounters a macro instruction, it processes a sequence of assembler-language statements that you have specified. You need to specify such a sequence only once. After that, you can specify that the sequence is to be processed by issuing the macro instruction.

The definition and use of a sample macro instruction are shown in Figure 2. Once the macro definition has been stored, issuing the macro instruction would have the same effect as issuing the FORTRAN statement or the sequence of assembler-language statements also shown in Figure 2.

After a program has been written and assembled or compiled, you may want to run (execute) it immediately, store it for future execution, or both.

Programs are executed by a machine unit called a processor-controller. This unit includes a central processing unit (CPU), which carries out aritnmetic and logical operations, and main (core) storage, where programs are executed.

There normally isn't room in main storage for all your programs to be there all the time. Most programs are stored in disk storage and then read into main storage when they are to be executed.

MPX allows you to manage storage and execution of your programs in several different ways. You tell the system to store and execute programs by issuing control statements that activate various programs within MPX.

Com	Statement Number	Cont	
1	2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
			A = B + C * (D(I) - E) / F

FORTRAN Statement

Label	Operation	F	T	
21 25	27 30	32	33	35 40
	LDX	I	1	SGT1
	LD	L	1	D
	S	L		E
	M	L		C
	SLT			16
	SRT			16
	D	L		F
	A	L		B
	STO	L		A

Assembler-Language Statements

```
01100101100000000000000000000111110
11000101000000000000000000000111000
10010100000000000000000000000111000
10100100000000000000000000000100010
00010000010010000
00011000010010000
10101100000000000000000000000111010
10000100000000000000000000000100000
11010100000000000000000000000011110
```

Machine-Language Statements

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55
	SMAC			
	CALC			SGT,AD,SUB,MULT,DIV,MORE,SPC
	LDX	I	1	SGT
	LD	L	1	AD
	S	L		SUB
	M	L		MULT
	D	L		DIV
	A	L		MORE
	STO	L		SPC
	MEND			

Macro Definition

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55
	CALC			SGT1,D,E,C,F,B,A

Macro Instruction

Figure 2. FORTRAN, Assembler-Language, and Machine-Language Statements

Every real-time MPX system must include the Executive and at least one partition. The number and size of other partitions depend on your needs.

Multiprogramming allows programs to be executed faster--and thus provides faster response to real-time events--by letting different programs use different resources of the system at the same time.

Any program, at any given time, requires only a fraction of the total resources of the system. For example, while a program is using the central processing unit to do addition or subtraction, it isn't using a printer; while a program is using a disk drive, it isn't using the CPU. Under MPX, different programs can use the CPU, printers, and disk drives, as well as other input/output devices at the same time, thus shortening the execution times of some or all of the programs.

Let's look at an example of the advantages of multiprogramming. Suppose programs A, B, and C are being executed in a system without multiprogramming. Program A reads some information from disk, operates on it, and prints out a report. Program B performs some computation, prints a message, performs some more computation, and writes the result on disk. Program C performs some computation, reads some information from disk, performs some more computation, and writes the result on disk.

Figure 3 illustrates the sequence in which various operations would be performed when the three programs are executed one after the other. Notice that while any one part of the system, such as a disk drive, is being used, the other parts, such as the CPU, are idle.

