IBM Data Processing Techniques

Analysis

Elements of Run Time

This manual describes some of the basic concepts used to provide optimum computer processing of information. These include simultaneous operations, data channels, channel interference with the processor, buffers, process-limited runs, input/output-limited runs, run balance and run timing as a part of system selection.

Scientific

This edition, C20-1648-1, includes Technical Newsletter N20-0060-0.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

INTRODUCTION

The purpose of this manual is to describe some of the basic concepts used to provide optimum computer processing of information. It is assumed that the reader has some knowledge of computer systems, but limited experience in run timing and systems planning.

The concepts presented include simultaneous operations, data channels, channel interference with the processor, buffers, process-limited runs, input/output-limited runs, run balance and run timing as a part of system selection.

Familiarity with the concepts in this manual should enable the reader to (1) evaluate various methods of run timing and (2) review existing and planned programs to determine whether they are using the full capabilities of the data processing systems.

The performance of a data processing system depends on the equipment, the programs, and data characteristics. These are reviewed to provide answers to questions such as:

- What must be known about the data processing system, the data, and the programs to produce a reliable estimate of run time?
- What are concurrent operations?
- What is a data channel? Why are some data channels different from others?
- What is channel interference?
- What is a process-limited run? An I/O limited run? A balanced run?
- What are the effects of blocking and buffering on run time? How can these techniques be used to improve run balance?
- Why does an apparently balanced run sometimes require more or less time than originally estimated?
- What is the function of run timing in the larger activity of system selection?

WHAT IS A RUN?

A data processing run is an interaction of machines, programs, and data. It follows, then, that any procedure designed to time a run must take into account the machines that make up the data processing system, the programs that perform the processing, and the data involved in the run. The accuracy desired in the run time estimate determines the amount of information required about each of these elements.

THE FUNCTIONAL PARTS OF A DATA PROCESSING SYSTEM

A data processing system can be divided into four functional parts:

1. A control unit — to direct the sequence of operations and set up the proper circuits to perform the program
2. A storage unit — to hold instructions and data
3. A processor — to operate upon data
4. Input/output devices

The part of a computer that is called on most frequently is the storage unit. The way in which this part is used and organized has a great deal to do with the ability to perform simultaneous operations in a data processing system. The parts of a computer contend for the use of storage. The control unit must have access to storage for its instructions. The processor must periodically make reference to storage either to fetch or to store the data being processed. The I/O units require the use of main storage during read and write operations.

SIMULTANEOUS OPERATIONS

A wide variety of IBM data processing equipment is available with performance characteristics suitable for all applications. These performance characteristics include speed, capacity, and the ability to perform more than one operation at one time. To illustrate simultaneous operations, a data processing system may be considered capable of performing three operations: input, processing, and output. These may be reading a card (input), accumulating a total (processing), and printing some information from the card (output). A minimum data processing system, consisting of one processor and two input/output devices, could operate in four different ways, according to the equipment capabilities, the program, and the job being done:

- No simultaneous operations — the input operation must be completed before the processor can be used, and the printer must wait until the processing is complete. Operations are sequential.
- Processing done at the same time as one I/O operation, such as reading a card, then processing the card information while printing the information from the previously processed card.
- Concurrent input and output operations, followed by processing, such as reading a card and printing a line at the same time, then processing the card that was read.
- Concurrent input, processing, and output operations.

These four modes of operation are shown in Figure 1 using an input card file and a minimum data processing system. Again, the mode of operation depends on the capabilities of the data processing system, the programming techniques, and the job being done. It is apparent from Figure 1 that when several operations are performed at the same time, job time is reduced.

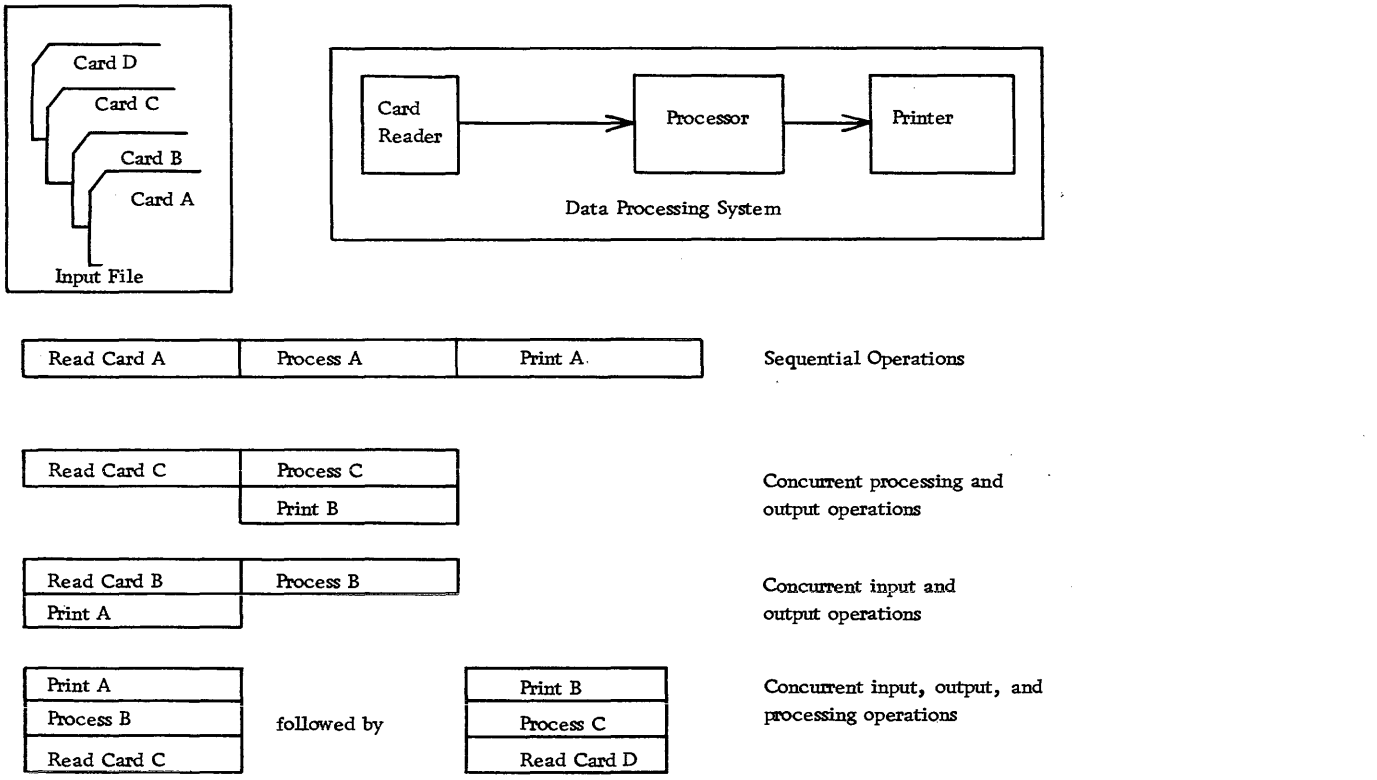

Figure 1. Concurrent and sequential operations

TIME UNITS

The units of time used in expressing computer operations are shown in Figure 2. The relationship between them can be seen in the following example:

