Introduction to IBM Data Processing Systems

Student Text

IBM

Introduction to IBM Data Processing Systems

Student Text

Preface

All IBM Data Processing Systems, regardless of size, type, or basic use, have certain common fundamental concepts and operational principles. This manual presents these concepts and principles as an aid in developing a basic knowledge of computers. The manual is designed for use in training programs where a basic knowledge of computers is the end objective or is a prerequisite to the detailed study of a particular IBM system.

Each section is organized to present a logical association of related concepts and operational principles. The sections may be used in a progressive sequence to develop a concept of the computer system, or they may be used independently as reference material. The subject matter has been generalized and refers to actual machines and systems as little as possible. Specific systems are mentioned only to illustrate a general principle, not to compare one system with another.

Throughout this manual you will be reading about data processing concepts and devices supported by IBM. Such specifics as memory requirements, device capacities and speeds, and special features will be discussed. However, because of the dynamic nature of data processing, where changes and improvements are being made at a very rapid pace, the reader is advised to refer to the IBM Systems Reference Library and other IBM publications for the most current information.

Fifth Edition -- June 1977, Reprinted July 1978.

This edition is a major revision of and obsoletes all previous editions of GC20-1684. It is intended as a partial update of the previous edition (GC20-1684-3) incorporating numerous changes that have occurred since publication of that edition.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Address comments concerning the contents of this publication to IBM Corporation, DPD Education Development - Publishing/Media Support, Education Center, South Road, Poughkeepsie, New York 12602.

Contents

Preface i

Section 1: Introduction 1-1
Data Processing in the Past. 1-4
Data Processing at Present 1-8
Data Processing in the Future 1-8
The Data Processing System 1-8
Stored Programs. 1-10
Functional Units 1-12
Central Processing Unit. 1-12
Storage 1-13
Input and Output Devices 1-13
Console. 1-16
System/3. 1-17
System 32 1-18
IBM 5100 1-18

Section 2: Data Representation 2-1
Computer Data Representation 2-3
Binary States 2-4
Binary Number System. 2-5
Computer Codes 2-6
Code Checking 2-6
Standard BCD Interchange Code 2-8
Six-Bit Alphameric Code (Binary Coded Decimal System). . . . 2-10
Eight-Bit Alphameric Code (Extended Binary Coded Decimal Interchange Code - EBCDIC) 2-11
Eight-Bit Alphameric Code (ASCII-8) 2-13
Computer Number Systems and Conversion. 2-13
Binary System. 2-13
Hexadecimal System 2-13
Integer Conversion 2-14
Hexadecimal to Decimal 2-14
Decimal to Hexadecimal 2-15
Hexadecimal to Binary and Binary to Hexadecimal. 2-16
Decimal to Binary. 2-16
Binary to Decimal. 2-16
Fraction Conversion 2-17
Decimal to Hexadecimal 2-17
Hexadecimal to Decimal 2-18
Usage of Hexadecimal 2-20
Data Recording Media 2-21
Cards 2-21
The 80-Column Card Code. 2-21
System/3 Card. 2-22
Representation of Information 2-23
Zone and Digit Portions of a Column. 2-23
Paper Tape. 2-24
Eight-channel Code (Hole Pattern) 2-24
Five-channel Code (Hole Pattern). 2-25
Magnetic Tape. 2-26
Seven-Track Coding on Tape (BCD Mode). 2-27
Seven-Track Coding on Tape (Binary Mode). 2-27
Nine-Track Coding on Tape 2-28
Character Reading 2-30
Magnetic Ink Characters 2-30
Optically Read Characters 2-31
Visual Output 2-32

Section 3: Storage Devices 3-1
Main Storage. 3-4
Magnetic Disk Storage 3-5
Mass Storage. 3-9
Storage and Data Processing Methods 3-11

Section 4: Central Processing Unit (CPU). 4-1
Functional Units 4-2
Register 4-2
Counter 4-5
Adder. 4-5
Machine Cycles 4-5
Instruction Cycle 4-6
Execution Cycle. 4-7
Serial and Parallel Operation. 4-8
Fixed - Length and Variable - Length Words. 4-9
Floating - Point Operation 4-9

Section 5: Input/Output Devices. 5-1
Control Units 5-2
Channels 5-3
Validity Checks 5-4
Indicators, Keys and Switches. 5-4
Punched Card Devices 5-4
Card Readers 5-4
Card Punches 5-5
Diskette Input/Output Devices 5-6
Magnetic Tape Units 5-8
Loading Tape Units. 5-9
Reading From and Writing on Magnetic Tape 5-10
Checking Magnetic Tape Data. 5-11
Tape Records, Interblock Gap and Tapemark. 5-13
Tape Unit Characteristics 5-14
Maximum and Effective Character Rates. 5-15
Load-Point and End-of-Reel Markers. 5-15
File Protection 5-16
Direct Access Storage Devices. 5-16
Punched Tape Devices 5-17
Paper Tape Reader 5-17
Paper Tape Punch. 5-18
Printers. 5-18
3800 Printing Subsystem. 5-18
Impact Printer. 5-21
Character Recognition Input Units 5-24
Magnetic Character Readers 5-24
Optical Character Readers 5-26
Consoles 5-27
Graphic Display Units 5-28
Terminals 5-30
Data Buffering. 5-30

Section 6: Teleprocessing 6-1
Elements of a Teleprocessing Network. 6-2
Host Processor. 6-2
Communications Control Devices. 6-2
Modulation/Demodulation Devices. 6-3
Communications Lines. 6-4
Configuration 6-4
Transmission Direction. 6-4
Type 6-4
Transmission Mode 6-5
Codes. 6-5
Line Disciplines 6-5
Binary Synchronous Communications 6-5
Synchronous Data Link Control 6-5
Teleprocessing Applications 6-6

Section 7: Stored Program Concepts . . . 7-1
Instructions . . . 7-1
Two-Address Instructions . . . 7-2
Instructions and Data . . . 7-2
Developing a Program . . . 7-3
System Flowchart . . . 7-5
Program Flowchart . . . 7-7
Flowcharting Worksheet . . . 7-7
Miscellaneous Techniques . . . 7-9
Flowcharting by Computer . . . 7-11
Reading Data . . . 7-11
Calculating . . . 7-12
Logical Operations . . . 7-14
Comparing . . . 7-19
Indexing . . . 7-20
Linking . . . 7-22
Chaining . . . 7-24

Section 8: Programming Languages and Techniques . . . 8-1
Program Preparation . . . 8-1
Types of Programming Languages . . . 8-2
Symbolic Language . . . 8-2
Language Translation . . . 8-4
Machine-Oriented Programming Languages . . . 8-5
Macro Instructions . . . 8-6
The Macro Instruction Statement . . . 8-7
The Macro Definition . . . 8-7
The Macro Library . . . 8-7
Varying the Generated Statements . . . 8-7
COBOL . . . 8-7
FORTRAN . . . 8-8
PL/I . . . 8-9
APL . . . 8-10
Report Program Generator . . . 8-12
Program Checkout . . . 8-13
Testing Techniques . . . 8-13
Storage Printout . . . 8-14
Tracing . . . 8-14
Improved Programming Technologies . . . 8-14
Structured Programming . . . 8-14
Structured Programming Theory . . . 8-15
Top-Down Program Development . . . 8-17
Chief Programmer Teams . . . 8-18
HIPO . . . 8-18
Structured Walk-Throughs . . . 8-21
Inspections . . . 8-22

Section 9: Programming Systems . . . 9-1
Input/Output Control Systems . . . 9-1
Operating System . . . 9-2
Virtual Storage . . . 9-2
DOS/VS . . . 9-5
Component Programs . . . 9-5
Control Programs . . . 9-5
Processing Programs . . . 9-5
Data Management . . . 9-6
Multiprogramming . . . 9-7
Virtual Storage Support . . . 9-8
Paging . . . 9-8
OS/VS1 . . . 9-10
Job Management . . . 9-10
Task Management . . . 9-11
Data Management . . . 9-11
Recovery Management . . . 9-11
Service Aids . . . 9-12
GTF (Generalized Trace Facility) . . . 9-12
HMASPZAP Program . . . 9-12
Virtual Storage Support . . . 9-12
JES (Job Entry Subsystem) . . . 9-12
Compatibility . . . 9-13
Other Features . . . 9-13
Shared DASD (Direct Access Storage Device) . . . 9-13
SMF (System Management Facilities) . . . 9-13
Checkpoint/Restart . . . 9-13
MCS (Multiple Console Support) . . . 9-14
Time-Slicing Facility . . . 9-14
OS/VS2 . . . 9-14
Job Management . . . 9-14
Multiple Console Support - MCS . . . 9-14
System Log . . . 9-15
Hardcopy Log . . . 9-15
Checkpoint/Restart . . . 9-15
System Management Facilities - SMF . . . 9-15
Other Features . . . 9-16
System Integrity . . . 9-16
Time Sharing Option - TSO . . . 9-16
Virtual Machine Facility/370 (VM/370) . . . 9-18
Elements of VM/370 . . . 9-18
Virtual Machine Operating Systems . . . 9-19
VM/370 Applications . . . 9-19
Programming . . . 9-19
Operations . . . 9-20
Interactive Use . . . 9-21
Access Methods . . . 9-22
Sequential Access Method - SAM . . . 9-23
Basic Direct Access Method - BDAM . . . 9-23
Indexed Sequential Access Method - ISAM . . . 9-23
Virtual Storage Access Method - VSAM . . . 9-24
Alternate Indexes . . . 9-25
Relative Record Data Set . . . 9-25
Get Previous . . . 9-25
Telecommunications Access Methods . . . 9-25
Basic Telecommunications Access Methods - BTAM . . . 9-26
Virtual Telecommunications Access Method - VTAM . . . 9-26
Telecommunications Access Methods - TCAM . . . 9-28

Section 10: Data Security and Integrity . . . 10-1
Backup Facilities . . . 10-2
Data Controls . . . 10-3
Input Controls . . . 10-3
Processing Controls . . . 10-4
Record Count . . . 10-4
Limit Check or Reasonableness Check . . . 10-5
Tape and Disk Labels . . . 10-5
Housekeeping Check . . . 10-6
Sequence Check . . . 10-6
Output Controls . . . 10-6
Control Total . . . 10-6
Console Error Messages . . . 10-7
Checkpoint and Restart . . . 10-7
Procedural Controls . . . 10-7
Separation of Responsibility . . . 10-7
Resolving Exceptions . . . 10-9
Overriding Programmed Controls . . . 10-9
Programming Controls . . . 10-9
Establishing Programming Controls . . . 10-10
Operations Control Groups . . . 10-11
Controlling Programs . . . 10-11
Network Security . . . 10-12
Terminal Security . . . 10-13
System Controls and Procedures . . . 10-13

Index

Section 1:

Introduction

Technological advance in data processing is both dynamic and extensive. The ways in which data processing systems can be used seem almost boundless. Each new application demonstrates how such systems can be used to help man enlarge his capabilities.

Data processing systems ordinarily consist of a combination of programs and physical equipment designed to handle business or scientific data at electronic speeds with self-checking accuracy. The physical equipment (Figure 1-1) consists of various units, including input, storage, processing and output devices. Figure 1-2 pictures a *teleprocessing*, (telecommunications plus data processing) system applied to airline reservation activities.

Machines are devised by men for a purpose. In the case of data processing machines, the purpose can be expressed simply: they offer man a means to increase his productivity.

They do this in two ways. First, they enable man to increase his output per hour and the quality of his output (this is true whether it be in research, production, problem solving, or the distribution of goods and services). Second, they increase productivity by encouraging careful and intelligent planning.

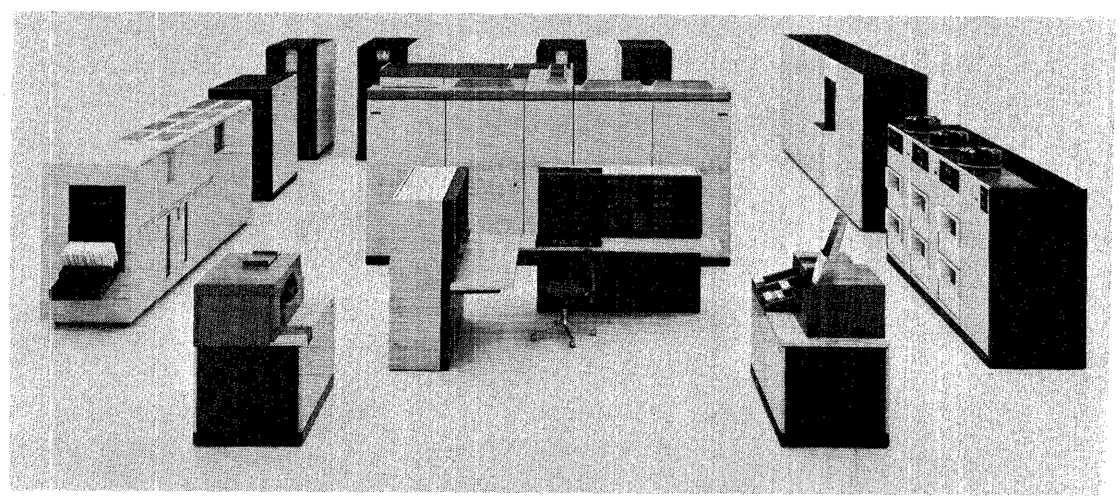


Figure 1-1 IBM System/370 Model 168 Data Processing System

Data processing machines came into being primarily to meet the increased need for information under increasingly complex conditions.

As a manufacturing economy developed during the 19th century, it became clear that expanded markets would require mass production techniques. Machinery was introduced to increase productivity. It became possible to turn out more and more goods with less human effort.

During the last quarter century, further changes have taken place. Science has moved into the forefront of human activity. Research has grown to a multibillion-dollar-a-year undertaking. New technology has provided a new impetus for corporate growth. Service industries have multiplied. Patterns of consumer spending have changed.

As these changes gained force, they manifested themselves in many ways. Informational needs greatly increased. Data assumed new importance. Clerical tasks multiplied. It seemed that paper handling alone would overwhelm all productive activities, for clerical mechanization had not kept pace with production line developments in the factory.

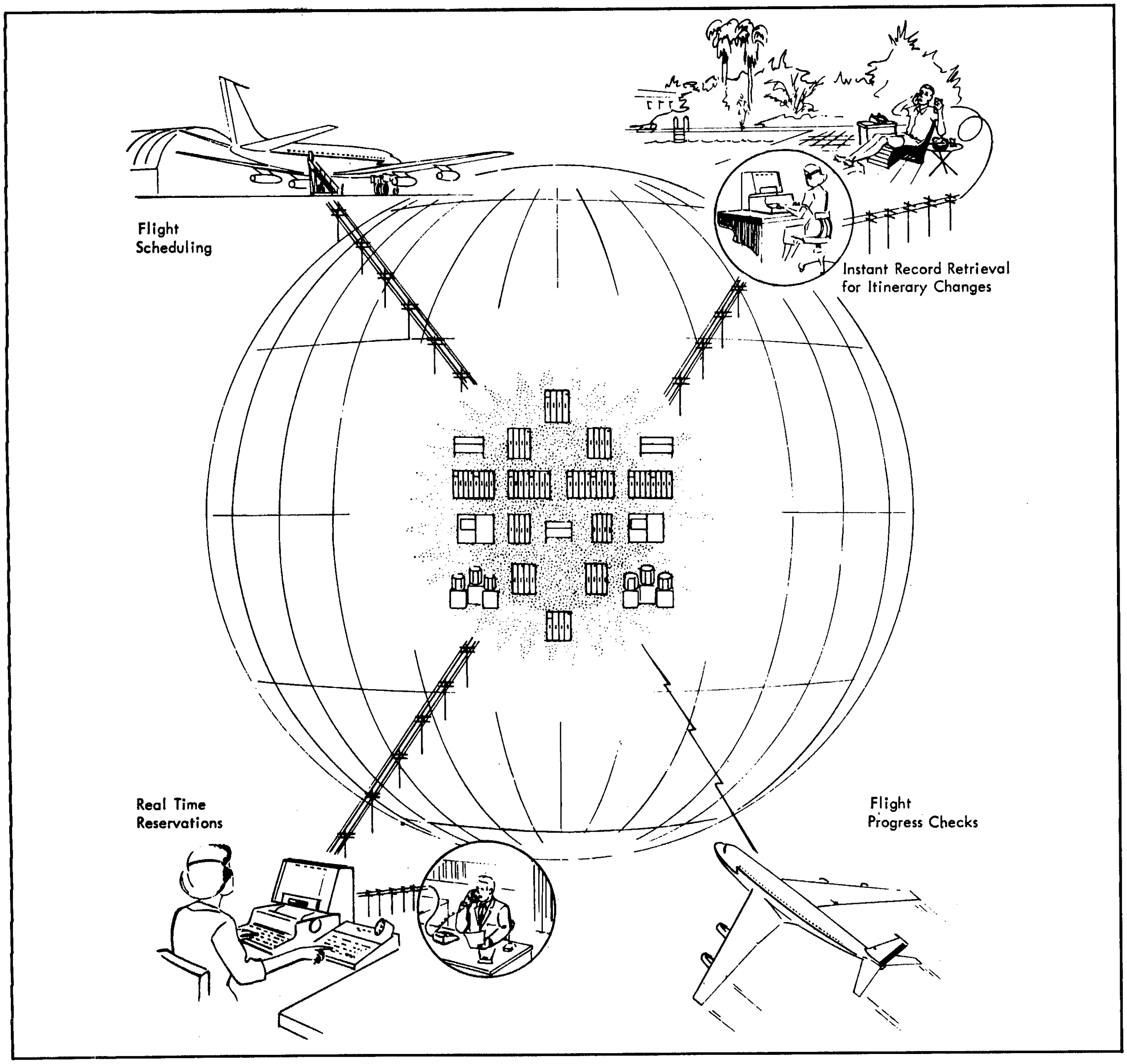


Figure 1-2 Data Processing System Application

Great opportunities and challenges lie ahead. An example of what can be done is the development a few years ago of magnetic character sensing for the banking industry. The estimated 30 billion checks that circulate annually in the United States present a staggering task in data handling for banks. Each check drawn on a bank must be handled at least six times before it is cancelled and returned. Even when business machines were introduced to handle part of this chore, operators were needed to transfer data from the checks to a form in which the data could be used by the machines.

Magnetic character sensing, developed by computer manufacturers in cooperation with the American Bankers Association (ABA), permits data to be read directly by both man and machine (Figure 1-3). By agreement among computer manufacturers, check printers, and the ABA, such banking documents as checks, deposit slips, and debit and credit memos can be printed in magnetic ink. Printed information about the bank of origin, depositor's account number, and other essential data can be read directly by the machine. Only the specific amount of each check or deposit slip need be recorded on the document in magnetic print, and this need be done only once by an operator to process the document through its entire routine.

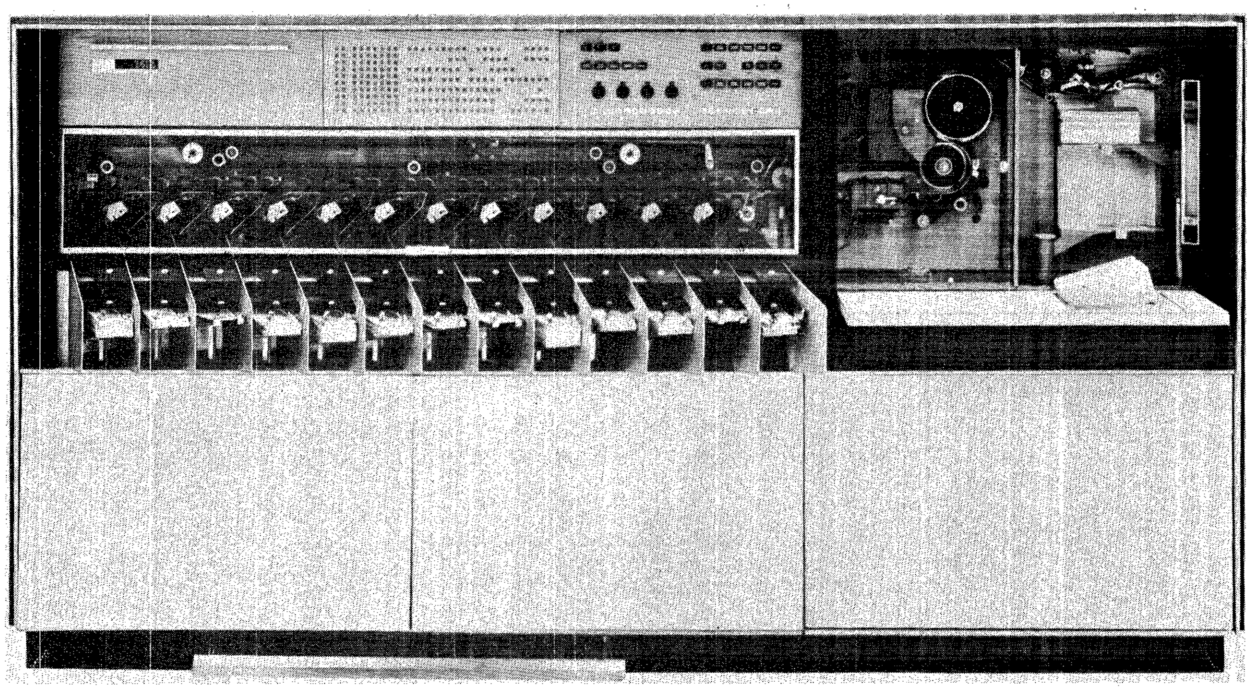


Figure 1-3 Magnetic Character Sensing - IBM 1419 Magnetic Character Reader

In addition to the growing need for mechanization of clerical routines and management procedures, there is the tremendously expanded need for data processing to match the new rate of technological growth and scientific research. The demands for information are enormous. Data processing systems are increasingly relied upon for information to assist in running enterprises, administering institutions, directing research, and planning future activities. To this end, data processing centers are, increasingly, offering time-sharing services to their users. The users can enter problems to be solved, requests for information, and data to be processed - all from remote terminals located either on site or possibly thousands of miles away. The automatic reservation systems for airlines and motels are examples of the long-distance entries in intracompany time sharing.

Two other areas of remarkable advances are image processing and audio response. The processing of the Mariner IV pictures transmitted from the planet Mars to Earth was an example of image processing.

Rapid microfilm scanning was combined with automatic interpretation of dark and light spots into 1's and 0's for computer storage. These pictures were then displayed on viewers' screens.

Although it is possible, experimentally, to dictate or speak directly to a binary recording device that will compute from the dictation, the converse is being done daily at the New York Stock Exchange. There, the latest stock prices are quoted on request, by a recorded voice. The message is selected and assembled from spoken words previously stored in the computer.

Regardless of the product or problem, the nature of the enterprise or institution, wherever there is need for information upon which human judgments can be based, there may also exist a need for a data processing system.

Data Processing in the Past

Although data processing systems are tools of astonishing versatility, the automatic processing of data is so recent that its biggest period of growth can be traced within the last 40 years.

Punched cards were introduced during the census of 1890, but the data processing industry, as recently as 1937, amounted to little more than a fledgling.

World War II caused a swift change of pace in data processing developments. Much of the momentum came from the urgent demands of science. In aircraft design and ordnance development, new and prodigious requirements for data were encountered. As work got under way on the atomic bomb, scientists found themselves suddenly faced with new dimensions in calculation

Both here and abroad, the first two large-scale computers were developed in university laboratories. The earliest, the ENIAC, came from the University of Pennsylvania; Europe's first, the EDSAC, came from the laboratories of Cambridge University in England.

In these machines, the switching and control functions, once entrusted to relays, were handled by vacuum tubes. Thus, the relatively slow movements of switches in electromechanical computers were replaced by the swift motion of electrons. By this changeover, it became possible to increase the speed of calculation and perform computations 1000 times as fast as before.

Almost concurrently with the use of electronics came another major development that was to widen the capabilities of data processing systems and expand their opportunities for application. This advance is embodied in what is called a stored program computer. At the start, machine instructions were programmed on interchangeable control panels, cards, or paper tapes. Detailed instructions had to be wired in or read into the machine as the work progressed. Data put into the computer was processed according to the instructions contained in these preset devices. Only in a limited way could the computer depart from the fixed sequence of its program.

It soon became apparent that these programming techniques inhibited the performance of the computer. To give the computer greater latitude in working problems without operator assistance, scientists proposed that the computer store its program in a high-speed internal memory or storage unit. Thus, the computer would have immediate access to instructions as rapidly as it called for them. With an internal storage system, the computer could process a program in much the same way that it processed data. It could even be made to modify its own instructions as dictated by developing stages of work.

The earliest computer to incorporate this feature was completed in 1948. Later computers extended the principle until it became possible for a computer to generate a considerable part of its own instructions.

Because the computer is capable of making simple decisions, and because it is capable of modifying instructions, the user is relieved of a vast amount of costly and repetitive programming.

Concurrently with the development of stored programs for computers, teleprocessing was being born, although it was not known by that name for more than a decade. In 1940, the U.S. Air Corps voiced a need for a machine to automatically punch IBM cards with the data received over telegraph lines in the form of punched paper tape To answer this need, IBM produced a tape-controlled card punch and a card-controlled tape punch. During the last two years of World War II, 4 to 5 million cards per month were transmitted from point to point by telegraph.

The next major advance in teleprocessing was the introduction of the IBM Data Transceiver (1954) to provide direct card-to-card transmission over voice grade (telephone) channels, as well as microwave, short-wave radio, and telegraph channels.

The early 1950s saw the introduction of medium and large scale data processing systems, specifically designed to take over the burdensome clerical chores that beset so many growing companies.

Though essentially similar to previous computers in the way they processed data, these new business systems differed substantially in various parts of their makeup. In scientific research, most problems call for relatively few items to be subjected to intensive machine processing. In business operations, the reverse is more often true. Here the need is for machines that accommodate vast numbers of items, while the processing, by comparison, is ordinarily quite simple.

Modifications in these new business systems were addressed to the twin problems of input and output.

Early computers had used punched cards and paper tapes for the input of information. Then a method was developed for storing information as magnetized spots on *magnetic tape*. This new technique provided input speed 50 to 75 times that of cards and brought improvement in input, output, and storage. More recent advances in magnetic tape technology have greatly increased the original input/output rate.

After the Korean War, man's need seemed to be constantly one jump ahead of the computer's ability to handle the logical and arithmetic labors of his reasoning. The demand quickened especially in such fields as nuclear physics and space technology, where work on the H-bomb and ballistic missiles presented problems that put a severe strain on the capacities of existing machines. Still more speed was needed.

A substitute for earlier storage devices appeared in the early 1950s - the *magnetic core*, which is a small ring of ferromagnetic material. When strung on a complex of fine wires (Figure 1-4), magnetic cores can be made up into a high-speed internal storage system. An array of cores - some magnetized in one direction, some in the other - represents items of information. Items in a core storage can be located and made ready for processing in a few millionths of a second.

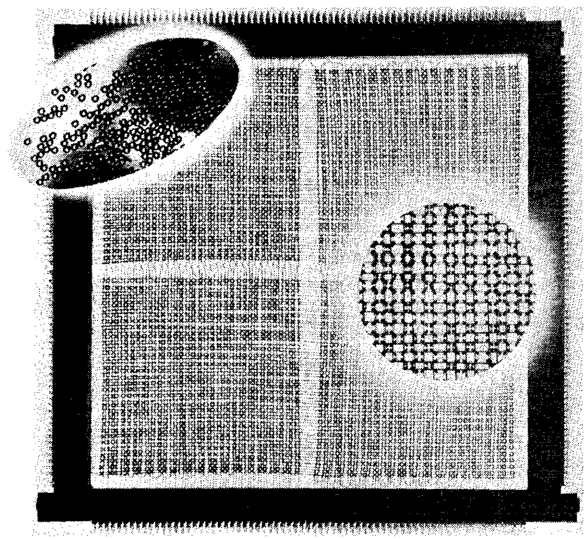


Figure 1-4 Magnetic Core Plane

Almost at the same time, other engineers developed magnetic drum storage. Access to information stored on the drum was substantially slower than with the core system, but storage capacity was substantially increased, and access was still faster than with magnetic tape.

Other conditions peculiar to business led to still more developments. A major one is a system that overcomes a problem - batching -often encountered in data processing. For example, magnetic tapes store information sequentially, and the user must accumulate information in batches before putting it on tape. Otherwise, the computer would be prohibitively costly and time consuming. But when this limitation is applied to business practice, it means that each item of information can be only as current as the batch in which it is bundled for delivery to the computer. In ordinary operations, hours and sometimes days may elapse between batches.

Because of the sequential nature of tapes, the limitation is compounded when the user calls for the retrieval of a piece of information. The computer is forced to search through a long reel of information for the piece. Access is slow; time may be lost.

Batching and searching requirements frequently present serious drawbacks, even in scientific work. In business, the difficulty becomes much more acute, especially in accounting procedures.

Inline data processing was provided in the mid-50s with the introduction of random (direct) access storage units. These allow direct access to the desired data record addressed, thus reducing the processing time required in sequential processing. The first direct access storage units consisted essentially of a stack of magnetically coated, rotating disks, each disk containing data tracks. Information can be recorded on, or retrieved from, the data tracks without regard to the sequential order of recorded data.

Meanwhile, continuing developments in electronics and solid state physics led to still newer and better components.

In some switching functions, the vacuum tube was replaced by a smaller semiconductor diode that has the advantage of demanding less power. A further advance came when tiny transistors were introduced in place of vacuum tubes in the computer. Not only can these transistors be packed into smaller units (Figure 1-5), but they have greater reliability. The changeover to transistors was accomplished, creating what has frequently been referred to as the "Second Generation" of computers.

The next technological advance miniaturized and refined components of the Second Generation. This, when done, led to a concept known as Solid Logic Technology. The use of these components (Figure 1-5) ushered in the "Third Generation" of computers.

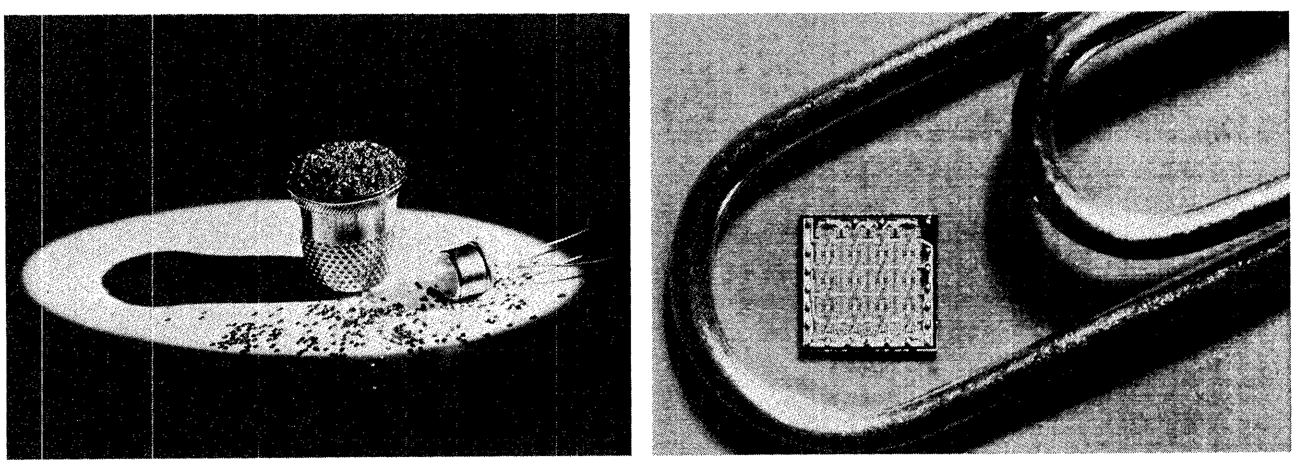


Figure 1-5 Second and Third Generation Computer Components

Data Processing at Present

A significant advance in input/output technique is the development of the various types of graphic displays. These are similar in appearance to television sets, but the "picture" is computer output in the form of printed characters or graphic designs specified by the program and data. In some cases, the user can change the output display by using a *light pen* to "erase" a character (or part of a line) from the display screen. Then, by using the light pen or the associated keyboards, or both, he can alter the displayed information. More and more frequently, display and other devices are used to enter data into the computer from remote locations by means of communications lines (usually telephone lines) connecting the devices known as *terminals* with the computer. When multiple terminals are connected to the computer, concurrent usage of the computer for problem solving and program development on an *interactive* basis often increases substantially.

Along with other advances in computer technology, computer data storage capacity has increased immensely. This has led to the development of large on-line interrelated collections of data referred to as *data bases*. Many large centralized data bases utilizing remote terminals have evolved in recent years. Such systems of large networks have integrated widely scattered business operations. This has occurred since, in general, information can be communicated and processed more accurately and with less cost by network-integrated processing systems.

More recently, data communications systems with processing and disk storage capabilities have been developed. These systems located at sites remote from the central (host) computer are capable of processing much of the input data locally. Data that has been deemed necessary or desirable for processing or maintenance at the central site is transmitted to the host computer. This is often referred to as *distributed function*.

Data Processing in the Future

Computers of the future, as well as programs, will probably be quite different from those of today. Storage and processing units will be drastically reduced in physical size, yet speed and capacity will be greatly increased.

Research scientists have already advanced to still further stages in design. Some are studying magnetic bubbles on a chip for use as secondary storage in the computer. Others are considering electron beam techniques for main storage.

As always, the objective is to develop a better, more versatile, more useful computer - one that will work faster, store more information, require less power, occupy less space, and cost less. Computers of the future will inevitably introduce changes in the way we work, in the way we learn, and even in the way we provide for our armed defense.

The Data Processing System

Data processing is a series of planned actions and operations upon information to achieve a desired result. The procedures and devices used constitute a data processing system (Figure 1-6). The devices may vary: all operations may be done by machine, or the devices may be only pencil and paper. The procedures, however, remain basically the same.

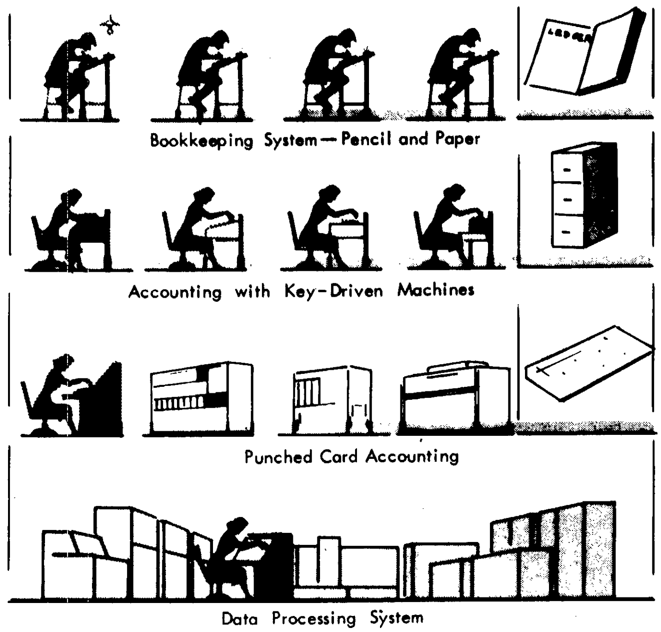


Figure 1-6 Data Processing Systems

There are many types of IBM data processing systems. These vary in size, complexity, speed, cost, levels of programming systems, and application. But, regardless of the information to be processed or the equipment used, all data processing involves at least three basic considerations:

1. The source data or *input* entering the system
2. The orderly, planned processing within the system
3. The end result or output from the system

Input may consist of any type of data: commercial, scientific, statistical, engineering, and so on (Figure 1-7).

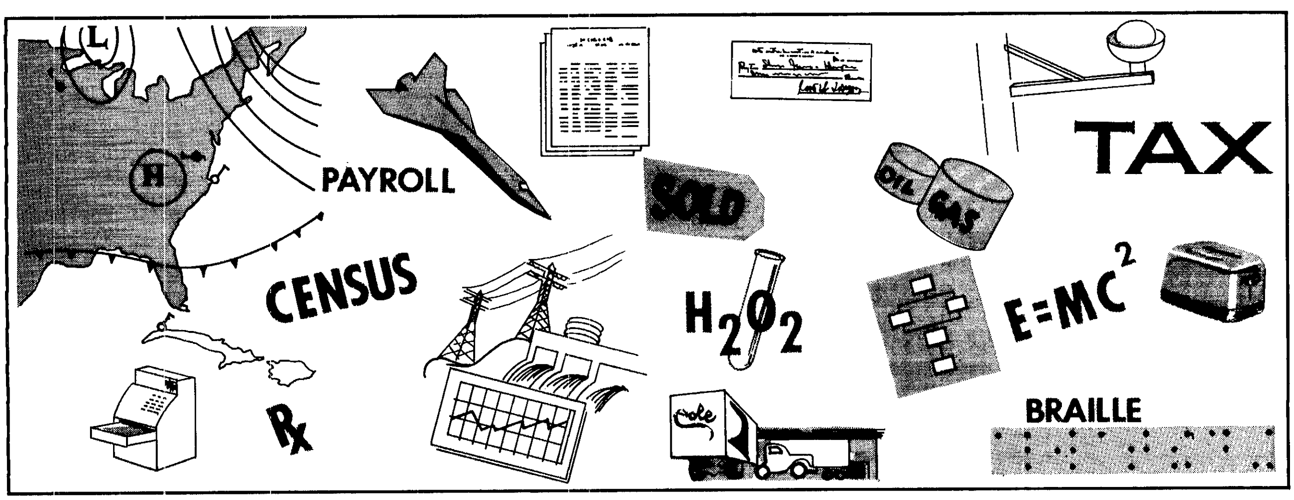


Figure 1-7 Sources of Data

Processing is carried out in a preestablished sequence of instructions that are followed automatically by the computer (Figure 1-8). The plan of processing is always of human origin. By calculation, sorting, analysis, or other operations, the computer arrives at results that may be used for further processing or recorded as reports or sets of data.

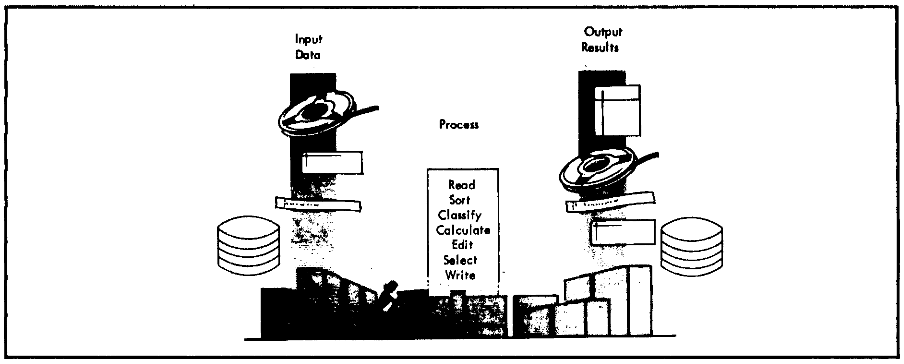


Figure 1-8 Data Processing by Computer

Stored Programs

Each data processing system is designed to perform a specific number and type of operations. It is directed to perform each operation by an instruction. The instruction defines a basic operation to be performed and identifies the data, device, or mechanism needed to carry out the operation. The entire series of instructions required to complete a given procedure is known as a *program*.