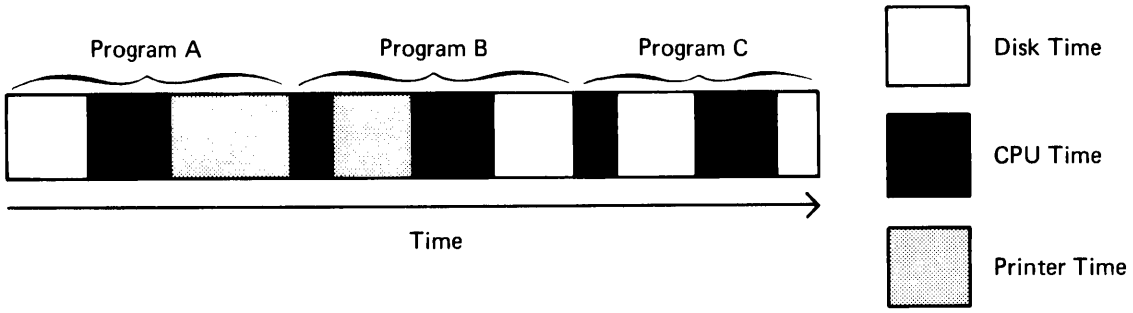


Figure 3. Sequential Execution of Programs

Figure 4 shows the sequence in which the same functions might be carried out under MPX. Note that the three resources involved (CPU, disk drive, printer) are in some cases all used at the same time. Note also that Program A is completed just as soon, and Programs B and C are completed sooner, than they were in the system without multiprogramming.

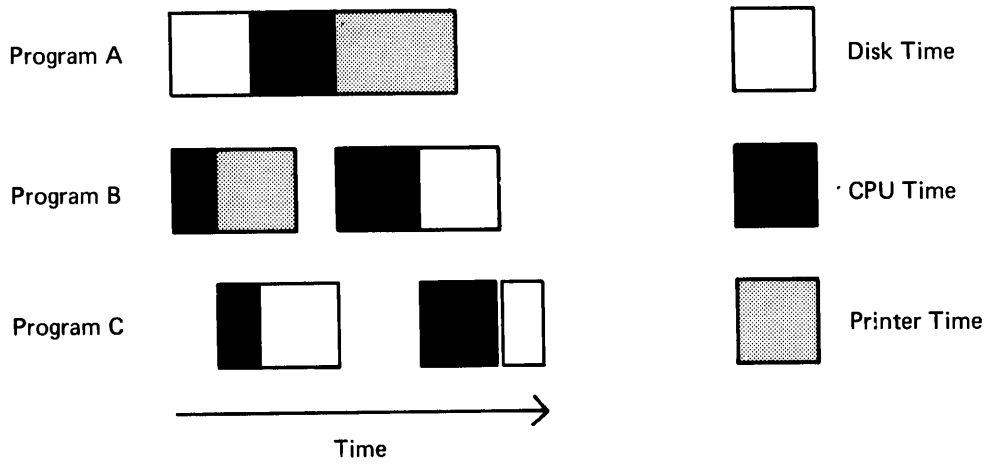


Figure 4. Concurrent Execution of Programs

INTERRUPTS

In order that MPX respond to a real-time event, it must have some way of knowing that the event has taken place. To accomplish this, each 1800 can be connected so that physical events cause interrupts. An interrupt is the recognition of an event that alters the sequence of program execution by causing execution of a specific program. When you set up your system, you specify which program is to be executed in response to each interrupt. The program is said to service the interrupt.

When the interrupt occurs, a physical indicator is set. The 1800 uses the indicator to determine the source of the interrupt. Then MPX executes the program that has been specified to service that interrupt.

Interrupts can, as we've said, be caused by real-time events. For example, if an 1800 were controlling the testing of resistors as they were manufactured, an interrupt might be generated every time the testing mechanism finished testing a good resistor. A different interrupt would be generated when the resistor being tested was found to be defective. Different programs would be executed to service the two interrupts. In an actual process, the two programs might cause the resistors to be moved to different locations.

An interrupt is also generated each time an input/output device finishes an operation. Programs within MPX are executed in response to those interrupts. Interrupts can also be generated by programs. For example, in the resistor-testing application described above, an interrupt was generated when a defective resistor was encountered. The program executed in response to that interrupt might move the defective resistor to a specified spot, update some stored production data, and generate another interrupt. The program servicing this second interrupt could begin the testing of another resistor.