750	nanoseconds is equal to
.750	microseconds, which is equal to
.000 750	milliseconds (ms), which is equal to
.000 000 750	seconds

Unit	Part of a Second	Decimal Equivalent
Second	One full second	1.0
Millisecond	One thousandth	.001
Microsecond	One millionth	.000 001
Nanosecond	One billionth	.000 000 001

Figure 2. Time units and decimal equivalents

IBM 407 SIMULTANEOUS OPERATIONS

A list-accumulate job on an IBM 407 Accounting Machine illustrates a data processing run with some simultaneous operations. The 407 has a card-reading mechanism (input device), a printing mechanism (output device), and a processor. Consider the following job specifications:

1. Read 1500 cards. Of these, 900 have an X-punch in column 80, and 600 do not have an X-punch in column 80 (input).
2. Accumulate 900 numbers, one from each card with an X-punch in column 80 (processing).
3. Accumulate 500 minor totals (processing).
4. Print 1500 detail lines, one from each card (output).
5. Print 500 minor total lines (output).
6. Print one final total line (output).

The rated speed of the 407 — that is, 150 cards per minute and 150 lines per minute — results in a run time summary as follows:

1500 read operations (1500 cards)	10.0 minutes
1400 processing operations (900 plus 500)	9.3 minutes
2001 printing operations (1500 plus 500 + 1)	13.3 minutes

Since the card reading, accumulating, and detail printing functions can be handled in the time required for card reading, these could be considered as simultaneous operations — input, processing, and output at the rated speed of the machine. A closer look, however, reveals that the actual simultaneous operations are (1) reading the card, (2) accumulating, and (3) positioning the printing mechanism. The printing operation follows, but is still within the rated speed. The total of 2001 printing operations includes 500 minor total lines, and one final total line. Printing each of these 501 lines is a non-simultaneous operation. The total job time is 13.3 minutes, the time required for 2001 printing operations, since the reading and processing operations were handled at the same time as 1500 of the printing operations.

If a 523 Gang Summary Punch is attached to the 407 as an additional output device, and the job changed to include punching 500 summary cards (one for each minor total), the run time is increased. Since the 523 requires 10.0 minutes for punching 500 summary cards, and the 407 must wait while the 523 is punching, the run time is increased to 23.3 minutes, the sum of the printing and punching output operations.

It is interesting to note that in both cases, the runs are I/O-limited: process time is less than the longest I/O time. By definition, a run is I/O-limited when a reduction in the time required for internal processing does not reduce the run time.

The 407-523 is a data processing system with limited potential for simultaneous operations. A system's potential for simultaneity is limited when the operation of one component of the system delays the operation of some other component, or when several activities of a system all make use of the same component.

A SEQUENTIAL DATA PROCESSING SYSTEM

The IBM 1401 is a good example of a sequential data processing system. To illustrate, assume the 407 run discussed previously, but with the input medium changed from cards to magnetic tape and an increased amount of internal processing. The job specifications are now:

1. Read 1500 eighty-character records from a 729 II tape unit
2. Process each record for 20 ms
3. Print 1500 detail lines
4. Print 500 minor total lines
5. Print one final total line

The functional summary is:

1. Read 1500 records at 14.52 ms each	.36 minutes*
2. Process 1500 records at 20.00 ms each	.50 minutes
3. Print 2000 lines at 100.00 ms each	3.33 minutes
Total	4.19 minutes

In keeping with the idea that the basic 1401 is a sequential or nonsimultaneous system, run time may be estimated as the sum of all of the functions that the system performs (read time plus process time plus print time), or about 4.19 minutes. This is a close enough estimate for all practical purposes.

The 1401 main storage must serve both the tape unit and the printer that are attached to the system. During a tape read, main storage is busy receiving each character from the tape unit as it is read. During the print cycle, main storage is busy with the transmission of data to the printer. However, since tape units and printers are mechanical devices, they do not operate instantaneously. These devices require some time for starting before data transmission begins and some time for stopping after data transmission ends. On the 1401, main storage is not busy during the printer or tape unit stop time, and is therefore available to other parts of the system.

Tables such as the one following show (1) what actually happens if the job is programmed in the most straightforward way and (2) the potential for concurrent operations that exists in the basic 1401 system. In the run there are two basic sequences of operations, one sequence in which it is necessary to print a minor total line and one in which it is not. The most frequent (no-minor-total) sequence is shown in Figure 3 and timed in this way:

Operation	Overall time required for operation	Next operation can start after	Cumulative time
1. Read tape	14.52 ms	12.42 ms	12.42
2. Process	20.00 ms	20.00 ms	32.42
3. Print detail line	100.00 ms	84 ms	116.42

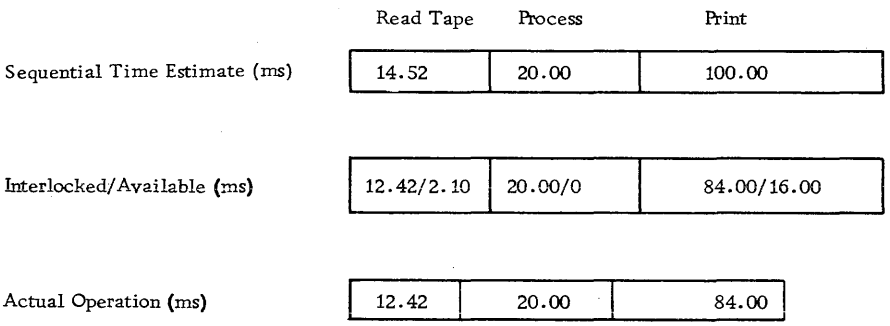

Figure 3. Preliminary estimate and actual timing of tape-to-print operation

*Using the 729 II read timing shown in Tape Input/Output Instructions, IBM Systems Reference Library (A24-3069), page 12, as follows:

Start time	10.50 ms per record
Data transfer time	1.92 ms per record (.024 per character)
Stop time	2.10 ms per record
Total	14.52 ms

When reading tape or printing, it is possible to begin the next operation a short time before the print or read operation is finished during the device stop time. The 729 II tape unit needs 2.1 ms of the total 14.52 ms to come to a stop after the last character of a record has been read. This time can be used to process the record just read. The 1403 Printer requires 16 ms of its 100-ms cycle to come to a stop. During this time main storage can be used for processing. In this example, the 1401 system can accomplish 134.52 ms of strictly sequential operations in an elapsed time of 116.42 ms. This sequence of operations occurs a thousand times during the execution of the program. Extending the elapsed time of 116.42 ms per record, a partial run time of 1.94 minutes is developed.