For example, the computer may have the operation of multiplication built into its circuits in much the same way that the ability to add is built into a simple desk adding machine. There must be some means of directing the computer to perform multiplication, just as the adding machine is directed by depressing keys. There must also be a way to instruct the computer where in storage it can find the factors to multiply.

Further, the comparatively simple operation of multiplication implies other activity that must precede and follow the calculation. Assume that the multiplicand and the multiplier are read into storage by an input device. This device must previously have had access to the record or records from which these factors are to be supplied. Once the calculation is performed, the product must be returned to storage at a specified location, from which it may be written out by an output device.

Any calculation, therefore, involves reading, locating factors in storage, perhaps adjusting the result, returning the result to storage, and writing out the completed result. Even the simplest portion of a procedure involves a number of planned steps that must be spelled out to the computer if the procedure is to be accomplished.

An entire procedure is composed of these individual steps grouped in a sequence that directs the computer to produce a desired result. Thus, a complex problem must first be reduced to a series of basic machine operations before it can be solved. Each of these operations is coded as one instruction or a series of instructions, in a form that can be interpreted by the computer, and is placed in the main storage unit as a portion of a stored program.

The possible variations of a stored program provide the data processing system with almost unlimited flexibility. A computer can be applied to a great number of different procedures simply by reading in or loading the proper program into storage. Any of the standard input devices can be used for this purpose, because instructions can be coded into machine language just as data can.

The stored program is accessible to the machine, providing the computer with the ability to alter the program in response to conditions encountered during an operation. Consequently, the program selects alternatives within the framework of the anticipated conditions.

A brief introduction to various types of programs and systems operations follows. All of the terms are discussed in greater detail later in the manual.

To make possible the teleprocessing networks and the orderly operation of many types of input/output devices that may be on-line with a computer, control programs have been developed by IBM and users of IBM computers. Control programs are also known as monitor programs or supervisory programs and they act as traffic directors for all the other programs. The others, called processing programs or problem programs, solve a problem or carry out a particular operation or process on a set of data and later relinquish control of the computer to the control program. Multiple levels of control programs are possible. However, each higher level control program considers the next level control program as a problem program. The control program may be constructed to allow the computer to handle random inquiries from remote terminals, switch from one problem program within the computer to another, control external equipment or do whatever the application calls for.

The concept of maintaining optimum computer usage by interleaving and interspersing processing programs under the direction of control programs gives rise to the use of two terms - time sharing and multiprogramming.

Briefly, time sharing may be thought of as the cooperative use of a central computer by more than one user (company, division or branch of a company, institution, or government agency). Each user receives a share of the time available, with the result that many jobs are being performed within a congruent time (either simultaneously or seemingly simultaneously). This service may be achieved by interspersing programs rapidly on one computer system, by multiprogramming (described later), or by using two computers that are joined to permit the sharing of each other's facilities (multiprocessing).

Multiprogramming is usually thought of as a system of control programs and computer equipment that permits many processing or operating programs of one or more users to go on concurrently. This is accomplished by interleaving the programs with each other in their use of the central processing unit, storage, and input/output devices. To do this, the control programs and equipment must be able to identify the point at which a problem program that is being executed must "wait" for the completion of some event. At that point, the control program begins another processing task that is ready to be executed. When that is done, the control program must be able to go on to something else or go back to the former (unfinished) program, if it is ready to continue. Since many programs may be in stages of partial completion, successful multiprogramming usually requires scheduling levels of priority for the different tasks.

Time sharing, multiprogramming, and multiprocessing are closely linked, and may be combined in many ways. While one user has the computer on a time sharing basis, his problem may involve several different tasks that can be interleaved by a computer and programming system that provides for multiprogramming. It is also perfectly possible for teleprocessing messages to be coming in and going out of certain types of computers at the same time that process (problem) programs are being run. These are but two examples of possible combinations of time sharing and multiprogramming.

Functional Units

Data processing systems can be divided into four types of functional units: central processing unit, storage, input devices, and output devices.

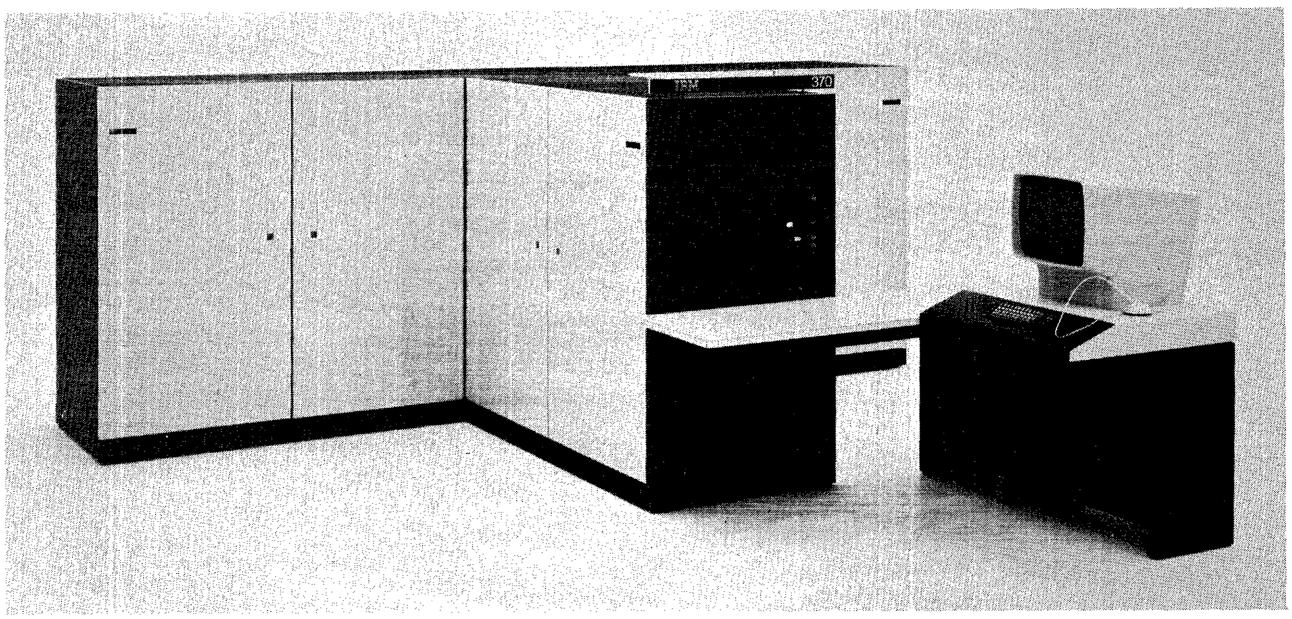


Figure 1-9 Central Processing Unit and Console

Central Processing Unit

The central processing unit (Figure 1-9) is the controlling center of the entire data processing system. It can be divided into two parts:

1. The arithmetic/logical unit;
2. The control section

The arithmetic/logical unit performs such operations as addition, subtraction, multiplication, division, shifting, moving, comparing, and storing. It also has a logical capability to test various conditions encountered during processing and to take action accordingly.

The control section directs and coordinates the entire computer system. Its functions involve controlling the input/output units and the arithmetic/logical operation of the central processing unit, and transferring data to and from storage, within given design limits. This section directs the system according to the procedure originated by its human operators and programmers.

Storage

Storage is somewhat like an electronic filing cabinet, completely indexed and instantaneously accessible to the computer.

All data must be placed in storage before it can be processed by the computer. Information is read into storage by an input unit and is then available for internal processing. Each position or section of storage has a specific location called an *address*, so that the stored data can be readily located by the computer as needed.

The computer may rearrange data in storage by sorting or combining different types of information received from a number of input units. The computer may also take the original data from storage, calculate new information, and place the result back in storage.

The size or capacity of storage determines the amount of information that can be held within the system at any one time. In some computers, storage capacity is measured in millions of digits or characters (bytes), providing space to retain entire files of information. In other systems, storage is smaller, and data is held only while being processed. Consequently, the capacity and design of storage affect the method in which data is handled by the system.

In System/370, *main storage* is thought of as consisting of the following: main data storage, (also referred to as *real* storage), which may vary in size from 65,536 to over eight million characters of programs and other data; control storage, which often contains special built-in *microprograms* to assist the computer in carrying out its own operations; and local storage, consisting of high-speed working areas (registers) for floating-point arithmetic, fixed-point arithmetic, and other types of processing.

In addition, much more storage is provided by the direct access storage devices described in a later section. These direct access devices and the tape units provide what is sometimes called secondary storage or *auxiliary storage.*

Storage is designed in such a way that information can be put there in many forms - as complete records, portions of records, digits, symbols, characters, code patterns, signals, and so on. However, capacity is usually stated in characters, meaning letters of the alphabet, digits, and special symbols of accounting, scientific notation, and report writing. In System/370, the word *byte* is used instead of "character". It is possible to *pack* two numeric digits into the same storage space that is required for letters of the alphabet, special characters, and the other symbols usually referred to as characters.

Input and Output Devices

The data processing system requires, as a necessary part of its information-handling ability, features that can enter data into the system and record data from the system. These functions are performed by input/output devices (Figure 1-10) linked directly to the system.

Input devices *read* or sense coded data that is recorded on a prescribed medium and make this information available to the computer. Data for input is recorded in cards and paper tape as punched holes, on magnetic tape as magnetized spots along the length of the tape, on paper documents as characters or line drawings created with the light pen and associated keyboards, etc.

The method of recording data for machine use and the characteristics of each medium are discussed in later sections.

Output devices record or *write* information from the computer on cards, paper tape, and magnetic tape; they print information on paper; generate signals from transmission over teleprocessing networks; produce graphic displays, microfilm images, and take other specialized forms. The number and type of input/output devices connected directly to the computer depend on the design of the system and its application.

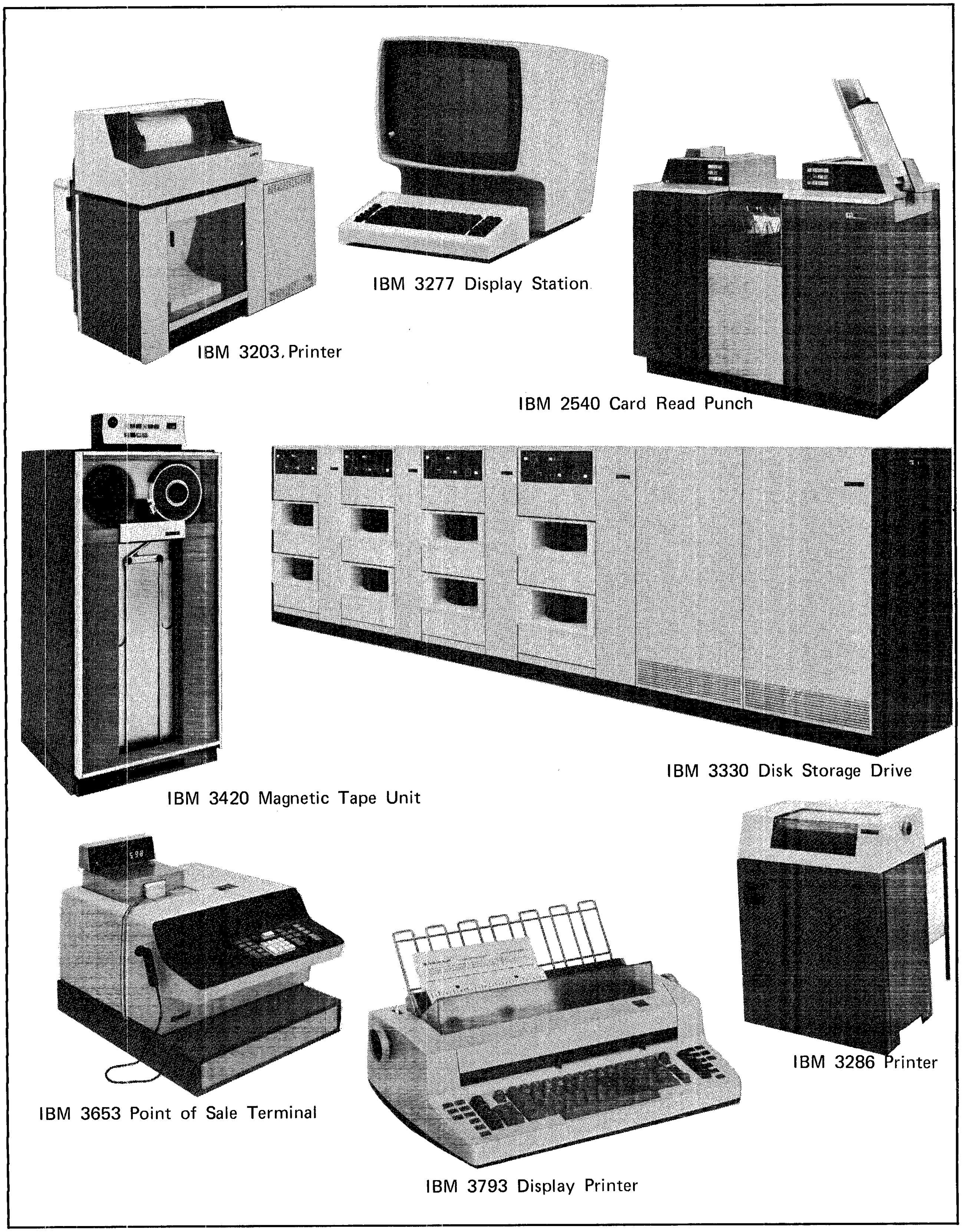


Figure 1-10 Input/Output Devices

Console

The console (Figure 1-11) is an input/output device that provides external control of the data processing system. Keys turn power on or off, start or stop operation, and control various devices in the system. Data may be entered directly by manually depressing keys. Lights are provided so that data on the system may be displayed visually.

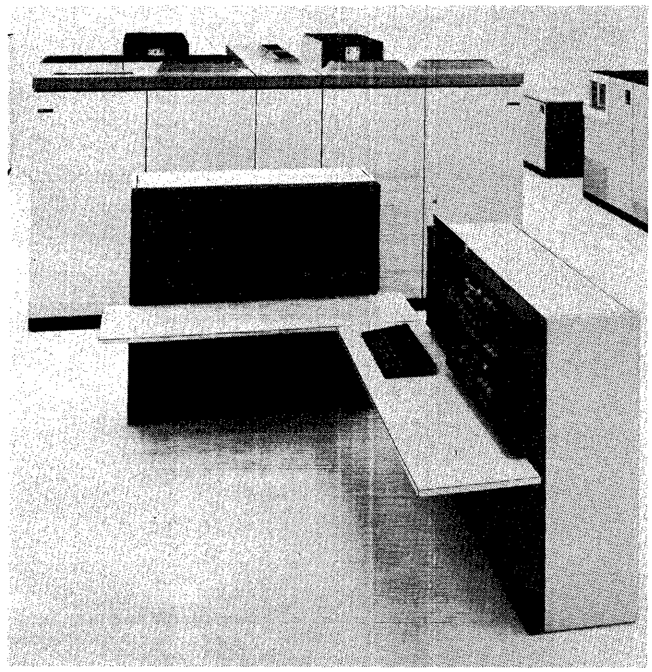


Figure 1-11 IBM System/370 Model 168 with IBM 3066 System Console

On some systems, a console printer (and/or display screen) and keyboard provide limited output or input. The input/output device may print or display messages, signaling the end of processing or an error condition. It may also print or display totals or other information that enables the operator to monitor and supervise operation, or it may give instructions to the operator. On the other hand, it may be used to key in meaningful information (such as altering instructions) to a data processing system that is programmed to respond to such messages.

A remote console (Figure 1-12) may offer increased efficiency and flexibility by providing duplicate operator controls at a station removed from the processing unit.

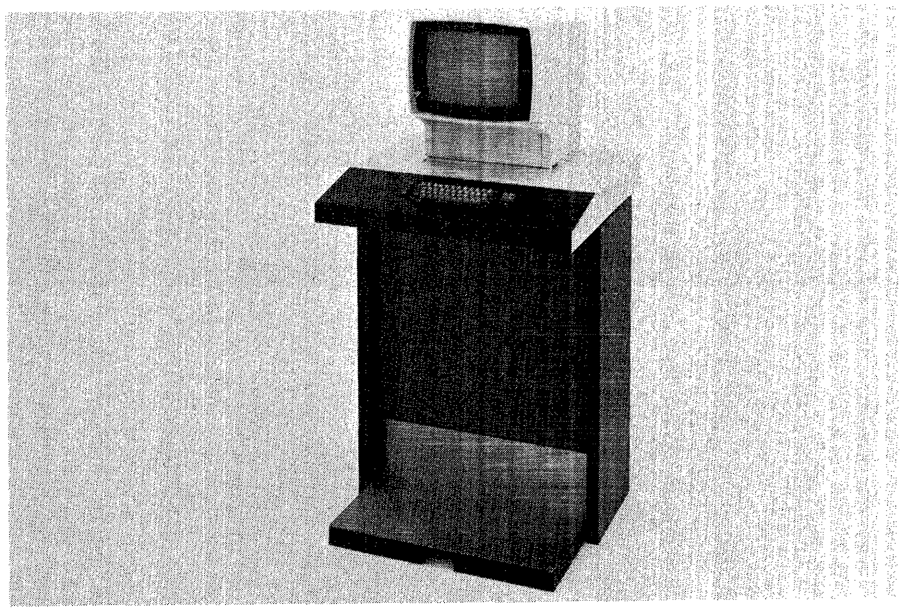


Figure 1-12 IBM 3056 Remote System Console

System/3

IBM System/3 (Figure 1-13) is a low-cost, general-purpose system for commercial data processing and interactive problem solving. The system offers integrated card processing, stored-program capability, calculating and logic capabilities, and the flexibility of disk storage. In addition, local and binary synchronous communications adapters are available for attaching a wide variety of teleprocessing devices. System/3, which is primarily for small business, extends the use of stored-program data processing to the small data processing user.

Processor (main) storage ranges in capacity from 8,192 bytes to 262,144 bytes. All models of System/3 can communicate with System/370 over appropriate communications lines.

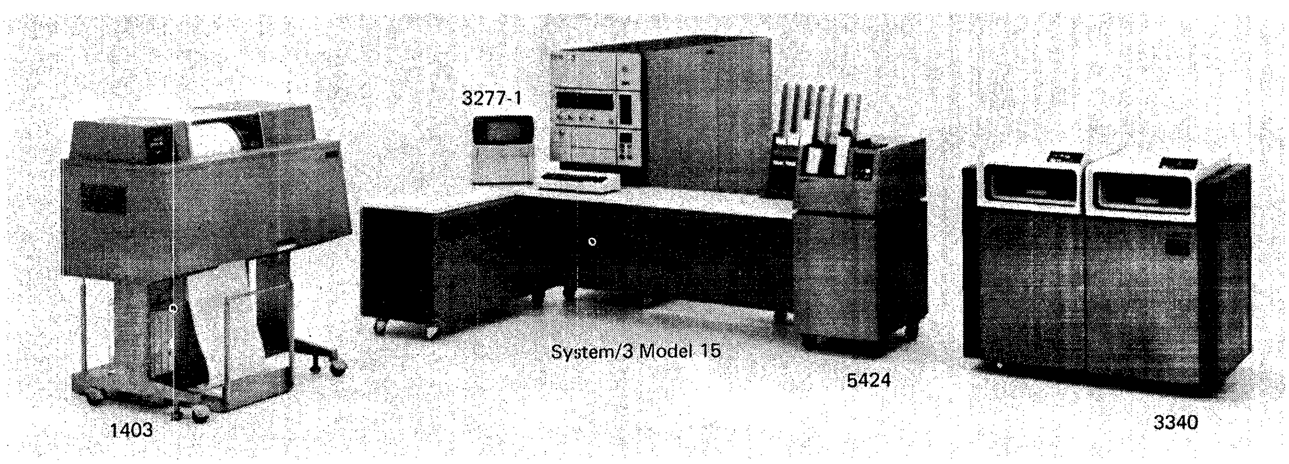


Figure 1-13 IBM System/3 Model 15

System 32

IBM System/32 (Figure 1-14) is a desk-size, operator-controlled, general-purpose computing system, designed for use as a data processing tool in a wide variety of industries. It is composed of the IBM 5320 System Unit, which features keyboard data entry and disk processing. Components of the 5320 include a keyboard, a processing unit, disk storage, a display screen, a diskette drive, and a printing unit.

Processor storage ranges in capacity from 16,384 bytes to 32,768 bytes. With appropriate attachments, System/32 can communicate with System/370.

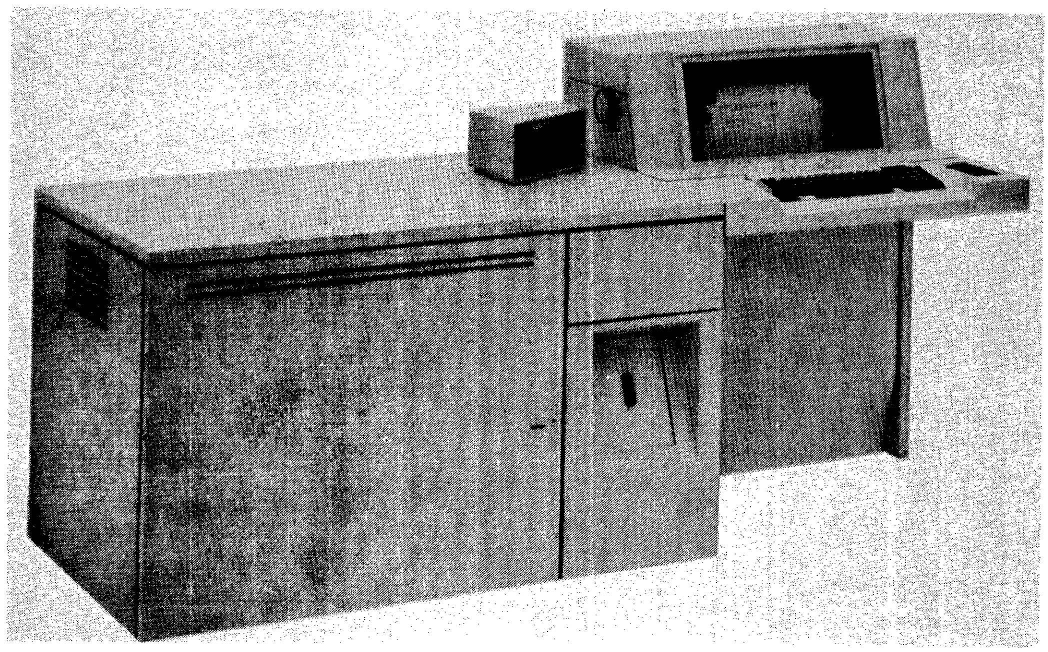


Figure 1-14 IBM System/32

IBM 5100

The IBM *5100 Portable Computer* (Figure 1-15) provides the scientist or engineer with personal, local computing using an *interactive* high-level language, either APL or BASIC. With maximum memory in the machine, the user has an active workspace of approximately 60K bytes; and with one or two tape cassettes, an additional 200K or 400K bytes are at his disposal.

In addition to the standard keyboard, character screen, and tape cassette unit, the 5100 has available as optional equipment a second tape cassette unit, a printer, a communications adapter, and a serial input/output (SIO) adapter. The communications adapter enables the 5100 to be used as a terminal in conjunction with large, interactive systems. The SIO adapter provides for the attachment and local control of devices with a communications feature.

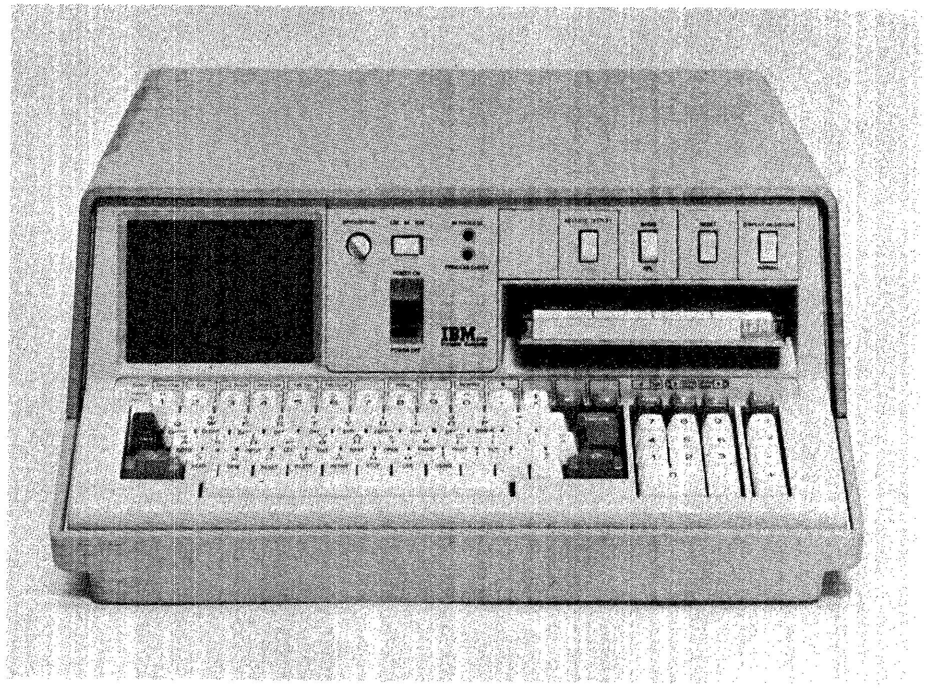


Figure 1-15 IBM 5100 Portable Computer

Section 2:

Data Representation

Symbols convey information; the symbol itself is not the information but merely represents it. The printed characters on this page are symbols that convey one meaning to some persons, a different meaning to others, and no meaning to those who do not know their significance (Figure 2-1).

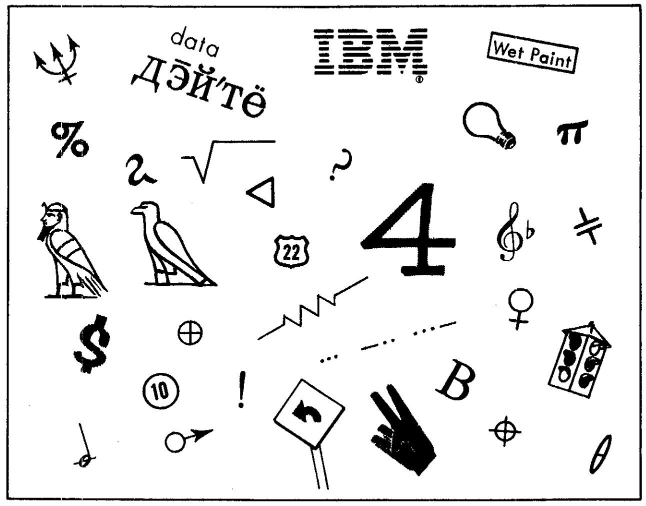


Figure 2-1 Symbols for Communication

Presenting data to the computer system is similar in many ways to communicating with another person by letter. The intelligence to be conveyed must be reduced to a set of symbols. In the English language, these are the familiar letters of the alphabet, numbers and punctuation. The symbols are recorded on paper in a prescribed sequence and transported to another person who reads and interprets them.

Similarly, communication with the computer system requires that data be reduced to a set of symbols that can be read and interpreted by data processing machines. The symbols differ from those commonly used by people, because the information to be represented must conform to the design and operation of the machine. The choice of these symbols (and their meaning) is a matter of convention on the part of the designers. The important fact is that information can be represented by symbols, which become a language for the communication between people and machines.

Information to be used with the computer systems can be in the form of punched cards, paper tape, magnetic tape, direct access storage devices *(DASD)*, magnetic ink characters, optically recognizable characters, microfilm and display screen images, communication network signals, etc. The list is growing larger each year. Some are pictured in Figure 2-2.

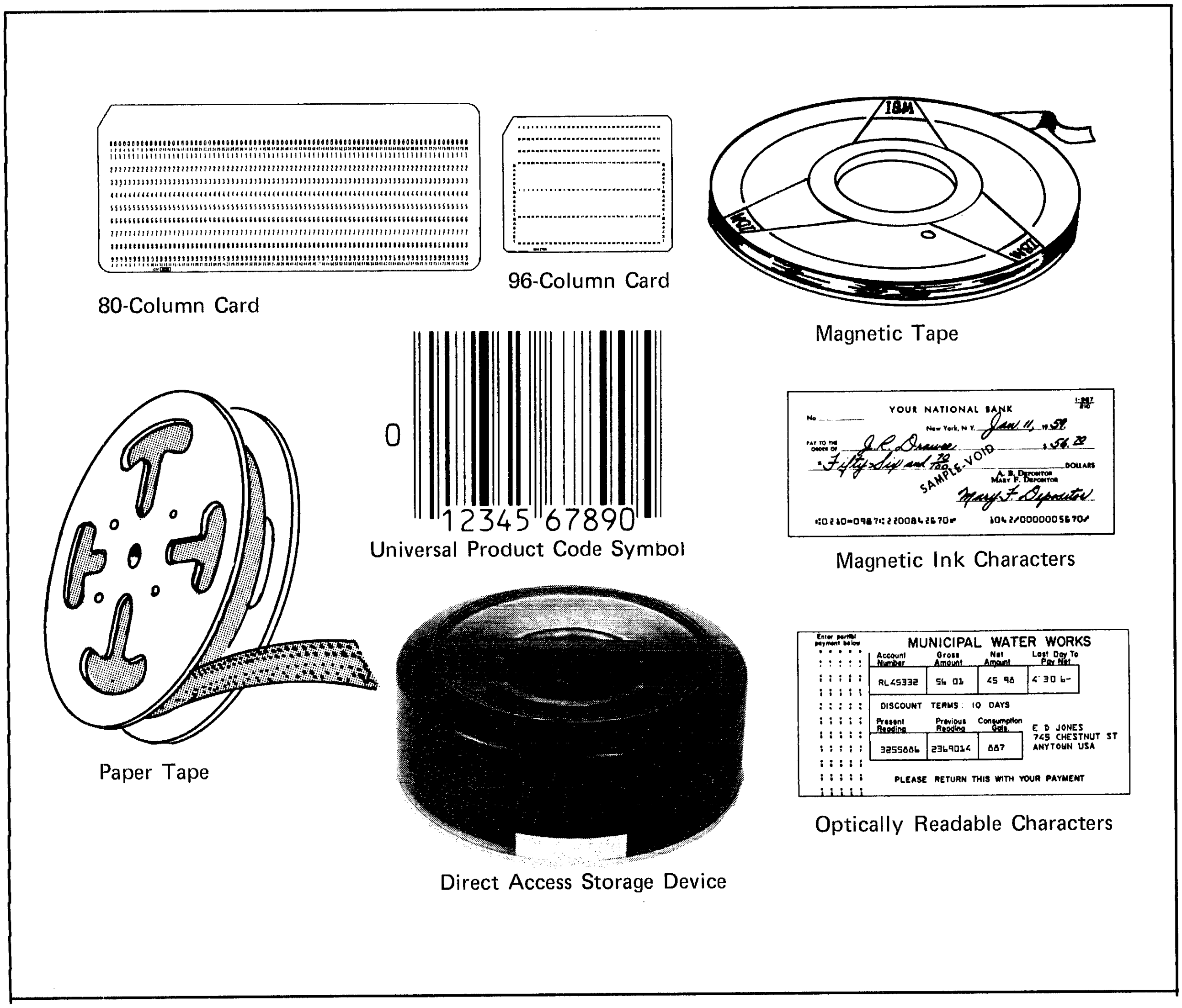


Figure 2-2 Data Recording Media

Data is represented on the punched card by the presence of small holes in specific locations of the card. In a similar manner, small circular holes along a paper tape represent data. On magnetic tape or on DASD, the symbols are small magnetized areas, called spots or bits, arranged in specific patterns. Magnetic ink characters are printed on paper. The shape of the characters and the magnetic properties of the ink permit the printed data to be read by both man and machine. The shape of the optical characters, together with the contrast with the background paper, permits optical characters to be read by the machine and by people.

Each medium requires a code or specific arrangement of symbols to represent data. These codes are described later in this section.

An input device of the computer system is a machine designed to sense or read information from one of the recording media. In the reading process, recorded data is converted to, or symbolized in, electronic form; the data can then be used by the machine for data processing operations.

An output device is a machine that receives information from the computer system and records it on the designated output medium.

All input/output devices cannot be used directly with all computer systems. However, data recorded on one medium can be transcribed to another medium for use with a different system. For example, data on cards or paper tape can be transcribed onto magnetic tape. Conversely, data on magentic tape can be converted to cards, paper tape, printed reports, or plotted graphs.

As there is communication between people and machines, there is also communication from one machine to another (Figure 2-3). This intercommunication may be the direct exchange of data (in electronic form) over wires, cables, or radio waves, or recorded output of one machine or system to be used as input to another machine or system.

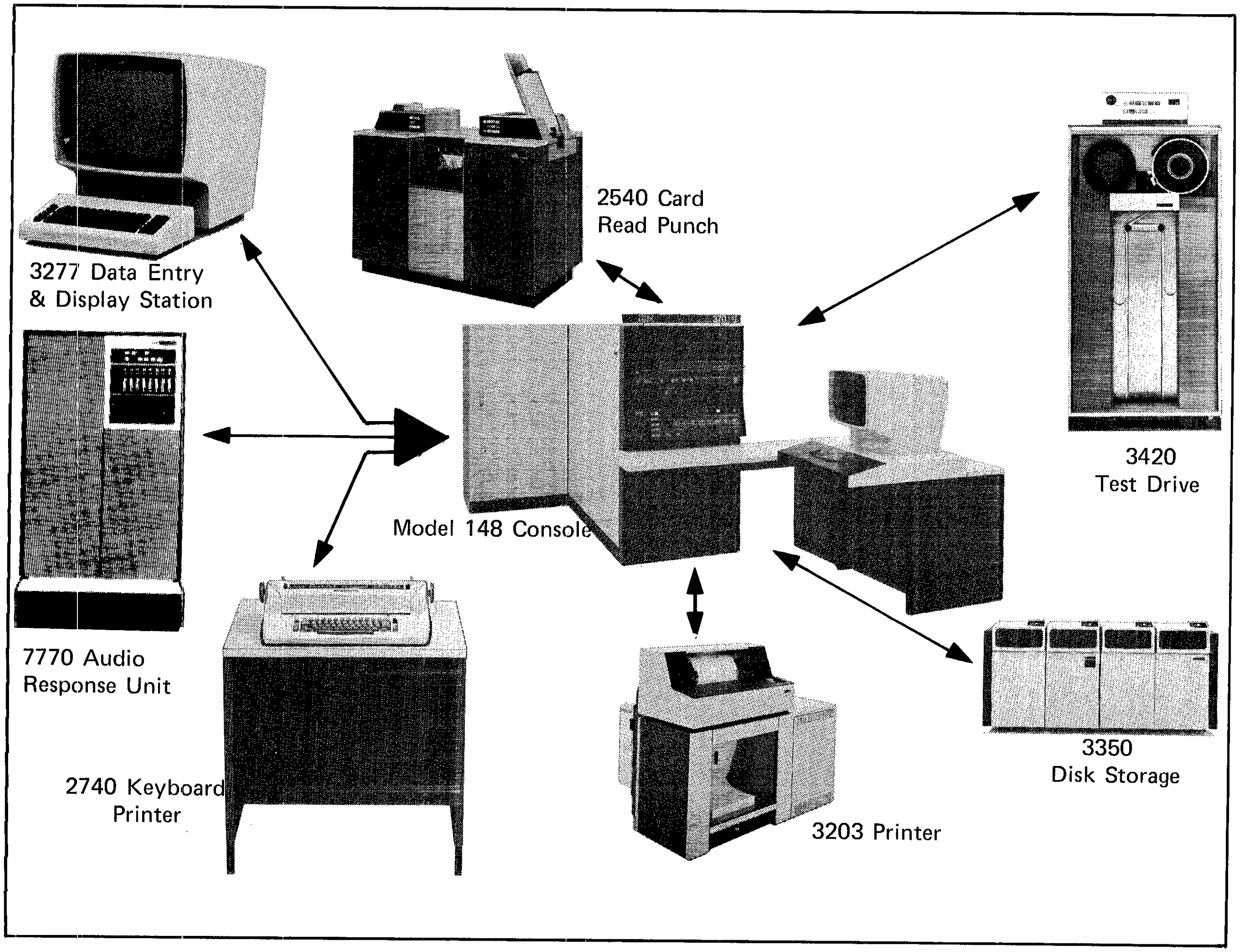


Figure 2-3 Machine-to-Machine Communication

Computer Data Representation

Not only must there be a method of representing data on cards, on paper tape, on magnetic tape, and in magnetic ink characters, but there must also be a method of representing data within a machine.

In the computer, data is represented by many electronic components: transistors, magnetic cores, wires, and so on. The storage and flow of data through these devices are represented as electronic signals or indications. The presence or absence of these signals in specific circuitry is the method of representing data, much as the presence of holes in a card represents data.

Binary States

Computers function in *binary* states; this means that the computer components can indicate only two possible states or conditions. For example, the ordinary light bulb operates in a binary mode, that is, it is either on or off. Likewise, within the computer, transistors are maintained either conducting or nonconducting; magnetic materials are magnetized in one direction or in an opposite direction; and specific voltage potentials are present or absent (Figure 2-4). The binary states of operation of the components are signals to the computer, as the presence or absence of light from an electric light bulb can be a signal to a person.