MPX can distinguish up to 744 different interrupts. The actual use of interrupts at your installation depends on your needs.

instructions in which you specify the location of the data to be transferred, where it's to be sent, how much data is to be transferred, and what's to be done when the transfer is complete. For example, a call to the input/output control subroutine for the card read punch might specify that the contents of one card in the card read punch are to be read into a specified location in main storage. A call to the input/output control subroutine for the 2311 disk storage drive might specify that 29 characters beginning at a certain location in main storage are to be copied onto a disk pack, beginning at a certain location on the disk pack.

The following MPX input/output control subroutines control data-processing I/O devices:

- BULKN--handles output to and input from the 1810 disk storage unit and sections of 2311 disk storage drives that are treated like 1810s.
- CARDN--handles the reading and punching of cards on the 1442 card read punch.
- FILEN--handles input to and output from the 2311 disk storage drive.
- MAGT--handles input to and output from the 2401 and 2402 magnetic tape units.
- PAPTN--handles input from the 1054 paper tape reader and output to the 1055 paper tape punch.
- PLOTX--handles output to the 1627 plotter.
- PRNTN--handles printing and line control on the 1443 printer.
- TYPEN, WRTYN--handles input from the 1816 printer keyboard and output to the 1816 and the 1053 printer. (TYPEN and WRTYN are two names for the same subroutine.)

The following input/output control subroutines control process I/O devices:

- AIRN--handles input of data to main storage from a random set of analog input points.
- AISN, AISQN--handles input of data to main storage from a sequential set of analog input points. (AISN and AISQN are two names for the same subroutine.)
- DAOP--handles output to digital and analog output devices.
- DIEXP--handles reading of a group of digital input points and storing of the values in a special format.
- DINP--handles reading and checking of several groups of digital input points.

The following input/output control subroutines control communications I/O devices:

- BSCIO--handles input from and output to the computers and terminals attached to the 1800 by the communications adapter.
- 2790 Input/Output Control Subroutine--handles input from the 2791 and 2793 area stations, 2795 and 2796 data entry units, and 1035 badge readers, and output to the 2791 area stations and 1053 printers attached to area stations.

To run MPX, you must have at least one 1442 card read punch, one 1053 printer, and one disk drive (2311 or 1810 Model A). If you don't have 2311s, then the 2790 system requires an 1810 Model A2 or B2. The 1053 printer and the disk drive are used by some of the MPX system programs, so the input/output control subroutines for those devices must be in the Executive. You can put CARDN in the Executive or a coreload.

Of the input/output control subroutines for other devices, you need to include only those for devices you are using in your system. Any of these other input/output control subroutines, except PRNTN, can be put in either the Executive or a coreload. If PRNTN is included in the system, it must be placed in the Executive.

Table 1. Minimum and Maximum Machine Configurations

Device	Number in Maximum System	Number in Minimum System
2311 Disk Storage Drive	8	1 or 1 (Model A1 or B1)
1810 Disk Storage Unit	1 (Model A3 or B3)	
1442 Card Read Punch	2	1
1816 Printer Keyboard	8 (including up to two 1816s)	1
1053 Printer		
1443 Printer	1	0
1627 Plotter	1	0
1054 Paper Tape Reader	1	0
1055 Paper Tape Punch	1	0
2401/2402 Magnetic Tape Unit	2	0
Communications Line Adapter	8	0
2790 Loop Adapter	2	0

Table 2. Maximum 2790 Loop Configuration

Device	Number in Maximum Loop
2791 Area Station or 2793 Area Station	100
2795 Data Entry Unit	32 per 2791 Model 1 or 2793 area station; maximum of 1024
2796 Data Entry Unit	
1035 Badge Reader	300 (3 per 2791 Model 1 area station)
1053 Printer	100 (1 per 2791 Model 1 or 2793 area station)
User-Supplied Input Device	100 (1 per 2791 Model 1 area station)

DISK UTILIZATION PROGRAMS

The Disk Pack Initialization Program (DPIP), the BOM Disk Write Addresses Program (BDWAP), and the Disk Management Program (DMP) are programs that give you further assistance in the use of disk storage.