The other operation sequence, which requires a minor total to be printed, occurs 500 times and is illustrated in Figure 4. It is timed like this:

Operation	Time required for operation	Point of time in the operation when next operation can start
1. Read tape	14.52 ms	12.42 ms
2. Print minor total	100.00 ms	96.42 ms
3. Process	20.00 ms	116.42 ms
4. Print detail line	100.00 ms	200.42 ms

In this case, 234.52 ms of sequential activity is accomplished in 200.42 ms. Five hundred executions of the minor total sequence require 1.67 minutes. Adding the two partial run time estimates (1.94 minutes plus 1.67 minutes) gives a total of 3.61 minutes as against the original estimate of 4.19 minutes. For a run of this size, the difference seems unimportant. It is interesting, however, that the difference between the first and second estimates is a reduction in time of almost 14%. This is significant on a longer run. The difference on a four-hour job, for example, would be more than a half hour.

Activity	Interlock Time	+	Available Time	=	Total Time
Read Tape	12.42		2.10		14.52 ms.
Print Minor Total	84.00		16.00		100.00 ms.
Process	20.00		None		20.00 ms.
Print Detail Line	84.00		16.00		100.00 ms.
	200.42		34.10		234.52

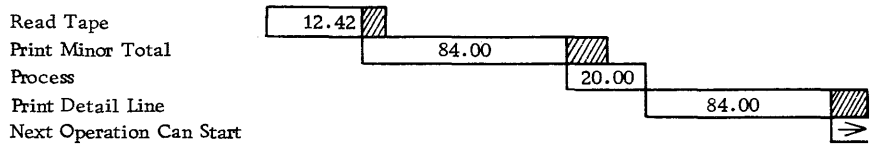

Figure 4. Minor total sequence

MACHINE FEATURES ENABLING SIMULTANEOUS OPERATIONS

It can be seen that the 1401 system is able to outperform a 407 on a read and print operation by almost four to one. The printer on the 1401 is four times faster than the 407 and, in the example, the 729 II tape unit achieves a reading rate 200 times that of the 407. A realistic comparison of internal processing speed is more difficult to make; however, a 407 can perform as many as 18 six-digit additions during its 150-ms cycle, while a 1401 can execute about 650 in the same time. This is a ratio of 36 to 1 in favor of the 1401.

These great differences in speed will not usually exist when comparing two computer systems. It is more likely that one computer will perform more efficiently than another because of its ability to do several things at the same time. This, in turn, depends largely on the amount of storage in the system and the way in which it is organized and used.

BUFFERS

A buffer is a small storage unit used to relieve main storage of the task of directly serving I/O devices. Generally speaking, buffering becomes more important as the difference in speed between main storage and the I/O device increases. For example, every time a line is printed, the 1401 main storage is kept busy for 84 ms transmitting data to the 1403 Printer. The 1401 main storage can transmit almost 87,000 characters per second; the 1403 can achieve a maximum data rate of 1320 characters per second. With this disparity in speed, a buffer (Print Storage option) for the 1403 allows better utilization of 1401 main storage.

Print Storage is a core storage buffer large enough to hold one line of information for 1403 printing. Print Storage can be filled from main storage in 2 ms. Then, during the actual printing operation, main storage is accessible to the other components of the system. The addition of Print

Storage to a 1401 system makes an additional 82 ms of processing time available on every print cycle taken by the 1403 and increases the potential for concurrent processor and printer operations from 16% to 98%.

CHANNELS

Logically, a channel is a data path from main storage to an I/O device. Every data processing system, therefore, must have at least one channel. The physical expression of a channel may take many forms. In the 1401, for example, all data movement in or out of the main storage is by way of registers that are integral parts of the processor. The processor in this case is also the data channel, which means that I/O operations cause processing to be suspended unless special features are installed to permit simultaneity.

In practice, however, the word "channel" has come to mean, more specifically, a group of components that provide a data path in and out of main storage without unnecessarily interfering with the operation of the processor. Channels vary widely in capability; the more powerful ones are actually small computers with the ability to address the main storage of the parent system. Such a channel will contain several registers capable of performing operations very similar to those that take place in the CPU, including a limited amount of arithmetic. Typical channel functions include:

1. Controlling I/O devices, usually through some intermediate I/O synchronizer such as the 2841 Control Unit.
2. Addressing locations in main storage for the purpose of fetching or storing data.
3. Incrementing or decrementing the channel's registers as the channel handles data transfers under count control.
4. Providing status information to the processor and the program control unit as I/O operations are started or completed.

CHANNEL INTERFERENCE WITH THE CPU

Channels designed to perform numerous operations for themselves interfere very little with the CPU. A channel of this kind contains and executes a stored program of its own, interfering with internal processing only when it requires a main storage cycle at the same instant as the processor. When such a conflict occurs, processing is suspended momentarily, and the channel takes precedence.

The more powerful channels are able to perform functions that the processor would otherwise be required to do. Data chaining (the rearrangement of data as it is fetched from or inserted into main storage) and command chaining (issuing a series of commands to an I/O device) are examples.

It is comparatively easy to predict the effect of channel operations on the processor of a machine like the IBM 1410, because:

1. All data is stored or fetched by the 1410 main storage one character at a time.
2. All internal operations are done by the 1410 in a storage-to-storage mode. Therefore, any arithmetic or logical operation requires the use of main storage.
3. The 1410 channels are limited in that they operate only upon contiguous locations in storage. and cannot by themselves execute a series of commands.