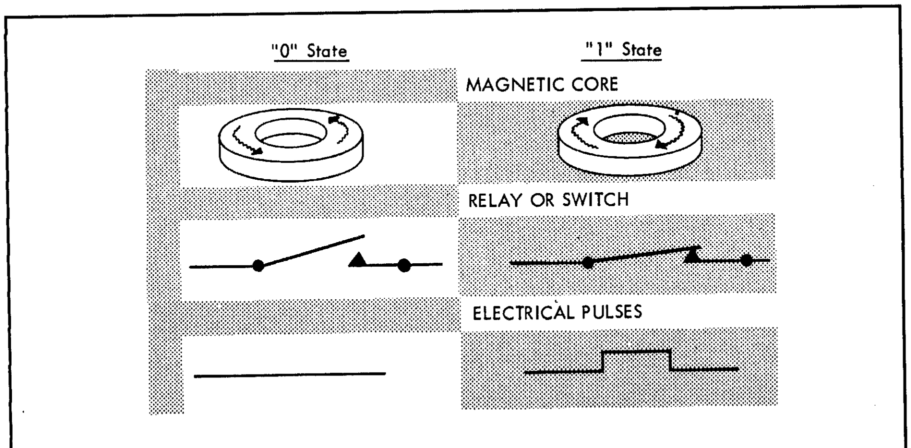


Figure 2-4 Binary Components

Representing data within the computer is accomplished by assigning or associating a specific value to a binary indication or group of binary indications. For example, a device to represent decimal values could be designed with four electric light bulbs and switches to turn each bulb on or off (Figure 2-5).

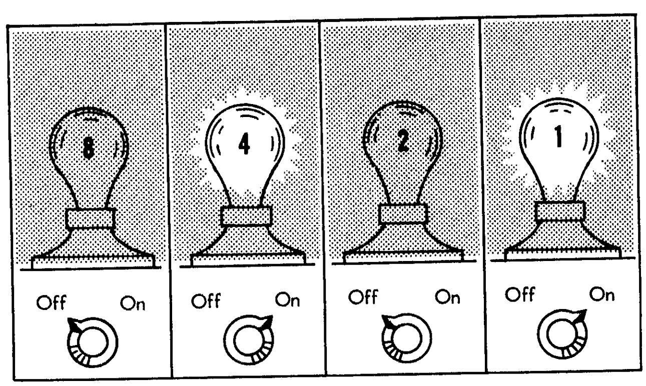


Figure 2-5 Representing Decimal Data with Binary Components

The bulbs are assigned decimal values of 1, 2, 4, and 8. When a light is on, it represents the decimal value associated with it. When a light is off, the decimal value is not considered. With such an arrangement, the single decimal value represented by the four bulbs will be the numeric sum indicated by the lighted bulbs.

Decimal values 0 through 15 can be represented. The numeric value 0 is represented by all lights off; the value 15, by all lights on; 9, by having the 8 and 1 lights on and the 4 and 2 lights off; 5, by the 1 and 4 lights on and the 8 and 2 lights off; and so on.

The value assigned to each bulb or indicator in the example could have been something other than the values used. This change would involve assigning new values and determining a new scheme of operation. In a computer, the values assigned to a specific number of binary indications become the code or language for representing data.

Because binary indications represent data within a computer, a binary method of notation is used to illustrate these indications. The binary system of notation uses only two symbols, zero (0) and one (1), to represent specific values. In any one position of binary notation, the 0 represents the absence of a related or assigned value, and the 1 represents the presence of a related or assigned value. For example, to illustrate the indications of the light bulb in Figure 2-5, the following binary notation would be used: 0101.

The binary notations 0 and 1 are commonly called *bits*. Properly, they are called 0 bit and 1 bit. Occasionally, however, they are loosely spoken of as no bit (0 bit) and bit (1 bit). For example, the binary notation 0101 of Figure 2-5 would be described as having a 1 bit in the 1 and 4 bit positions and a 0 bit in the 2 and 8 bit positions.

Binary Number System

In some computers, the values associated with the binary notation are related directly to the binary system. This system is not used in all computers, but the method of representing values using this numbering system is useful in learning the general concept of data representation.

The common decimal number system uses ten symbols or digits to represent numeric values, and the place value of the digits signifies units, tens, hundreds, thousands, and so on. The binary or base-two number system uses only two symbols or digits: 0 and 1. The position value of the bit symbols (0 or 1) is based on the progression of powers of 2; the units position of a binary number has the value of 1; the next position, a value of 2; the next, 4; the next, 8; the next, 16; and so on (Figure 2-6).

8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Figure 2-6 Place Value of Binary Numbers

In pure binary notation, the binary digits or bits indicate whether the corresponding power of 2 is absent or present in each position of the number. The 1 bit represents the presence of the value, and the 0 bit represents the absence of the value. The place value of the digits does not signify units, tens, hundreds, or thousands, as in the decimal system; instead, the place value signifies units, twos, fours, eights, sixteens, and so on. Using this system, the quantity 12, for example is expressed with the symbols 1100, meaning $(1x2^3)+(1x2^2)+(0x2^1)+(0x2^0)$, or $(1x8) + (1x4) + (0x2) + (0x1)$.

Figure 2-7 shows the binary representation of the decimal values 0 through 16.

Decimal Value	Place Value 16	8	4	2	1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0

Figure 2-7 Binary Representation of Decimal Values 0-16

Note that the decimal digits 0 through 9 are expressed by four binary digits. The system of coding or expressing decimal digits in an equivalent binary value is known as *binary coded decimal (BCD)*. For example, the decimal value 265,498 would appear in binary coded decimal form as shown in Figure 2-8. This is also known as *packed decimal* digits.

Decimal Digits	2	6	5	4	9	8
Binary Value	0 0 1 0	0 1 1 0	0 1 0 1	0 1 0 0	1 0 0 1	1 0 0 0
Place Value	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1

Figure 2-8 Binary Coded Decimal Representation of Decimal Number 265,498

Computer Codes

The method used to represent (symbolize) data is known as a *code* or a system. In the computer, the code relates data to a fixed number of binary indications (symbols). For example, a code used to represent numeric and alphabetic characters may use eight positions of binary indication. By the proper arrangement of the binary indications (0 bit, 1 bit), all characters can be represented by a different combination of bits.

Some computer codes in use are six-bit *alphameric* (alphabetic, numeric, and special symbol) code, eight-bit alphameric code, two-out-of-five count code, seven-bit alphameric code, and six-bit (packed) numeric code.

Code Checking

Most computer codes are self-checking; that is, they are provided with a built-in method of checking the validity of the coded information. This code checking occurs automatically within the machine as the data processing operations are carried out. The method of validity checking is part of the design of the code.

In some codes, each unit or character of data is represented by a specific number of bit positions that must always contain an even number of 1 bits. Different characters are made up of different combinations of 1 bits, but the number of 1 bits in any valid character is always even. With this code system, a character with an odd number of 1 bits is detected, and an error is indicated. Likewise, a code may be used in which all characters must have an odd number of 1 bits; an error is indicated when characters with an even number of 1 bits are detected.

This type of checking is known as a *parity check*. Codes that use an even number of 1 bits are said to have even parity. Codes that use an odd number of bits are said to have odd parity.

In other codes, the number of 1 bits present in each unit of data is fixed. For example, a code may use eight bit positions to code all characters, but exactly four 1 bits will be present in each character. Characters having more or fewer than four 1 bits cause an error indication. This system of checking is known as a *fixed-count check* and is often used for data transmission in teleprocessing networks.

Six-Bit Alphameric Code (Binary Coded Decimal System)

In this code, all characters - numeric, alphabetic, and special - are represented (coded) using six positions of binary notation (plus a parity bit position). These positions are divided into three groups: one check position, two zone positions, and four numeric positions (Figure 2-9).

Check Bit	Zone Bits		Numeric Bits			
C	B	A	8	4	2	1

Figure 2-9 Bit Positions, Six-Bit Alphameric Code

The four numeric positions are assigned decimal values of 8, 4, 2, and 1, and represent, in binary coded decimal form, the numeric digits 0 through 9 (Figure 2-10). Note that 0 is represented as 1010, actually the binary number for 10. The B and A zone bits are not present (00) when the numeric digits 0 through 9 are represented.

Decimal Digit	Place Value 8	4	2	1
0	1	0	1	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Figure 2-10 Numeric Bit Configurations, Decimal Digits 0-9, Six-Bit Alphameric Code

Combinations of zone and numeric bits represent alphabetic and special characters. The B and A bits provide for four possible bit combinations: 10, 01, 11, and 00.

The C position, known as the check bit is used for code checking only. Because the six-bit alphameric code is usually an even parity code, the total number of bits used to represent a character must have an even number of 1 bits, or the character is considered invalid. A 1 bit is added to a character when the sum of the zone and numeric bits used to represent the character is odd. If the number of bits in a character is even without the C bit, the C bit is a 0.

Standard BCD Interchange Code

To provide compatibility of data for interchangeability among various computer systems, the standard BCD interchange code has been developed. This coding structure consists basically of 64 different characters. Figure 2-11 shows the BCD standard interchange code used on older IBM data processing systems and seven-track IBM tape units.

COLLATING SEQUENCE (Low → High)

CHARACTER Report	CHARACTER Program	CARD CODE	BCD CODE (Core Storage)						
b		No Punches	C						
.		12-3-8		B	A	8		2	1
⌑	)	12-4-8	C	B	A	8	4		
[		12-5-8		B	A	8	4		1
<		12-6-8		B	A	8	4	2	
⧧		12-7-8	C	B	A	8	4	2	1
&	+	12	C	B	A				
$		11-3-8	C	B		8		2	1
*		11-4-8		B		8	4		
]		11-5-8	C	B		8	4		1
;		11-6-8	C	B		8	4	2	
△		11-7-8		B		8	4	2	1
—		11		B					
/		0-1	C		A				1
,		0-3-8	C		A	8		2	1
%	(	0-4-8			A	8	4		
∼		0-5-8	C		A	8	4		1
\		0-6-8	C		A	8	4	2	
⧻		0-7-8			A	8	4	2	1
ƀ		2-8			A				
#	=	3-8				8		2	1
@	'	4-8	C			8	4		
:		5-8				8	4		1
>		6-8				8	4	2	
√		7-8	C			8	4	2	1
?		12-0	C	B	A	8		2	
A		12-1		B	A				1
B		12-2		B	A			2	
C		12-3	C	B	A			2	1
D		12-4		B	A		4		
E		12-5	C	B	A		4		1
F		12-6	C	B	A		4	2	
G		12-7		B	A		4	2	1
H		12-8		B	A	8			
I		12-9	C	B	A	8			1
!		11-0		B		8		2	
J		11-1	C	B					1
K		11-2	C	B				2	
L		11-3		B				2	1
M		11-4	C	B			4		
N		11-5		B			4		1
O		11-6		B			4	2	
P		11-7	C	B			4	2	1
Q		11-8	C	B		8			
R		11-9		B		8			1
‡		0-2-8			A	8		2	
S		0-2	C		A			2	
T		0-3			A			2	1
U		0-4	C		A		4		
V		0-5			A		4		1
W		0-6			A		4	2	
X		0-7	C		A		4	2	1
Y		0-8	C		A	8			
Ƶ		0-9			A	8			1
Ø		0	C			8		2	
1		1							1
2		2						2	
3		3	C					2	1
4		4					4		
5		5	C				4		1
6		6	C				4	2	
7		7					4	2	1
8		8				8			
9		9	C			8			1

NOTE: Tape may use even parity.

Figure 2-11 Standard BCD Interchange Code

Shown are the *collating sequence* (ascending sequence), graphics, card code, and BCD code for each of the 64 different bit combinations. The C bit, used for parity checking purposes, is dependent upon the specific computer design using the standard BCD interchange code. If the system uses odd parity, a C bit will be automatically placed in each C position of a character that contains an even number of bits. Conversely, a system using even parity will have a C bit placed in each C position of a character that contains an odd number of bits.

Five of the standard BCD bit combinations print out as two different characters (called graphics), depending upon the type set used in the printer. The two variations are called graphic subset 1 and graphic subset 2. (Figure 2-12).

BCD Code	Graphic Subset 1 Print Arrangement A	Graphic Subset 2 Print Arrangement H
8-2-1	#	=
8-4	@	'
A-8-4	%	(
B-A	&	+
B-A-8-4	⌑	)

Figure 2-12 Graphic Subsets 1 and 2

Graphic subset 1 is used primarily for computer report writing and most commercial uses, while graphic subset 2 is used for such programming languages as FORTRAN and COBOL, and meets general requirements for mathematical symbolism. Figure 2-13 indicates the preferred standardized terminology for special characters.

SYMBOL	NAME	SYMBOL	NAME
¢	Cent Sign	>	Greater-than Sign
.	Period, Decimal Point	?	Question Mark
<	Less-than Sign	:	Colon
(	Left Parenthesis	#	Number Sign
+	Plus Sign	@	At Sign
|	Logical OR	'	Prime, Apostrophe
&	Ampersand	=	Equal Sign
!	Exclamation Point	"	Quotation Mark
$	Dollar Sign	⑁	Chair
*	Asterisk	⑂	Fork
)	Right Parenthesis	⑀	Hook
;	Semicolon	|	Vertical Line
¬	Logical NOT	`	Accent Grave
-	Minus Sign, Hyphen	~	Tilde
/	Slash	{	Opening Brace
,	Comma	}	Closing Brace
%	Percent	\	Reverse Slant
_	Underscore		

Figure 2-13 Special Graphic Characters

Eight-Bit Alphameric Code (Extended Binary Coded Decimal Interchange Code - EBCDIC)

This code (Figure 2-14) uses eight binary positions for each character format, plus a position for parity checking. By using eight bit positions, 256 different characters can be coded. This code permits, for instance, the coding of uppercase and lowercase alphabetic characters, a much wider range of special characters, and many control characters that are meaningful to certain input/output devices. At present, many bit patterns have no assigned function (control or graphic). They are reserved for future assignment. EBCDIC is the principal alphameric coding scheme for future assignment. EBCDIC is the principal alphameric coding scheme for System/360 and System/370. The internal coding structure of System/3 and System 32 is EBCDIC.

EBCDIC	Bit Configuration	ASCII-8
NUL	0000 0000	NUL
SOH	0000 0001	JOH
STX	0000 0010	STX
ETX	0000 0011	ETX
PF	0000 0100	EOT
HT	0000 0101	ENQ
LC	0000 0110	ACK
DEL	0000 0111	BEL
GE	0000 1000	BS
RLF	0000 1001	HT
SMM	0000 1010	LF
VT	0000 1011	VT
FF	0000 1100	FF
CR	0000 1101	CR
SO	0000 1110	SO
SI	0000 1111	SI
DLE	0001 0000	DLE
DC1	0001 0001	DC1
DC2	0001 0010	DC2
TM(DC3)	0001 0011	DC3
RES	0001 0100	DC4
NL	0001 0101	NAK
BS	0001 0110	SYN
IL	0001 0111	ETB
CAN	0001 1000	CAN
EM	0001 1001	EM
CC	0001 1010	SUB
CU1	0001 1011	ESC
IFS	0001 1100	FS
IGS	0001 1101	GS
IRS	0001 1110	RS
IUS	0001 1111	US
DS	0010 0000	SPACE
SOS	0010 0001	!
FS	0010 0010	"
	0010 0011	#
BYP	0010 0100	$
LF	0010 0101	%
ETB	0010 0110	&
ESC	0010 0111	'
	0010 1000	(
	0010 1001	)
SM	0010 1010	*
CU2	0010 1011	+
	0010 1100	,
ENQ	0010 1101	–
ACK	0010 1110	.
BEL	0010 1111	/
	0011 0000	0
	0011 0001	1
SYN	0011 0010	2
	0011 0011	3
PN	0011 0100	4
RS	0011 0101	5
UC	0011 0110	6
EOT	0011 0111	7
	0011 1000	8
	0011 1001	9
	0011 1010	:
CU3	0011 1011	;
DC4	0011 1100	<
NAK	0011 1101	=
	0011 1110	>
SUB	0011 1111	?
SPACE	0100 0000	@
	0100 0001	A
	0100 0010	B
	0100 0011	C
	0100 0100	D
	0100 0101	E
	0100 0110	F
	0100 0111	G
	0100 1000	H
	0100 1001	I
¢	0100 1010	J
.	0100 1011	K
<	0100 1100	L
(	0100 1101	M
+	0100 1110	N
|	0100 1111	O
&	0101 0000	P
	0101 0001	Q
	0101 0010	R
	0101 0011	S
	0101 0100	T
	0101 0101	U
	0101 0110	V
	0101 0111	W
	0101 1000	X
	0101 1001	Y
!	0101 1010	Z
$	0101 1011	[
*	0101 1100	\
)	0101 1101	]
;	0101 1110	¬
¬	0101 1111	_
–	0110 0000	`
/	0110 0001	a
	0110 0010	b
	0110 0011	c
	0110 0100	d
	0110 0101	e
	0110 0110	f
	0110 0111	g
	0110 1000	h
	0110 1001	i
¦	0110 1010	j
,	0110 1011	k
%	0110 1100	l
_	0110 1101	m
>	0110 1110	n
?	0110 1111	o
	0111 0000	p
	0111 0001	q
	0111 0010	r
	0111 0011	s
	0111 0100	t
	0111 0101	u
	0111 0110	v
	0111 0111	w
	0111 1000	x
`	0111 1001	y
:	0111 1010	z
#	0111 1011	{
@	0111 1100	¦
'	0111 1101	}
=	0111 1110	~
"	0111 1111	DEL
	1000 0000	
a	1000 0001	
b	1000 0010	
c	1000 0011	
d	1000 0100	
e	1000 0101	
f	1000 0110	
g	1000 0111	
h	1000 1000	
i	1000 1001	
	1000 1010	
	1000 1011	
	1000 1100	
	1000 1101	
	1000 1110	
	1000 1111	
	1001 0000	
j	1001 0001	
k	1001 0010	
l	1001 0011	
m	1001 0100	
n	1001 0101	
o	1001 0110	
p	1001 0111	
q	1001 1000	
r	1001 1001	
	1001 1010	
	1001 1011	
	1001 1100	
	1001 1101	
	1001 1110	
	1001 1111	
	1010 0000	
~	1010 0001	
s	1010 0010	
t	1010 0011	
u	1010 0100	
v	1010 0101	
w	1010 0110	
x	1010 0111	
y	1010 1000	
z	1010 1001	
	1010 1010	
	1010 1011	
	1010 1100	
	1010 1101	
	1010 1110	
	1010 1111	
	1011 0000	
	1011 0001	
	1011 0010	
	1011 0011	
	1011 0100	
	1011 0101	
	1011 0110	
	1011 0111	
	1011 1000	
	1011 1001	
	1011 1010	
	1011 1011	
	1011 1100	
	1011 1101	
	1011 1110	
	1011 1111	
{	1100 0000	
A	1100 0001	
B	1100 0010	
C	1100 0011	
D	1100 0100	
E	1100 0101	
F	1100 0110	
G	1100 0111	
H	1100 1000	
I	1100 1001	
	1100 1010	
	1100 1011	
ʃ	1100 1100	
	1100 1101	
Ψ	1100 1110	
	1100 1111	
}	1101 0000	
J	1101 0001	
K	1101 0010	
L	1101 0011	
M	1101 0100	
N	1101 0101	
O	1101 0110	
P	1101 0111	
Q	1101 1000	
R	1101 1001	
	1101 1010	
	1101 1011	
	1101 1100	
	1101 1101	
	1101 1110	
	1101 1111	
	1110 0000	
	1110 0001	
S	1110 0010	
T	1110 0011	
U	1110 0100	
V	1110 0101	
W	1110 0110	
X	1110 0111	
Y	1110 1000	
Z	1110 1001	
	1110 1010	
	1110 1011	
⑁	1110 1100	
	1110 1101	
	1110 1110	
	1110 1111	
0	1111 0000	
1	1111 0001	
2	1111 0010	
3	1111 0011	
4	1111 0100	
5	1111 0101	
6	1111 0110	
7	1111 0111	
8	1111 1000	
9	1111 1001	
LVM	1111 1010	
	1111 1011	
	1111 1100	
	1111 1101	
	1111 1110	
EO	1111 1111	

Figure 2-14 Configurations, Extended Binary Coded Decimal Interchange Code (EBCDIC)

Eight-Bit Alphameric Code (ASCII-8)

The American National Standard Code for Information Interchange (ASCII) is a seven-bit code developed through the cooperation of users of equipment of communications and data processing industries, in an attempt to simplify and standardize machine-to-machine and system-to-system communication. ASCII offers 128 possible characters. Because the System/360 and System/370 have an eight-bit character capacity (with 256 possible characters), an eight-bit representation of ASCII referred to as ASCII-8 was proposed and was made available on System/360. This eight-bit code has since been rejected and is not available on System/370.

Computer Number Systems and Conversion

Binary System

Computers using the binary system of data representation are typified by the IBM System/370.

For these systems, the basic unit of information is a *byte*. Four bytes constitute a word consisting of 32 consecutive bit positions of information which are interpreted as a unit, much as a character or a digit in other systems.

The bit sections within the word have a place significance related to the binary number system. That is, the place position of a bit in the word determines the value of the bit. In the binary number system, the decimal values of the places (from right to left) are 1, 2, 4, 8, 16, 32, 64, and so on as shown in Figure 2-6.

Although the place values of the bits of a word are always those of the binary number system, they can be interpreted or processed in such a way as to represent other than a binary number. For example, a 32-bit word (Figure 2-15) can be interpreted as one 32-place binary number, as an eight-digit hexadecimal (base 16) number, as four alphameric characters (which may be alphabetic or numeric), or as any predetermined representation established by the programmer.

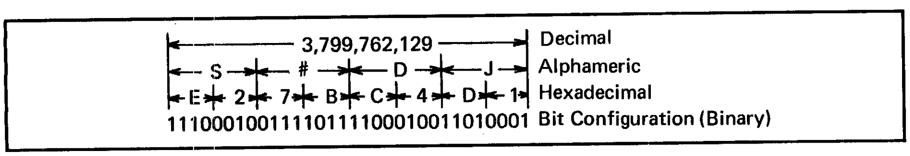


Figure 2-15 The 32 Bit Word

Hexadecimal System

It is apparent that binary numbers require several times as many positions as decimal numbers to display the equivalent number. In talking and writing, these binary numbers are bulky. A long string of ones and zeros cannot be effectively transmitted from one individual to another. Some shorthand method is necessary. The *hexadecimal* number system fills this need. Because of its simple relationship to binary, numbers can be converted from one system to another by inspection. The base or radix of the hexadecimal system is 16. This means that six more symbols are required than in the decimal system. The symbols used are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. It takes four bits to count to F (15) in a computer. Figure 2-16 shows how it is done.

DECIMAL SYSTEM	HEXADECIMAL SYSTEM	BINARY SYSTEM 8 4 2 1 Bit values
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
10	A	1 0 1 0
11	B	1 0 1 1
12	C	1 1 0 0
13	D	1 1 0 1
14	E	1 1 1 0
15	F	1 1 1 1

Figure 2-16 Relationship Among Decimal, Hexadecimal, and Binary Systems

Remember that the internal circuitry of the computer understands only binary ones and zeros. The hexadecimal system is used to provide a shorthand method of reading and writing binary numbers. In the hexadecimal system, the base number is 16. The digits of the number represent the coefficients of the ascending powers of 16. Consider the hexadecimal (hex) number:

$$173 = (1 \times 16^2) + (7 \times 16^1) + (3 \times 16^0)$$
$$= 256 + 112 + 3$$
$$= 371 \text{ (decimal)}$$

Similarly:

Hex 173

3 units	= 3
7 sixteens	= 112
1 two hundred fifty-sixes	= 256

By remembering what a number represents in the hexadecimal system, the number can be converted to its decimal equivalent by the method shown above. As the numbers get bigger, this method becomes more difficult to use.

The following section provides detailed methods for converting from one system to another with and without the use of reference tables.

Integer Conversion

Hexadecimal to Decimal

Suppose we have a hexadecimal number such as A4B5. How do we convert it to a decimal-system number? First, think of the rightmost (low-order) position as position 1, the next position to the left as position 2, the next position to the left as position 3, and the left-most position as position 4. Looking at Figure 2-17, note that:

5 in hex (hexadecimal) position 1 equals	5
B in hex position 2 equals	176
4 in hex position 3 equals	1,024
A in hex position 4 equals	40,960
	42,165

The sum is the decimal value of A4B5.

Without the use of the table, the rule is as follows: Continuously multiply by 16, and add the next hexadecimal digit (using decimal arithmetic).

```
    A      4      B      5
   10
 x 16
  160
+   4
  164
 x 16
 2624
+  11
 2635
 x 16
42160
+   5
42165
```

The high-order digit is multiplied by 16, and the next-lower-order digit is added to the result. The resultant answer is then multiplied by 16, and the next-lower-order digit is added to the result. When the low-order digit has been added to the answer, the process ends.

Decimal to Hexadecimal

Reversing the procedure, to convert the decimal number 16,428 to hex, look up in Figure 2-17 the next smaller number than 16,428. Note the hex equivalent and position number. Subtract the decimal value of that hex digit from 16,428, and look up the remainder in Figure 2-17. The process works as follows:

Find the hex equivalent of decimal	16,428
4 in hex position 4 equals	16,384
Remainder	44
0 in position 3 equals	0
Remainder	44
2 in position 2 equals	32
Remainder	12
C in position 1 equals	12

Therefore, 402C is the hex equivalent of 16,428.

Without tables, continuously divide the decimal number (using decimal arithmetic) by 16, and develop the hexadecimal number from the remainders of each step of the division.

```
16  |16428
16   |1026    Remainder   12 (C)
16     |64    Remainder    2
16      |4    Remainder    0      =402C
        |0    Remainder    4
```

We first divide the original number to be converted by 16. The remainder of this division becomes the low-order digit of the conversion (12 which is expressed as C). We then divide the quotient (received from the first division) by 16. Again the remainder becomes a part of the answer (next-higher order, 2). This process is continued until the quotient is zero. In the following examples, where multiplication or division is used, detailed explanations will not be used because the proceedings are similar.

Hexadecimal to Binary and Binary to Hexadecimal

Rule: Express the number in binary groups of four starting at the assumed binary point.

Example:

```
Hex to Binary                     Binary to Hex
  4    0    2    C                100000000101100
 0100 0000 0010 1100              0100 0000 0010 1100
= 100000000101100                  4    0    2    C
                                   = 402C
```

Decimal to Binary

Rule: Divide the decimal number by 2, and develop the binary number from the remainders.

Example: Convert 149 to its binary equivalent

```
2 |149                 Remainder 1
 2 |74                 Remainder 0
  2 |37                Remainder 1
   2 |18               Remainder 0
    2 |9               Remainder 1 = 1001 0101
     2 |4              Remainder 0
      2 |2             Remainder 0
       2 |1            Remainder 1
           0
```

Binary to Decimal

Rule: Continuously multiply by 2, and add the next binary digit.

Example: Convert 1001 0101 to its decimal equivalent.

```
  10010101
X 2
  2
+ 0
  2
X 2
  4
+ 0
  4
X 2
  8
+ 1
  9
X 2
 18
+ 0
 18
X 2
 36
+ 1
 37
X 2
 74
+ 0
 74
X 2
148
+ 1
149
```

OR 1001 0101

$= 1\ (2^7) + 0\ (2^6) + 0\ (2^5) + 1\ (2^4) +$

$0\ (2^3) + 1\ (2^2) + 0\ (2^1) + 1\ (2^0)$

$= 128 + 16 + 4 + 1$

$= 149$

Fraction Conversion

Decimal to Hexadecimal

To convert from decimal to hex, find the next lower decimal value and its hex equivalent in Figure 2-18. Subtract the found decimal value from the desired decimal value, and use this to locate the next hex equivalent. Repeat this procedure for the number of positions required.

Example: Convert .13 decimal to hex.

Decimal number to convert	.1300	to	Hex
Next-lower decimal number	.1250	equals	.2
Remainder	.0050 0000		
Next-lower decimal number	.0039 0625		.01
Remainder	.0010 9375 0000		
Next-lower decimal number	.0009 7656 2500		.004
Remainder	.0001 1718 7500		
Next-lower decimal number	.0001 0681 1523 4375		.0007
.13 decimal approximately equals		hex	.2147

In the absence of convenient tables, multiply by 16 (using decimal arithmetic), and develop the hexadecimal number from the carry:

```
.    .1300
      x 16
2    .0800
      x 16
1    .2800
      x 16
4    .4800
      x 16
7    .6800
.2147 hex
```

Hexadecimal to Decimal

To convert from hex fractions to decimal fractions, use Figure 2-18 to find the sum of the decimal equivalents for each position of the fraction.

Example Convert .ABC hex to decimal

.A hex in position 1 equals	.6250
.0B hex in position 2 equals	.0429 6875
.00C hex in position 3 equals	.0029 2968 7500
.ABC hex equals decimal	.6708 9843 7500

HEX	DEC	HEX	DEC	HEX		HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
Hexadecimal Positions	8		7		6		5		4		3		2		1

Figure 2-17 Hexadecimal-Decimal Integer Conversion Table

HEX 0123	DEC	HEX 4 5 6 7	DECIMAL	HEX 0 1 2 3	DECIMAL	HEX 4 5 6 7	DECIMAL EQUIVALENT
.0	.0000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000 0000
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375
Hexadecimal Positions	1		2		3		4

Figure 2-18 Hexadecimal-Decimal Fraction Conversion Table

Without tables, express as powers of 16, and add.

$$
\begin{aligned}
.ABC &= 10\,(16^{-1}) + 11\,(16^{-2}) + 12\,(16^{-3}) \\
&= 10/16 + 11/256 + 12/4096 \\
&= 2748/4096 \\
&= .6708984375
\end{aligned}
$$

Hexadecimal to Binary and Binary to Hexadecimal

Rule: The same rule applies for fractions as for whole numbers.

Example:

```
 .A     B     C    .101010111100
.1010  1011  1100  .1010 1011 1100
                     .A    B    C
```

Binary to Decimal

Rule: The same rule applies as for whole numbers.

Example:

$.001001100010$
$= 1(2^{-3}) + 1\,(2^{-6}) + 1\,(2^{-7}) + 1\,(2^{-11})$
$= 1/8 + 1/64 + 1/128 + 1/2048$
$= 305/2048$
$= .1489$ plus
or $.149$

Usage of Hexadecimal

Since System/370 has eight bits (plus a parity bit) in each byte of its storage, each byte can be thought of as being two hexadecimal-system digits; for example:

Decimal	248	
Binary	1111	1000
Hexadecimal	F	8

Remember that the hexadecimal system is simply a shorthand notation used to express the bit patterns within a computer such as System/370. Thus, it is also related to the other code structures defined previously. In extended binary coded decimal interchange code (EBCDIC), this eight-bit character (called F8 in hexadecimal) would be an 8; it could have other meanings in other codes. But regardless of the code meaning, it is expressed as F8 in the hexadecimal system.

Data Recording Media

Cards

The 80-column punched card is one of the most successful media for communication with machines. Information is recorded as small rectangular holes punched in specific locations in a standard size card (Figure 2-19). Information, represented (coded) by the presence or absence of holes in specific locations, can be read or sensed as the card is moved through a card-reading machine.

Reading or sensing the card is basically a process of automatically converting data, recorded as holes, to an electronic impulse and thereby entering the data into the machine. Cards are used both for entering the data into the machine and for recording or punching information from a machine. Thus the card is not only a means of transferring data from some original source to a machine, but also is a common medium for the exchange of information between machines.

80-column cards provide twelve punching positions in each column. The twelve punching positions form twelve horizontal rows across the card. One or more punches in a single column represents a character. The number of columns used depends on the amount of data to be represented.

The card is often called a *unit record* because the data is restricted to the 80 columns, and the card is read or punched as a unit of information. The actual data on the card, however, may consist of part of a record, one record, or more than one record. If more than 80 columns are needed to contain the data of a record, two or more cards may be used. Continuity between the cards of one record may be established by punching identifying information in designated columns of each card.

Information punched in cards is read or interpreted by a machine called a *card reader* and is recorded (punched) in a card by a machine called a *card punch*. Data is transcribed from source documents to punched cards by manually operated card punch machines.

The 80-Column Card Code

The standard *card code* uses the twelve possible punching positions of a vertical column on a card to represent a numeric, alphabetic, or special character (Figure 2-19). The twelve hole positions are divided into two areas, numeric and zone. The first nine hole positions from the bottom edge of the card are the numeric hole positions and have an assigned value of 9, 8, 7, 6, 5, 4, 3, 2, and 1, respectively. The remaining three positions, 0, 11, and 12 are the zone positions. (The 0 position is considered to be both a numeric and a zone position).

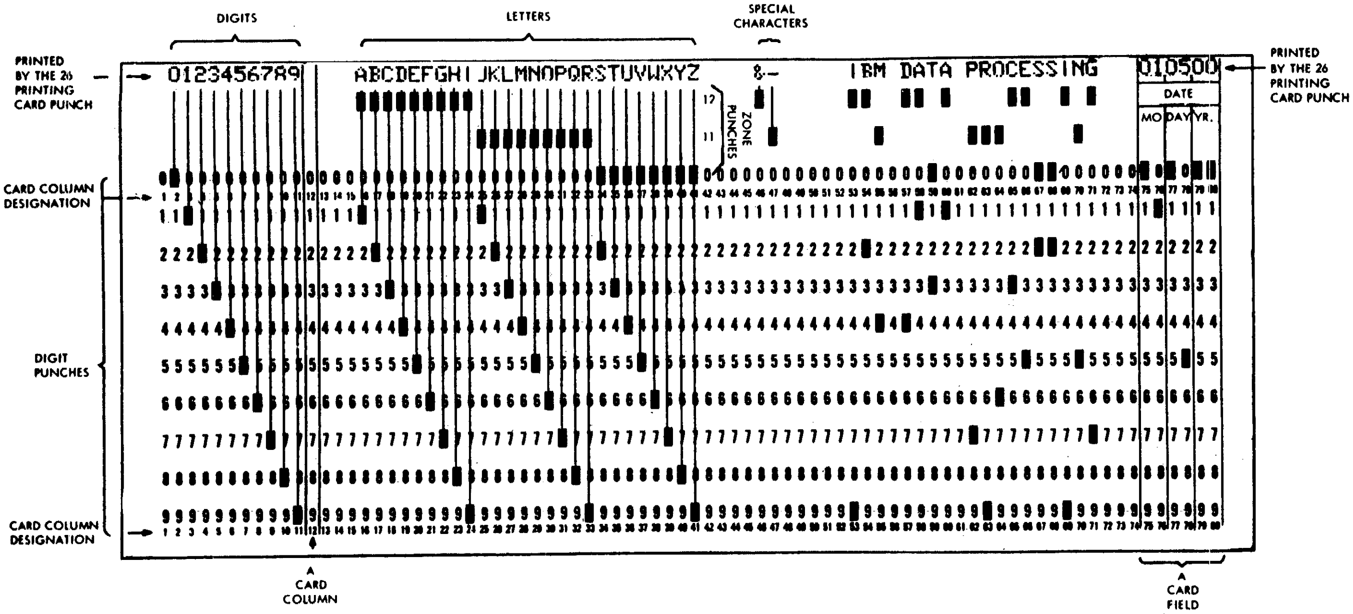


Figure 2-19 IBM 80 Column Punched Card, Standard Hole Pattern

The numeric characters 0 through 9 are represented by a single hole in a vertical column For example, 0 is represented by a single hole in the 0 zone position of the column.

The alphabetic characters are represented by two holes in a single vertical column, one numeric hole and one zone hole. The alphabetic characters A through I use the twelve zone hole and a numeric hole 1 through 9, respectively. The alphabetic characters J through R use the 11 hole and a numeric hole 1 through 9, respectively. The alphabetic characters S through Z use the 0 zone hole and a numeric hole 2 through 9, respectively.

The standard special characters $ * % and so on, are represented by one, two, or three holes in a column of the card and consist of hole patterns not used to represent numeric or alphabetic characters.

System/3 Card

The System/3 card is approximately one-third the size of the 80-column card, yet can hold up to 20 per cent more information. The unique size of the System/3 card permits the use of newer card-handling techniques which have resulted in smaller, simpler equipment.

As shown in Figure 2-20, the 96-column card has two sections. The lower section of the card is the punch area, while the upper section of the card is the print area.

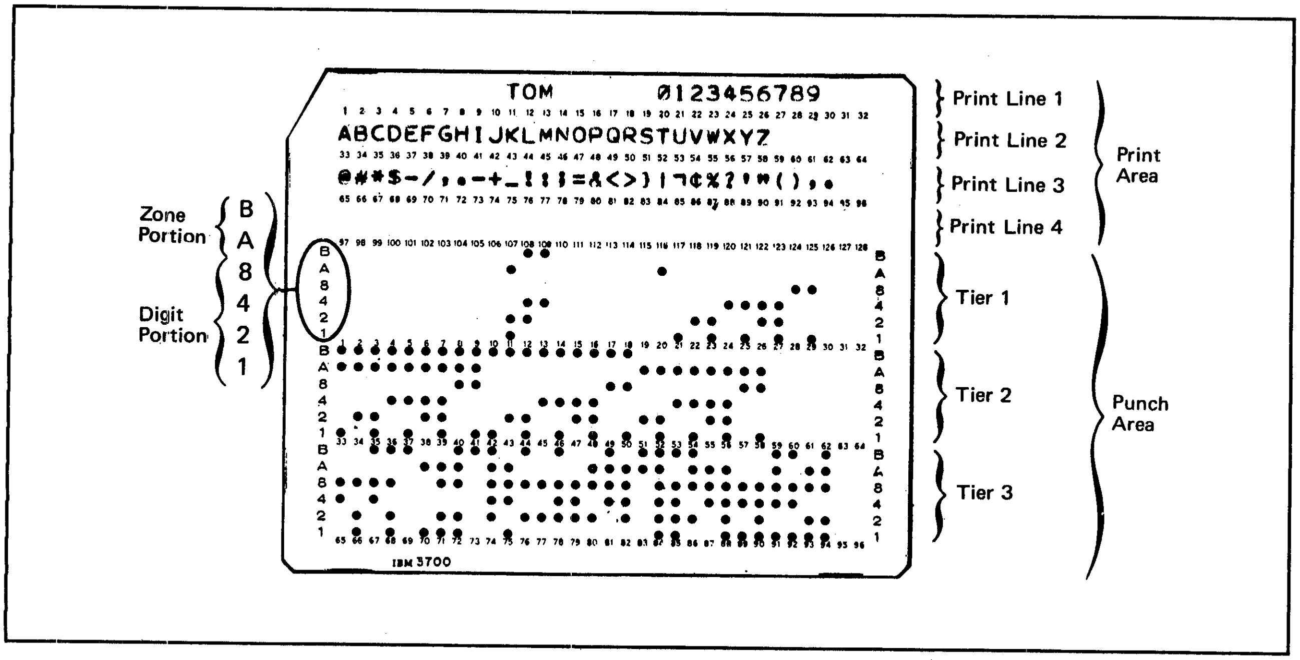


Figure 2-20 IBM 96-Column Punched Card

The system reads the punched holes in the punch area to determine what information is on the card. To make reading easier for you, the data recorder can print at the top of the card the information punched at the bottom.