DPIP allows you to test a 1316 disk pack (used with 2311 disk storage drives) for defects and initialize it for use. BDWAP initializes a 2315 disk cartridge (used with 1810 disk storage units). By issuing control statements that activate parts of DMP, you can carry out such functions as storing programs and data on disk, deleting programs and data previously stored, copying disk packs and cartridges, and reserving disk space without actually storing anything.

9. Accessibility of subroutines that are used by many programs

Some subroutines may be used as part of the execution of many different programs. Storing a copy of such a subroutine with each coreload that uses it would require extensive disk space. The partitions where the coreloads are executed would also have to be large enough to accommodate the subroutine along with the rest of the coreload. You can save space by placing such subroutines in the Executive. Then they can be accessed by a program executing in any partition.

10. The ability to interrupt a subroutine, use it, and then complete the interrupted execution properly

If a program is being executed on an interrupt level, and a higher-level interrupt occurs, execution of the program is suspended until the higher-level interrupt is serviced. However, a problem arises when a subroutine in the Executive is interrupted in this way and the program servicing the higher-level interrupt calls the same subroutine.

Let's look at an example. Say you have two communications adapters, one of which generates level 1 interrupts and the other of which generates level 2 interrupts. The input/output control subroutine BSCIO services all interrupts for both communications adapters. Say BSCIO is transmitting or receiving data with the level 2 communications adapter when an interrupt from the level 1 communications adapter occurs. The first execution of BSCIO is suspended, and another execution of BSCIO is begun to service the level 1 interrupt. If BSCIO were a conventional subroutine, the partial results of the first execution would be lost. The communications that had taken place during the first execution would have to be repeated.

To prevent this kind of problem, MPX subroutines can be written to be reentrant. When execution of a reentrant subroutine on a given interrupt level is interrupted, all the results accumulated so far are saved in an area called the level work area, and execution can later be resumed at the point where it was interrupted. There's a level work area for each interrupt level in the system.

All MPX system subroutines that may be called from more than one level are available in reentrant form. This means they can be called from different interrupt levels with no loss of partial results. Many of the system subroutines are available in both reentrant and nonreentrant forms. The reentrant forms generally require less main storage, but require more execution time than the nonreentrant forms. You can choose the form that better meets your needs.

11. The ability to perform some tasks on a timed basis

Assume that your 1800 is controlling the packaging of peanut butter. You might want the 1800 to weigh a finished jar of peanut butter every two seconds, check the flow of salt into the peanut butter every two minutes, and print an inventory report every two hours.

Timers built into the 1800 system and into MPX allow you to do things on a timed basis. A timer is a device that generates an interrupt every time a specified interval of time elapses. Three physical timers are built into the 1800 hardware. Two of the physical timers are available for your use; MPX uses the third to keep track of the time of day and

Appendix A. Pulse Count and External Alarm for 1800/2790 Data Communication System

The information in this appendix is to be used only as a planning aid, because the features described are not available at this time. This information is subject to modification between announcement and release of the external alarm and pulse count features of the 1800/2790 Data Communication System.

This appendix briefly describes the pulse count adapter for the 2793 area station, the external alarm for the 2791/2793 area station, and MPX programming support to be provided for these features.

Information describing the 2790 Data Communication System units and MPX programming support is in the IBM 1800 MPX Operating System, 1800/2790 Data Communication System Programming manual, Order Number GC26-3732. The descriptions in this appendix assume that the reader has a general understanding of the 2790 system as described in the above manual.

Pulse Count

The pulse count adapter permits the recording of events from up to 63 mechanical contact closures (contact sense points). A pulse count adapter can be attached to each 2793 area station on the 2790 loop.

Each counter can record from 0 to 29,999 events and can be read, set to a value, or tested by use of the specific device address assigned to each contact sense point.

A visual readout attachment can be located at the area station. This attachment will provide a means for displaying the value in each of the contact sense counters. Operation of this attachment is independent of the 2790 Data Communication System.