For each character transmitted by a 1410 channel, one storage cycle is required; and since the processor is always working with data in main storage, every storage cycle used by a channel is taken at the expense of internal processing. However, the 1410 channel, being unable to chain data or commands, never needs to refer to core storage for anything but its object data. Therefore, channel interference for any I/O operation is one storage cycle (4.5 microseconds) for each character in the block of data being read or written.

Such a precise calculation of channel interference is not always possible in other machines, notably the IBM System/360. Some of the variables are:

1. The width of the data path. The Model 40 has 2.5-microsecond access to two bytes of data. This means that the channel must make only one reference to core storage for each two bytes fetched or stored, thus reducing interference for data transfer to a little more than one-half a storage cycle per byte.
2. The amount of local storage available to the processor. If the processor can work from registers during some of its more lengthy arithmetic and logical operations, it may continue to compute for some time without requiring access to the main storage and thus reduce the number of times that the processor and a channel are contending for the same storage cycle.
3. The self sufficiency of the data channel. Some data channels not only use an occasional storage cycle, but also interfere with the CPU directly; that is, they use CPU circuitry to increment or decrement counts and to perform I/O operations.
4. The kind of I/O operation in progress and the type of I/O device on which the operation occurs. A System/360 channel is a stored-program device and, as such, takes its commands from the main storage just as the processor does.

If a series of commands is executed in a channel program, the channel must make references to main storage for each command. Command chaining

has the effect of reducing the capacity of the channel to transmit data. Some I/O devices, notably disk drives and data cells, are normally operated in this fashion. A single channel command, however, can control the transfer of more than 65,000 bytes of data from or to contiguous locations in storage. In addition to command chaining, the channel may be doing a data chaining operation. Data chaining is transmitting data in or out of noncontiguous areas of main storage. System/360 channels operate under a count control in the channel command word (CCW). When a channel has transferred the requisite number of bytes and data chaining is indicated, it must obtain the next CCW from main storage and continue the data transfer to or from a new storage area. This, like command chaining, increases the probability that the processor and the channel will contend for the use of main storage.

5. The kind of program being executed by the processor. Channel operations will interfere less with a program that makes extensive use of local storage — fixed- and floating-point registers — than with one that consists mostly of data movements and decimal (storage-to-storage) arithmetic.

A channel is not only more or less independent of the processor, but also independent of any other channels that may be attached to the system. Theoretically, as many I/O operations may be performed along with internal processing as there are channels on the system. With several high-speed data channels operating at once, however, it is possible to exceed the capacity of main storage to transmit data. To illustrate, consider an IBM 1410 with a storage cycle of 4.5 microseconds and 729 VI tape drives attached. A 729 VI has a data rate of 90,000 characters per second. With a storage cycle of 4.5 microseconds, data can be transferred at a rate of about 222,000 characters per second. Two channels of 729 VI tape can be operating simultaneously (90KC + 90KC = 180KC), but not three, since the aggregate data transfer rate of three 729 VI tapes drives is 270KC, a rate that requires a storage cycle faster than 4.5 microseconds.

Two channels on such a system would be feasible; however, with both operating at once, the processor would not have access to main storage for computing.

CHANNEL SIGNALS

If a data processing system is to achieve any substantial amount of simultaneity, some means must be found to make internal processing more or less independent of input and output operations. To illustrate why this is so, suppose that a system is capable of reading a record, performing internal processing, and writing a record, all at the same time. If the program is written in a strictly sequential manner (read record A, process record A, write record A, read record B, process record B, write record B, etc.), none of the potential simultaneity of the system is realized. Obviously, processing cannot begin until the record is read, and writing cannot begin (except to start an I/O device moving) until processing has been completed.

If an input unit, an output unit, and the processor are all to be employed simultaneously, each must be busy with a different record. It is then possible to be simultaneously writing record A (previously processed), processing record B (previously read), and reading record C (to be processed next). This is shown in Figure 5.

Internal processing and I/O operations are now independent, in the sense that each component of the system operates upon a different record, and no component is forced to wait for any other to complete a task.

This type of operation is highly efficient and therefore much to be desired, but it presents some difficulties. When I/O operations and internal processing are performed in an independent fashion, there is a need for a control program to schedule

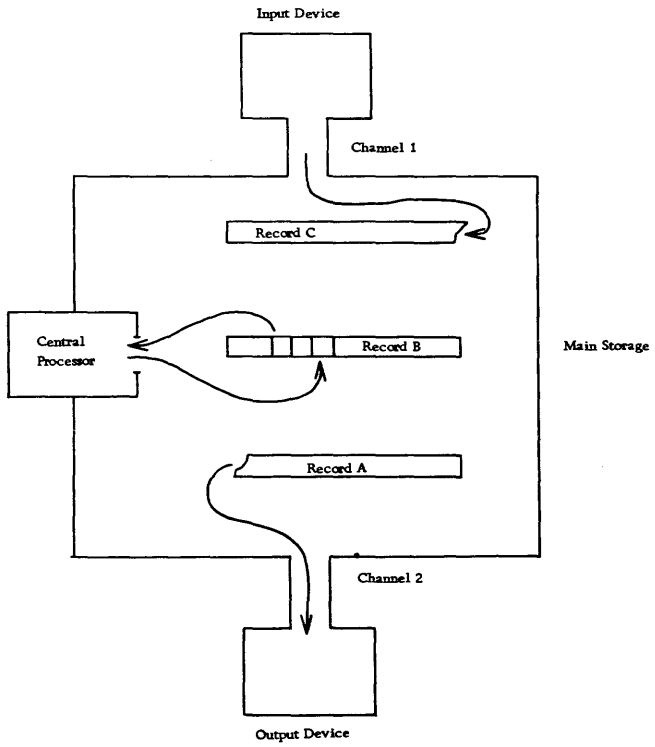

Figure 5. Simultaneous reading, writing, and internal processing

and coordinate the two types of activity. In general, the idea behind such a control program is that the data channels ought to be kept busy whenever possible; as soon as a channel completes one I/O operation, it should be set to work on another. In order to do this, some means of determining channel status is essential, and the most efficient way is to have the channels themselves keep the control program informed of the status of I/O operations.

Some machine features that facilitate maximum use of data channels are variously called "priority processing", "trapping", and "automatic interruption". Regardless of the name, the function is always the same: The data channel supplies signals to the control program when some event requires a service routine. The event is usually the completion of an I/O operation, although any of several error conditions might also require a service routine. The channel signal received by the control program causes a transfer of control to the appropriate routine for whatever condition has occurred. When the data channel has been serviced, control is passed back to the mainline program.