The positions in the print area (print lines 1-3) are numbered from 1-96 to correspond with the columns in the punch area. Figure 2-20 shows how information may be represented on a card as punched holes and as printed characters. The name, TOM, was punched in columns 11, 12 and 13 of the punch area and printed in the corresponding positions of the print area.

As you can see in Figure 2-20, the data which was punched in tier 1 was printed on print line 1. Similarly, tier 2 corresponds to print line 2 and tier 3 to print line 3. Print line 4 (positions 97-128) does not have a corresponding tier but may be used for printing.

Representation of Information For every character of information that is keyed, a particular combination of holes is punched into a column. Using the six punch positions of a column, it is possible to form 64 different punch combinations.

Because 64 different punch combinations can be formed, it is possible to represent 64 different characters: A-Z, 0-9, and 28 special characters, such as the comma, dollar sign, period, and the blank.

Zone and Digit Portions of a Column Look at a column of punch positions in Figure 2-20. The B and A punch positions are called the zone portion. The 8, 4, 2, and 1 positions are referred to as the digit portion.

Digits are usually represented by one or more holes in the digit portion of a column (Figure 2-20). The number 1 consists of a hole in the 1 punch position. Likewise, a hole in the 2 punch position would represent a 2. Because there is no 3 punch position, a 3 is represented by holes made in both the 1 and the 2 punch positions (1 plus 2 equals 3). Notice that the digit 0 is represented not by a digit punch, but by a single punch in the A position. This is the one exception.

Alphabetic characters are represented by a combination of punches made in the digit (8, 4, 2, 1) and zone (B, A) portions of a column, as showin in Figure 2-20.

Special characters are represented by a combination of punches in the digit and/or zone portions of a column. Figure 2-20 shows the punch combinations associated with each character.

There is no need to memorize the punch combinations. The appropriate holes are automatically punched by pressing the keys on the data recorder.

Paper Tape

Punched paper tape serves much the same purpose as punched cards. Developed for transmitting telegraph messages over wires, paper tape is used for data processing communication as well. For long-distance transmission, machines convert data from cards and keyboard strokes to paper tape, send the information over telephone or telegraph wires to produce a duplicate paper tape at the other end of the wire, and reconvert the information to punched cards for later processing.

Data is recorded as a special arrangement of punched holes, precisely arranged along the length of a paper tape (Figure 2-21 and 2-22). Paper tape is a continuous recording medium, as compared to cards, which are fixed in length. Thus, paper tape can be used to record data in records of any length, limited only by the capacity of the storage medium into which the data is to be placed or from which the data is received.

Data punched in paper tape is read or interpreted by a paper tape reader and recorded by a paper tape punch.

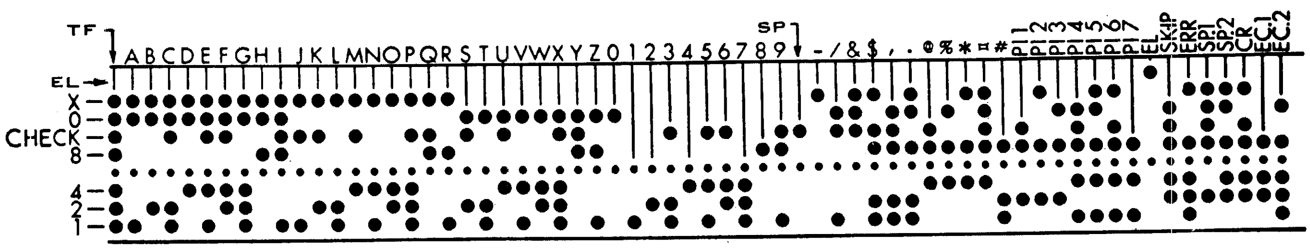


Figure 2-21 Paper Tape, Eight-Channel Code

Eight-Channel Code (Hole Pattern)

Data is recorded (punched) and read as holes located in eight parallel channels along the length of the paper tape. One column of the eight possible punching positions (one for each channel) across the width of the tape is used to code numeric, alphabetic, special, and function characters. Figure 2-21 shows a section of paper tape with the eight channels and several coded characters.

The lower four channels of the tape (excluding the feed holes) are labeled 1, 2, 4, and 8 and are used to record numeric characters. The numeric characters 0 through 9 are represented as a punch or punches in these four positions.

The sum of the position values indicates the numeric value of the character. For example, a hole in channel 1 is used to represent a numeric 1: a combination of a 1 and a 2 punch represents a numeric 3.

The X and O channels are similar to the zone punches in 96-column punched cards. These channels are used in combination with the numeric channels to record alphabetic and special characters. The coding for the alphabetic and special characters is shown in Figure 2-21.

To check that each character is recorded correctly, each column of the tape is punched with an *odd* number of holes. A check hole must be present in the check channel for any column whose basic code (X, 0, 8, 4, 2, 1) consists of an even number of holes.

A punch in the EL (end-of-line) channel is a special function character used to mark the end of a record on the tape. The tape feed code consists of punches in the X, O, 8, 4, 2, and 1 channels, and is used to indicate blank character positions. The paper tape reader automatically skips over the areas of tape punched with the tape feed code.

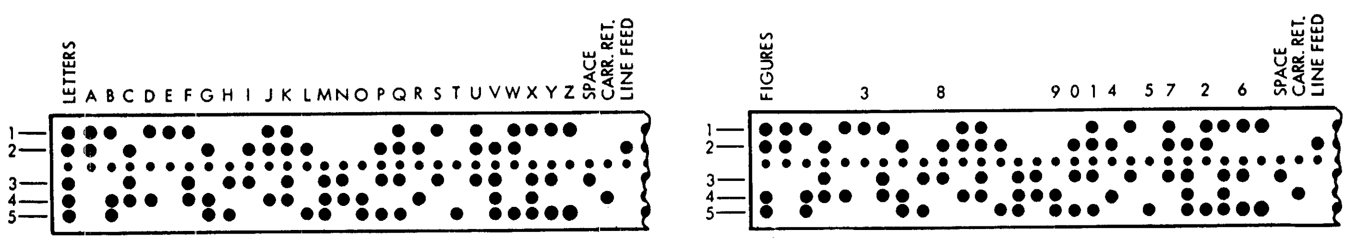


Figure 2-22 Paper Tape, Five-Channel Code

Five-Channel Code (Hole pattern)

Data is recorded (punched) and read as holes in five parallel channels along the length of the paper tape. One column of the five possible punching positions (one of each channel) across the width of the tape is used to code numeric, alphabetic, special, and function characters. Figure 2-22 shows a section of paper tape with the five channels and several coded characters.

Because there are only 32 possible combinations of punches, using the five punching positions, a shift system is used to double the number of available codes. When the letters (LTRS) code punch precedes a section of tape, the characters that follow are interpreted as alphabetic characters (Figure 2-22). When the figures (FIGS) code punch precedes a section of tape, the coded punches are interpreted as numeric or special characters.

Ten of the 32 characters are used for coding both the alphabetic character P, Q, W, E, R, T, Y, U, I, and O and the decimal digits 0 through 9, respectively. Interpretation depends on the shift code, LTRS or FIGS, preceding these characters. Likewise, the code for special characters is identical to that of other alphabetic characters. The actual alphabetic code that is equivalent to a given special character code varies, depending on customer requirements.

The function characters - space, carriage return (CR), and line feed (LF) - are the same in either LTRS or FIGS shift. The space code is used to indicate the absence of data on tape. The actual function of the CR and LF characters depends on the machine with which they are used.

A disadvantage of this code is evident in data communications. The shift characters increase the lengths of messages. For intermixed alphabetic and numeric data, this increase can be significant.

Magnetic Tape

Magnetic tape is one of the principal input/output recording media for computer systems. It is also used extensively for compact storage or large files of data.

Magnetic tape units offer high-speed entry of data into the computer system, as well as efficient, extremely fast recording of processed data from the system. Highly reliable input/output data rates of up to 2.5 million numeric characters per second are possible.

The magnetic tape unit functions as both an input unit and an output unit for the computer system. It moves the magnetic tape across a read/write head and accomplishes the actual reading and writing of information on the tape.

Information is recorded on magnetic tape as magnetized spots called bits. The recording can be retained indefinitely, or the recorded information can be automatically erased and the tape reused many times with continued high reliability.

So that tape can be easily handled and processed, it is wound on individual reels or in dust-resistant cartridges. Tape on the individual reels is 1/2 inch wide and is supplied in lengths of up to 2400 feet per reel.

Several features built into the IBM 3420 magnetic tape unit ensure reliability and ease of operation. An automatic reel latch mechanically seats the file reel in position and pneumatically locks it on the hub for tape movement. With automatic threading and cartridge loading, tape mounting and demounting times are significantly reduced. Optical tachometers, built into the drive, sense small variations in the speed of the capstan and the tape, and generate corrective signals. This precise control is one of the keys to the 3420's fast read access and rewind times. (Rewinding of a full 2,400 foot reel takes only 45 seconds).

Tape wear and contamination are minimized by extensive use of air bearings and surface treatments. The path that the tape takes through the drive, from reel to reel, allows the recording side of the tape to touch only two surfaces during read/write operations: the tape cleaner and the read/write head. Additionally, the cleaning mechanism provided on certain models removes loose contaminants from the tape and protects the read/write head from tape media contamination during high-speed rewind and tape loading and unloading.

Data is recorded in parallel channels or tracks along the length of the length of the tape.

The tracks across the width of the tape provide one row of data. The spacing between the vertical rows is automatically generated during the writing operation and varies, depending on the character density used for recording. Character densities as high as 6250 characters per inch are available.

A longer space is automatically generated between blocks of records on the tape; this space is called the *interblock gap* (formally called interrecord gap).

The IBM 3400 series tape units usually have a nine-track read/write head. A seven-track read/write head is available as an option. The nine-track heads read or write information in nine tracks along the length of the tape. With the single-density feature, certain models operate at 1,600 bytes per inch. With the dual-density feature, operations are at both 800 and 1,600 bytes per inch for these models and at 1,600 and 6,250 bytes per inch for other models. Each byte consists of eight data bits and a parity bit. Each byte may represent two packed decimal digits, eight binary digits, or one special or alphameric character.

Packed mode is limited to recording decimal numbers, but it offers the advantage of recording two numbers in a single tape row, effectively doubling the reading and writing speed for numeric data on certain data processing systems.

The seven-track feature is intended for compatibility with tapes written on older tape drives. Data for IBM computers may be coded on seven-track magnetic tape in two modes, binary coded decimal (BCD) or binary. The code used depends on the computer that originates the data.

Seven-Track Coding on Tape (BCD Mode)

Letters of the alphabet, decimal numbers, and special characters may be recorded on magnetic tape using the binary coded decimal code (Figure 2-23).

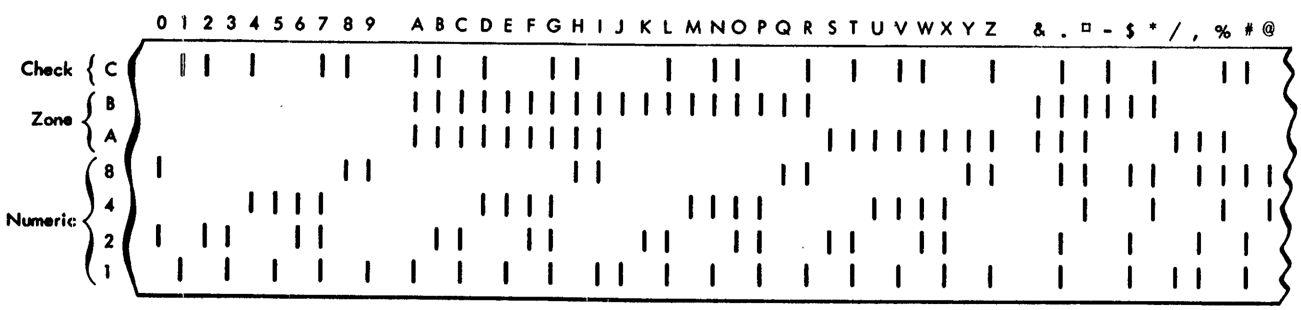


Figure 2-23 Magnetic Tape, Seven-Track, Seven-Bit Alphameric Code

Seven-Track Coding on Tape (Binary Mode)

Some computer systems record data on magnetic tape in binary notation (Figure 2-24).

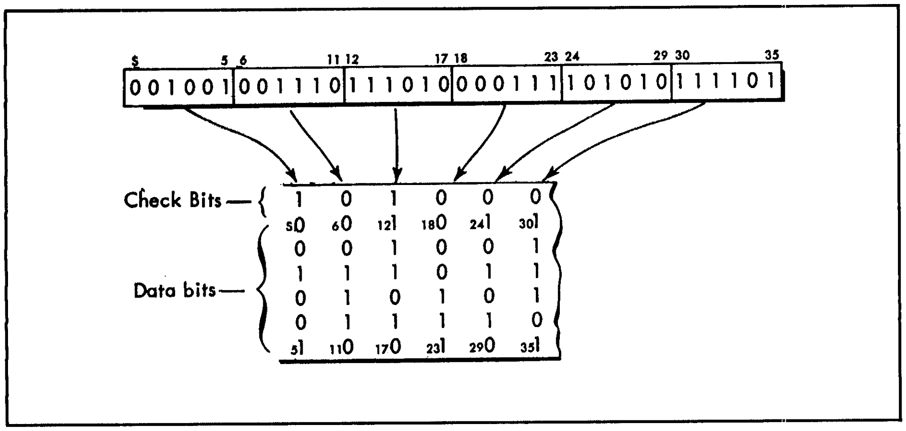


Figure 2-24 Magnetic Tape, Seven-Track, Binary Recording

As in BCD recording, the C track is used to verify accuracy of tape reading and writing. With binary tape, however, each row of bits must contain an odd number of 1 bits.

The longitudinal parity check in binary mode is similar to that for BCD mode; the total number of bits in each horizontal track of a record block must be even.

Nine-Track Coding on Tape

The nine-track magnetic tape for Series 3400 tape units accepts the System/370 central processing unit coding, shown in Figure 2-25, as System/370 Eight-Bit Code. It adds an odd-parity bit, however, and the order of the bits in the character (byte) row is rearranged as shown in Figure 2-26.

Hexa-decimal	Mnemonic	Graphic & Control Symbols BCDIC	Graphic & Control Symbols EBCDIC	Punched Card Code	System/360 8-Bit Code
00			NUL	12-0-1-8-9	0000 0000
01			SOH	12-1-9	0000 0001
02			STX	12-2-9	0000 0010
03			ETX	12-3-9	0000 0011
04	SPM		PF	12-4-9	0000 0100
05	BALR		HT	12-5-9	0000 0101
06	BCTR		LC	12-6-9	0000 0110
07	BCR		DEL	12-7-9	0000 0111
08	SSK			12-8-9	0000 1000
09	ISK			12-1-8-9	0000 1001
0A	SVC		SMM	12-2-8-9	0000 1010
0B			VT	12-3-8-9	0000 1011
0C	(EBCDIC +)		FF	12-4-8-9	0000 1100
0D	(EBCDIC -)		CR	12-5-8-9	0000 1101
0E			SO	12-6-8-9	0000 1110
0F			SI	12-7-8-9	0000 1111
10	LPR		DLE	12-11-1-8-9	0001 0000
11	LNR		DC1	11-1-9	0001 0001
12	LTR		DC2	11-2-9	0001 0010
13	LCR		TM	11-3-9	0001 0011
14	NR		RES	11-4-9	0001 0100
15	CLR		NL	11-5-9	0001 0101
16	OR		BS	11-6-9	0001 0110
17	XR		IL	11-7-9	0001 0111
18	LR		CAN	11-8-9	0001 1000
19	CR		EM	11-1-8-9	0001 1001
1A	AR		CC	11-2-8-9	0001 1010
1B	SR		CU1	11-3-8-9	0001 1011
1C	MR		IFS	11-4-8-9	0001 1100
1D	DR		IGS	11-5-8-9	0001 1101
1E	ALR		IRS	11-6-8-9	0001 1110
1F	SLR		IUS	11-7-8-9	0001 1111
20	LPDR		DS	11-0-1-8-9	0010 0000
21	LNDR		SOS	0-1-9	0010 0001
22	LTDR		FS	0-2-9	0010 0010
23	LCDR			0-3-9	0010 0011
24	HDR		BYP	0-4-9	0010 0100
25	LRDR		LF	0-5-9	0010 0101
26	MXR		ETB	0-6-9	0010 0110
27	MXDR		ESC	0-7-9	0010 0111
28	LDR			0-8-9	0010 1000
29	CDR			0-1-8-9	0010 1001
2A	ADR		SM	0-2-8-9	0010 1010
2B	SDR		CU2	0-3-8-9	0010 1011
2C	MDR			0-4-8-9	0010 1100
2D	DDR		ENQ	0-5-8-9	0010 1101
2E	AWR		ACK	0-6-8-9	0010 1110
2F	SWR		BEL	0-7-8-9	0010 1111
30	LPER			12-11-0-1-8-9	0011 0000
31	LNER			1-9	0011 0001
32	LTER		SYN	2-9	0011 0010
33	LCER			3-9	0011 0011
34	HER		PN	4-9	0011 0100
35	LRER		RS	5-9	0011 0101
36	AXR		UC	6-9	0011 0110
37	SXR		EOT	7-9	0011 0111
38	LER			8-9	0011 1000
39	CER			1-8-9	0011 1001
3A	AER			2-8-9	0011 1010
3B	SER		CU3	3-8-9	0011 1011
3C	MER		DC4	4-8-9	0011 1100
3D	DER		NAK	5-8-9	0011 1101
3E	AUR			6-8-9	0011 1110
3F	SUR		SUB	7-8-9	0011 1111
40	STH		SP	no punches	0100 0000
41	LA			12-0-1-9	0100 0001
42	STC			12-0-2-9	0100 0010
43	IC			12-0-3-9	0100 0011
44	EX			12-0-4-9	0100 0100
45	BAL			12-0-5-9	0100 0101
46	BCT			12-0-6-9	0100 0110
47	BC			12-0-7-9	0100 0111
48	LH			12-0-8-9	0100 1000
49	CH			12-1-8	0100 1001
4A	AH		¢	12-2-8	0100 1010
4B	SH		.	12-3-8	0100 1011
4C	MH	¤)	<	12-4-8	0100 1100
4D		[	(	12-5-8	0100 1101
4E	CVD	<	+	12-6-8	0100 1110
4F	CVB	‡	|	12-7-8	0100 1111
50	ST	& +	&	12	0101 0000
51				12-11-1-9	0101 0001
52				12-11-2-9	0101 0010
53				12-11-3-9	0101 0011
54	N			12-11-4-9	0101 0100
55	CL			12-11-5-9	0101 0101
56	O			12-11-6-9	0101 0110
57	X			12-11-7-9	0101 0111
58	L			12-11-8-9	0101 1000
59	C			11-1-8	0101 1001
5A	A		!	11-2-8	0101 1010
5B	S	$	$	11-3-8	0101 1011
5C	M	*	*	11-4-8	0101 1100
5D	D	]	)	11-5-8	0101 1101
5E	AL	;	;	11-6-8	0101 1110
5F	SL	Δ	¬	11-7-8	0101 1111
60	STD	–	–	11	0110 0000
61		/	/	0-1	0110 0001
62				11-0-2-9	0110 0010
63				11-0-3-9	0110 0011
64				11-0-4-9	0110 0100
65				11-0-5-9	0110 0101
66				11-0-6-9	0110 0110
67	MXD			11-0-7-9	0110 0111
68	LD			11-0-8-9	0110 1000
69	CD			0-1-8	0110 1001
6A	AD			12-11	0110 1010
6B	SD	,	,	0-3-8	0110 1011
6C	MD	% (	%	0-4-8	0110 1100
6D	DD	Y	_	0-5-8	0110 1101
6E	AW	\	>	0-6-8	0110 1110
6F	SW	⧺	?	0-7-8	0110 1111
70	STE			12-11-0	0111 0000
71				12-11-0-1-9	0111 0001
72				12-11-0-2-9	0111 0010
73				12-11-0-3-9	0111 0011
74				12-11-0-4-9	0111 0100
75				12-11-0-5-9	0111 0101
76				12-11-0-6-9	0111 0110
77				12-11-0-7-9	0111 0111
78	LE			12-11-0-8-9	0111 1000
79	CE			1-8	0111 1001
7A	AE	b̸	:	2-8	0111 1010
7B	SE	# =	#	3-8	0111 1011
7C	ME	@	@	4-8	0111 1100
7D	DE	:	'	5-8	0111 1101
7E	AU	>	=	6-8	0111 1110
7F	SU	√	"	7-8	0111 1111
80	SSM			12-0-1-8	1000 0000
81			a	12-0-1	1000 0001
82	LPSW		b	12-0-2	1000 0010
83	(Diagnose)		c	12-0-3	1000 0011
84	WRD		d	12-0-4	1000 0100
85	RDD		e	12-0-5	1000 0101
86	BXH		f	12-0-6	1000 0110
87	BXLE		g	12-0-7	1000 0111
88	SRL		h	12-0-8	1000 1000
89	SLL		i	12-0-9	1000 1001
8A	SRA			12-0-2-8	1000 1010
8B	SLA			12-0-3-8	1000 1011
8C	SRDL			12-0-4-8	1000 1100
8D	SLDL			12-0-5-8	1000 1101
8F	SRDA			12-0-6-8	1000 1110
8F	SLDA			12-0-7-8	1000 1111
90	STM			12-11-1-8	1001 0000
91	TM		j	12-11-1	1001 0001
92	MVI		k	12-11-2	1001 0010
93	TS		l	12-11-3	1001 0011
94	NI		m	12-11-4	1001 0100
95	CLI		n	12-11-5	1001 0101
96	OI		o	12-11-6	1001 0110
97	XI		p	12-11-7	1001 0111
98	LM		q	12-11-8	1001 1000
99			r	12-11-9	1001 1001
9A				12-11-2-8	1001 1010
9B				12-11-3-8	1001 1011
9C	SIO			12-11-4-8	1001 1100
9D	TIO			12-11-5-8	1001 1101
9F	HIO			12-11-6-8	1001 1110
9F	TCH			12-11-7-8	1001 1111
A0				11-0-1-8	1010 0000
A1				11-0-1	1010 0001
A2			s	11-0-2	1010 0010
A3			t	11-0-3	1010 0011
A4			u	11-0-4	1010 0100
A5			v	11-0-5	1010 0101
A6			w	11-0-6	1010 0110
A7			x	11-0-7	1010 0111
A8			y	11-0-8	1010 1000
A9			z	11-0-9	1010 1001
AA				11-0-2-8	1010 1010
AB				11-0-3-8	1010 1011
AC				11-0-4-8	1010 1100
AD				11-0-5-8	1010 1101
AE				11-0-6-8	1010 1110
AF				11-0-7-8	1010 1111
B0				12-11-0-1-8	1011 0000
B1				12-11-0-1	1011 0001
B2				12-11-0-2	1011 0010
B3				12-11-0-3	1011 0011
B4				12-11-0-4	1011 0100
B5				12-11-0-5	1011 0101
B6				12-11-0-6	1011 0110
B7				12-11-0-7	1011 0111
B8				12-11-0-8	1011 1000
B9				12-11-0-9	1011 1001
BA				12-11-0-2-8	1011 1010
BB				12-11-0-3-8	1011 1011
BC				12-11-0-4-8	1011 1100
BD				12-11-0-5-8	1011 1101
BE				12-11-0-6-8	1011 1110
BF				12-11-0-7-8	1011 1111
C0		?		12-0	1100 0000
C1		A	A	12-1	1100 0001
C2		B	B	12-2	1100 0010
C3		C	C	12-3	1100 0011
C4		D	D	12-4	1100 0100
C5		E	E	12-5	1100 0101
C6		F	F	12-6	1100 0110
C7		G	G	12-7	1100 0111
C8		H	H	12-8	1100 1000
C9		I	I	12-9	1100 1001
CA				12-0-2-8-9	1100 1010
CB				12-0-3-8-9	1100 1011
CC				12-0-4-8-9	1100 1100
CD				12-0-5-8-9	1100 1101
CE				12-0-6-8-9	1100 1110
CF				12-0-7-8-9	1100 1111
D0		!		11-0	1101 0000
D1	MVN	J	J	11-1	1101 0001
D2	MVC	K	K	11-2	1101 0010
D3	MVZ	L	L	11-3	1101 0011
D4	NC	M	M	11-4	1101 0100
D5	CLC	N	N	11-5	1101 0101
D6	OC	O	O	11-6	1101 0110
D7	XC	P	P	11-7	1101 0111
D8		Q	Q	11-8	1101 1000
D9		R	R	11-9	1101 1001
DA				12-11-2-8-9	1101 1010
DB				12-11-3-8-9	1101 1011
DC	TR			12-11-4-8-9	1101 1100
DD	TRT			12-11-5-8-9	1101 1101
DE	ED (3)			12-11-6-8-9	1101 1110
DF	EDMK (3)			12-11-7-8-9	1101 1111
E0		‡		0-2-8	1110 0000
E1				11-0-1-9	1110 0001
E2		S	S	0-2	1110 0010
E3		T	T	0-3	1110 0011
E4		U	U	0-4	1110 0100
E5		V	V	0-5	1110 0101
E6		W	W	0-6	1110 0110
E7		X	X	0-7	1110 0111
E8		Y	Y	0-8	1110 1000
E9		Z	Z	0-9	1110 1001
EA				11-0-2-8-9	1110 1010
EB				11-0-3-8-9	1110 1011
EC				11-0-4-8-9	1110 1100
ED				11-0-5-8-9	1110 1101
EE				11-0-6-8-9	1110 1110
EF				11-0-7-8-9	1110 1111
F0		0	0	0	1111 0000
F1	MVO	1	1	1	1111 0001
F2	PACK	2	2	2	1111 0010
F3	UNPK	3	3	3	1111 0011
F4		4	4	4	1111 0100
F5		5	5	5	1111 0101
F6		6	6	6	1111 0110
F7		7	7	7	1111 0111
F8	ZAP (3)	8	8	8	1111 1000
F9	CP (3)	9	9	9	1111 1001
FA	AP (3)			12-11-0-2-8-9	1111 1010
FB	SP (3)			12-11-0-3-8-9	1111 1011
FC	MP (3)			12-11-0-4-8-9	1111 1100
FD	DP (3)			12-11-0-5-8-9	1111 1101
FE				12-11-0-6-8-9	1111 1110
FF				12-11-0-7-8-9	1111 1111

Figure 2-25 IBM System/370 Eight-Bit Code

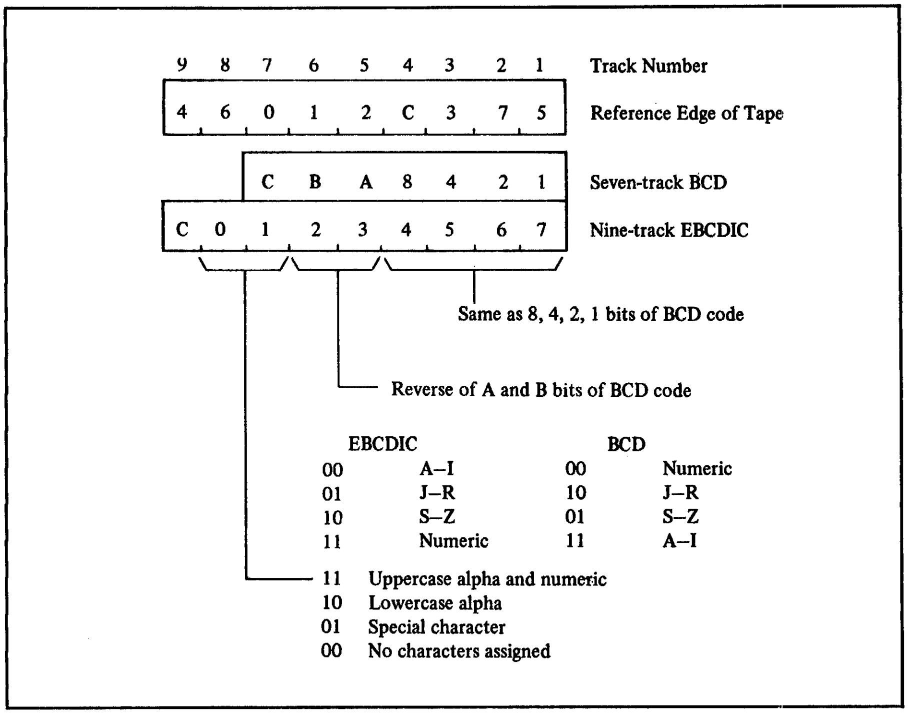


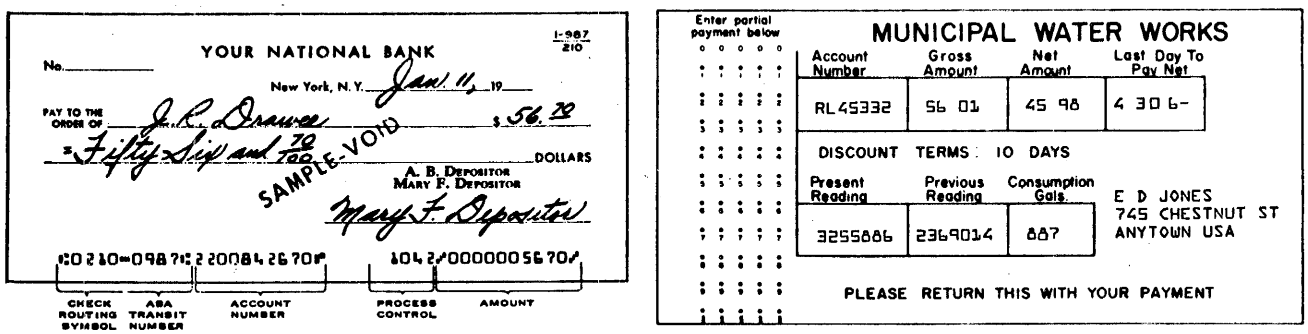
Figure 2-26 Comparison of Seven-Track and Nine-Track Alphabetic Code

The 4-7 recording positions of nine-track tape parallel the function of the 8, 4, 2, 1 bit positions of seven-track tape. The 2 and 3 recording positions are the exact reverse of the A and B bit positions of seven-track. Positions 0 and 1 are the two additional recording channels that group the characters into one of four classifications: uppercase alpha and numeric, lowercase alpha, special characters, and no assigned character. Note that the actual channels on nine-track tape do not run in 0-7 sequence.

Character Reading

Magnetic Ink Characters

Another method of representing data on paper media for machine processing is with magnetic ink characters - a language readable by both man and machine. Magnetic ink characters are printed on paper, as in Figure 2-27. The shape of the characters permits easy visual interpretation; the special magnetic ink allows reading or interpretation by machine.



Magnetic Ink Characters

Optically Readable Characters

Figure 2-27 Magnetically and Optically Readable Characters

The printing (inscribing) of magnetic ink characters on the paper documents is done by machine. The paper documents may be random size paper or cards ranging from 2 3/4 inches to 3 2/3 inches wide, from 6 inches to 8 3/4 inches long, and from .003 inch to .007 inch thick.

The IBM 1260 Electronic Inscriber, in addition to performing the normal proving functions related to banking procedures, inscribes documents. After inscription, the IBM 1419 Magnetic Character Reader reads the inscribed information from the documents and converts it to a machine language. At this point, the information enters directly into an IBM data processing system. The 1419 can sort the documents as well.

Optically Read Characters

Another method of representing data on paper documents for input to a data processing system uses optically readable characters (Figures 2-27).

The IBM 1287 Optical Reader can read typed and machine-printed alphameric data and hand-printed numbers, letters, and marks from cut-form documents having a variety of formats, orientations, types of data and field lengths. The data of source documents can be organized in fixed or variable-length fields, in columns or rows, and can be read in any sequence. Just as easily, the 1287 can also read numeric data and special symbols from journal rolls, such as cash-register and adding-machine tapes.

The IBM 3881 Optical Mark Reader reads penciled and machine-printed marks on a variety of document sizes, weights, and formats. It can be easily adjusted so as to read data from forms ranging in size from 3-by-3-inch bill stubs to 9-by-12-inch sets. When equipped with the BCD feature, the 3881 can read *turnaround documents*, which are forms printed by line printers as output from a data processing system, distributed for action, then returned to be read as input to the system.

The IBM 3666 Checkout Scanner is an optical recognition device designed to read the Universal Product Code symbols (Figure 2-28) on items as they are pulled across the scanner slot in the checkstand in a supermarket. Item symbols are read at a rate of up to 100 inches per second as they are moved across the scanner window. The scanner improves checker productivity and accuracy, and provides automatic recording of item movement data.

0 12345 67890
11-Digit Symbol

0 003190
7-Digit Symbol

Figure 2-28 Universal Product Code Symbols

Visual Output

Visual display units in several sizes, capacities, speeds, and capabilities to handle complexities of information permit the user of a computing system to see, on a cathode ray tube, graphic reports that would take many times longer to produce by normal printing methods. The use of a visual display unit as a system operator console is a typical application. Another is the retrieval and presentation of a client's account record during a telephone inquiry. It is possible to update the record immediately (by using an entry keyboard) and return the corrected data to storage.

The display units present (on the cathode ray tube screen) tables, graphs, charts, and alphameric letters and figures. The IBM 2250 Display Unit has a display area containing over one million points that can be addressed by X and Y coordinates. It can display 52 lines of 74 characters each - all on a twelve-inch-square area. Figure 2-29 shows an example of a display on the 2250.

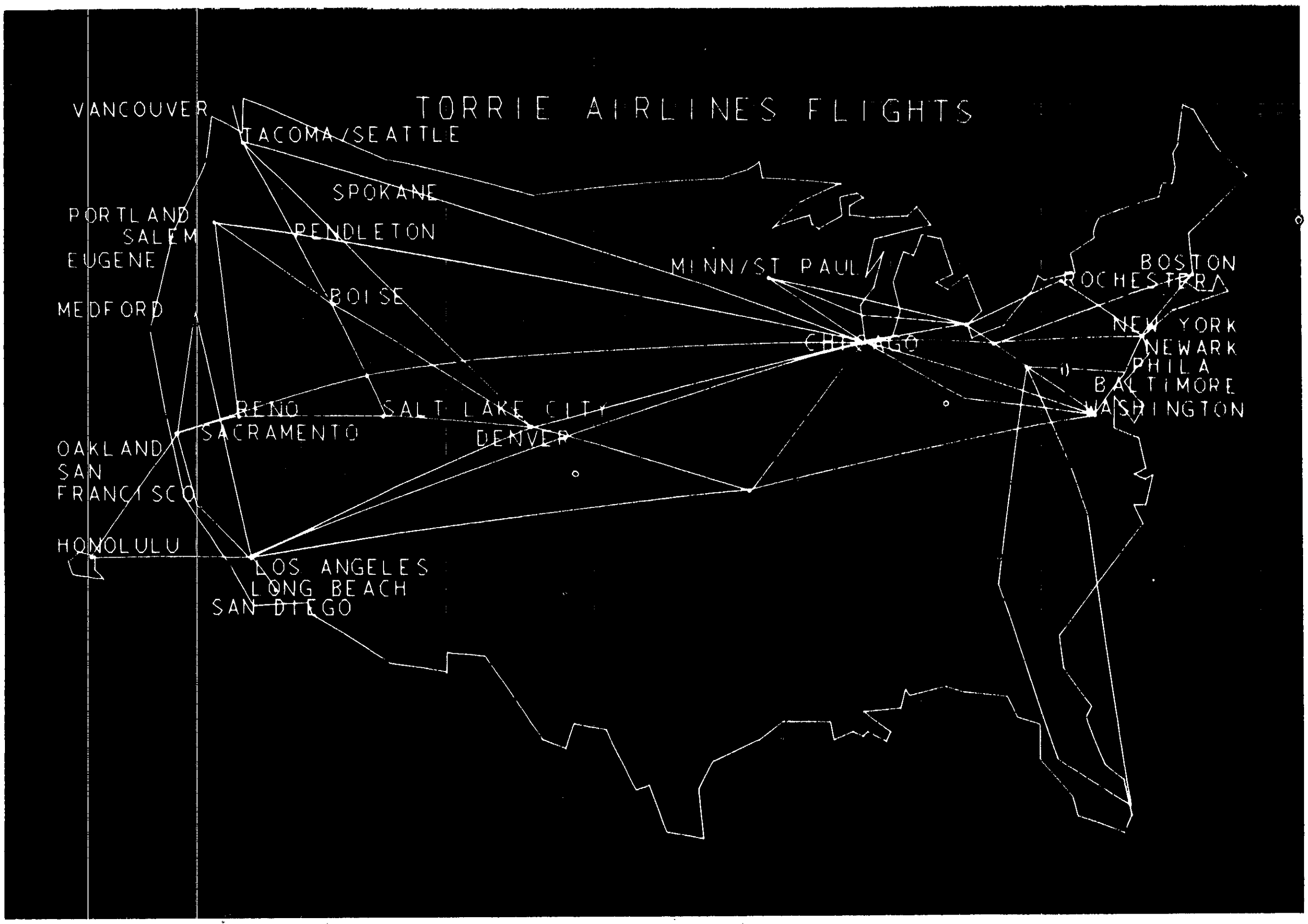


Figure 2-29 IBM 2250 Screen Contents. A Display Including Absolute Vector Graphics, Point Plotting, and Two Sizes of Alphabetic Characters.

The 2840 Control Unit, connected to the 2250, accepts and stores data from the computer at up to 500,000 characters per second. As many as 60,000 characters, or lines, can be displayed per second. Horizontal and vertical lines may be drawn by specifying only the end points of the lines; in addition, a special feature enables lines to be drawn at any angle. Points may be displayed as fast as 16.8 microseconds (millionths of a second) per point.

Visual output utilizing a display unit capable of displaying alphameric characters has become common for situations not requiring hard copy reports or cards as output. One model of the IBM 3277 Display Station displays up to 1,920 characters in 24 lines of up to 80 characters per line (Figure 2-30). The standard character set includes 36 alphameric and 27 special characters. A feature is available, however, that expands the character set to include upper and lower case alphabetic characters, additional graphics and special characters, and the full 133 character APL character set. (APL is a programming language described in a later section).

The 3277 can be attached to an IBM 3272 Control Unit connected *locally* (directly to a channel of a System/370). The 3272 provides local data transfer rates of up to 650,000 characters per second.

The 3272 and the 3277 are components of the IBM 3270 Information Display System. Additional information about the 3270 appears in another section.