External Alarm

The external alarm feature provides a means of alerting anyone in the vicinity of a 2791 Model 1 or 2793 Model 1 area station that a condition you have designated exists. The alarm can be a bell, light, or any device you provide that can be operated by the momentary closure of contacts. An area station having this feature must also have a 1053 printer adapter. The alarm is activated by the receipt of the character for bell at the area station.

Programming Support for Pulse Count

The pulse count adapter for the 2793 area station is supported by the 1800/2790 MPX Data Communication System operating under a real-time MPX operating system.

The pulse count feature is supported by six new macro instructions and a subroutine. Four of the macro instructions are used to read the counters,

set them to values, and reset them to zero. The two remaining instructions will be used to test the status of pulse counters after a read, set, or reset has been performed.

The PULSE subroutine reads the counters, sets them to a predetermined value, and resets the counters from a coreload other than the 2790 SPAR coreload. The PULSE subroutine can be called from a FORTRAN or assembler-language program.

READC MACRO INSTRUCTION

This instruction reads from one to seven counters on loop 1 or 2 (m in the following example). The counters to be read are specified in groups according to area station number (000-127, aaa in the following example) and the counter number (1-63, nn in the following example). Up to seven counter groups may be defined in one macro instruction. A group may range from one area station with seven counters to seven area stations with one counter each.

The reset parameter, r, allows you to specify normal end (no reset), reset counter to zero, and reset on overflow condition (when counter exceeds 29,999). The r parameter can also be used to reset the count test bit. This bit is described under the RESET macro instruction.

The format of READC is:

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55 60
	READC			r,,m,,(aaa,,nn)

In a series of pulse count instructions, it will be necessary to specify the loop number and counter group parameters in only the first macro instruction. Subsequent pulse count instructions will pick up the loop number, area station, and counter numbers from the I/O buffer currently being processed. When parameters are omitted in the instructions, the commas that would normally precede omitted parameters must be shown.

The loop number and area station number can also be omitted from the pulse count instructions. The required information will be taken from the 2793 area station which initiated the transaction. That 2793 must have the pulse count adapter.

SETC MACRO INSTRUCTION

This instruction sets the counters specified in the counter group (aaa,nn) on loop m to the value specified in parameter vvvvv. Up to seven counter groups may be defined in each SETC instruction so long as the maximum of seven counters is not exceeded.

The format of SETC is:

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55 60
	SETC			v,v,v,v,v,,m,,(aaa,,nn)

The loop number and counter group(s) or the loop number and area station number may be omitted under the same conditions as described for the READC instruction.

PSETC MACRO INSTRUCTION

This instruction sets the counter specified in the counter group (aaa,nn) to a value that is 30,000 less than the value specified in the vvvvv parameter. This method of setting up the counter takes advantage of the overflow transmission code that is generated by the area station when a counter value exceeds 29,999.

Up to seven counter groups may be defined in each PSETC instruction so long as the maximum of seven counters is not exceeded.

The format of PSETC is:

Label		Operation		F	T		Operands & Remarks
21 25		27 30		32	33		35 40 45 50 55 60
		PSETC					v,v,v,v,v,,m,,(a,a,a,,n,n)

The loop number and the counter group or the loop number and area station number may be omitted under the same conditions as described for the READC instruction.

RESET MACRO INSTRUCTION

This instruction resets the counter specified in the area station group (aaa,nn) in a manner determined by the reset parameter, r. Up to seven counters may be specified in up to seven counter groups. The m parameter contains the loop number.

This instruction resets the counter and/or the count test bit. A count test bit is associated with each counter. The counters are so designed that any time the counter is incremented the count test bit is turned on. This bit can therefore be used to determine whether a particular counter has been operating since the last time the bit was reset.

The reset parameter determines how the counter and/or count test bit are reset. The options available include reset counter to zero, reset counter unconditionally, reset on overflow condition (when counter exceeds 29,999), and reset the count test bit.