STANDARD PROGRAMMING TECHNIQUES THAT AFFECT SYSTEM PERFORMANCE

Data processing equipment requires good programming if its full potential is to be realized. This is particularly true for the portion of the program that coordinates input/output activity with the process routines. In general, when a run is timed, it should be assumed that such well known techniques as record blocking and use of multiple I/O areas will be employed. These techniques are standard features of IBM input/output control systems (IOCS).

In run timing the effect of operating systems should be considered. Every computer installation spends significant amounts of time in loading each program into storage before running it. Tape and disk labels are checked, and storage is cleared before loading programs. Checkpoints are taken and portions of programs are loaded, overlaid, and reloaded.

Programs may be relocated to make optimum use of storage. Also, to allow the computer to use any input/output device of the proper capacity that happens to be available, the assignment of data files to input/output units may be accomplished just before the program is executed.

All of these can be done automatically by an operating system, and the speed at which they can be done by computer is often much greater than the speed at which they can be done (if they can be done at all) with operator intervention. The fact remains, however, that they do involve significant amounts of time. Specific figures and the methods for calculating these factors cannot be covered in an introductory manual such as this. Where operating system features are used extensively, further refinement of run times is necessary.

MULTIPLE I/O AREAS IN MAIN STORAGE

If a data processing system is to accomplish simultaneous processing and I/O operations, it is necessary to have more than one record within main storage at all times. This practice is sometimes called "buffering", but this programmed buffering within the main storage of the computer should not be confused with buffering accomplished by machine features like the 1401 Print Storage feature mentioned earlier.

The use of multiple I/O areas in the main storage of a data processing system makes it possible for I/O and processing operations to be carried out in a logically independent and simultaneous fashion. Most IBM-provided input/output control systems furnish several programmer options as to the number and type of areas that may be specified.

RECORD BLOCKING

A certain amount of access time is associated with any mechanical I/O or storage device. Record blocking is a widely used technique for the minimization of access time. The idea is a simple one — the combination of several logical records into one physical record so that all may be read or written with only one access to a device. The IBM 7330 tape unit affords a good illustration of the use of this technique. This device requires an average of 20.7 ms for starting and stopping each time it is called upon by the program. The average time required to read or write a record at high density may be calculated by the formula 20.7 + .050N, where N is the number of characters in the record.* For an 80-character record, this is 24.7 ms.

Now, suppose that five 80-character logical records are grouped together to form one physical record on the tape. The file is said to have a blocking factor of five, or be "blocked five". This tape block of 400 characters can be read or written in an average time of 40.7 ms. Average access time to one logical record is thus reduced from 24.7 to just over 8 ms. A higher blocking factor will further reduce average access time. With the file blocked ten, average access time to one record becomes a little more than 6 ms. This increase in

*IBM 729, 7330, and 727 Magnetic Tape Units, IBM Systems Reference Library (A22-6589).

efficiency provides a time advantage, but more main storage space is required. In practice, the amount of record blocking that can be done depends on the amount of main storage available for the larger I/O areas that are required to accommodate long physical records. If a file is used in several runs, the run with the least amount of storage available for record blocking will determine the block size for all the other runs.

The following table shows the effect of record blocking on file passing time. The file contains 20,000 eighty-character records. The I/O device being used is 7330 tape; density is 556 characters per inch.

Blocking factor	Block length	Average time to read or write a block	Number of blocks	Average file pass time
1	80 ch.	24.7 ms	20,000	8.23 min.
2	160 ch.	28.7 ms	10,000	4.78 min.
5	400 ch.	40.7 ms	4,000	2.71 min.
10	800 ch.	60.7 ms	2,000	2.02 min.

One other advantage of blocked records should be mentioned. Although blocked records require more space in the main storage of the computer, they require much less space on a magnetic tape or disk. This is because blocking reduces the number of interrecord gaps and consequently increases the amount of space used for data. The interrecord gap on the 7330 tape averages .75 inches in length. For a record length of 80 characters at 556-cpi density, the amount of blank space on the tape reel exceeds the amount of recorded space until the file is blocked five, at which point the two are about equal. A 2400-foot reel of tape can contain about 32,000 eighty-character records unblocked; if the records are blocked ten, it can contain about 130,000.

RUN BALANCE

Record blocking and the use of multiple I/O areas each have a different purpose. Record blocking increases the efficiency of an I/O operation. The use of multiple I/O areas is necessary for concurrent processing and I/O operations. Both techniques are often used together in order to achieve a balanced run.

Record blocking will always improve the performance of a system that cannot perform simultaneous operations; it will often improve the performance of a system with this ability. On a sequential system — without the ability to do simultaneous I/O and processing — run time is the sum of all I/O plus all processing. It follows, then, that in a sequential system any reduction in either I/O time or processing time will be directly and completely reflected in the time required for the run.

When a data processing system has the potential for simultaneous operations, the benefits of record blocking are less certain. A reduction in I/O time does not necessarily cause a corresponding reduction in total run time. The minimum time for a run performed on a system with the ability to do simultaneous operations is internal processing time plus all channel interference time. Therefore, if the time required for I/O operations is brought below this minimum, further record blocking will have no effect. The run is then process-limited. By definition, a run is process-limited when a reduction in the time required for I/O operations does not reduce the total run time. When the time required for I/O operations is approximately equal to the sum of internal processing time plus channel interference time, a run is considered to be balanced.

To illustrate how record blocking and multiple I/O areas may be used together to bring a run into near balance, assume that:

1. 10,000 eighty-character records are to be read, processed, and written.
2. 12 ms is the average processing time required for each record.
3. A two-channel 1410 with 7330 tapes is the data processing system.
4. 1600 positions of core storage are available for I/O areas.

In one approach, the file might be blocked ten — as highly as possible with the storage available — and simultaneous reading and writing (but not processing) performed. The sequence of operations would be:

1. Perform an initial read of one 800-character tape block (10 logical records) into one of the I/O areas.
2. Process for 120 ms (10 records x 12 ms per record) the data records in the tape block just read.
3. Modify the program so that for the next loop the other I/O area is used for processing.
4. a. Start writing the block just processed.
   b. Start reading the next block into the other I/O area.
5. Wait 60.7 ms for I/O operations to be completed — that is, 20.7 + (800 ch. x .050 ms) — and then return to step 2. Note that the two I/O areas are used alternately as process areas.