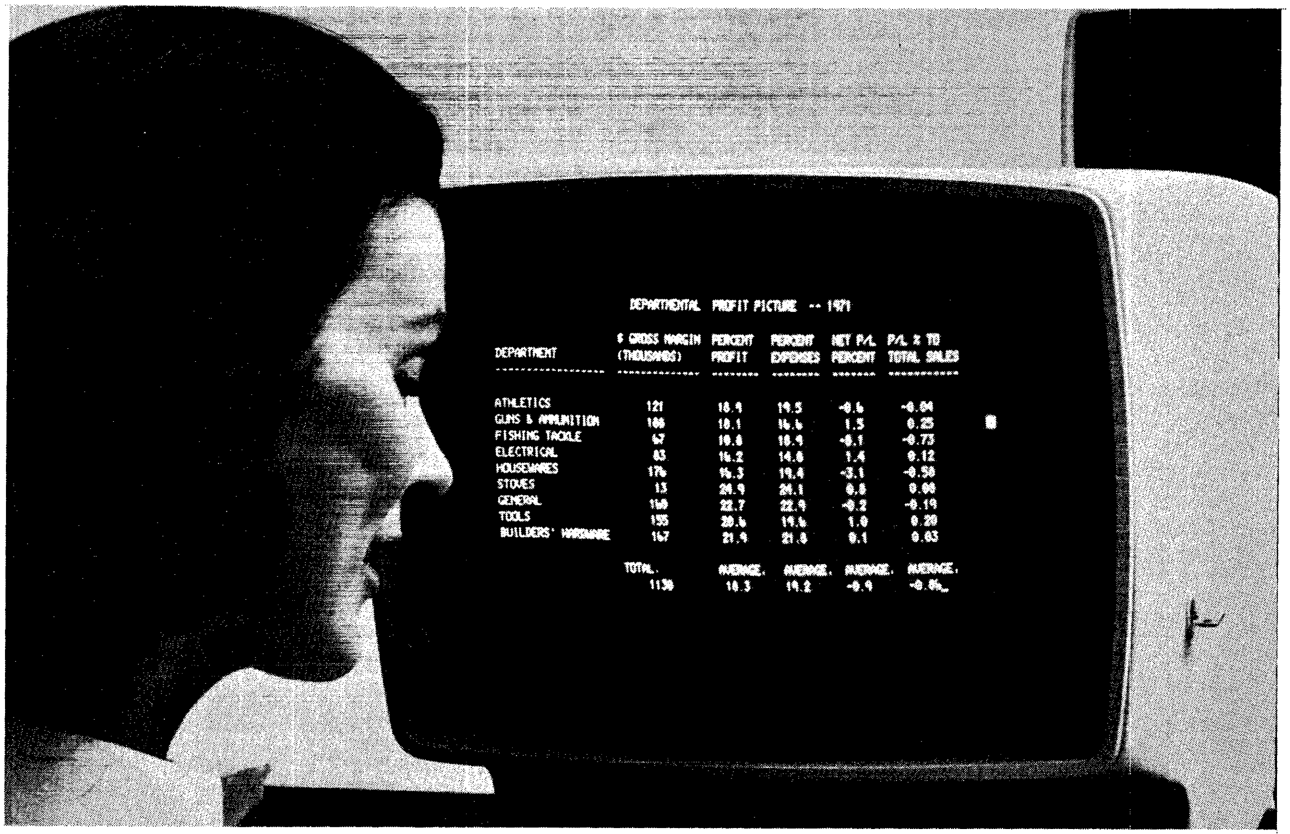
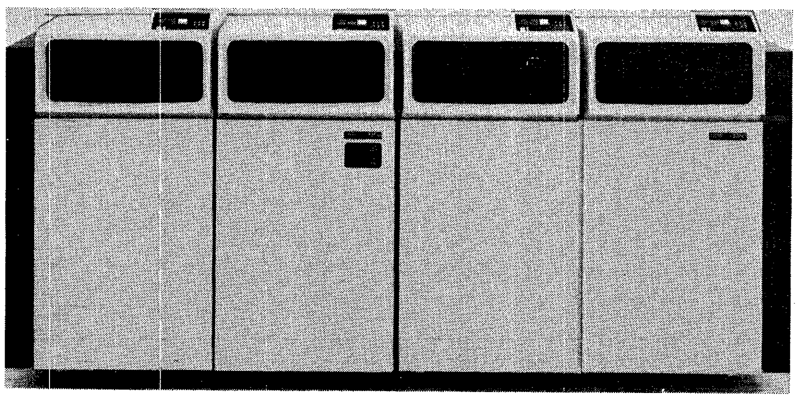


Figure 2-30 IBM 3277 Screen Display

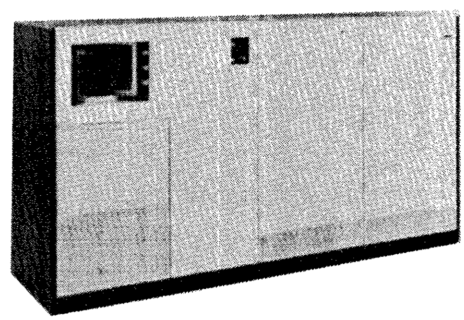
Section 3:

Storage Devices

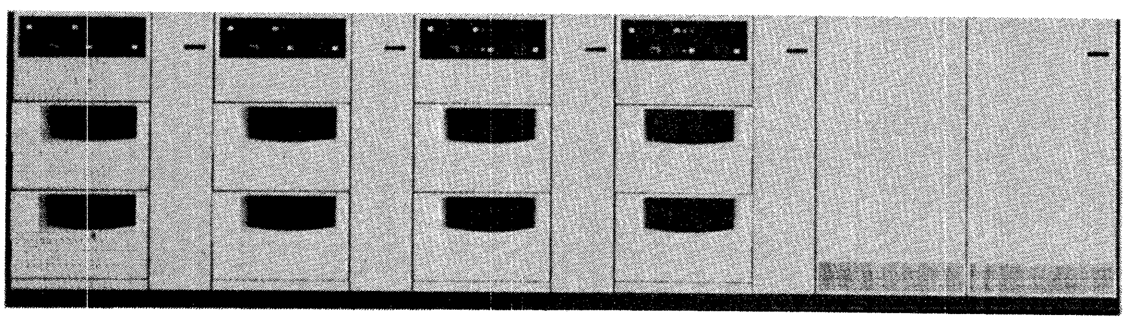
Several types of IBM storage are presently available: semiconductor, magnetic disk, and mass storage (Figures 3-1 and 3-8). Sometimes magnetic tape is thought of as storage rather than as an input/output medium.



IBM 3340 Disk Storage



IBM 2305 Fixed Head Storage



IBM 3330 Disk Storage

Figure 3-1 IBM Storage Devices

Information can be placed into, held in, or removed from computer storage as needed. The information can be:

1. Instructions to direct the central processing unit
2. Data (input, in-process, or output)
3. Reference data associated with processing (tables, code charts, constant factors, and so on).

Storage is classified as main or auxiliary, as in System/370 (Figure 3-2). Main storage is sometimes called memory. It can consist of core storage, but in today's larger systems such as System/370, it is usually made up of semiconductor integrated circuits.

Auxiliary refers to all other storage and is of two types:

1. *Direct access.* Disk and mass storage devices in which records can be accessed without having to read from the beginning of a file to find them.
2. *Sequential.* Tape units where reels must be read from the beginning in order to read or write a desired record.

Main storage accepts data from an input unit, exchanges data with and supplies instructions to the central processing unit, and can furnish data to an output unit. All data to be processed by any system must pass through main storage. This unit must therefore have capacity to retain a usable amount of data and the necessary instructions for processing.

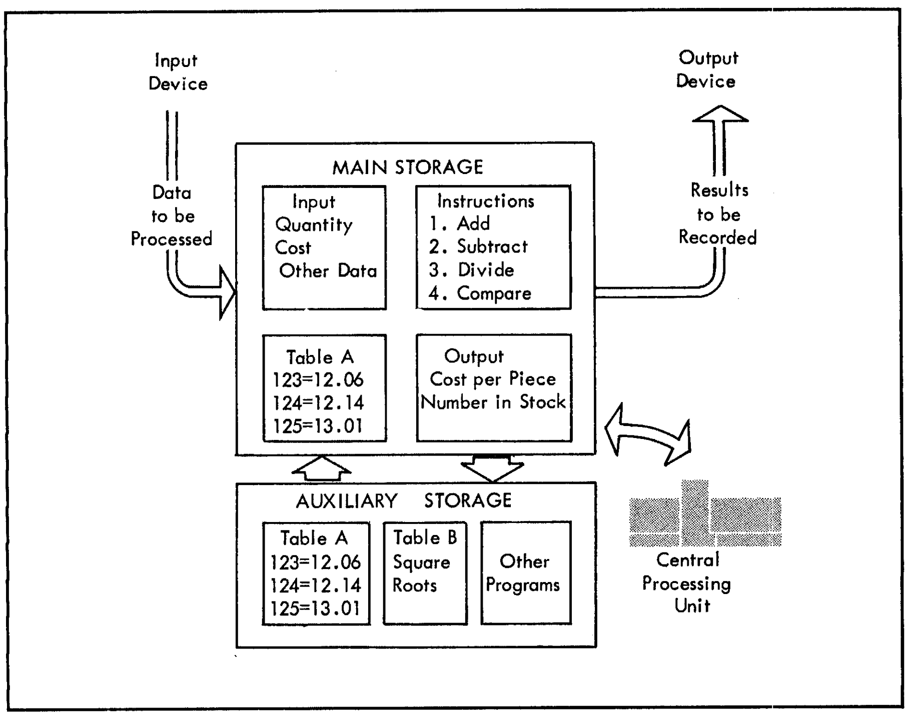


Figure 3-2 Schematic, Main, and Auxiliary Storage

Applications can require additional storage. If so, the capacity of main storage is augmented by an auxiliary storage unit. All information to and from auxiliary storage must be routed through main storage.

Storage is arranged somewhat like a group of numbered mail boxes in a post office (Figure 3-3).

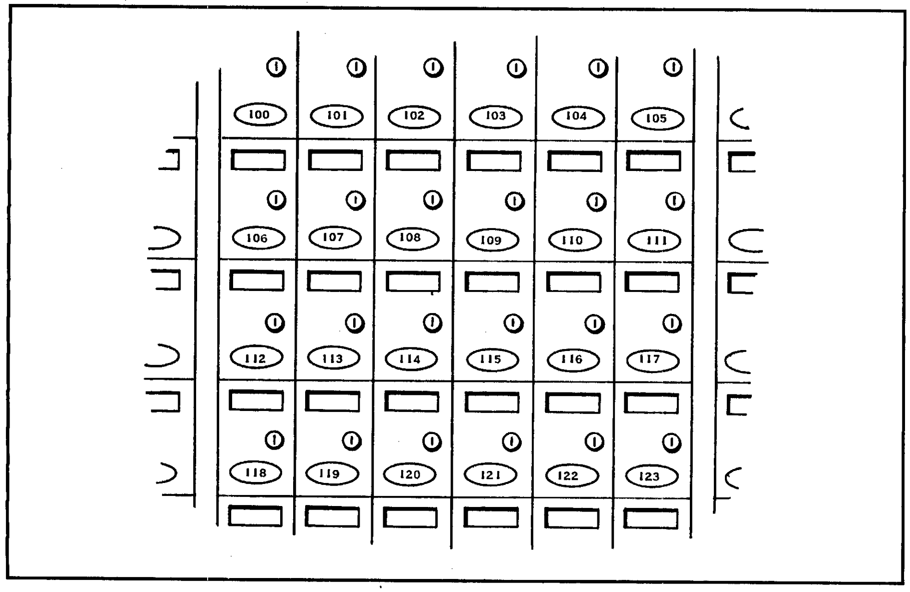


Figure 3-3 Post Office Mail Boxes

Each box is identified and located by its number. In the same way, storage is divided into locations, each with an assigned address. Each location holds a specific unit of data. Depending on the system, the unit of data may be a character, a digit, an entire record, or a word. To insert or remove data at a location, the address must be known, either to the programmer or to a control program (explained later).

When information enters a location, it replaces the previous contents of that location. However, when information is taken from a location, the contents remain unaltered. Thus, once located in storage, the same data may be used many times. In effect, a duplicate of the information is made available for processing.

The computer requires some time to locate and transfer information to or from storage. This is called *access time.* Storage units are available whose access time is so brief that it is measured in billionths of a second. To appreciate such a minute interval of time, consider a spaceship of the future traveling at 100,000 miles per hour. In one-millionth of a second, the spaceship would travel about 1 3/4 inches, in a nanosecond (one-billionth of a second), it would travel about one-thousandth of 1 3/4 inches.

Because so many references must be made to storage in all data processing operations, the access speed has a direct bearing on the efficiency of the entire system.

For example, semiconductor main storage is the most expensive storage device in terms of cost per storage location. However, main storage also provides the fastest access time; thus, it may be the most economical in terms of cost per machine calculation. Disk storage offers the advantages of lower direct cost to offset slower speed. Disk devices also offer the advantage of capacity in billions of digits. The largest storage system is the IBM 3850 Mass Storage System which has a capacity of 472 billion eight-bit characters.

Main Storage

The main storage of most computers, such as System/370, consists of micro-miniature integrated circuits which are designed to store data in addressable locations called bit cells. Electrical devices--such as transistors-- in each bit cell are capable of being in one of two states: on or off. The two states are used, therefore, to represent 0 or 1, plus or minus, yes or no, etc. This is the basis of the computer's binary system of storing information.

Since any specified location of storage must be instantly accessible (called random access), the bits are arranged so that any combination of ones (1s) and zeros (0s) representing a character can be electronically written into or read back from main memory when needed. Other electronic circuits are used to locate the desired bit cell as an x-y intersection on an imaginary grid covering all of the main storage. Thus any memory bit can be addressed (or accessed) in a few nanoseconds.

In a complicated manufacturing process, the electronic memory circuits are built into a single chip of semiconductor material. They are termed monolithic--derived from the Greek words denoting a "single stone"--because the electrical components are formed within a single chip of silicon. Transistors, diodes, resistors and capacitors can all be produced within such a monolithic structure.

Large numbers of electrical components on a single silicon chip are connected to create integrated electronic circuits by applying a thin metal film to the top of the chip to provide contacts to the components and interconnections between them. A typical IBM semiconductor chip measures about one-sixth of an inch per side and contains 2,048 bit cells formed by almost two meters of aluminum "wire" connecting over 14,000 electrical devices.

Due principally to the miniaturization of the bit cells made possible by semiconductor technology, main storage is usually found in the same box as the central processing unit. In contrast, auxiliary storage is located in separate boxes which are cabled to the central processor. Since electronic signals travel at the constant speed of light, shrinking the size of the main memory devices and locating them very close to the processor allows the computer to operate at extremely fast speeds.

It is beyond the scope of this book to fully explain semiconductor storage. There are innumerable ways to design the bit cell circuitry and many variations in the monolithic manufacturing processes. In all semiconductor main storage, however, the basic principle is to use tiny integrated electronic circuits to store zeros (0s) and ones (1s), resulting in binary codes which can represent any data required by the computer.

Magnetic Disk Storage

Disk storage provides IBM data processing systems with the ability to record and retrieve stored data sequentially or randomly (directly). It permits immediate access to specific areas of information without the need to examine sequentially all recorded data. Magnetic tape operations do not have this ability; tape searching must start at the beginning of the tape reel and continue sequentially through all records until the desired information area is found.

For an example of the application of direct access operations, as compared to sequential operations, consider the search for a word in a large unabridged dictionary. If the contents of the dictionary were stored on magnetic tape, the complete dictionary could be machine-read in about two minutes. A wide range of individual words would require an average of one minute to be found and read by the magnetic tape sequential method of searching. Using the dictionary, a human being would average about 1/5 of a minute per word, simply because he would limit his search for each word to an appropriate portion of the whole dictionary. That is, he would immediately go to a specific letter rather than start at the beginning of the dictionary and check each entry. This concept of limiting a search to a small section of the whole would permit direct access storage to perform the dictionary word search in a few thousandths of a second.

The high-speed access to data storage locations provided by direct access data processing permits the user to maintain up-to-date files and to make frequent direct reference to the stored data.

The magnetic disk is a thin metal disk coated on both sides with magnetic recording material. Disks are mounted on a vertical shaft; they are slightly separated from one another to provide space for the movement of read/write assemblies. The shaft revolves, spinning the disk (Figure 3-4).

Data is stored as magnetized spots in concentric tracks on each surface of the disk. Some units have 808 tracks on each surface. The tracks are accessible for reading and writing by positioning the read/write heads between the spinning disks.

The IBM 3336 Disk Pack is a high-speed, removable, interchangeable disk storage unit for the 3330 Disk Storage Unit. Due to its compact size, it is easy to move and handle. It can be interchanged between all modules of the same 3330 model, making communications between many different systems possible. The disk pack offers unlimited storage capacity. Because of the ease of removal and interchange of 3336s, storage capacity to be used in combination with any system is virtually unlimited. A pack consists of eleven disks, with data recording on 19 surfaces.

The magnetic disk data surface can be used repetitively. Each time new information is recorded and stored on a track, the old information is erased. The recorded data may be read as often as desired; data remains until written over.

The access mechanisms containing the read-write heads of a 3330 disk drive move in unison. Hence, without moving the access mechanisms, each read-write head has access to a particular track on its associated surface. The totality of all tracks of a disk that are accessible without motion of the access mechanisms is referred to as a *cylinder*. Therefore, a cylinder has 19 tracks.

Up to 13,030 bytes on each track, available under each of the ten access mechanisms (Figure 3-4), provide 247,570 bytes per cylinder. Data transmission is at the rate of 806,000 bytes per second. Minimum access time is 10 milliseconds; average is 30 milliseconds; maximum is 55 milliseconds.

Standard functions provided include:

Rotational Position Sensing, which increases channel availability by releasing the channel during most of record search time.

Multiple Requesting, which permits up to eight channel command sequences (one per disk drive) to be active in the storage facility, thereby permitting maximum use of the facility and contributing to improved response to I/O requests.

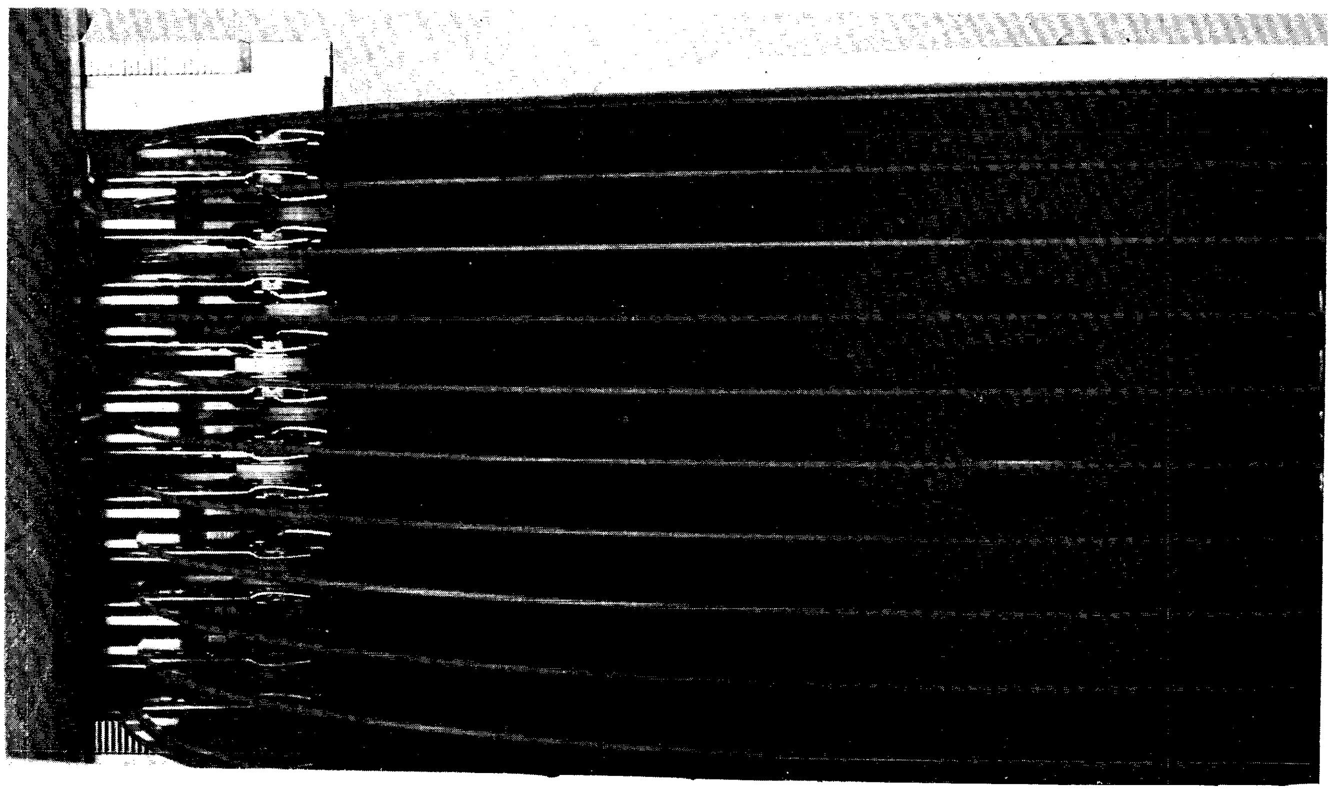


Figure 3-4 A Disk Storage Drive Showing Mounted Disk Pack and Access Arms

Command Retry, which enhances error recovery by permitting the channel and control unit to retry operations without CPU program intervention.

Extensive Error Detection and Correction Capabilities, which enhance data integrity and reliability.

A 3330 may have one or two independent modules, each storing up to 200 million 8-bit bytes. Up to four 3330's may be connected to a control unit. With eight removable and interchangeable 3336's per control unit, a total of 1.6 billion bytes of on-line storage and virtually unlimited off-line storage are provided.

Another type of disk storage is the 2305 FHSF (Fixed Head Storage Facility) shown in Figure 3-5. The 2305 Fixed Head Storage modules physically consist of non-removable rotating media and multiple element recording heads. The recording media consist of six 14-inch oxide coated disks revolving in an environmentally controlled air system.

The 2305 Model 1 has 384 addressable tracks, each with two read/write heads and a maximum capacity of 14,136 bytes per track (or 5.4 million bytes).

The 2305 Model 2 has 768 tracks, each with its own read/write head, and a maximum capacity of 14,660 bytes per track (or 11.2 million bytes).

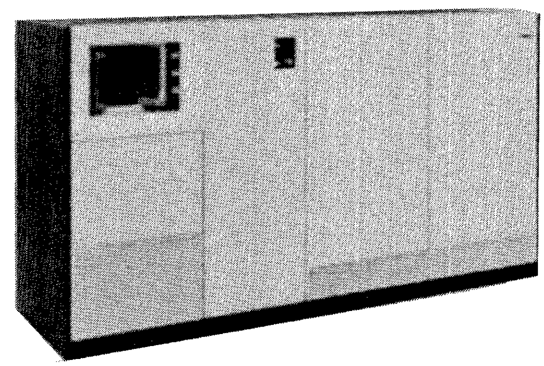


Figure 3-5 IBM 2305 Fixed Head Storage Facility

The 2305 features a high data transfer rate (3 million bytes per second for the smaller but faster Model 1 and 1.5 million bytes per second for the Model 2). Any record location can be accessed in an average of 2.5 and 5.0 milliseconds and within a maximum of 5.1 and 10.2 milliseconds for Models 1 and 2 respectively.

In order to increase data reliability without sacrificing the flexibility of earlier disk drives having removable disk packs, the IBM 3348 Data Module (Figure 3-6) was developed for use in IBM 3340 Disk Storage Drives (Figure 3-7). The recording medium is a sealed assembly (removable from the 3340) that contains magnetic disks, access arms, read/write heads, and spindle. Combining these components into a sealed integrated unit results in greater data reliability than that of earlier IBM disk drives, because read/write head alignment differences between drives are eliminated and also because the clean-air environment guards against airborne contamination.

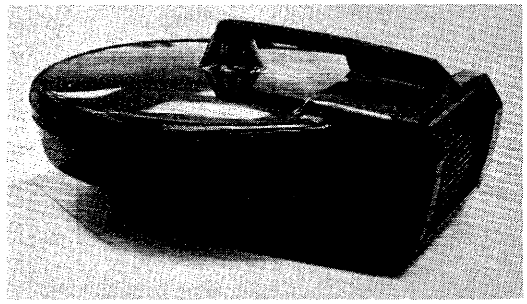


Figure 3-6 IBM 3348 Data Module

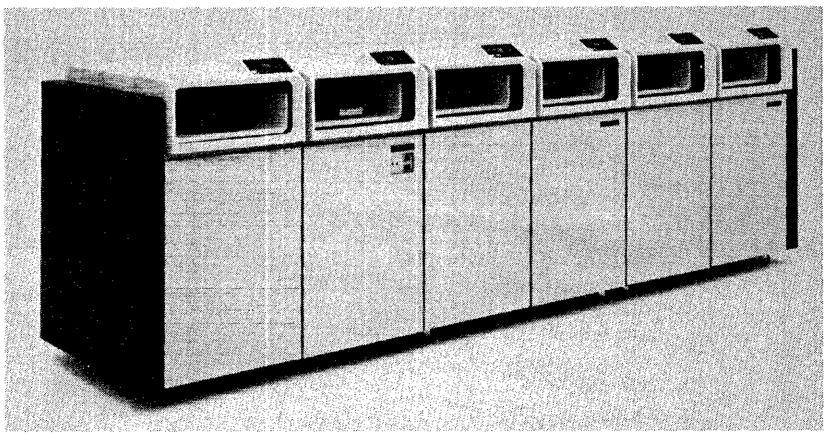


Figure 3-7 IBM 3340 Direct Access Storage Facility and IBM 3350 Direct Access Storage

Several models of the IBM 3348 Data Module are available ranging in size from 34.9 million bytes to 69.9 million bytes. One model having a total capacity of 69.9 million bytes has .5 million bytes accessible by fixed heads with zero seek time.

With the continued increase in the use of data communications and the significantly larger disk storage capacities available, a growing number of users, particularly those with large installations, rarely if ever change many of their disk packs. For all practical purposes, once they mount a disk pack containing application data, it remains online for as long as they plan to use it.

With non-removable disks, improved direct access storage performance at lower costs can be achieved. One such device is the IBM 3350 Direct Access Storage (Figure 3-7). The 3350 is a high-performance, large capacity subsystem that uses a fixed (sealed) storage medium, containing disks, spindle, access arms, and read/write heads. Each drive provides high-density storage for a maximum of 317.5 million bytes of data. Up to 8 drives may be connected to a controller yielding a total capacity of 2.54 billion bytes of on-line storage per controller.

Data transmission rates for the 3340 and 3350 are 885,000 and 1,198,000 bytes per second respectively. The average access time required is 25 milliseconds except for the fixed head portion of certain models (which portion has zero seek time). For the 3340, the average rotational delay time is 10.1 milliseconds while that for the 3350 is 8.4 milliseconds.

Mass Storage

Low-cost mass storage for as many as 472 billion bytes (the equivalent of 4,760 disk packs - 3336 Model 1 - or 2,360 Model 11 disk packs) under the control of a System/370 is provided by the IBM 3850 Mass Storage System (Figure 3-8). All of this data is accessible without manual intervention and at sharply reduced costs per megabyte transferred when compared to disk storage costs. The 3850 combines many of the advantages of tape and disk systems. Direct access device utilization is improved because only active data occupies direct access device space.

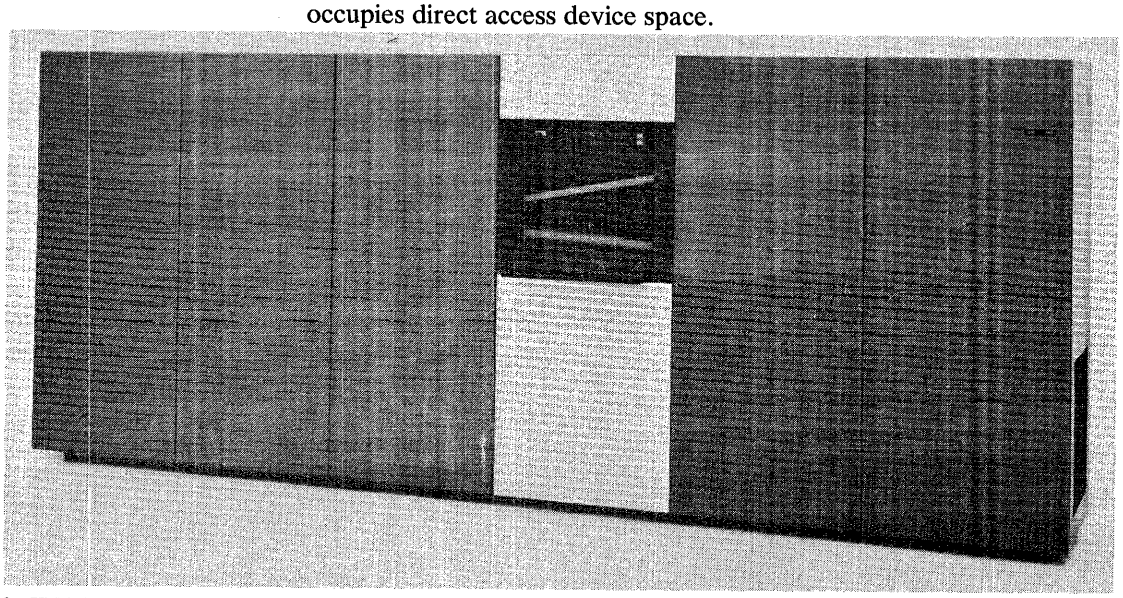


Figure 3-8 IBM 3851 Mass Storage Facility

Data under control of the 3850 Mass Storage System is stored on data cartridges. These data cartridges are stored in *cartridge storage cells* (Figure 3-9) within the 3850. When data is requested by System/370, it is transferred by the mass storage system from the cartridges to 3330 disk storage drives in a process called *staging*. Once data has been staged, it behaves the same way as any other data resident on a 3330 drive in terms of organization and accessing. After the data has been updated, it is *destaged* back onto the data cartridge.

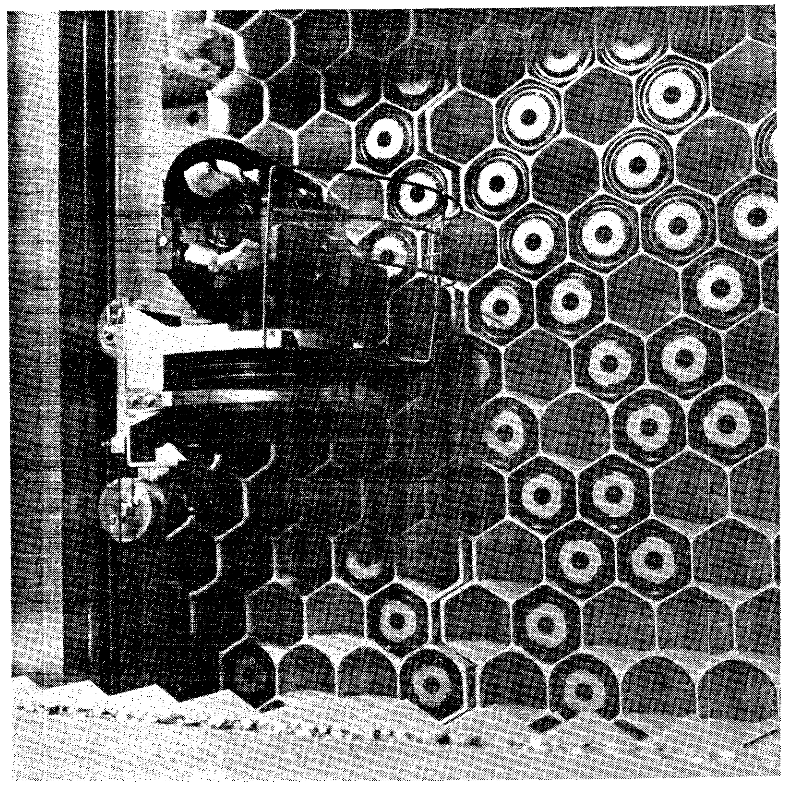


Figure 3-9 IBM 3850 Cell Structure

The data cartridge (Figure 3-10) is a plastic cylinder approximately 4 inches long and 2 inches in diameter. It contains a spool of magnetic tape approximately 3 inches wide and 770 inches long. This spool is removed from the cartridge when it is loaded into a data recording device. Data is written on the magnetic tape in the form of 3336 Disk Pack (Model 1) images. Two cartridges equal the capacity of one 3336 Disk Pack (Model 1). A pair of cartridges is called a *mass storage volume.*

Figure 3-10 Data Cartridges

Since all cartridges under control of the mass storage system are physically resident within the 3851 Mass Storage Facility, floor space required for storage can be greatly reduced.

Two accessors and their associated accessor controls move data cartridges from one location to another within the mass storage facility. Locations can be a storage cell, a data recording device, or the cartridge access station.

The cartridge access station permits manual entry and/or removal of data cartridges into and/or out of the mass storage facility.

Storage capacity ranges from 706 cartridges to 4,720 cartridges. From two to eight data recording devices are available.

Storage and Data Processing Methods

IBM data processing systems use two methods of data handling - sequential or batch processing and inline or direct access processing (see Figure 3-11). The application requirements determine which method is needed.

In either case, all data pertaining to a single application is maintained in files (often called data sets).

In sequential processing, these files are stored outside the computer - usually on magnetic tape - and they can be arranged in a predetermined sequence. The data may concern inventory, accounts receivable, accounts payable, payroll, and the like. Each file (data set) is made up of records, each containing information required to describe completely a single item. The sequence may be by item number, name, account number, or man number, but all files pertaining to a single application must be in the same sequence.

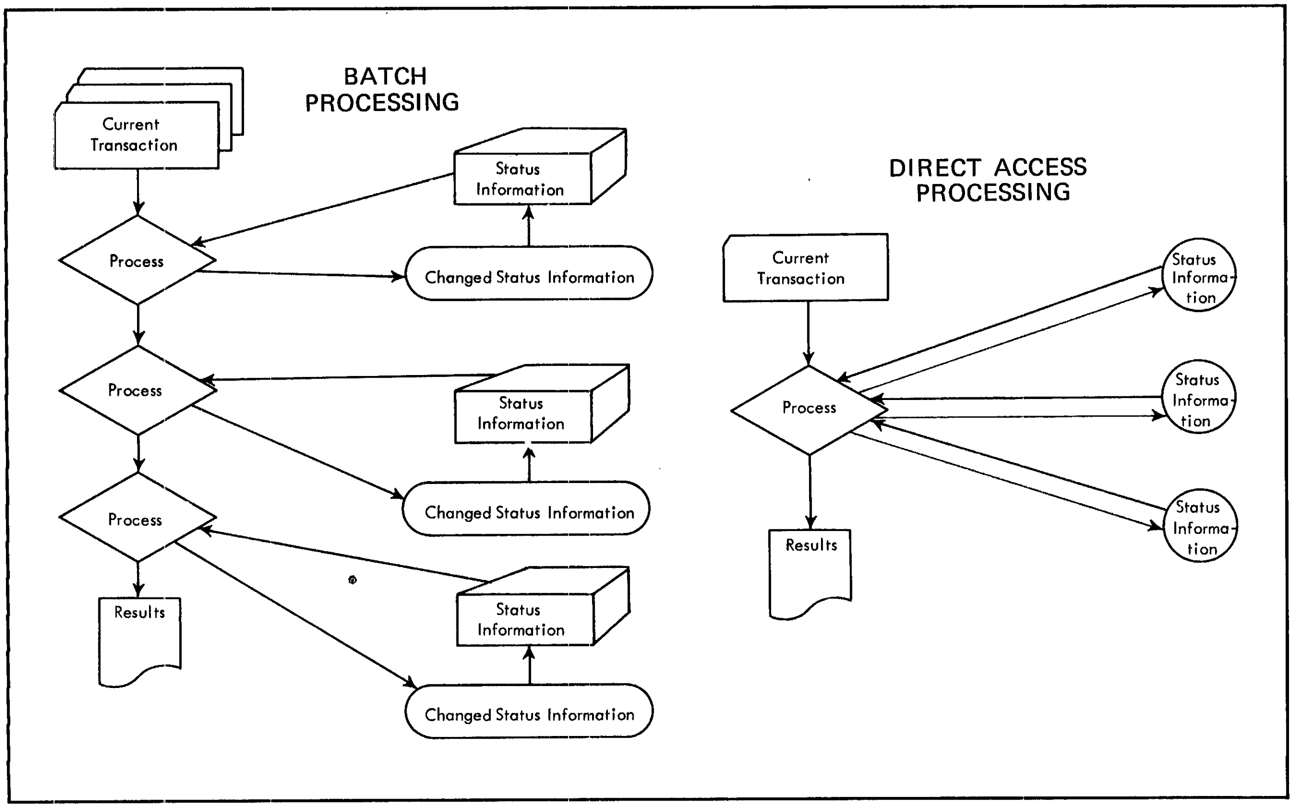


Figure 3-11 Batch and Direct Access Processing

In many cases, processing involves not only performing calculation on some parts of each record to arrive at balances, amounts, or earnings, but also involves adding, changing, or deleting records as new transactions occur. However, before transactions can be applied against the main or master file, they must also be arranged in the same sequence as the master file. For this reason, they are accumulated in convenient groups or batches.

The two files (data sets), master and transaction, now become input to the data processing system. One record or a small group of records (also called a block) is read into storage at a time. These are processed, and the result is written as output. When magnetic tape files are used, the output records with the updated results of current processing must be recorded on a separate tape, producing a new master that will be used as input the next time the job is to be done. The next group of records is read in, and the process is repeated. The series of repetitive operations continues under the direction of program instructions, record by record, until the input files are exhausted. The results form a revised master file, updated according to the current transactions. The new master file is in the same sequence as the original files.

Other output may also be produced as a by-product of the processing. This output may be records of delinquent accounts, bank orders, earnings statements, payroll checks, and so on. In every case, however, the sequence of all output remains the same as the sequence of the incoming data.

With sequential processing, the information in storage is transient. Consequently, the storage unit needs only enough capacity for program instructions, plus the largest element of data to be processed.

When direct access processing is used, transactions affecting the contents of the file (data set) are fed to the computer directly, as they occur. In this case, the computer locates the corresponding record or data in storage and adjusts this master record accordingly. Accounts or balances are constantly maintained and are available as output when needed. Transactions are not batched, and they need not be sorted before processing.

Section 4:

Central Processing Unit (CPU)

The central processing unit controls and supervises the entire computer system and performs the actual arithmetic and logical operations on data. From a functional viewpoint, the CPU consists of two sections - control and arithmetic/logical (Figure 4-1).