The format of RESET is:

Label		Operation		F	T		Operands & Remarks
21 25		27 30		32	33		35 40 45 50 55 60
		RESET					r,,m,,(a,a,a,,n,n)

The loop number and counter group (aaa,nn) or the loop number and the area station number may be omitted under the same conditions as described for the READC instruction.

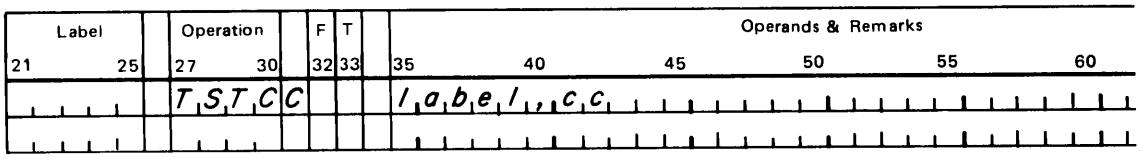
TSTCC MACRO INSTRUCTION

This instruction tests the condition code for each of the counters specified in the previous read, set, or reset operation.

Each time a read, set, or reset operation is performed on the counters, a code is placed in a specific word for each counter. This code indicates successful completion of the set or reset operation, the status of the count test bit (on or off), counter offline or area station bypassed, an overflow condition, and other error or status conditions.

If the condition code specified in the TSTCC instruction is equal to the code for any counter being processed, a branch is made to the instruction "label" in the transaction control list. If none of the counters being tested has the condition code specified in this instruction, the next instruction in the transaction control list is executed.

The format of TSTCC is:

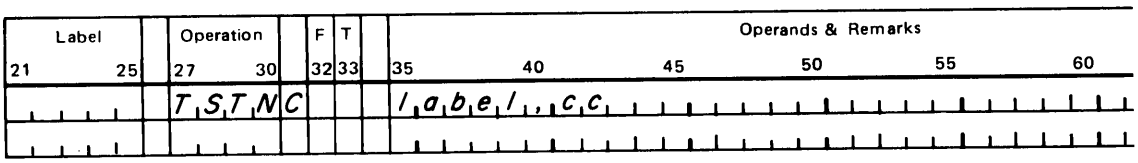


TSTNC MACRO INSTRUCTION

This instruction is similar to TSTCC because the same codes are tested. If the completion code cc specified in this instruction is not equal to any of the codes examined, a branch is made to the instruction "label."

This instruction also causes a branch to "label" if the data in the I/O buffer is not from a previous pulse count operation.

The format of TSTNC is:



PULSE SUBROUTINE

The PULSE subroutine will be able to read counters on a 2793 area station, set them to values, and reset them. The subroutine may be called from a FORTRAN or assembler-language program and can be executed in a coreload other than the 2790 coreload.

Assembler-Language Call:

Label	Operation	F	T	Operands & Remarks
21 25	27 30	32	33	35 40 45 50 55 60
	CALL			PULSE Call 2790 IOCR
	DC			LIST Address of I/O List
	.			
	.			
	.			
LIST	DC			*-* Link/Busy Indicator
	DC			Ø Exit Type
	BSS			4 System Reserved Words 1-4
	DC			*-* Completion Code
	DC			/XXXX Control Parameter
	DC			AREA I/O Area Address
	.			
	.			
	.			
AREA	DC			WDCNT Word Count and
	DC			/AANN Area Station/Counter Number
	DC			*-* Read/Set Completion Code
	BSS			3 Data/Set Value

The calling sequence is similar to that used for MPX input/output control subroutines. A description of the first six words of the I/O list is in the MPX Subroutine Library manual, Order Number GC26-3724, listed under, "Calling Sequences -- IOCRs." The remaining parameters are described here as they apply to the PULSE subroutine.

Completion Code

The completion code parameter indicates successful completion of the call or one of several error conditions. The error conditions are: a call was made to PULSE when the 2790 coreload was not in main storage, a call specified the number of a loop that was not active, and one of the counters has an error condition to report and the completion code must be checked for further error information. The completion codes are described later in this section.