If each file is on a different channel, almost complete simultaneity of reading and writing can be achieved (see Figure 6). Since records are blocked to the limit of the storage available, I/O operations have been made as efficient as possible. Storage interference is not a factor in the run timing calculation, since no processing takes place during the read-write operation.

This, however, is not the best way to design the run. None of the processing is being overlapped. The program performs I/O for about 60.7 ms, then processes for about 120 ms, and so on. Run time will be about 181 seconds:

$$\frac{1000 \text{ blocks} \times (120 \text{ ms} + 60.7 \text{ ms})}{1000 \text{ ms per second}} = 180.7 \text{ sec.}$$

This run can be changed to make more efficient use of the 1600 positions of storage that are available for I/O operations. If the blocking factor is reduced to 5, an additional area in core storage can be provided for each file, making a total of four I/O areas for the two files, two input areas, and two output areas. Now the I/O and processing can proceed simultaneously. When the records in an input area have all been processed, the channel can be set to work to refill it. In the meantime, records in the alternate area can be processed. When an output area is filled with processed records, it can immediately be written out, and the alternate area made available to the program for the assembly of a new output tape block. It is now possible to perform simultaneous reading, writing, and processing. With simultaneous operations being performed, storage interference becomes a factor in the calculation of run time. Each time a block of 400 characters is read or written, 400 storage cycles are made unavailable to the processor. This is equal to 1.8 ms (400 x .0045 ms). A table of operations can be constructed that will show time for one process cycle, that is, time required to write one tape block, process a second, and read a third:

Read one tape block	40.7 ms	
Write one tape block (simultaneous operation)		40.7 ms
Perform internal processing (simultaneous operation)		40.7 ms
Do additional processing (60.0 ms - 40.7 ms)	19.3 ms	
Storage interference 2(400 x .0045 ms)	3.6 ms	
Total	63.6 ms	

The total, multiplied by 2000 tape blocks, equals 127.2 seconds, a substantial improvement over the 180 seconds estimated for the first design. This run time cannot be improved by further record blocking, for it is already process-limited (see Figure 7).

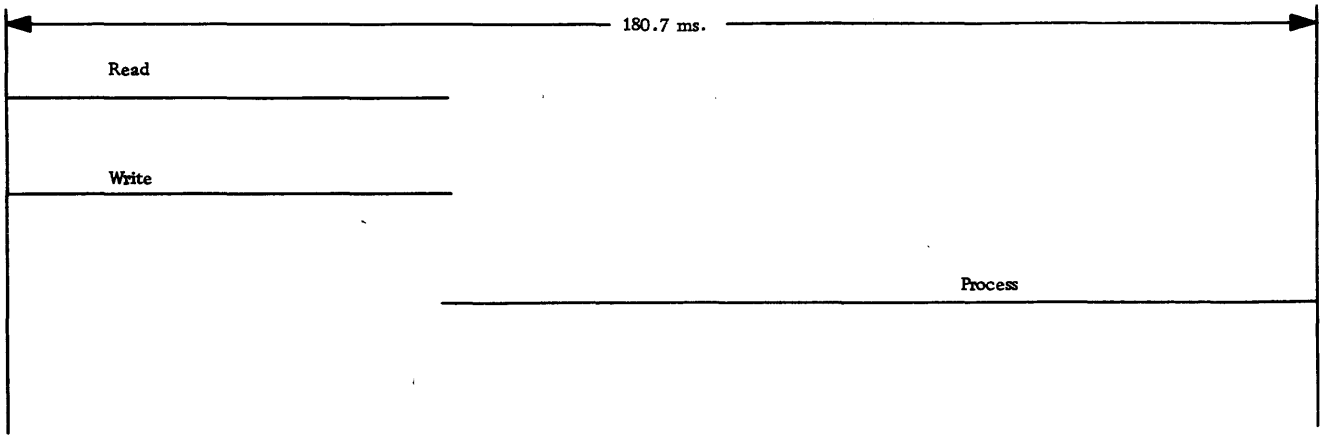

Figure 6. Simultaneous reading and writing, blocked 10.

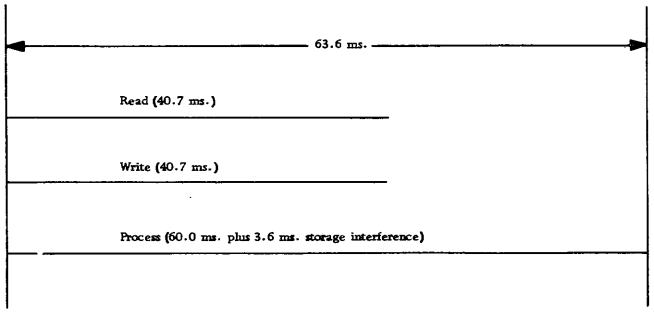

Figure 7. Simultaneous reading, writing, and processing, blocked 5

Process time for a block of five records is 60 ms, but input and output for the block total only 40.7 ms. When 3.6 ms of storage interference is deducted from the 40.7 ms available during I/O, it is evident that the processor can actually accomplish only 37.1 ms of the 60.0 ms of processing required. Therefore, a somewhat lower blocking factor could be used without increasing run time. As a matter of fact, this run is approximately balanced between I/O and processing when the files are blocked three. Blocked three, a tape block is 240 characters. Average time for I/O is 20.7 + (240 x .050), or 32.7 ms. Storage interference when one block is being read and another written is (2 x 240 x .0045 ms), or 2.16 ms. Therefore, while I/O operations are being carried out, available process time is 32.7 ms minus 2.16 ms storage interference, or 30.54 ms. Since 36 ms are required to process the three records, the job is still process-limited when the file is blocked three, although not by much — only 5.46 ms per block. When a blocking factor of three is used, the table of operations looks like this:

Read one tape block	32.70 ms	
Write one tape block (simultaneous operation)		32.70 ms
Perform internal processing (simultaneous operation)		32.70 ms
Do additional processing (36.00 ms - 32.70 ms)	3.30 ms	
Storage interference 2(240 x .0045 ms)	2.16 ms	
Total	38.16 ms	

This total, multiplied by 3333 tape blocks, again equals 127.2 seconds. Decreasing the blocking factor has not changed the estimated run time, because the run is still process-limited.

When the blocking factor is further reduced to two, however, the run becomes tape-limited. In this case, not only will all processing be performed during I/O operations, but the processor will be idle for about 3.26 ms per tape block (see Figure 8). Estimated run time is the time required to read the file — that is, 5000 blocks x 28.7 ms, or about 143.5 seconds.