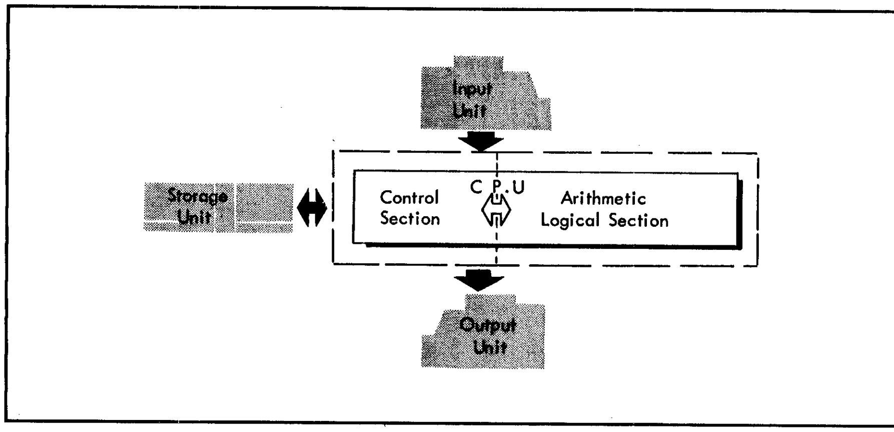


Figure 4-1 Central Processing Unit in the Data Processing System

The *control section* directs and coordinates all operations called for by instructions. This involves control of input/output devices, entry or removal of information from storage, and routing of information between storage and the arithmetic/logical section. Through the action of the control section, automatic, integrated operation of the entire computer system is achieved.

In many ways, the control section can be compared to a telephone exchange. All possible data transfer paths already exist, just as there are connecting lines between all telephones serviced by a central exchange (Figure 4-2).

The telephone exchange has a means of controlling instruments that carry sound pulses from one phone to another, ring the phones, connect and disconnect circuits, and so on. The path of conversation between one telephone and another is set up by appropriate controls in the exchange itself. In the computer, execution of an instruction involves opening and closing many paths or gates for a given operation. Some functions of the control section are to start or stop an input/output unit, to turn a signal device on or off, to rewind a tape reel, or to direct a process of calculation. In some System/370 models, a part of this section consists of a control device called *read-only storage* that contains circuits for performing operations designated by the operation codes. It also houses the *emulator* circuits that the user may select to make his System/370 perform programming instructions written for other computers.

The *arithmetic/logical section* contains the circuitry to perform *arithmetic* and *logical* operations. . The former portion calculates, shifts numbers, sets the algebraic sign of results, rounds, compares, and so on. The latter portion carries out the decision-making operations to change the sequence of instruction execution.

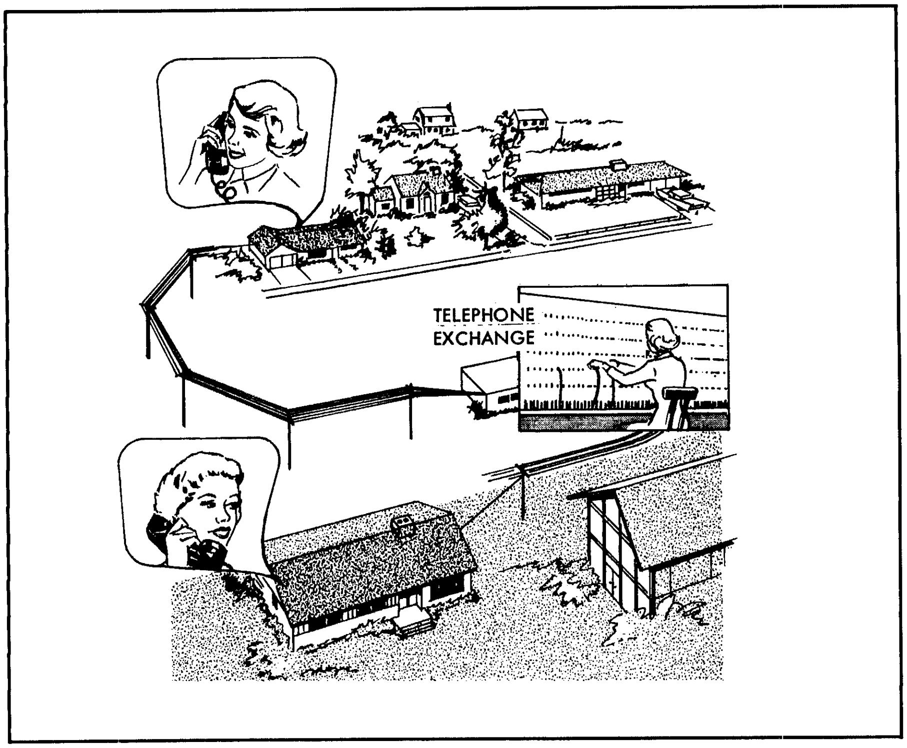


Figure 4-2 Telephone Exchange System

Functional Units

Register

A *register* is a device capable of receiving information, holding it, and transferring it as directed by control circuits. The electronic components used may be magnetic cores or transistors.

Registers may be named according to their function: an accumulator accumulates results; a multiplier-quotient holds either multiplier or quotient; a storage register contains information taken from or being sent to storage; an address register holds the address of a storage location or device; and an instruction register contains the instruction being executed (Figure 4-3). System/370 has general purpose registers, which are used for several functions, including storage addresses, index addresses, and data that is to be processed logically or arithmetically.

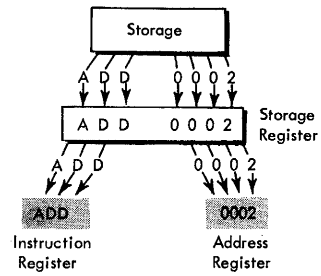


Figure 4-3 Register Nomenclature and Function

Registers differ in size, capacity, and use. In some cases, extra positions detect possible *overflow* conditions during an arithmetic operation. For example, if two eleven-digit numbers are added, it is possible that the result is a twelve-digit answer (Figure 4-4).

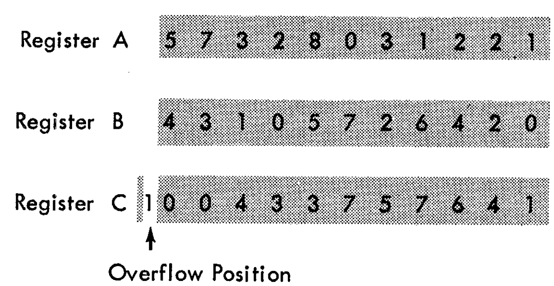


Figure 4-4 Overflow Condition Resulting from Addition

In this figure, register A holds one factor, and register B holds the other factor. The two factors are combined, and the result is placed in register C, where an overflow condition is indicated by the presence of data in the overflow position. The contents of other registers can be *shifted* right or left within the register and in some cases, even between registers. Figure 4-5 shows shifting of register contents three positions to the right. Positions vacated are filled with zeros, and numbers shifted beyond register capacity are lost.

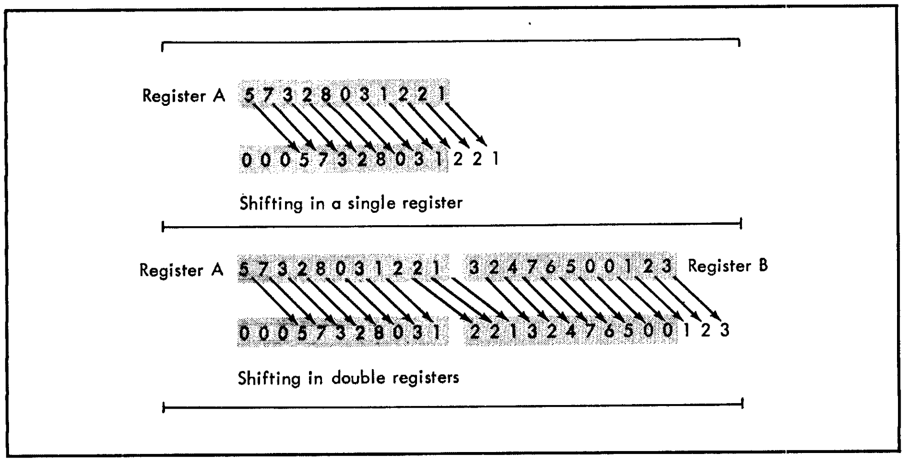


Figure 4-5 Types of Computer Register Shifting

In other instances, a register holds data while associated circuits analyze the data. For example, an instruction can be placed in a register, and associated circuits can determine the operation to be performed and locate the data to be used. Data within specific registers may also be checked for validity.

The more important registers of a system, particularly those involved in normal data flow and storage addressing, may have small lights associated with them. These lights are located on certain machine consoles (Figure 4-6) for visual indication of register contents and various program conditions.

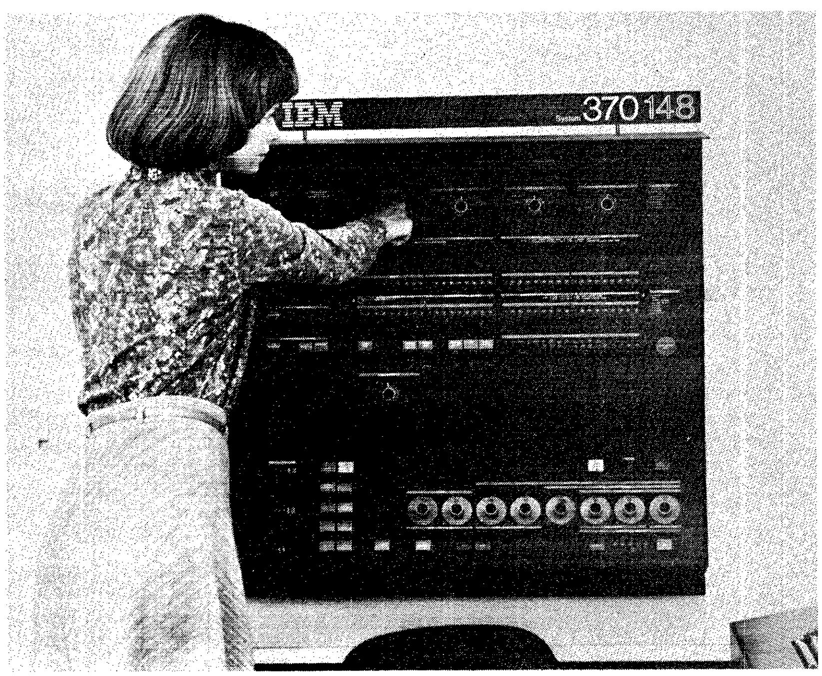


Figure 4-6 Typical System Console (System/370 Model 148)

Counter

The *counter* is closely related to a register, and may perform some of the same functions. Its contents can be increased or decreased. The action of a counter is related to its design and use within the computer system. Like the register, it may also have visual indicators on the system console.

Adder

The *adder* receives data from two or more sources, performs addition, and sends the result to a receiving register or accumulator. Figure 4-7 shows two positions of an adder circuit, with input from registers A and B. The sum is developed in the adder. A carry from any position is sent to the next-higher-order position. The final sum goes to the corresponding positions of the receiving register.

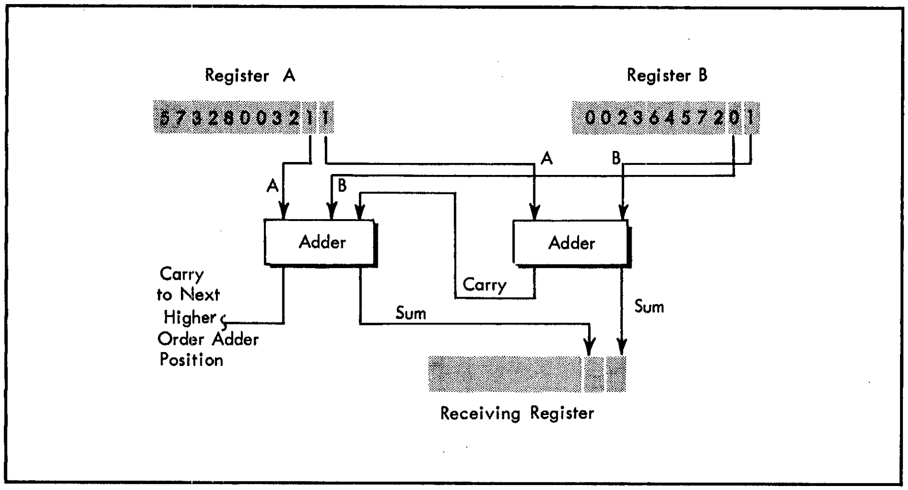


Figure 4-7 Adders in a Computer System

Machine Cycles

All computer operations take place in fixed intervals of time. These intervals are measured by regular pulses emitted from an electronic clock at frequencies as high as 17.2 million per second. A fixed number of pulses determines the time of each basic machine cycle.

In computer usage, time references are stated in such terms as *milliseconds*, *microseconds*, and *nanoseconds*. These terms may convey no meaning unless it is realized just how short an interval a millisecond is. For example, the blink of an eye takes about one-tenth of a second or 100 milliseconds!

The following table establishes some additional terms and abbreviations:

.1	=1/10 second
	=100 milliseconds
.001	=1/1,000 second
	=1 millisecond (ms)
.000001	=1/1,000,000 second
	=1 microsecond (μ sec)
.000000001	=1/1,000,000,000 second
	=1 nanosecond (ns)

Within a machine cycle, the computer can perform a specific machine operation. The number of operations required to execute a single instruction depends on the instruction. Various machine operations are thus combined to execute each instruction.

Instructions usually consist of at least two parts, an operation and an operand. The *operation* tells the machine which function to perform: read, write, add, subtract, and so on. The *operand* can be the address of data or an instruction in main storage, the address of data or programs in secondary storage, or the address of an input/output unit. It can also specify a control function, such as shifting a quantity in a register, or backspacing and rewinding a reel of tape. In System/370, most instructions contain two operands.

To receive, interpret, and execute instructions, the central processing unit must operate in a prescribed sequence, which is determined by the specific instruction and is carried out during a fixed interval of timed pulses.

Instruction Cycle

The first machine cycle required to execute an instruction is called an *instruction cycle*. The time for this cycle is instruction or I-time. During I-time:

1. The instruction is taken from a main storage location and brought to the central processing unit.
2. The operation part is decoded in an instruction register. This tells the machine what operation is to be performed.
3. The operand is placed in an address register. This tells the machine what factors are to be used in the operation.
4. The location of the next instruction to be executed is determined.

At the beginning of a program, an instruction counter is set to the address of the first program instruction. This instruction is brought from storage, and, while it is being executed, the instruction counter automatically advances (steps) to the location corresponding to the space occupied by the next stored instruction. If each instruction occupies one storage position, the counter steps one; if an instruction occupies five positions, the counter steps five. By the time one instruction is executed, the counter has located the next instruction in program sequence. The stepping action of the counter is automatic. In other words, when the computer is directed to a series of instructions, it executes them one after another until instructed to do otherwise.

Assume that an instruction is given to add the contents of storage location 2 to the contents of a general purpose register that will be used as an accumulator register. Figure 4-8 shows the main registers involved and the information flow lines.

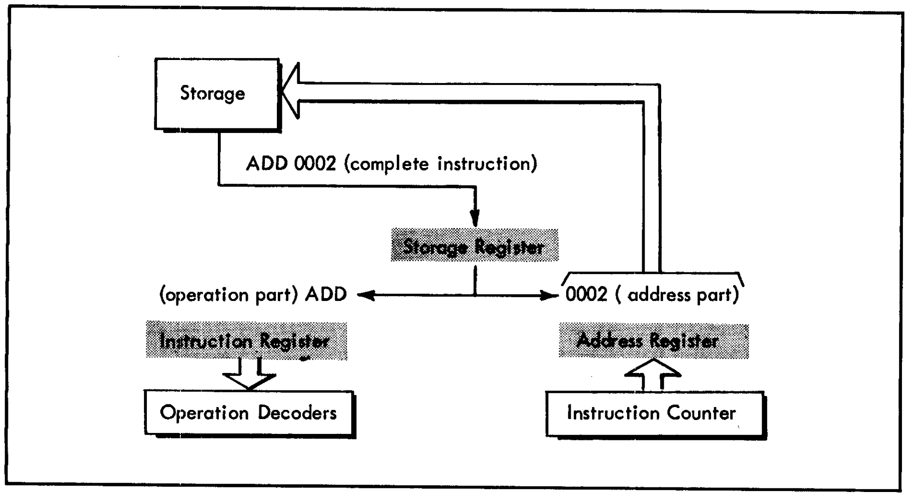


Figure 4-8 Computer I-Cycle Flow Lines

I-time begins when the instruction counter transfers the location of the instruction to the address register. This instruction is selected from storage and placed in a storage register. From the storage register, the operation part is routed to the instruction register and the operand to the address register. Operation decoders then condition proper circuit paths to perform the instruction.

Execution of instructions does not necessarily have to proceed sequentially. Certain instructions alter the process of sequential execution unconditionally. In this case, an instruction brought from storage indicates that the next sequential instruction is not to be executed but that one located in another position is next. The normal stepping of the instruction counter can also be reset back to the beginning of the program so that the entire program can be repeated for another incoming group of data.

This *branching* (transfer) to alternative instructions may also be conditional. The computer can be directed to examine some indicating device and then branch if the indicator is on or off. Such an instruction could say, in effect, "look at the sign of the quantity in the accumulator; if the sign is minus, take the next instruction from location 5000; if the sign is plus, proceed to the next instruction in sequence." The instruction counter is set according to one of the two possible storage locations (5000 or the location of the next instruction in sequence). The logical path followed by the computer (that is, the precise sequence of instructions executed) may be controlled either by unconditional branching or by a series of conditional tests applied at various points. However, the arrangement of instructions in storage is not normally altered.

Execution Cycle

I-time is usually followed by one or more machine cycles that occur during execution for E-time. The number of *execution cycles* required depends on the instruction to be executed. Figure 4-9 shows the data flow following I-time illustrated by Figure 4-8.

The E-cycle starts by removing from storage the information located at the address (0002) indicated by the address register. This information is placed in the storage register. In this case, one of the factors to be added is placed in the adders together with the number from the accumulator. The contents of the storage register and the accumulator are combined in the adders, and the sum is returned to the accumulator.

The address register may contain information other than the storage location of data. It can indicate the address of an input/output device or a control function to be performed. The operation part of the instruction tells the computer how to interpret this information.

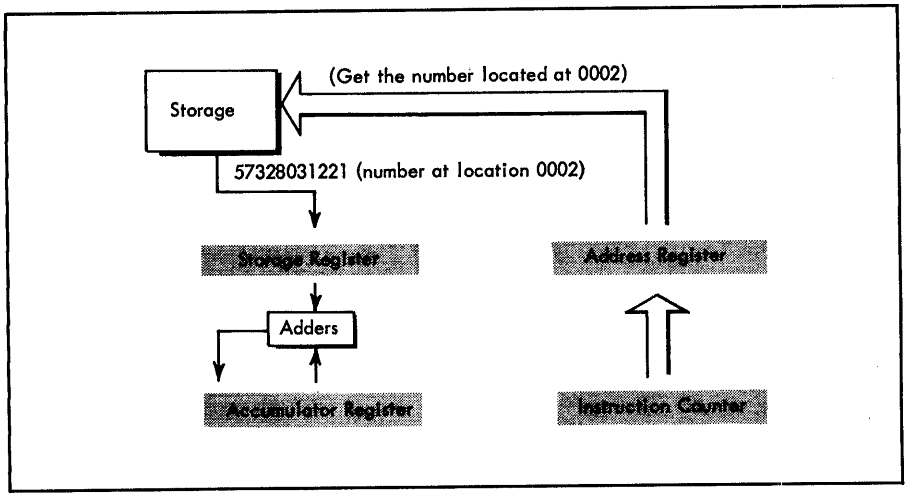


Figure 4-9 Computer E-Cycle Following an I-Cycle

Serial and Parallel Operation

Computers and portions of computers are classified as either serial or as parallel, depending on the method used to perform arithmetic. Essentially, all arithmetic is performed by addition.

In a serial computer, numbers to be added are considered one position at a time (unit, ten, hundred, etc.), in the same way that addition is done with paper and pencil. Whenever a carry is developed, it is retained temporarily and then added to the sum of the next-higher-order position.

The time required for serial operation depends on the number of digits in the factors to be added. Serial addition is shown in Figure 4-10.

	1st Step	2nd Step	3rd Step	4th Step
Addend	1234	1234	1234	1234
Augend	2459	2459	2459	2459
Carry	1	1		
Sum	3	93	693	3693

Figure 4-10 Serial Addition

In a parallel computer, addition is performed on complete data words. The words are combined in one operation, including carries. Any two data words, regardless of the magnitude of the numbers contained in the words can be added in the same time. Figure 4-11 shows parallel addition.

	00564213
Numbers being added	00000824
Carry	1
Final Result	00565037

Figure 4-11 Parallel Addition

Fixed-Length and Variable-Length Words

Data can be addressed and processed by a computer system using either fixed-length and variable-length words.

In operations using fixed-length words, information is handled and addressed in units or words containing a predetermined number of positions. The size of a word is designed into the system, and it normally corresponds to the smallest unit of information that can be addressed for processing in the central processing unit. Records, fields, characters, or factors are all manipulated in parallel as words; registers, counters, accumulators, and storage are designed to accommodate a standard word.

In operations using variable-length words, data handling circuitry is designed to process information serially as single characters. Records, fields, or factors may be of any practical length within the capacity of the storage unit. Information is available by character instead of by word.

Operation within a given data processing system may be entirely fixed-length, entirely variable, or a combination.

Floating-Point Operation

Mathematicians and scientists use logarithms to simplify mathematical manipulations of very large numbers and very small fractions. Similarly, in computers having the capability, *floating-point* instructions are used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

All central processing units that handle floating-point arithmetic do it by converting numbers (integers, fractions, or improper fractions) into the exponential form on the right-hand side of the equation. For example:

$N = b^e \times f$ where:
N = number
b = numbering system base (such as 2 for binary system, 10 for decimal system, 16 for hexadecimal system)
e = exponent (power to which the numbering system base is raised to make the expression equal to N - the same as the characteristic of a logarithm)
f = fraction (similar to the mantissa in a logarithm)

The base used depends on the internal numbering system. For System/370, it is 16.

The range of exponents that can be used depends on the space allowed and on whether a position in the computer is allocated to contain a sign (+ or -) for the exponent. The space allotment is seven bits without sign in System/370. The bits within the field can represent numbers from 0 through 127.

When no sign position is allowed for an exponent, a value in the center of the total range of possible exponents is chosen to represent the exponent 0. Positive exponents are added to this "middle" value; negative exponents are subtracted from it. This "scaled exponent" is called a *characteristic* (and therefore differs from the characteristic of a logarithm). In System/370, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess-64 notation. The range covered by the magnitude of a normalized floating-point number is therefore approximately from 7.2×10^{75} down through 5.4×10^{-79}. Both positive and negative quantities have a true fraction, the sign being indicated by the sign bit. The number is positive or negative, depending on whether the sign bit is zero or one, respectively.

The range of fractions in computers using fixed-length words is determined by the space remaining in the word (or words), allocated to express the floating point number after the exponent and sign have been provided for. In System/370, floating point data occupies a fixed-length form which may be either a four-byte (short) format, an eight-byte (long) format, or a 16-byte (extended) format. Therefore, the fraction field may have up to 6, up to 14, or up to 30 hexadecimal digits (24, 56, or 120 bits) depending upon whether one, two, or four 32-bit words are to be used in carrying out the precision of the fraction.

Floating-point is so called because, using the exponent x fraction method of expressing a number, we can vary the exponent and then "float" the hexadecimal point (in hexadecimal computers) correspondingly and still not change the value of the number (Figure 4-12).

	Hexadecimal Digit Values: Integer				Fraction		
	256	16	1	.	1/16	1/256	1/4096
37 = 1 x 37 = 16^0 x		2	5				
37 = 16 x 37/16 = 16^1 x			2	.	5		
37 = 256 x 37/256 = 16^2 x				.	2	5	
37 = 4096 x 37/4096 = 16^3 x				.	0	2	5
37 = 1/16 x 592 = 16^{-1} x	2	5	0	.			

Figure 4-12 Different Floating-Point Expressions of the Same Value

When the point falls just to the left of the high-order significant digit (in hexadecimal) of the fraction, the floating-point expression is called *normalized*. A quantity can be represented with the greatest precision by a floating-point number of given fraction length when that number is normalized. When numbers are initially entered into the central processing unit of a computer as floating-point numbers, they take this normalized form. As arithmetic operations are performed on them, the points may float to the right or to the left as the computer seeks to get all the numbers involved in the operation into the same power of 16 (this is similar to changing fractions with different denominators into fractions with a common denominator in order to add them). The numbers then become decidedly unnormalized. Eventually, however, the computer normalizes the final result. Of course, since normalization applies to hexadecimal digits, the three high-order bits of the fraction of a normalized number may be zero. It is also obvious that a number with a zero fraction cannot be normalized.

Converting a decimal number (assume the number 149.25) into a single floating-point word, as used in System/370, is done as follows:

1. Separate the number into the integer and the fraction;

   149.25 = 149 plus 0.25

2. Convert the decimal integer to hexadecimal;

   (see Figure 2-17) $149_{10} = 95_{16}$

3. Convert the decimal fraction to hexadecimal;

   (see Figure 2-18) $0.25_{10} = 0.4_{16}$

4. Combine the two and express in normalized form (as a fraction times an exponent to the base 16);

   $95.4_{16} = (0.954 \times 16^2)_{16}$

5. Since 64 is the midpoint of the characteristic range, add the exponent (2) to 64 to get the characteristic:

   64 + 2 = 66 = 1000010

6. Convert the fraction to binary, and group it hexadecimally;

   $.954_{16} = .1001\ 0101\ 0100$

7. The floating-point word for decimal 149.25 (hexadecimal 95.4) appears as

   Sign* Characteristic
   0 100 0010
   Fraction
   1001 0101 0100 0000 0000 0000

*Zero (0) is used, since the value of 149.25 is positive.

Section 5:

Input/Output Devices

An input/output unit is a device for putting in or getting out data from storage (Figure 5-1).

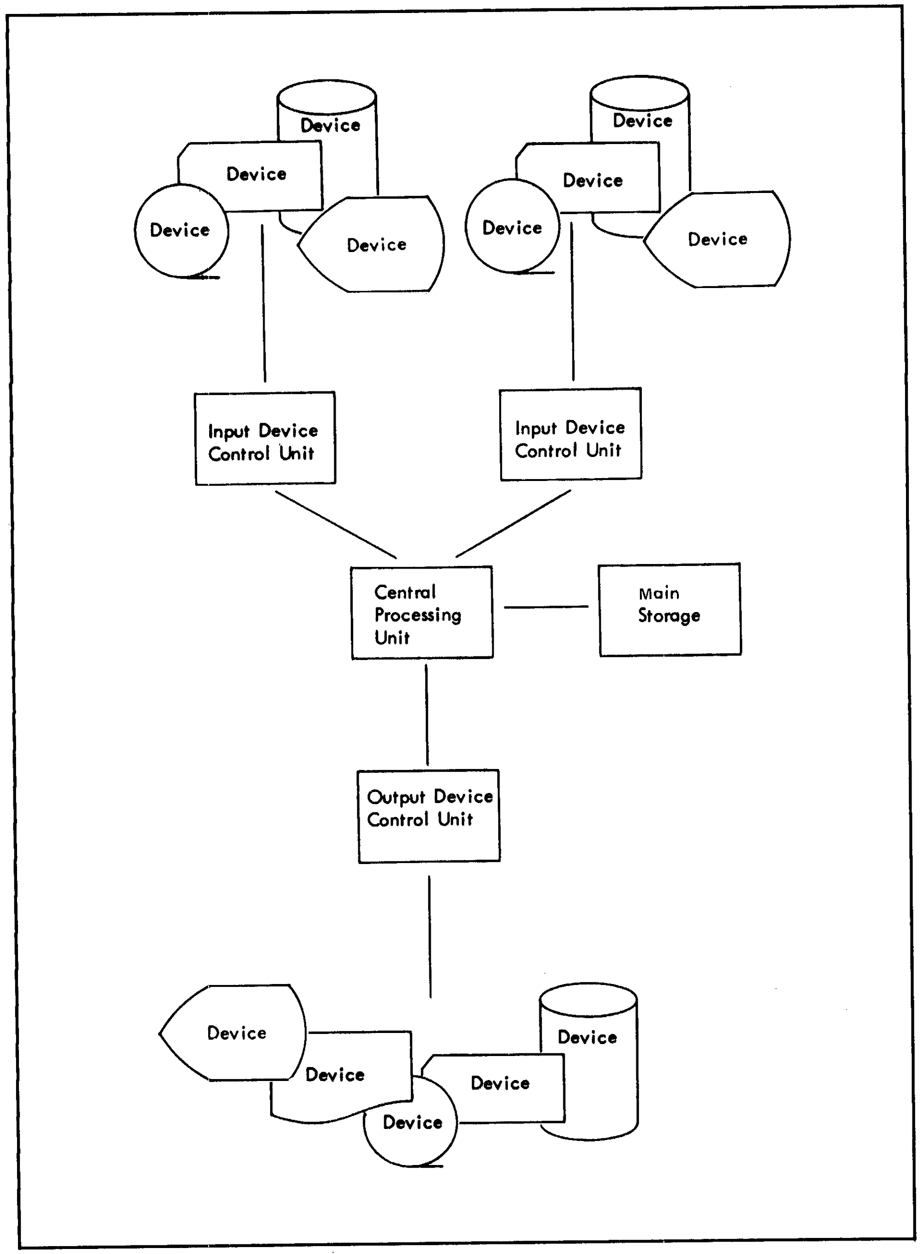


Figure 5-1 Input/Output Units in the Data Processing System.

Usually, device operation is initiated by a program instruction that generates a *command* to an input/output *channel*. A *control unit* decodes the command and effects operation of the device.

Some control units control a particular type of device, such as disk drives; for example, the IBM 3830 storage control units can control the 3330, 3333, 3340, 3344, and 3350 disk drives. Others control different input/output devices; for example, the IBM 3272 control units can control the 3277 display stations and the 3284, 3286 and 3288 printers.

Input devices sense or read data from cards, magnetic tape, paper, magnetic ink characters inscribed on paper documents, images on 35mm microfilm, or remote terminals via communication lines. The data is made available to the main storage of the system for processing. Output devices record or write information from main storage on cards, magnetic tape, and paper tape, prepare printed copy, produce microfilm images, make graphic displays, or transmit information over a teleprocessing network.

Reading takes place as the input medium physically moves through an input device. Information is sensed or read and is converted to a code used within the computer system. The information is then transmitted to main storage.

Writing involves transferring data from primary storage to an output device. The computer code is made compatible with the output medium.

Most input/output devices are automatic; once started, they continue to operate as directed by the stored program until the entire file is processed. Instructions in the program select the required device, direct it to read or to write, and indicate the storage location into which data will be entered or from which data will be taken.

Some I/O devices are used for manual entry, and no medium for recording data is involved. Instead, data is entered directly into storage using a keyboard or switches. Locally, these devices may be a console keyboard, local terminals (such as the IBM 2740s), or display terminals. Remotely, many types of teleprocessing terminals may be used. Instead of a recording medium, these terminals may require some amount of internal storage for holding (and perhaps analyzing) signals until a short message is completed, or until the terminals are polled (requested to transmit) and selected for data transfer.

Control Units

The type of information buffering required to coordinate the operations of the input/output device with the central processing unit (sometimes through transmission hookups) is one of the functions of the control unit. Other common functions are checking, coding, and decoding. If several similar devices are operating through one control unit, two principal functions are (1) determining priority of servicing, and (2) signaling device identification when requesting service for the input devices.

Conversely, on the way out, the control unit directs the data to the addressed output unit.

In some data processing systems, the traffic routing function of the control unit is referred to as orders. In such systems, the orders, as written by the programmer, consist solely of the address of the input/output device that the control unit must prepare to read or to write. In System/370, orders are part of a control command, relayed through a channel to the input/output control unit; they instruct the control unit to have a device perform a specified auxiliary operation - one that doesn't move data, such as rewind or seek.

Channels

Whereas the control unit is either included under the cover of an input/output device or located very close to a group of such devices, the channel (or channels) is contained within the central processing unit or is a separate piece of equipment near the CPU. The channel relieves the CPU of the burden of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O operations. It might be thought of as the computer's control unit for one or more input/output control units. It is almost a separate, small CPU devoted exclusively to managing the input/output control units and devices assigned to it. After the channel has once been activated by an initializing instruction from a program being executed in the CPU, it carries out one or more commands that are similar to a section (subroutine) of a program, but the important difference is that of overlapping operations. The program in the CPU can be continuing with other jobs while the channel is carrying out its own program of bringing data into or out of the main storage. Sometimes it is interleaving input and output in a seemingly simultaneous fashion, working with several input/output control units at once, and maintaining the proper destinations for the messages - whether they be storage allocation (for input) or control unit and device (for output).

The steps in a program in the CPU are called instructions; the steps in a program for a channel are called *commands*. Each command has an operation code that tells the channel what to do (for instance: read, write, control, sense, etc); if it is a command that involves a data transfer, the command also has an address telling where to get or where to put the data in the storage system of the computer; if it is a control command that does not involve a data transfer, either it contains the order to be passed on to the control unit, or (in some computer systems) it contains the address of a location in storage where the order is located.

Just as the CPU is free to continue with its program once it has given an instruction to start a channel on its independent program of commands, so a channel is free to step through other commands (probably starting or terminating some other input/output transfer of data) as soon as it has commanded the control unit what to do and given it an order specifying the particular device. Thus, a channel is an intermediary input/output device that is constantly juggling the various input/output operations to make the most efficient use of time, not only by overlapping different input and output but by doing so without tying up the CPU.

As soon as a particular input/output transaction is completed, the device control unit signals the channel, which, in turn, signals the CPU with an *interrupt,* meaning: "My particular job is done. As soon as convenient, use the data I have given you (if it was an input operation) and give me another comand."

This idea of automatic interrupts (built into the design of the data processing system components), combined with carefully preprogrammed commands to the channels and orders to the control units, leads to a far greater total amount of data handling per unit of time (sometimes described as *throughput*) than used to be possible.

Validity Checks

All data transferred between the input/output units and storage is automatically checked for validity. First, data is checked before being sent by the input device and checked when received by the output device. Second, certain data checks are also made within the central processing unit as it receives or sends data; for example, if a 5 is entered instead of a 4, this error cannot be detected by this means. However, if the indicated number of character is represented or coded incorrectly on the medium or within the machine, this is automatically detected.

Indicators, Keys, and Switches

All input/output units have indicator lights as well as operating keys and switches. (Figure 5-2). The indicator lights show the status of a unit: on, off, ready, selected, and so on. The operating keys and switches are used primarily to start and to stop operations manually. The specific functions and use of the indicators, keys and switches are described in the IBM manuals for particular machines and systems.

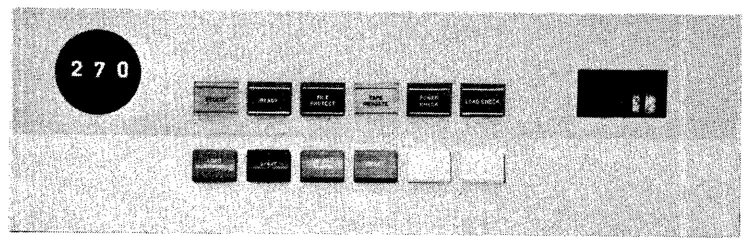


Figure 5-2 Operator Panel

Punched Card Devices

Card Readers

Card reading devices introduce IBM punched card data into the computer system. The card reader moves or feeds cards past a reading unit that converts the data on the card into an electronic form. Two types of reading units are used: reading brushes or photoelectric cells.

In the brush type reader, cards are mechanically moved from a card hopper, through the card feed unit, and under reading brushes. The reading brushes electrically sense the presence or absence of holes in each column of the card (Figure 5-3).

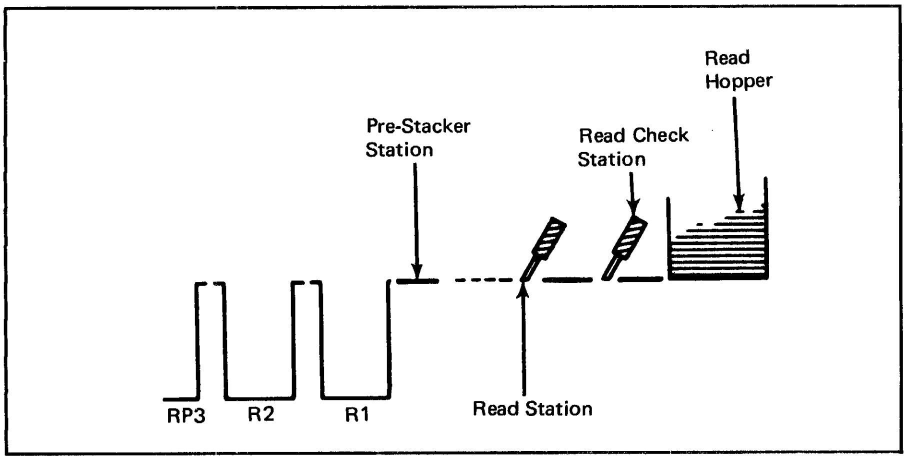


Figure 5-3 Read Feed

This electric sensing converts the information of the card to electrical impulses that can be detected by the card reader circuitry and stored as data. After cards are read, they are moved from the card feed unit and placed in the card stacker in the same sequence in which they were fed into the reader. Some readers have two sets of reading brushes. As a check on the validity of the reading process, each card can be read twice as it moves through the card feed unit.

The photoelectric type of card reader performs the same functions as the brush type; the difference is in the method of sensing the holes. Photoelectric cells are activated by the presence of light. As the punched card is passed over a light source in the card reader, light passing through the punched holes activates photoelectric cells, one cell for each column of the card.

Card reading speeds vary from about 12 cards to 1200 cards a minute, depending on the type of card reader.

Card Punches

Output from the computing system is recorded in cards by a card punching device.The card punch automatically moves blank cards, one at a time, from the card hopper, under a punching mechanism that punches data received from storage (Figure 5-4). After the card is punched, it is moved to a checking station, where the data is read and checked with the information received at the punching station. The card is then moved to the stacker.

Card punching speeds vary from about 12 to 500 cards per minute, depending on the type of card punch.