Control Parameter

The first digit of the control parameter is the function code, which determines if a counter is to be read, set to a value, reset, or placed online or offline.

The second digit of the control parameter is the function modifier code. If the function code is a read or reset operation, the modifier code can specify normal end (no reset), reset counter to zero, unconditional reset, and reset overflow condition (counter exceeds 29,999). The function modifier code can also be used to reset the count test bit.

If the function code specifies an online or offline operation, the modifier code determines if the counters specified in the I/O area are to be put on line or taken off line.

The third digit of the control parameter specifies the priority of the call and the fourth digit specifies the loop number for the area stations being addressed.

I/O Area Address

This parameter contains the address of the I/O area that will be used to specify the counters, reserve space for the completion code, and provide a data area for the read and set operations.

I/O Area (AREA)

The first word, WDCNT, contains the word count for the entire I/O area. The value is determined by the number of counters defined in the I/O area. Each counter requires five words, and the maximum number of counters that can be specified is limited by the amount of main storage available.

The five words required for each counter are:

Area Station and Counter. This word contains the area station and counter number, specified in hexadecimal.

Read/Set Completion Code. This word is used by the system to return a status or error code that applies to this counter. These codes indicate successful completion of a set or reset operation, status of the count test bit, counter off line, unsuccessful retry of a read or set operation, area station bypassed, invalid counter number, or a power on condition indicating that power has just been restored to the area station.

Data/Set Value (Three Words). These words hold the five digits and sign that are returned by a read counter operation. If the operation is set counter, these words will be used to specify the value to which the counter is to be set.

Programming Support for External Alarm

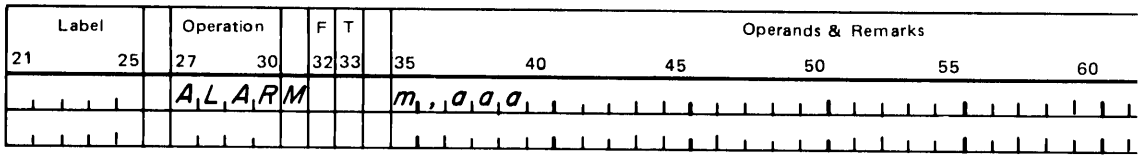
The external alarm feature for the 2791/2793 area station will be supported by the 1800/2790 MPX Data Communication System operating under a real-time MPX operating system.

A new macro instruction, ALARM, activates the alarm by transmitting the EBCDIC character for bell to the area station. The alarm will also be activated by the TMSG macro instruction if the message contains the EBCDIC character for bell.

ALARM MACRO INSTRUCTION

This instruction causes the external alarm contacts at area station aaa on loop m to be closed and activates the alarm. If the instruction is issued while the 1053 is busy, this message is queued in main storage until the 1053 is not busy.

The format of ALARM is:



The loop number and area station parameters can be omitted under the same conditions as described for the READC instruction.

2790 System Definition

The system definition macros for the 1800/2790 Data Communication System will be expanded to include provisions for defining the support required for pulse count and external alarm features.

Storage Requirements

Programming support for the pulse count feature will add approximately eight hundred words to the 2790 coreload. Each pulse count instruction will add approximately five words to the transaction control list.

This page intentionally left blank.

IBM **Technical Newsletter**

File No. 1800-36

Base Publ. No. GC26-3718-4

This Newsletter No. GN34-0047

Date November 1971

Previous Newsletter Nos. GN26-0615

IBM 1800 Multiprogramming Executive Operating System Introduction

This Technical Newsletter, a part of Version 3, Modification 2, of the IBM 1800 Multiprogramming Executive Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

Front cover, ii
iii–vi
34.1–34.8 (deleted)
Readers comment form

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This technical newsletter deletes from the manual an appendix that contained preliminary information about the pulse count and external alarm features for the 1800/2790 Data Communication System. With the release of these features in Version 3, Modification 2, updated material has been inserted in the *IBM 1800 Multiprogramming Executive Operating System 1800/2790 Data Communication System Programming* manual, GC26–3732.