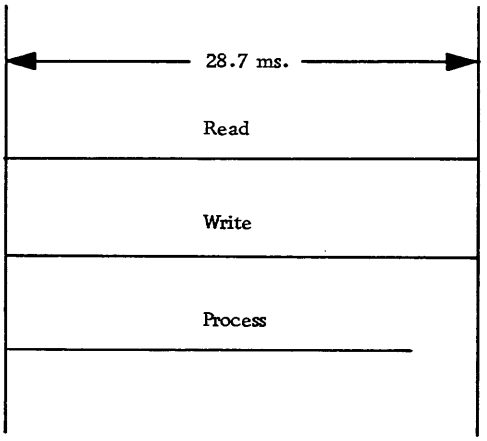

Figure 8. Simultaneous reading, writing and processing, blocked 2

DATA CHARACTERISTICS THAT AFFECT RUN TIME

The characteristics of the data in a run constitute an important determinant of run time. Some data characteristics have already been mentioned: file volumes and record lengths obviously are necessary inputs to any run timing procedure. However, when several files are present in a run, some secondary data characteristics become operative. Secondary data characteristics consist of the relationships among the files — the way in which one file interacts with another or with several others. These relationships are not always easily known, and even when known, they are likely to vary from one execution of a run to another.

SKEWED ACTIVITY WITHIN A FILE

When, during a run, an item in a file is changed, that item is said to be active. Activity is usually expressed as a percentage of the number of records in a file. For example, if an inventory master file contains 100,000 items, and if, during some period of time, the status of 20,000 of those items is altered by some business activity such as sales, purchases, or manufacturing, the file is said to be 20% active. It is important to know the percentage of activity, and also to realize that within that overall percentage, the exact pattern of activity across the file will affect the amount of time required for processing.

An uneven pattern of activity across a master file is sometimes called file skew. This condition can upset the balance between internal processing and I/O operations, which the designer of the run has apparently achieved. Suppose that a block of ten records requires 60.7 ms for reading or writing, and that the entire file is known to be 20% active. With an even distribution of activity, any block of ten records is likely to contain two records which are active. If each inactive record requires 1 ms for routine processing, and each active record requires 25 ms, the run is well balanced between I/O and internal processing. Process time for a block equals 8 ms for the eight inactive records, plus 50 ms for the two active records, for a total of 58 ms.

This, however, assumes an even distribution of activity across the file. It is more likely that many blocks will be encountered in which three, or perhaps five, or even all ten of the records are active. When this occurs, the run becomes process-limited for that block. On the other hand, any block which has fewer than two active records will be I/O-limited. If the activity across a file is severely skewed — that is, if some portions of the file are intensely active and others hardly active at all — the run may be process-limited part of the time and I/O-limited part of the time, and much of the expected simultaneity between processing and I/O functions may be lost.

BURSTS OF ACTIVITY

A 100,000-item master file that is 20% active may be affected by more than 20,000 transactions. In many cases, individual master records are active several times during the run. Several transactions all relating to the same master record are sometimes called a burst of activity.

Taking the example of a 100,000-item master file known to be 20% active, suppose that it is also known that the average run will process 60,000 transactions. The designer of the run knows that, on the average, each active master record will have three related transactions. This average figure may be deceptive, for in many applications some master records are much more active than others. Inventory is a good example; most inventory master files contain a large number of items with very low activity, and a relatively small number of items with high activity. In fact, a rule of thumb for the inventory application is that 80% of all the transactions in a run may be expected to relate to only 20% of the master file items.

Varying bursts of activity, like skew, can cause a run to be alternately process- and I/O limited. It is also possible for long bursts of transactions to disturb the balance between two I/O devices. Suppose, for example, that a transaction file is punched into cards, and that the run is designed with the idea that the card reader will operate once for each time that one block of tape records is read and written. If the card reader is on a separate channel from the tapes, and process requirements are such that one card reader cycle is approximately equal to the time required for a read-process-write cycle of the master tape, the I/O devices involved should not interfere with one another, and a large amount of simultaneity can be achieved between the several I/O operations. But if, in some cases, the master file must remain in place while the card reader reads a burst of 10 or 20 transactions, it is easy to see that an imbalance will exist and that, in effect, one data channel will interfere logically with the operation of other data channels on the system.

DEALING WITH SKEW AND BURSTS OF ACTIVITY FOR TIMING PURPOSES

For timing purposes, it is important to know that skew and bursts of activity will affect actual run time, and that they almost always are present to some extent. Any timing method that makes use of averages, such as an average of two active master records per block, with each active master having an average of three transactions, automatically has the effect of smoothing out uneven patterns of activity and assumes an optimum situation that rarely exists.

If the actual pattern of activity is known, and there is a significant amount of skew, the run may be considered as consisting of several segments, each segment having different skew characteristics.

As an example of how this might be done, and how it might affect a timing estimate, assume the following:

- A System/360, Model 30, that has one selector channel and several Model 2, 2400 series tapes attached.
- A master file of 100,000 eighty-byte records (blocked four) to be read and written.
- A 20,000-card transaction file that is read from a 2540 card read punch attached to the multiplexor channel.
- About 15 ms of processing time required for each active master record.
- About 6 ms of processing time required for the routine handling of each inactive master record.
- Just one transaction card for each active master record.

Functionally, the job looks like this:

Read 20,000 cards at 1000 cpm	20.00 minutes
Read 25,000 tape blocks at 13.34 ms each	5.55 minutes
Write 25,000 tape blocks at 13.34 ms each	5.55 minutes
Process 80,000 inactive records for 6 ms each	8.00 minutes
Process 20,000 active records for 15 ms each	5.00 minutes
*Selector channel interference (16,000,000 bytes at 1.5 microseconds per byte)	.40 minutes
**Multiplexor channel interference (2.7 ms per card)	.90 minutes

The theoretical minimum run time is 14.3 minutes, the sum of all processing and all memory interference. The selector channel time is 11.1 minutes — the sum of all tape reading and writing time for the run. Multiplexor channel time is 20 minutes — total card reading time. Since all other operations can apparently be accomplished within the time required for card reading, 20 minutes seems to be a good estimated run time.