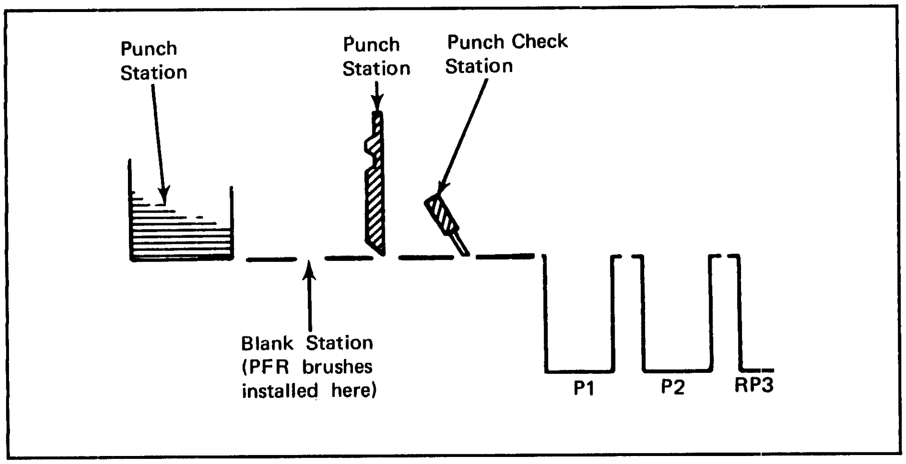


Figure 5-4 Punch Feed

Diskette Input/Output Devices

The IBM 3540 Diskette Input/Output Unit (Figure 5-5) is an efficient and economical data entry and output device for System/370. There is no contention for devices between data entry and processing programs. Designed around the IBM diskette (Figure 5-6), the 3540 reads up to 3,600 diskette records per minute and writes up to 2,200 records per minute.

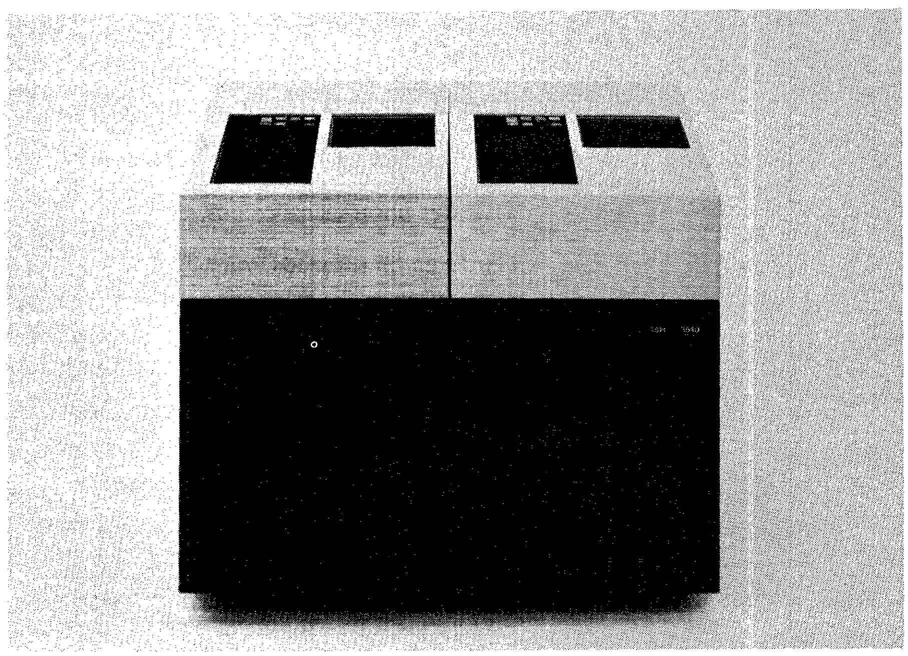


Figure 5-5 IBM 3540 Diskette Input/Output Unit

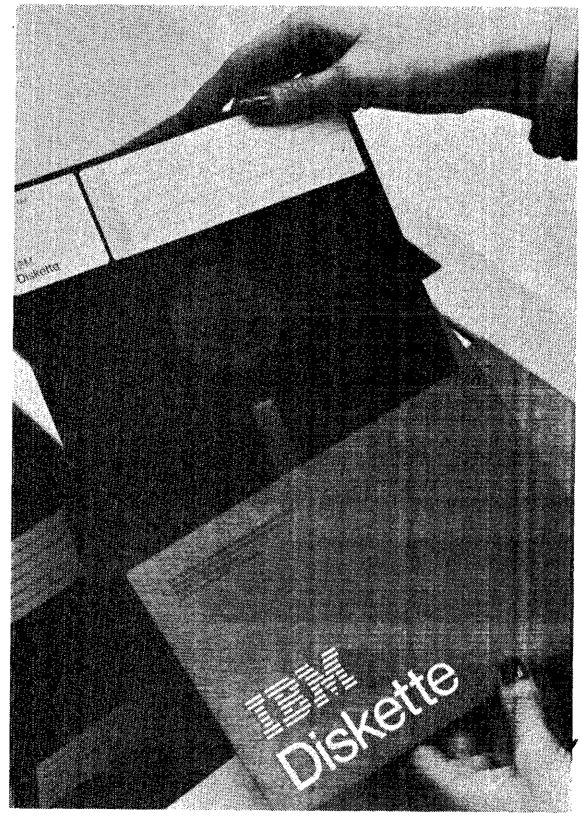


Figure 5-6 IBM Diskette

The diskette is a single magnetic disk, sealed in a plastic jacket about 8 inches square. It weighs less than 2 ounces, is reusable, is interchangeable with other diskette units, and can be easily corrected and updated.

The diskette can store as many as 1,898 128-character records, equivalent to as many characters as can be put on 3,036 80-column cards.

The 3540 is available with one or two diskette drives. Each drive has a separate hopper and a separate stacker holding up to 20 diskettes. Under program control, diskettes are automatically fed, one at a time, from the hopper and mounted on the drive spindle for a read or write operation. At the end of the operation, the diskette is automatically removed from the spindle and stacked, thus permitting uninterrupted processing. Diskettes are called and removed by the program while the unit is operating. Because each diskette goes through label checking before the reading of data, data integrity is assured.

Magnetic Tape Units

Increasing internal speeds of computers demand high-speed input and output devices so that system operations are not held back waiting for input or by an inability to get processed data out of the computer. Magnetic tape units, with their dual capability of input and output, have provided continued increases in the speed of data transmission to and from the computer and, at the same time have provided increased data storage capacity per reel of tape. At one time, in order to facilitate data entry from punched cards as well as the preparation of reports and punched card output, dedicated card-to-tape, tape-to-print, and tape-to-card devices were used. Such devices were no longer required when IBM computers with multiprogramming capabilities became available.

Magnetic tapes are frequently used to transport large volumes of data. For example, many companies submit income tax information to the federal government by means of magnetic tapes.

Magnetic tapes have been and are still being used extensively for the off-line storage of large volumes of data especially inactive files.

All magnetic tape units are basically similar in operation, but design improvements have brought about functional differences, increased tape applications, and easier operation.

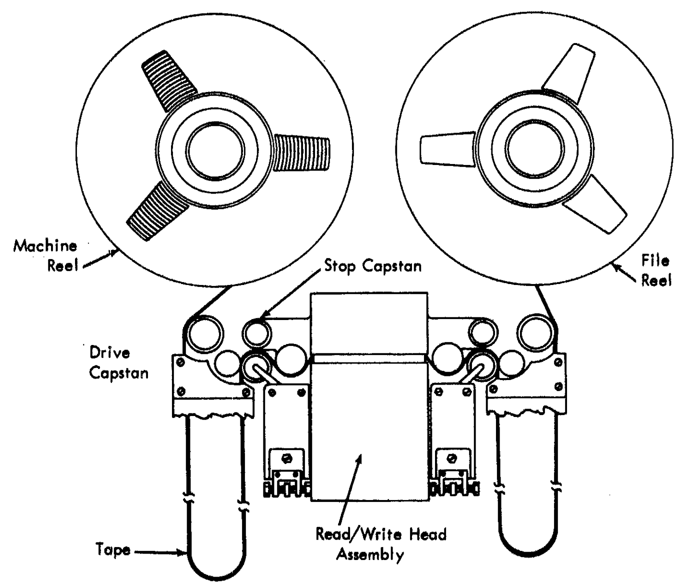
Figure 5-7 Tape Reel Cartridge Operation

The IBM 3400 series units transport tape past the recording head in a continuous movement and at a constant speed. The tape is always in motion during reading and writing. Tape moves at 12.5, 25, 50, 75, 125, or 200 inches per second, depending on the tape unit.

A full reel of 1/2 inch wide tape holds 2400 feet of tape, weighs about four pounds, and can contain data equivalent to about 1.8 million fully punched cards.

Loading Tape Units

Loading of tape cartridges is accomplished automatically after the dust-resistant cartrige is placed in the drive by the operator (Figure 5-7). Before operation, the tape is automatically threaded by the tape unit (Figure 5-8).



During reading or writing on all tape drives, tape is transferred from the file reel, past the read/write head, to the machine reel. During backspacing or rewinding, tape movement is from the machine reel to the file reel.

Figure 5-8 Tape Feed Schematic

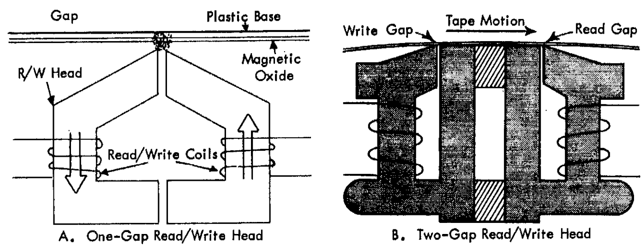
During operation, tape moves from the file reel through one vacuum column across the read/write head, through the other vacuum column to the machine reel. The loop in each vacuum column acts as a buffer to prevent high-speed starts and stops from breaking the tape. Vertical vacuum columns are used in some units; horizontal columns are used in the incremental and variable-speed machines.

Vacuum-actuated switches in the columns control magnetic clutches that permit the two reels to rotate independently. The file reel feeds tape when the loop reaches a minimum reserve length in the near vacuum column, and the machine reel winds tape when the loop reaches a point near the bottom of the other vacuum column.

Tape may be rewound or backspaced to the beginning of the reel. Rewind speeds are as high as 640 inches per second.

Reading From and Writing on Magnetic Tape

The magnetic tape unit reads or writes data as tape moves past the read/write head. Two types of heads have been used in IBM magnetic tape units, but the general principles of writing and reading tape are the same for both types (see Figure 5-9).



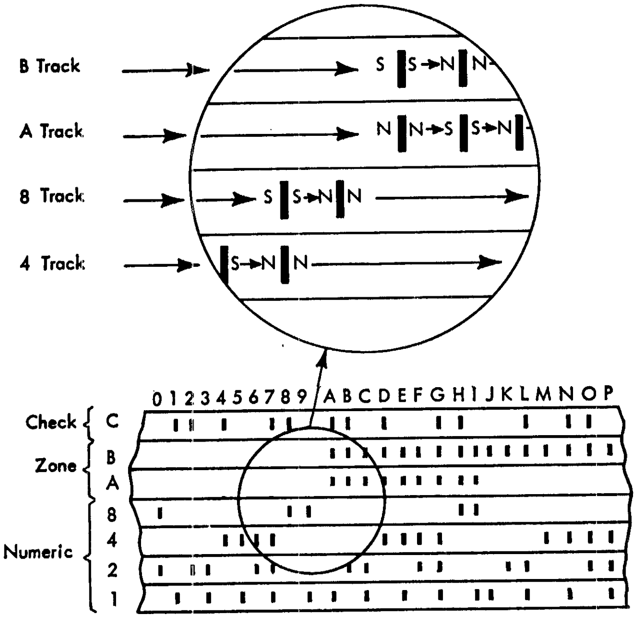
In the one-gap read/write head, reading and writing take place at the same gap. In the two-gap head, writing occurs at one gap, and reading occurs at the other. The two-gap head offers advantages discussed in the tape validity-checking sections.

Figure 5-9 Magnetic Tape Unit Read/Write Head

Writing on magnetic tape is destructive; that is, as new information is written, old information is destroyed. Reading is nondestructive; the same information can be read again and again.

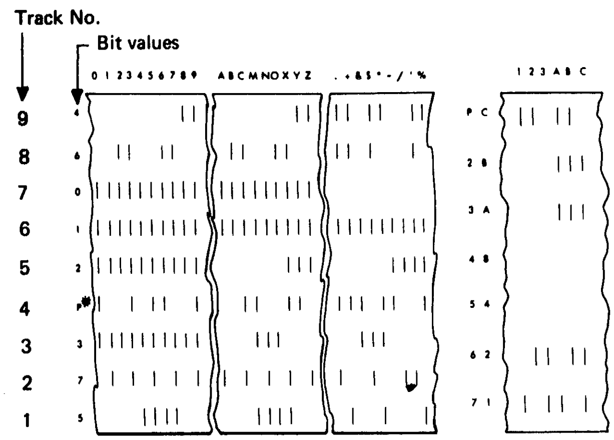
Information is written on tape by magnetizing areas in parallel tracks along the length of the tape.

There is one write coil in the write head for each recording track. Electrical current flowing through the coils magnetizes the iron oxide coating of the moving tape and erases previously written information (see Figure 5-10).



New data is written on magnetic tape by changing the direction of current flow and magnetic polarity from north (N) to south (S) in some of the write coils. This causes a change in the affected tracks. The coded pattern of 0 and 1 bits accross the width of the tape represents data received from the computer.

A. Magnetic recording of seven-track BCD code on tape



*The P bit position produces odd parity.

B. Nine-track (EBCDIC) and seven-track tape data format comparison

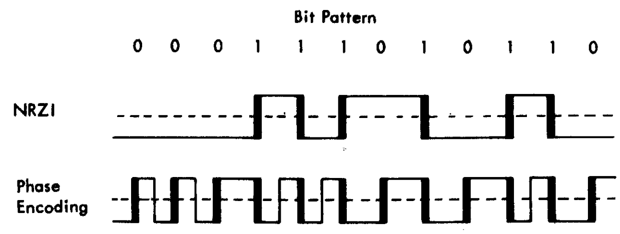
Figure 5-10 Data Recording on Magnetic Tape

In incremental tape units, the tape actually is motionless during writing, but the size of the recorded bits is almost the same as in the other tape units. Data generated on the incremental and variable-speed tape units is usable on other tape units adapted to seven-track data.

Checking Magnetic Tape Data

Data recorded on magnetic tape must be accurate so that errors are not sent through the system. Data is therefore checked to ensure that valid characters are recorded and to verify that the recorded bits are of effective magnetic strength.

Two methods of recording are used on IBM magnetic tape. The phase encoding method is used on newer IBM tape units; other magnetic tape units use the Non-Return-to-Zero-IBM (NRZI) method (see Figure 5-11).

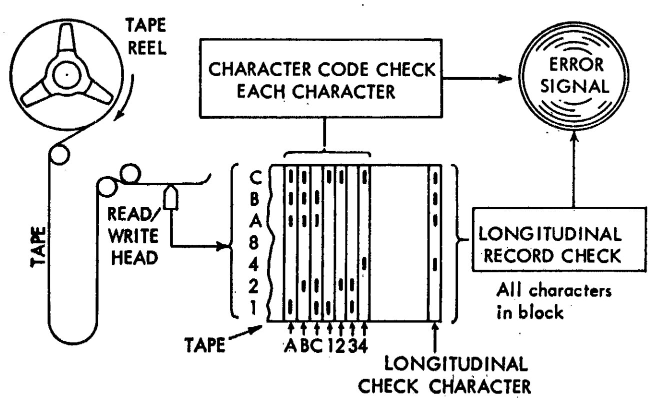


In the NRZI method of recording, a change in magnetic flux is interpreted as a 1 bit; lack of a flux change is interpreted as a 0 bit. The phase-encoded method of recording results in a continuous wave pattern, even when a record contains all zeros.

Figure 5-11 Comparison of NRZI and Phase-Encoded Bit Patterns

The NRZI method of data recoding is very reliable, but it has given way to the phase encoding method because of the increased densities of recording possible on the newer units.

The tape error detection system used on NRZI tapes uses the principle of simple parity checking. With this system, it is possible to detect virtually all tape reading and writing errors (Figure 5-12).



Information read from tape is checked two ways. A character code check (vertical check) is made on each column of information to ensure that an even number of bits exists for each character read. If an odd number of bits is detected for any character or column of bits, an error is indicated, unless the computer operates in odd parity. A longitudinal record check is made by developing an odd or even indication of the number of bits read in each of the seven bit tracks of the record, including the bits of the check character. If any bit track of the record block indicates an odd number of bits after it is read, an error is indicated, unless odd parity is required by system design.

Figure 5-12 Seven-Track Validity Checks, BCD Mode, Even Parity

Simple parity checking indicates the error, but not the kind of error. Similar double-bit errors in two characters of a record could conceivably cancel each other and indicate correct parity. However, this coincidence is extremely rare.

With certain models of the 3400 series, errors on a single track or combinations of two tracks simultaneously are corrected in flight without impairing tape performance. Errors may be corrected in all nine tracks of a single data block providing they occur in combinations of no more than two tracks at a time. As a result, most corrections are made without interrupting processing.

Tape units with two-gap heads provide increased checking while writing. Tape being written passes first over the write gap (to record data) and then over the read gap; the information that has been written is automatically read and checked.

When an error occurs during the writing operations, it is detected at the read gap, and an error indication is made. Programming must test the indicator and take an appropriate corrective action. The machine does not stop with the error section of tape positioned over the read gap; tape motion continues past the end of the record block. Then the machine may be instructed to backspace the tape and rewrite, again checking for an error.

Tape Records, Interblock Gap and Tapemark

Records on tape are not restricted to any fixed length of characters, fields, words, or blocks. Records may be any practical size within the limits of internal storage capacity.

Blocks of records, including blocks consisting of a single record, are separated on tape by an interblock gap, a section of blank tape varying in length from about .3 inch for 6250 bpi (bits per inch) tapes to about .75 inch for seven-track tapes. During writing, the gap is automatically produced at the end of each block of records. During reading, the block begins with the first character sensed after a gap and continues without interruption until the next gap is reached. The interblock gap also allows time for starting and stopping the tape between record blocks.

A *tapemark* (a special character represented in hexadecimal as 7F; see Figure 2-25) indicates the end of a file of records (see Figure 5-13). Most computers write and read tapemarks.

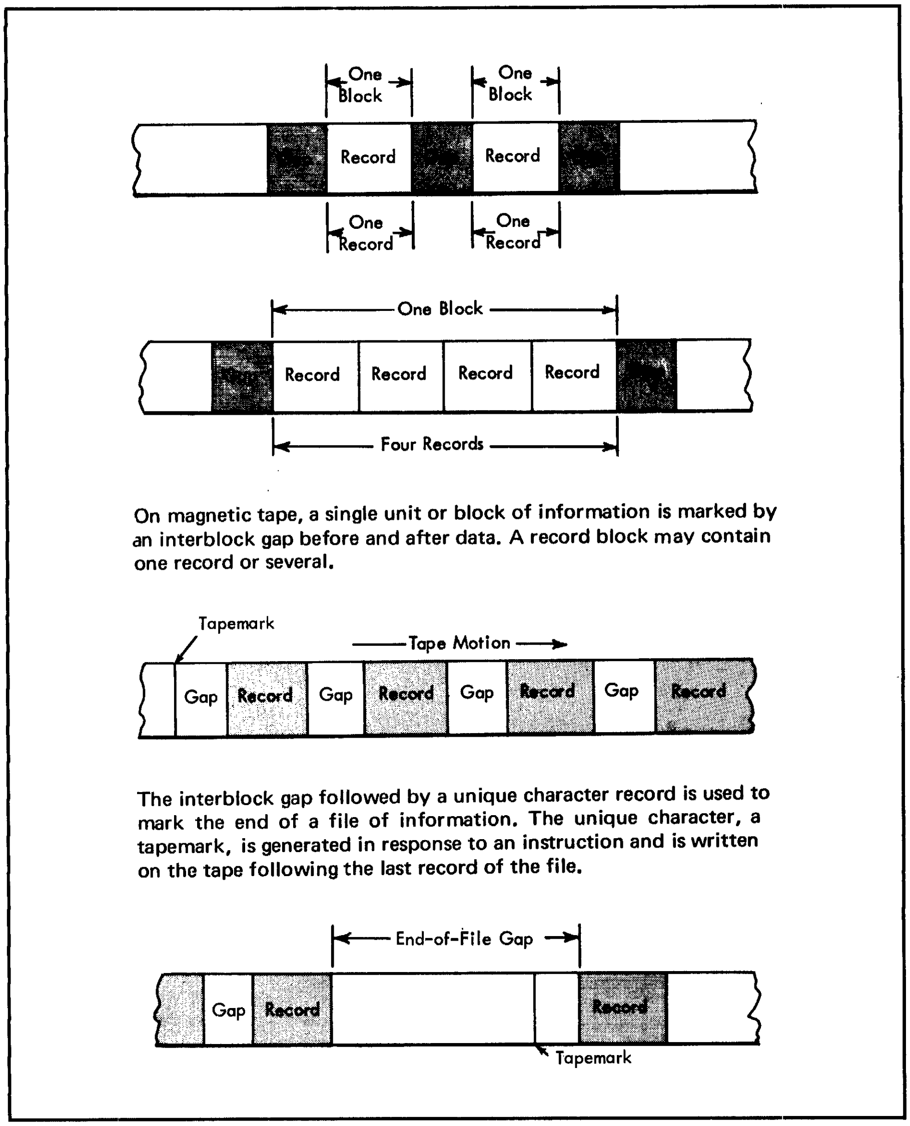


Figure 5-13 End-of-Block and End-of-File Indications on Tape

Tape Unit Characteristics

Among tape units, the major performance considerations are the speed at which tape is moved across the read/write head and the recording density of information on tape. These two factors determine important characteristics of character rate, tape access time (sometimes called gap time), and character time. Some other differences among tape units involve the length of the interblock gap, the extent and method of checking the validity of recorded data, and provisions for protecting recorded data. Figure 5-14 shows the major differences among tape units.

Magnetic Tape Unit		Nominal Data Rates (kilobytes/sec)					Tape Speed (inches/sec)	Nominal Read Access Time* (ms)	Nominal Interblock Gap (inch)			Nominal Interblock-Gap Time (ms)			Rewind Time (sec) (2,400' reel)	Rewind and Unload Time (2,400' reel)
Number	Model	at 200 bpi (7-track NRZI)	at 556 bpi (7-track NRZI)	at 800 bpi (7- or 9-track NRZI)	at 1,600 bpi (9-track PE)	at 6,250 bpi			7-track	9-track	at 6,250 bpi	7-track	9-track	at 6,250 bpi		
3410/3411	1	--	--	--	20	--	12.5	15	--	0.6	--	--	48	--	--	180
	2	--	--	20	40	--	25	12	--	0.6	--	--	24	--	--	180
	3	--	--	40	80	--	50	6	--	0.6	--	--	12	--	--	120
3420	3	--	41.7	60	120	--	75	4	0.75	0.6	--	10	8	--	60	66
	4	--	--	--	120	470	75	2.3	--	0.6	0.3	--	8	4	60	66
	5	--	69.5	100	200	--	125	2.9	0.75	0.6	--	6	4.8	--	60	66
	6	--	--	--	200	780	125	1.6	--	0.6	0.3	--	4.8	2.4	60	66
	7	--	111.2	160	320	--	200	2	0.75	0.6	--	3.75	3	--	45	51
	8	--	--	--	320	1,250	200	1.1	--	0.6	0.3	--	3	1.5	45	51

* The read access time is the interval of time from the beginning of a forward read, when the tape is not at load point, until the first data byte is read after the tape is brought up to speed from a stopped state. The times given for 3420 Models 4, 6, and 8 are for 6,250-bpi operation.

Note: bpi = bytes per inch.

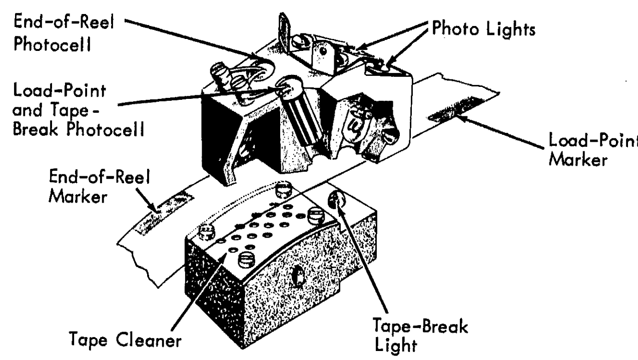
Figure 5-14 Characteristics of IBM 3400 Series Magnetic Tape Units

Maximum and Effective Character Rates

Because an interblock gap is placed between each record or block of records on tape, the total time required to read a record must include time to space over the gap; this is called access time to the data. Access time is given (Figure 5-14) for each tape unit on the basis of tape speed and length of interblock gap. Access time must be considered when determining the actual or effective character rate of a tape unit.

Load-Point and End-of-Reel Markers

Magnetic tape must have some blank space at the beginning and end of the reel to allow threading through the feed mechanism of the tape unit. Markers, called reflective strips, are placed on the tape to enable the magnetic tape unit to sense the beginning and the end of the usable portion of tape. Photoelectric cells in the tape units sense the markers as either the load-point marker (where reading or writing is to begin) or the end-of-reel marker (where writing is to stop). The tape unit does not recognize the end-of-reel marker when reading tape; a tapemark, written on the tape, signals an end-of-reel condition (see Figure 5-15).

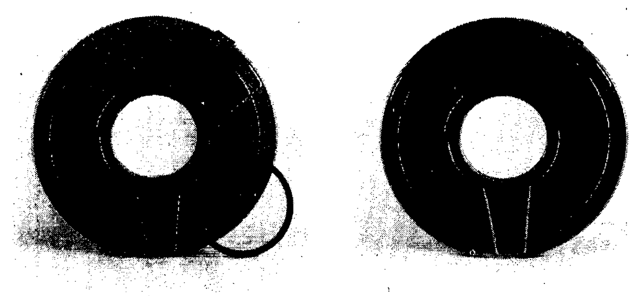


Tape markers are small pieces of transparent plastic with a thin vapor-deposited film of aluminum on one side. Pressure-sensitive adhesive covers the aluminum film. The markers are fastened manually to the base (uncoated) side of the tape. New reels of IBM tape have the markers in position. Photocells in the tape unit sense the markers as they pass; broken tape is also detected.

Figure 5-15 Photosensing Markers

File Protection

Because writing automatically destroys any previous information on the tape, there is a way to prevent accidental erasure of information. A plastic ring (see Figure 5-16) that fits into a round groove on the tape reel must be removed when tapes are to be saved for further reference. Without this ring the tape can only be read and cannot be written on.



On noncartridge tape, the file protection device is a plastic ring that fits into a round groove molded in the tape reel. When the ring is in place, either reading or writing can occur. When the ring is removed, writing is suppressed and only reading can take place; thus, the file is protected from accidental erasure.

Figure 5-16 File Protection Devices

Direct Access Storage Devices

These have been discussed in the section entitled "Storage Devices". However, DASD's may be thought of as input/output devices.

Punched Tape Devices

Paper Tape Reader

The paper tape reader shown in Figure 5-17 reads data represented as punched holes in five, six, seven, or eight-channel paper tape at a rate of up to 1000 characters per second. As it moves or feeds the tape past a reading unit, the presence or absence of holes in the tape is sensed and converted to electronic impulses that are used as data by the computer system. Accuracy of reading is determined by making a parity check (where characters are written with parity, as in eight-channel code). The speed of reading, from 150 to 1000 characters per second, depends on the type of reader and the lengths of the records.

Figure 5-17 IBM 2671 Paper Tape Reader and 2822 Paper Tape Reader Control

For faster paper tape input to the system, the data to be converted may be written on magnetic tape is an off-line operation at 150 paper tape characters per second (Figure 5-18). The recorded tape may then be placed on a magnetic tape unit and read into the computer system at the much higher magnetic tape input rates (see table in Figure 5-14).

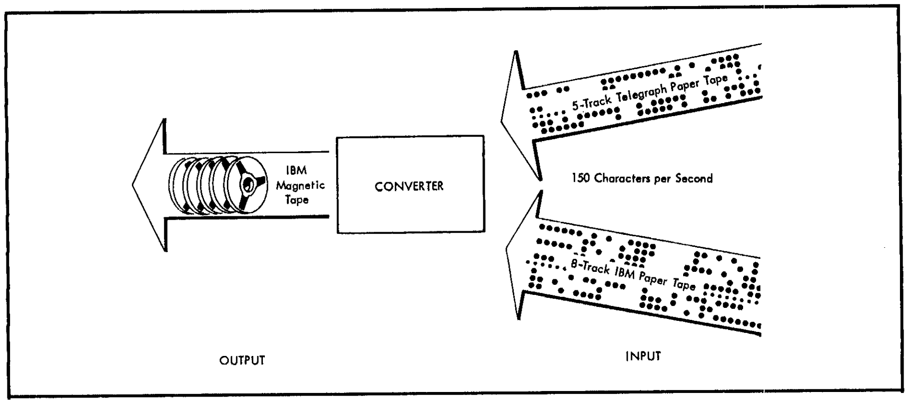


Figure 5-18 Data Conversion - Paper Tape to Magnetic Tape

Paper Tape Punch

Data from the computer system is recorded as punched holes in paper tape by an automatic tape punch. Data received from main storage is converted to a tape code and punched in blank tape as the tape is moved through a punching mechanism. Accuracy of data recorded is verified by a parity check for each character (eight-channel code, for instance). Tape is punched at a density of ten characters to the inch and at rates of 15 and 150 characters per second.

Printers

IBM printing devices provide a permanent visual record of data from the computer system. Speeds of printing vary from 15.5 to 68,136 characters per second.

As an output unit, the printer receives data, symbolized in electronic form, from the computer system. The electronic symbols enter appropriate circuity and cause printing elements to be actuated. All printing devices have a paper transport that automatically moves the paper as printing progresses.

The major printing devices consist of the electrophotographic printer, chain/train/belt printer, serial wire matrix printer, and the typewriter.

3800 Printing Subsystem

The fastest and most versatile IBM printer is the IBM 3800 Printing Subsystem (Figure 5-19). The IBM 3800 is a high-speed, nonimpact printer that produces characters on paper through electrophotographic and laser technology.

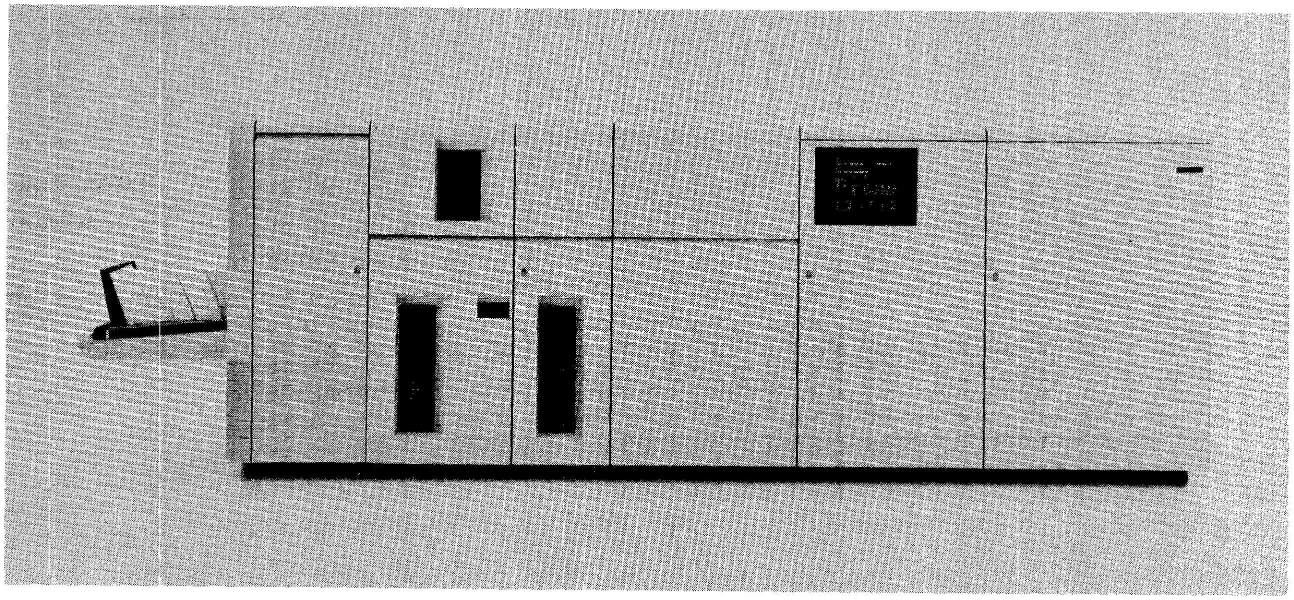


Figure 5-19 IBM 3800 Printing Subsystems

Features of the 3800 include:

1. Continuous forms input, transport, and stacking mechanism.
2. 52K byte storage for page buffering and control of printer operations.
3. Eighteen different character sets, including four special underscored sets, and 10-, 12-, and 15-pitch (characters per inch) sets, all of which may be printed separately or any combination of up to four may be mixed on a line. (Print line maximums are 136 positions at 10 characters per inch (cpi), 163 positions at 12 cpi, and 204 positions at 15 cpi).
4. Writable character generation storage organized into two 64-character writable generation modules to hold 128 characters. (An additional increment of 127 Writable Character Generation Storage Positions is optional. This represents up to 255 graphics online with no throughput loss).
5. The electrophotographic process, which includes the following:
   a. A continually revolving drum on which a charged photoconductive surface is selectively discharged by a low-power laser to produce images of the printed data.
   b. A developer station where black toner is attracted to the images.
   c. A transfer station where the toner is transferred to the paper forms.
   d. A fuser station to fuse the toner into the paper.
   e. A cleaning station to remove any residual toner from the drum after the page has been printed.
   f. A charge station to prepare the photoconductor.

6. A forms overlay station to expose the drum with form images or other fixed data.

In operation, data to be printed is moved from the CPU to the 3800 a line at a time, is translated into graphic code using a set of translate tables, and is stored in the page buffer. When the page buffer contains a full page, the code is used, through interaction with character generation storage, to modulate the laser in exposing the revolving drum. Exposure is by horizontal line scanning, similar to the way a cathode-ray gun scans a TV screen to produce a picture. The image is developed with toner, transferred to paper, and fused. The photoconductor surface of the drum is cleaned and reconditioned after each exposure. Finished copies are refolded and stacked in the continuous forms stacker, complete with job separation marking (optional). Figure 5-20 is a schematic diagram of the 3800 printing mechanism.

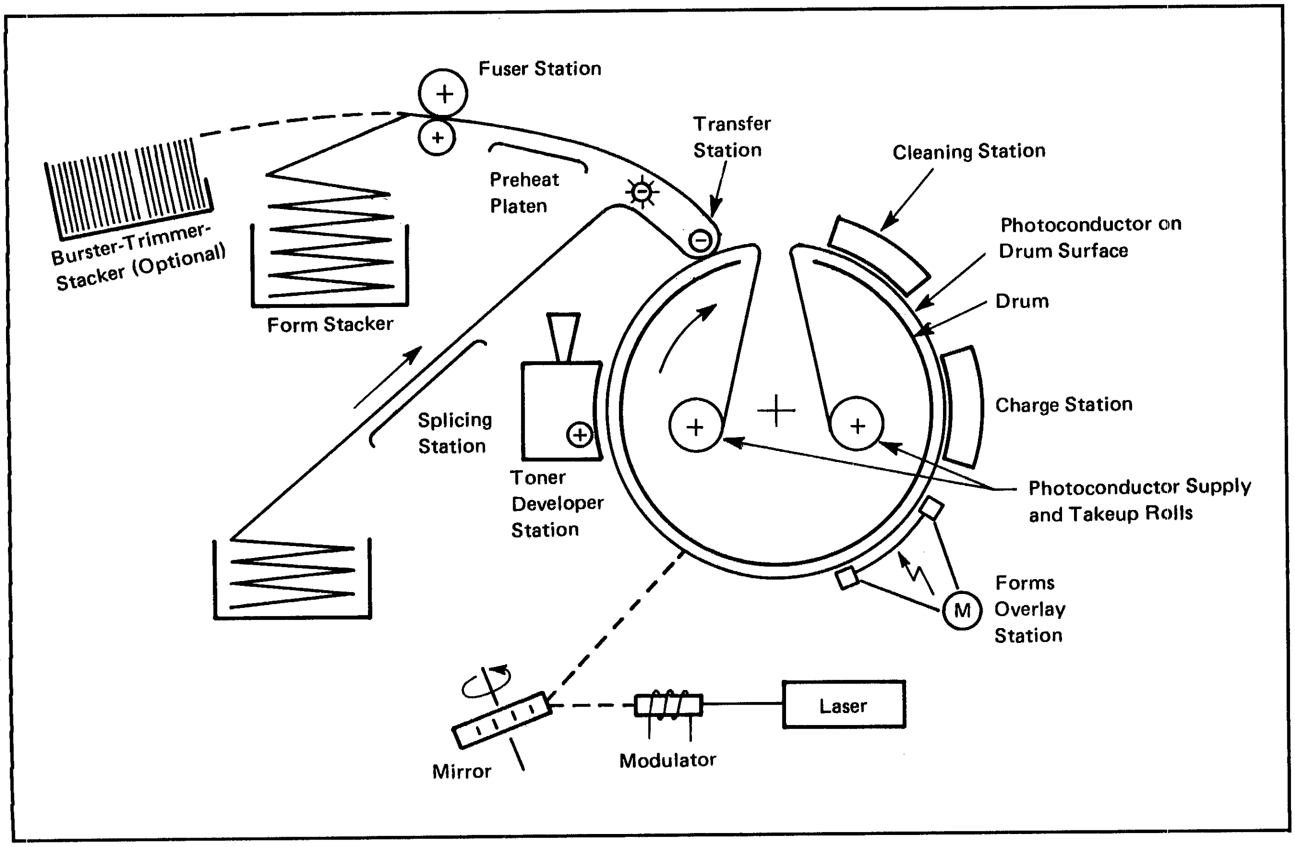


Figure 5-20 Path of Paper Forms Through the IBM 3800 Printing Subsystem

Graphic character modification allows user- or IBM-designed characters to take the place of an equal number of standard characters in character-generation storage. Line spacing is 6, 8, or 12 lines per inch and can be intermixed within a page.

The 3800 uses single-ply, edge-punched, perforated, and stacked continuous forms in any combination of five lengths and ten widths (common-use sizes). Preprinted forms may be used, or the form image can be printed simultaneously with text by the use of a forms overlay negative, by character formatting, or by any combination of these to suit the application.