Minor corrections to the manual have also been made.

Note. Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Systems Publications, Boca Raton, Florida 33432

PRINTED IN U.S.A.

File Number 1800-36
Order Number GC26-3718-4

IBM Systems Reference Library

IBM 1800 Multiprogramming Executive Operating System Introduction

Page of GC26-3718-4
Revised November, 1971
By TNL GN34-0047

Fifth Edition (June, 1970)

This is a major revision of, and renders obsolete, Form C26-3718-3 and Technical Newsletters N26-0598 and N26-0602. The entire manual has been rewritten to incorporate changes made to MPX in Versions 2 and 3.

This edition applies to Version 3, Modification 2 of the IBM 1800 Multiprogramming Executive Operating System, and to all subsequent versions and modifications unless otherwise indicated in new editions or Technical Newsletters. Changes may be made to the specifications in this manual at any time; before using this manual in connection with the operation of IBM systems, consult the latest SRL Newsletter, Order Number GN26-1800, for the editions that are applicable and current.

Forms for reader's comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, General Systems Division, Systems Publications, Department 707, Boca Raton, Florida 33432. Comments become the property of IBM.

For copies of this or any other IBM publication, see your IBM representative or call your local IBM branch office.

Preface: How to Use this Book

This manual is an introduction to the IBM 1800 Multiprogramming Executive (MPX) Operating System. It is intended for new and potential 1800 installation managers, programmers, and operators; the only prerequisite is knowledge of basic data-processing and process-control terms. The data-processing terms are explained in Introduction to IBM Data Processing Systems, Order Number GC20-1684.

The first chapter, "What the 1800 System Does," discusses kinds of applications in which the 1800 system is used, and the different kinds of work you can expect it to do.

The second chapter, "What MPX Does for You," discusses the programming functions that are carried out by the operating system. It describes needs that you might want the operating system to handle, and what MPX does about these needs.

The third chapter, "How MPX Is Organized," tells how the various parts of the system are put together and where they are located.

As you read this manual, you might want more detailed information about 1800 system physical units and MPX system programs. These books can be used for reference:

1800 System Summary, Order Number GA26-5920, which introduces the physical units that make up an 1800 system.

1800 Functional Characteristics, Order Number GA26-5918, which describes how the physical units work.

MPX Programmer's Guide, Order Number GC26-3720, which discusses MPX system organization and programming techniques.

MPX Subroutine Library, Order Number GC26-3724, which describes each of the system subroutines.

MPX Operating Procedures, Order Number GC26-3725, which tells how to generate, operate, and maintain MPX.

1130/1800 Assembler Language, Order Number GC26-3778, which tells how to use macro instructions and write programs in assembler language.

1130/1800 Basic FORTRAN IV Language, Order Number GC26-3715, which tells how to write programs in FORTRAN.

Communications Adapter Programming, Order Number GC26-3757, which tells how to write programs to carry out communications with other computers and terminals.

Binary Synchronous Communications--General Information, Order Number GA27-3004, which describes the programming conventions that govern communications between the 1800 and other computers and terminals.

1130/1800 Plotter Subroutines, Order Number GC26-3755, which describes MPX subroutines for controlling the 1627 plotter.

This page intentionally left blank.

Contents

READER'S COMMENT FORM

IBM 1800 MPX Operating System
Introduction

Order Number GC26-3718-4

Please comment on the usefulness and readability of this book, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you want a reply, be sure to give your name and address.

Name _____ Occupation _____

Address _____

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE. . .

Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 110
BOCA RATON, FLA
33432

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Boca Raton, Florida 33432

Attention: Systems Publications, Department 707

Fold

Fold

IBM 1800 (1800-36) Printed in U.S.A. GC26-3718-4

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM 1800 Printed in U.S.A. GC26-3718-4

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]