With a reasonably even distribution of active records across the master tape, this estimate will be very close to actual run time. But with a severely skewed activity pattern, the run could take considerably longer. Suppose that this is an inventory run, that 80% of all activity occurs on 20% of the master records, and that those active master records belong to the same product line and are grouped closely together on the master tape. There would be two activity patterns within the run, one very high and one very low. The first would look like this:

Read 16,000 cards (80% of all transactions)	16.00 minutes
Read 5000 tape blocks (20% of all master items)	1.11 minutes
Write 5000 tape blocks	1.11 minutes
Process 4000 inactive records	.40 minutes
Process 16,000 active records	4.00 minutes
Selector channel interference (3,200,000 bytes)	.08 minutes
Multiplexor channel interference (16,000 cards)	.72 minutes

The estimated partial run time is 16.00 minutes. This portion of the run is limited by the card reader.

The times for the second activity pattern would look like this:

Read 4000 cards (20% of all transactions)	4.00 minutes
Read 20,000 tape blocks (80% of all master items)	4.44 minutes
Write 20,000 tape blocks	4.44 minutes
Process 76,000 inactive records	7.60 minutes
Process 4000 active records	1.00 minutes
Selector channel interference (12,800,000 bytes)	.32 minutes
Multiplexor channel interference (4,000 cards)	.18 minutes

The estimated partial run time is 9.1 minutes, the sum of all processing and all channel interference time. CPU operations limit this segment of the run.

*With no data or command chaining.

**An approximation, assuming multibyte mode.

The estimated total run time is 25.1 minutes — the sum of both run segments. One segment of the run is limited by the card reader, and the other by the CPU. Notice that only the estimated time for the run has changed. Total card reader time is still 20 minutes; total processor time (internal processing plus storage interference) is still 14.3 minutes; and total tape time is still 11.1 minutes. It is only relationships within the run that have changed, causing an apparently well balanced run to become highly unbalanced. This is an extreme example, causing a difference of almost 25% between the first and second estimates.

This method of estimating the effects of a skewed activity pattern is not always completely realistic, for it assumes one line of demarcation between two physical file segments, one having relatively high activity and one having relatively low activity. In an actual case, segments of high and low activity may well be interspersed throughout the entire file, and the normal buffering provided by IOCS will tend to mitigate the effect of the skewed activity pattern. Even if it does not exactly fit the actual case, this method provides an approximation of the effect of a skewed activity pattern. It will almost always provide a more realistic estimate of run time than the assumption of an absolutely even distribution of activity.

RUN TIMING AS A PART OF SYSTEM SELECTION

Thus far, run timing has been treated as an end in itself, which, of course, it is not. Run timing is part of a larger problem called system selection, which is a four-step iterative process:

1. Design a system approach to the job.
2. Select a likely configuration of equipment.
3. Estimate run time.
4. Evaluate the results.

This process is repeated until it has yielded the best configuration of data processing equipment for the job at hand. Out of all of the systems considered, this "best" system may be the fastest, the least expensive, or the one that provides the lowest cost per unit of work. On the other hand, it may not be any of these. The "best" system can only be defined as the one that most nearly fulfills all of the user's requirements. The initial choice of systems to be studied in more detail must be based on a thorough knowledge of user objectives and policy.

CONSIDERATIONS IN RUN DESIGN

Run design is largely a matter of allocating the facilities of the data processing system to the elements of the run. System facilities are main storage, data channels, I/O devices, and direct access devices; run elements are programs and data files.

The designer of the run works with a finite system; he has a limited number of facilities available. Within these limits, however, many design variations are often possible. Data files may be switched from one I/O device to another; main storage may be used for long record blocks or for many I/O areas; or I/O areas may be traded between the several files in the run. Some general principles of run design follow:

1. The files that place the greatest burden on the system should be given priority for the fastest I/O devices, the largest blocking factors, and the most I/O areas.
2. The burden that a file places on the system depends on its size (the number of bytes of data that it contains), and also on the number of times that it is processed. A small file appearing in several runs may be a more important determinant of total job time than a large file used in only a few runs.
3. A high blocking factor improves performance for an I/O-limited run; multiple I/O areas improve performance for a process-limited run. In practice, however, many runs are both I/O- and process-limited at different times during the execution of the run.
4. In general, after having blocked records to the point of a theoretical balance between I/O operations and internal processing, the best use of main storage is to provide more I/O areas, rather than to increase the blocking factor still further.
5. At times, space on the external storage medium may be an important consideration, as when trying to fit a file within a direct access storage device. When this is so, a high blocking factor may be essential. However, this high blocking factor will require more main storage space for I/O areas.
6. When a file appears in several runs, the blocking factor for that file is fixed by the run that has the least amount of main storage available for I/O areas.

CONSIDERATIONS IN TIMING

When a data processing system has been selected for detailed timing, and the runs have been designed, the timing process can begin. Before it is begun, these questions should be answered:

1. Are job specifications current? (Estimates of file volumes, percentages of activity, average number of lines per form, etc., are sometimes not up to date. The figures gathered when the job was last studied may no longer be correct.)
2. When will the system be installed? Has an allowance been made for the expected increase in volumes during the preinstallation period?
3. Will the system handle peak loads? (Average volume figures are often deceptive, for volumes sometimes fluctuate considerably with the season of the year or the day of the week or month.)
4. Have all likely configurations been considered, or has the selection process stopped with the first workable one? (As many iterations of the design and timing cycle should be performed as are possible with the time and resources available.)
5. Is it necessary to time out every run in the job on each system being considered? (Often, comparative times on one or two important runs will establish the superiority of one system. The complete job can then be timed only for the selected system.)
6. What effect will the use of operating systems have on the timings? (Although operating systems can save large amounts of time or core storage, or prevent costly operator errors, the time that they require may be an important factor in run timing. This is particularly true if some features are used extensively, as in overlaying program segments for more efficient use of core storage.)

ACCURACY OF RUN TIME ESTIMATES

A timing estimate, however carefully done, should never be considered ironclad. This is because the timing equation is never complete. Some factors are always operative that cannot be measured completely and accurately, or cannot be considered at all. For this reason, estimated run times will vary somewhat from actual run times. Some of these hard-to-measure factors are:

1. Efficiency of the problem program. This depends on the skill of the individual programmer who does the work.
2. An unusually adverse or unusually favorable distribution of activity across a master file.
3. A business cycle that causes many exception routines to be executed.
4. A variance from the expected number of stepdowns in sequence when a file is sorted.

Any of these factors can cause estimated run times to vary from actual run times; often they will even cause the time required for a run to fluctuate noticeably from one execution of that run to another. But unless the total estimated load on the data processing system is very close to a maximum limit, these factors should not cause concern.

Printed in U.S.A. C20-1648-1

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)