The forms overlay negative, bearing the image of the form, is installed by the operator prior to printing. The image from the overlay negative may then be printed on any number of copies, starting with the first. Forms overlay may also be used for printing pictorial line art, or halftones, on copies.

The character formatting method uses a character set working under program control to create a line image.

Copy modification allows printing of predefined data on specified copies of the page of a data set. The data may be legends, column headings, or other information; or it may be blanks to eliminate the printing of data. The modification may vary from copy to copy.

The overall effect of forms overlay, character formatting, and copy modification is functionally equivalent to the use of conventional numbered, preaddressed, multiple-part forms with standard features such as legends, spot carbons, short plies, and printed blockout areas.

The maximum printing speeds of the IBM 3800 depend upon the number of lines per inch printed and the length of the forms used. With 3-1/2 inch forms and six lines per inch, up to 7,890 lines per minute can be printed. This increases to 20,040 lines per minute for 11 inch forms and 12 lines per inch.

A forms burster-trimmer-stacker that bursts printed output into individual sheets with the right and left carrier strips trimmed off is available as an optional feature. Jobs and/or single copies are separated by offsetting one job or copy from another in the stacker, and are ready for distribution without postprinting delays.

Impact Printer

Prior to the availability of the IBM 3800, the fastest IBM printers were various models of line printers utilizing the train/chain technique.

The train printer is an electromechanical line printer using engraved type. Alphabetic, numeric, and special characters are assembled in a train (Figure 5-21). As the train travels horizontally, each character is printed as it is positioned opposite a magnetically actuated hammer that presses the paper against one piece of type in the moving train. Up to 132 positions may be printed on one line, at speeds of up to 2000 lines per minute. The print train can be easily changed to provide a choice of print fonts.

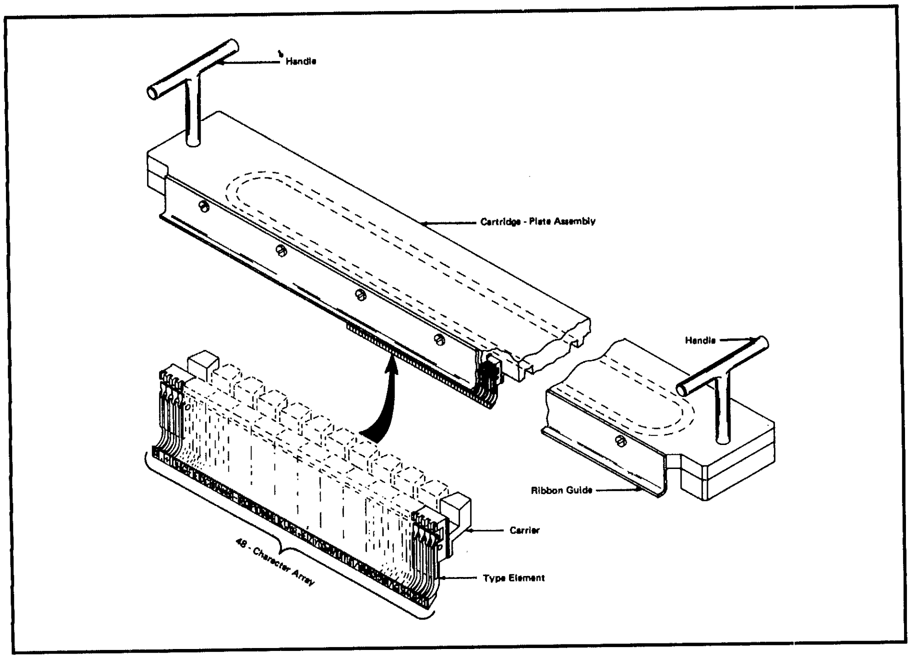


Figure 5-21 Print Train

On a belt printer, the character set is contained on a continuously rotating metal belt. As the belt rotates, a comparison is made between the character in front of each hammer and the character that is to print in that position. When the desired character is in position to print, the hammer for that position is fired to force the paper and ribbon against the character face to print the character.

Interchangeable train cartridges or print belts are available for System/370 line printers. Usually, the universal character set (UCS), which enables a user, with a customized print train, to obtain maximum printer efficiency, is standard. On certain printers, if needed, a user can have an expanded character set (up to 254 different characters on any one train) that is capable of using any of the 256 EBCDIC codes, except *null* or blank (hex 00 and 400).

A user can customize a print train by having only those characters, symbols, etc., necessary for a particular job. (IBM currently has a variety of print trains available that meet most needs). The user then assigns a code to represent each character on the train. The codes are entered into a buffer (located in the control unit) in the positions that the characters appear on the print train. When the user is printing, the printer prints the character that corresponds to the code in the buffer position.

Similarly, for belt printers, interchangeable belts are available. For example, the IBM 3776 console printers utilize 48-, 64-, and 94-character sets. (With the 48- or 64-character set belt, the 3776 will convert the 26 lowercase alphabetic characters to uppercase characters and print them).

Typically, the printing speed of these printes will vary depending upon the sizes of the character sets. For example, the IBM 3776 Model 2 can print at a maximum speed of 400 lines per minute using the 48-character set, 300 lines per minute using the 64-character set, and 230 lines per minute using the 94-character set.

In the serial wire matrix printer printed characters are formed in a dot pattern by pushing a vertical set of wires forward to contact an inked ribbon as the print mechanism moves across the print line. The number of dots used to form the characters depends upon the number of wires in the vertical arrangement. For example, the eight print wires available on the IBM 3774 console printer form characters in a 7 x 8 dot pattern. Figure 5-22 shows some characters formed by a 7 x 9 dot pattern.

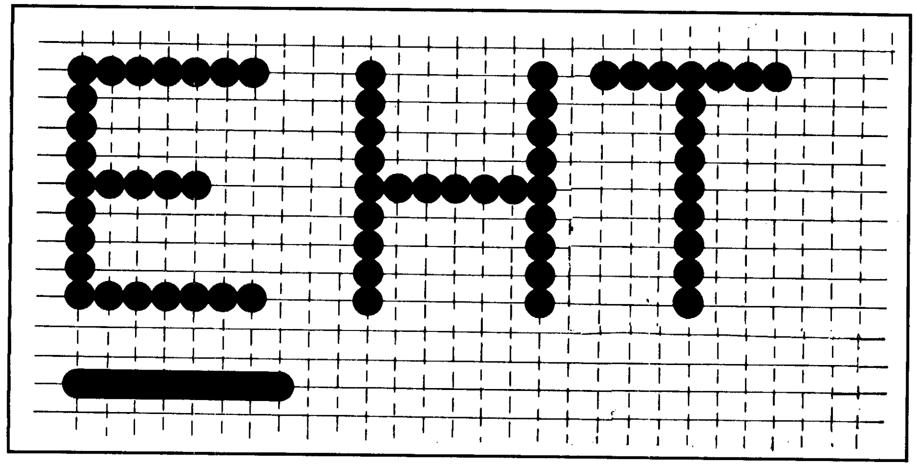


Figure 5-22 Examples of Monocase Characters Formed with 7 x 9 Dot Pattern

Some serial wire matrix printers such as the IBM 3774 offer bidirectional printing. This eliminates the mechanical delay encountered in moving the print mechanism back to the beginning of the print line after each line is printed, as is commonly done on most serial printers.

The typewriter that is used as an output device (Figure 5-23) is similar to the one used manually. The major difference is that control of the typewriter and the printing occurs automatically as directed by the stored program. Printing speed is about 600 characters per minute; spacing and carriage return are automatic. With the 3210 printer keyboard (using the "golf ball" printer), printing speed is 900 characters per minutes.

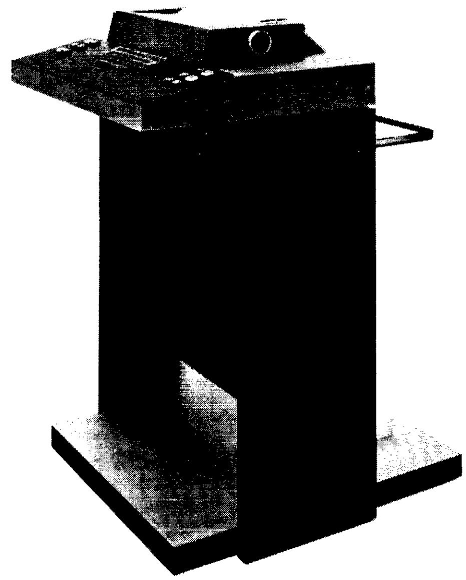


Figure 5-23 IBM 3210 Console Printer Keyboard

Character Recognition Input Units

With the advent of high speed automatic data processing systems, there came a realization that wide use could be made of input units capable of reading data that could also be read by people. The processing of over 30 billion checks yearly and the inestimable volume of other notices - insurance billings, magazine subscription renewals, invoices, manufacturing routing slips, utility bills, and so on - could be greatly speeded by the use of man-machine recognizable characters. Two systems that accomplish this are magnetic character readers and optical character readers.

Magnetic Character Readers

As a specialized means of input to computer systems, IBM magnetic character readers provide banks with a time-saving method of reading and processing large volumes of daily transactions. These machines read card and paper documents inscribed with the E13B type font approved by the American Bankers Association. A second important labor-saving feature of the magnetic character readers is their ability to sort the magnetically inscribed documents in off-line operations (See Figure 5-24).

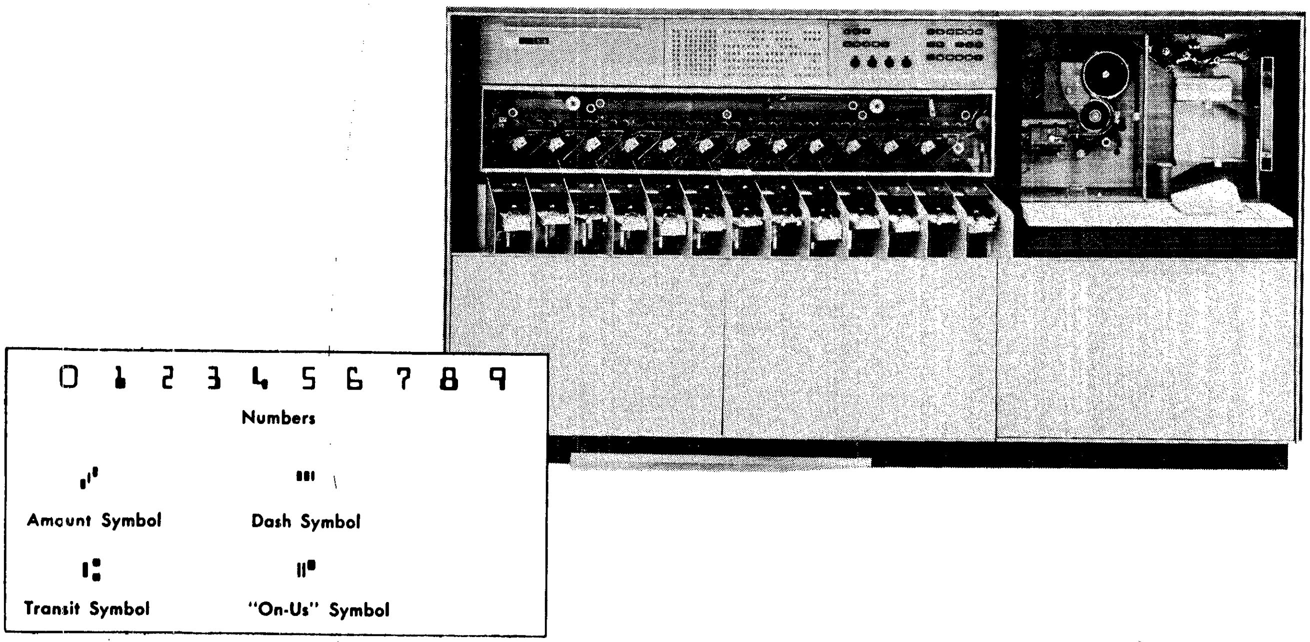
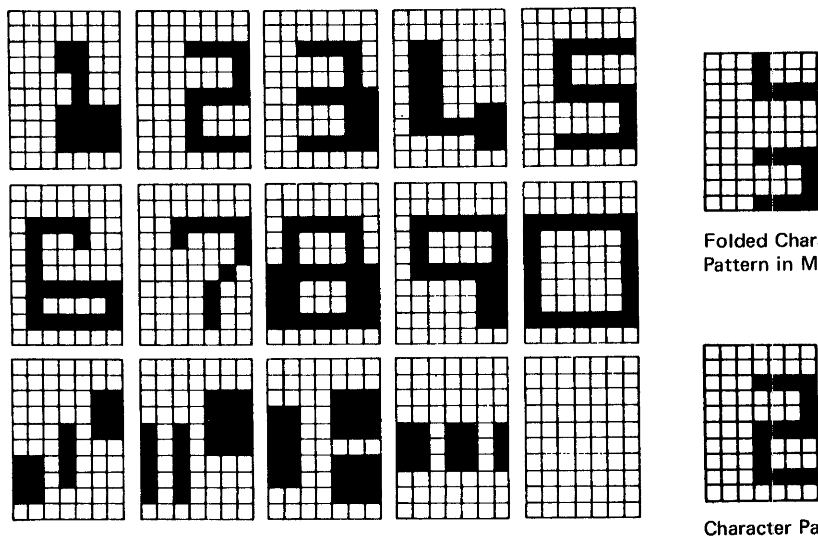


Figure 5-24 Magnetic Character Reader and Magnetic Ink Characters

The reader examines the shape of each magnetic ink character passing under the read head, and ten data channels send signals to an electronic storage device called the character matrix. The matrix has a storage location for each of 70 character segments, and, as documents pass under the read head, lack of any appreciable signal from a character area segment causes the machine to store a 0 bit in that storage location. The presence of a significant signal (indicating that magnetic ink is under the reading gap) causes the machine to store a 1 bit in the specified storage location. The bit structure entering the matrix is also displayed in the character matrix lights of the indicator panel.

After the entire character area has passed under the read head and all segments have been read, a pattern of the character shape is in the characer matrix as a configuration of 0 bits and 1 bits (see Figure 5-25). To verify the accuracy of processed data, the reader automatically checks each character as it is being read.



Each magnetic character passing the read head in the reader is sensed and examined. The reader looks for key recognition shapes and characteristics. If the character is slightly out of position as it passes under the read head, the signals sent to the matrix form what is called a folded character. The reader automatically unfolds the pattern by shifting it vertically to check for recognition.

Figure 5-25 Matrix Patterns of E13B Characters

An exhaustive study of the 14 characters shown in Figure 5-25 has determined that thousands of 0 bit and 1 bit configurations can be considered acceptable patterns for each character, even when portions of the character are missing. All other patterns are considered invalid.

When the machine determines that a character is valid, the reader stores the character in another storage location called the character register. The character remains there until it is no longer needed for processing. If the machine determines that the pattern is invalid, the recognition circuits provide the machine with an error signal.

Optical Character Readers

The optical character reader reads uppercase letters, numbers, and certain special characters from printed paper documents and introduces the data into a computer system. Transcribing of source data to cards or tape is eliminated, and the time between receipt of source documents and their entry into the data processing system is greatly reduced.

The principal operating action of the optical character reader is provided by a rotating drum that transports documents from a hopper past an optical scanning station. The scanner consists of a powerful light source and a lens system that distinguishes between black and white patterns of reflected light. These light patterns are read as a number of small dots and are converted into electrical impulses to develop a character pattern. When the pattern of the optically read character matches a character pattern in the reader's character recognition circuits, the character is recorded and transferred into the computer system for processing. The read and recognition operation is automatic and takes place at split-second speeds.

Optical readers can perform an additional operation known as mark-reading, the reading of ordinary pen or pencil markings. The mark, when placed in a specified location on the source document, represents specific information. This feature has many important applications, such as recording of partial payments directly on a customer's bill and immediate processing of payment, recording of meter reading cards at the customer's utility meter, and so on.

The IBM 3886 Optical Character Reader (Figure 5-26), uses several new technologies that make it a compact and highly reliable modular device. A powerful microprogrammed recognition and control processor performs machine-control and character-recognition functions, enabling the 3886 to perform sophisticated data and blank editing as well as output record formatting.

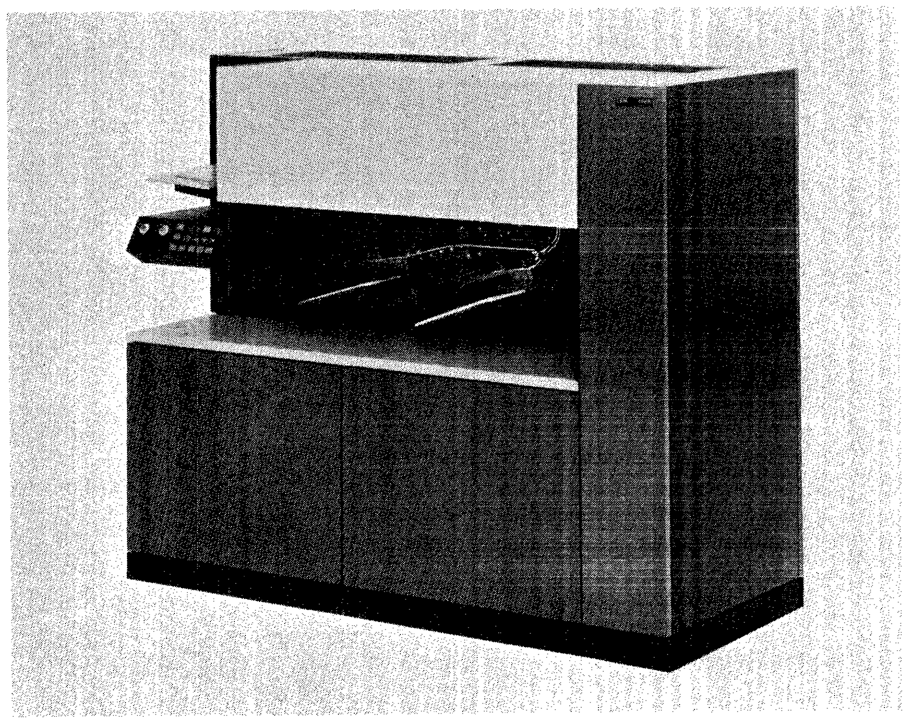


Figure 5-26 IBM 3886 Optical Character Reader

Consoles

The console of a data processing system (Figure 5-27) is used by the operator to control the system and monitor its operation. Depending upon the data processing system installed, different controls and displays such as a full keyboard, special purpose keys, switches, display lights, one or two CRT displays, audible tone signals, and so on, may be available to the operator to:

1. Start and stop the computation.
2. Manually enter and extract (or display) information from internal storage.
3. Determine the status of internal electronic switches.
4. Determine the contents of certain internal registers.

5. Alter the mode of operation so that, when an unusual condition occurs, the computer will either stop or indicate the condition and proceed.
6. Change the selection of input/output devices.
7. Reset certain types of computers when error conditions cause them to halt.
8. Otherwise interact with the system, for example, by presenting an attention signal.

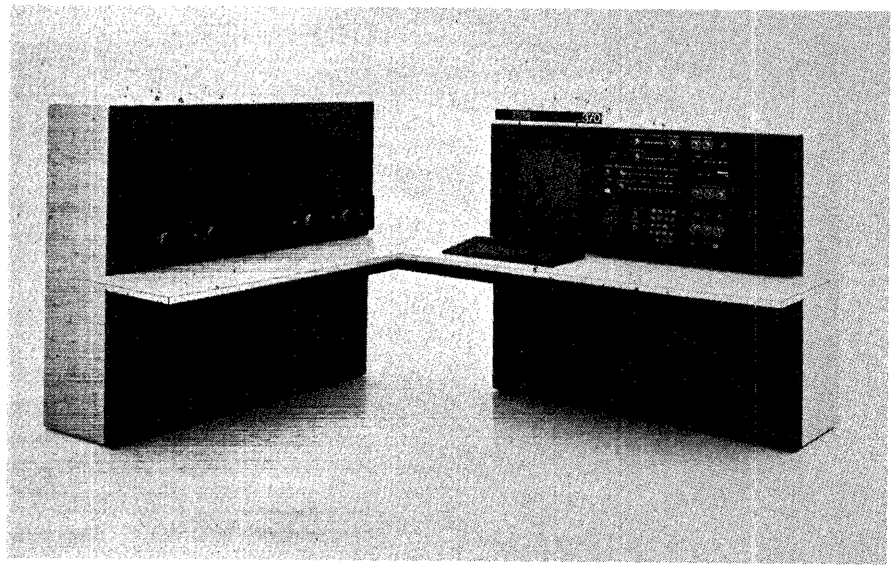


Figure 5-27 Console - IBM System/370 Model 168

In some data processing systems, the main console is connected only to the central pocessing unit and may be augmented by separate consoles that are used for engineering functions and for additional input/output control.

Graphic Display Units

More and more frequently, in data processing systems, terminals are used for data entry as well as for a visual display of records in main or secondary storage, as called for by an operator. Display stations of the IBM 3270 Information Display System (Figure 5-28) are commonly used. The IBM 3270 is a family of display products that can be tailored to meet the needs of all alphameric display applications. It offers improved response times and transaction rates based on increased transmission and operator efficiency. In addition, the 3270 display system is easy to operate and well suited to an office environment.

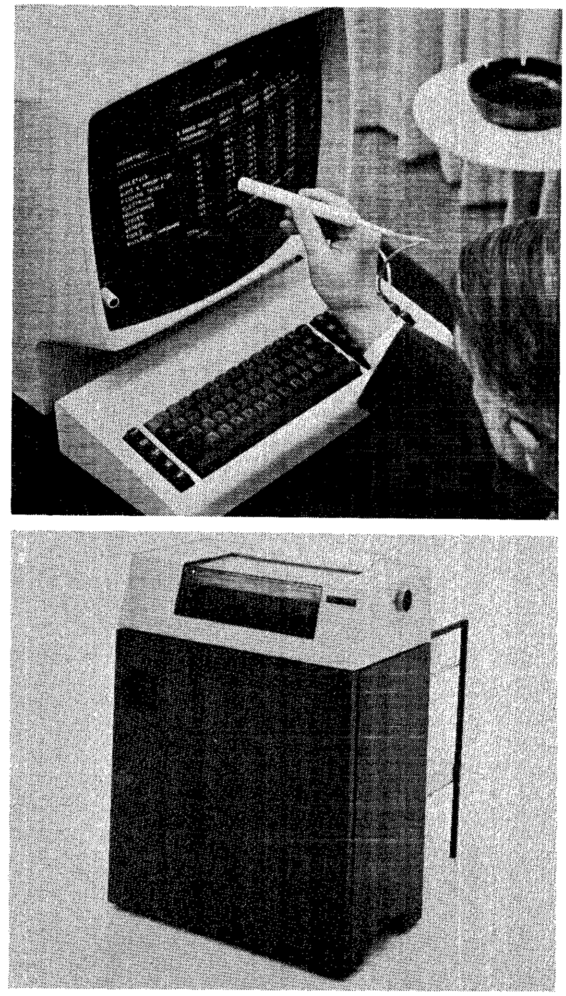


Figure 5-28 IBM 3270 Information Display System

The 3270 display system can be a standalone unit, a small cluster, or a large cluster (of up to 32 units), can include 480-character display stations or large capacity 1,920 character display stations or both, can also include printers (40 or 66 characters per second), and can be attached locally (directly to the channel of a System/370) or remotely.

The 3270 system offers typewriter, data entry, and operator console keyboards and a selector pen. It provides local data transfer rates of up to 650,000 characters per second and remote line speeds of up to 7200 bits per second. The 3270 system includes data security enhancement features, such as keylock and operator identification card reader, and the ability to enter data without displaying it.

Another display unit is the IBM 2250, described under "Visual Output" in the section entitled "Data Representation".

Terminals

IBM currently offers three classes of terminals used in teleprocessing operations. Because of the way they receive and transmit messages, they are called start/stop terminals, binary synchronous communication (BSC) terminals, and synchronous data link control (SDLC) terminals. Numerous terminals are available under each of these classes and include such devices as typewriter-like keyboards, printers, card readers, card punches, displays, magnetic tape terminals, etc.

The most conspicuous part of a teleprocessing operation is the use of data transmission facilities. These facilities are obtainable from communication common carriers or may be a privately owned or leased transmission medium.

The start/stop terminals include the IBM 1050, 1030, 1060, 2740, 2741, 2260, and 2265. The operation of these terminals is based upon asynchronous transmission (in which each information character is individually synchronized by a start character). In this type of transmission, each group of code elements corresponding to a character signal is preceded by a start signal, which serves to prepare the receiving mechanism for the reception and registration of a character. In addition, each character is followed by a stop signal, which serves to bring the receiving mechanism to rest in preparation for the reception of the next character. Transmission rates for start/stop terminals are within the 8 to 66 characters per second range.

The second classification of terminals is the binary synchronous communications (BSC) terminals. Greater flexibility of operation and substantially increased transmission speeds are provided by these classes of terminals. Included in the BSC terminals are the 2770, 2710, 2790, 1130, 1800, etc. The transmission speeds available in BSC operations are from 150 characters per second to 28,800 characters (or 57,600 packed decimal digits) per second.

Synchronous data link controls (SDLC) is a more sophisticated line discipline than BSC. SDLC offers a fixed format for transmissions which eliminates many of the control characters required for BSC transmission. SDLC terminals and communications systems include the 3270, 3600, 3650, 3767, 3770, and the 3790.

Data Buffering

All data processing procedures involve input, processing, and output. Each phase takes a specific amount of time. The usefulness of a computer is often directly related to the speed at which it can complete a given procedure. Any operation that does not use the central processing unit to full capacity prevents the entire system from operating at maximum efficiency. Ideally, the configuration and speed of the various input/output devices should be so arranged that the CPU is always kept busy with useful work.

The efficiency of any system can be increased to the degree in which input, output, and internal data-handling operations can be overlapped or allowed to occur simultaneously.

Input is divided into specific units or logical associations of data that enter storage under control of the program. A number of output results may be developed from a single input, or conversely, several inputs can be combined to form one output result. Figure 5-29a shows the basic time relationship between input, processing, and output with no overlap of operations. In this type of data flow, processing is suspended during reading or writing operations. Inefficiency is obvious, because much of the available time of the central processing unit is wasted.

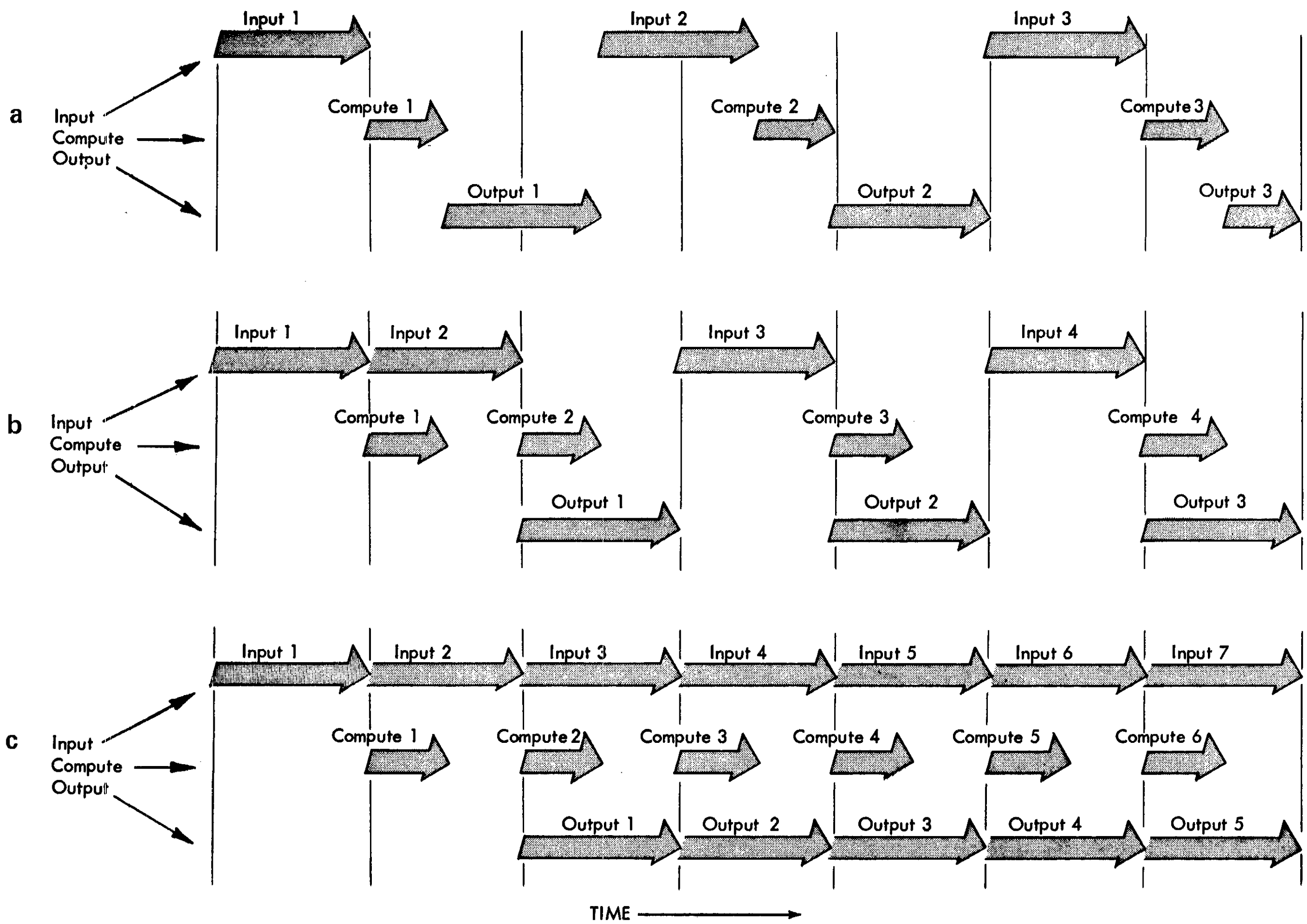


Figure 5-29 Data Buffering

Figure 5-29b shows a possible time relation between input/output and computing when a buffered system is used. Data is first collected in an external unit called a *buffer*. When summoned by the program, the contents of the buffer are transferred to the main storage unit. The transfer takes only a fraction of the time that would be required to read the data directly from an input device. Also, while data is being assembled in the buffer, internal manipulation or computing can occur in the computer. Likewise processed data from main storage can be placed in the buffer at high speed. The output device is then directed to write out the contents of the buffer. While writing occurs, the central processing unit is free to continue with other work.

If several buffered devices are connected to the system, reading, writing, and computing can occur simultaneously (Figures 5-29c).

Further development of the buffering concept has led to the use of main storage as the primary buffer. Data is collected from, or sent to, the input/output devices in words or in fixed groups of characters. Transmission of words is interspersed automatically with computation, but the time required for the transmission of single words is relatively insignificant. The effect is that of overlapping internal processing with both reading and writing. The principal advantage here is that the size or length of the data handled is restricted only by the practical limits of main storage. When external buffers are used, the amount of data handled at any time is limited to the capacity of the buffer. Overlapping operations up to this point have demonstrated a principle of synchronous operation; that is, the action of the input/output devices is made to occur at fixed points in the program and in a sequence established by the programmer.

In some computers, design features allow for automatic interruption of processing by the input/output devices; synchronous operation is not required. The input or output device signals the central processing unit when it is ready to read or to write. The central processing unit responds to these signals and either accepts the data as input or transmits the required information as output. In real-time teleprocessing systems, this type of input/output is likely to be nonsequential and unpredictable.

The problem arises of how to fill in the gaps in central processing unit time. The answer is to somehow queue the various tasks and programs to step in and, without interfering with one another, to use the otherwise idle time. This is the basis for multiprogramming, a subject described in more detail under "Programming Systems".

Section 6:

Teleprocessing

Teleprocessing is the processing of data that is received from or sent to remote locations by way of communications facilities.

A teleprocessing *network* consists of a number of communications lines (communications facilities) connecting a central data processing system with remote teleprocessing devices (Figure 6-1). Such devices can be terminals, control units, or other data processing systems. In this overview, any machine or group of machines capable of generating and/or receiving signals transmitted over communications lines will be referred to as a terminal or terminals. Thus, terminals may be data processing systems, communications systems such as the IBM 3270 Information Display System, or a single unit such as the IBM 2740 Communication Terminal.

As an example of how a teleprocessing network functions, a clerk in an insurance company's branch office receives a telephone call asking for information about an insured's account. Asking the caller to hold the line, the clerk enters the information request into a terminal, and the request is sent over a communications line to the System/370 at the insurance company's main office. When the request reaches the computer, several things happen. The computer interrupts processing whatever job it is working on and saves all necessary data and instructions so that it can resume processing the job at the exact point of interruption. As the information is received over the communications lines, the communications module in the control program converts the data into machine language, stores it in a buffer area, and checks to see that it was transmitted correctly.

The nature of the request may dictate that a number of operations be performed. To process the request, the teleprocessing program directs the System/370 to examine the appropriate policy file and bring the insured's record from storage. The program then searches the record for the information requested and sends it over the communications lines to the clerk who originated the request. The clerk reads the information as it is displayed or typed out at his terminal and relays the information to the policyholder or adjuster waiting on the telephone.

At the main office, the control program has returned the System/370 to its status prior to the interruption, and the computer has resumed processing. As a result of the telephone inquiry, the clerk in the branch office may update the insured's record and transmit this information to the System/370 at the main office at a later time.

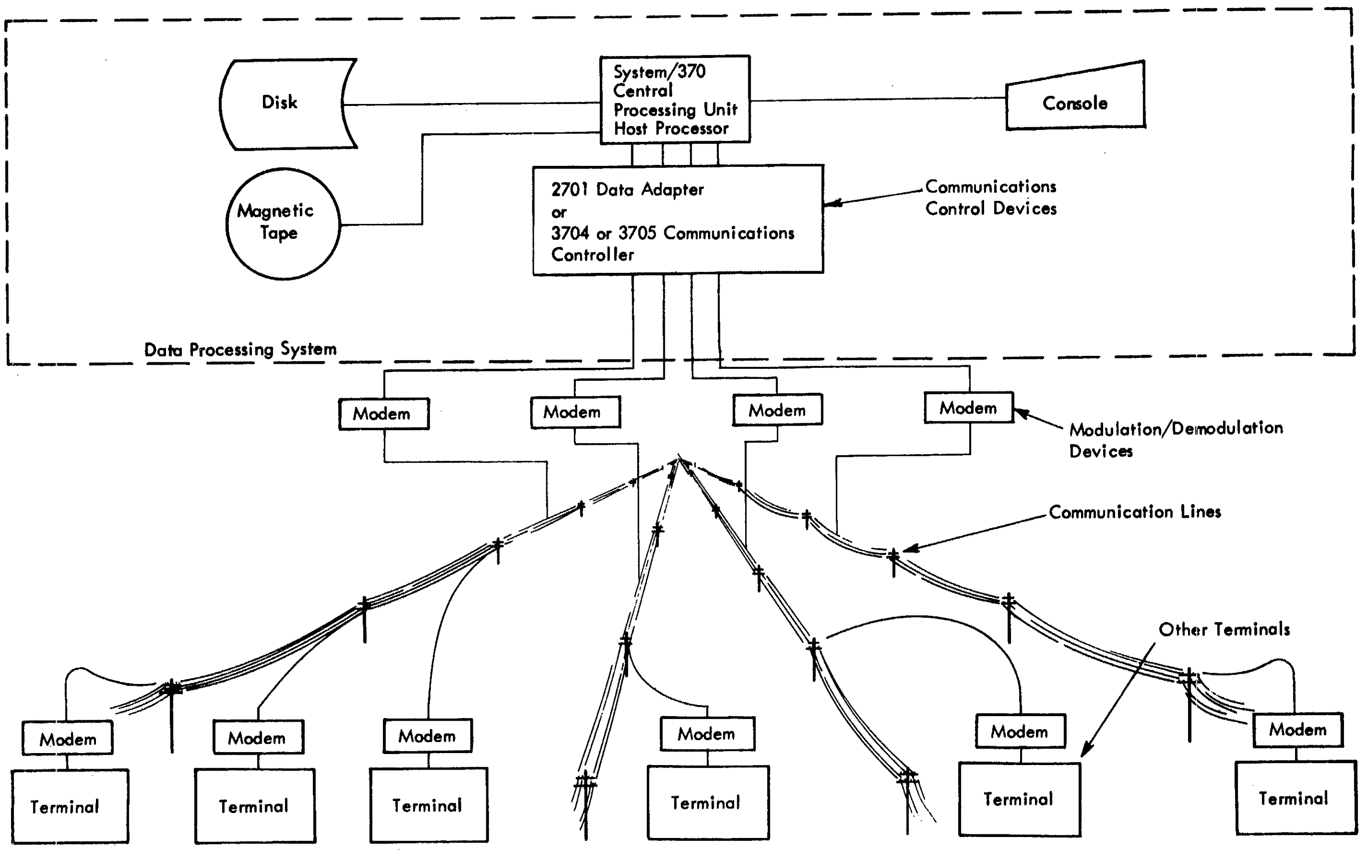


Figure 6-1 A Teleprocessing Network

Elements of a Teleprocessing Network

The elements of a teleprocessing network (Figure 6-1) consist of a host processor (central data processing system), communications control devices, modulation/demodulation devices (modems), communications lines, other terminals, and programming systems. Three of these elements, the communications control devices, modems, and communications lines, constitute a *data link* (Figure 6-2).

Host Processor

System/370 is designed so that it can serve as the host processor in a teleprocessing network. Requirements for the host processor include multiprogramming capability, adequate storage capacity, storage protection, adequate speed for the applications required, and, for planning purposes, the potential for expanding storage capacity and speed.

The host processor for a teleprocessing network must be able to handle random and unscheduled input, as well as serialized and scheduled input.

Communications Control Devices

Communications control devices are hardware components that link the communications lines to the host processor. These devices can be external to the processor, such as the IBM 3704 or 3705 Communications Controller, or they can be a part of the processor of a System/370, such as the integrated communications adapter feature. When control devices are external units, they can be classified as data transmission multiplexers.

The transfer of data requires noninformation transmissions for setting up, controlling, checking, and terminating information exchange. These noninformation exchanges constitute *data link control*. Communications control devices handle data link control; thus, functions of these devices include:

- Synchronization (getting the receiver in step with the transmitter)
- Identifying the sender and receiver
- Delimiting the beginning and ending of information (code translation)
- Error detection and recovery

In order for a host processor to send data over communications lines, the data must be converted (serialized) to a serial stream of binary digits. Likewise, when the host processor receives data from a remote terminal, this data must be reconverted (deserialized) into machine language for processing (Figure 6-2). Control devices perform this function.

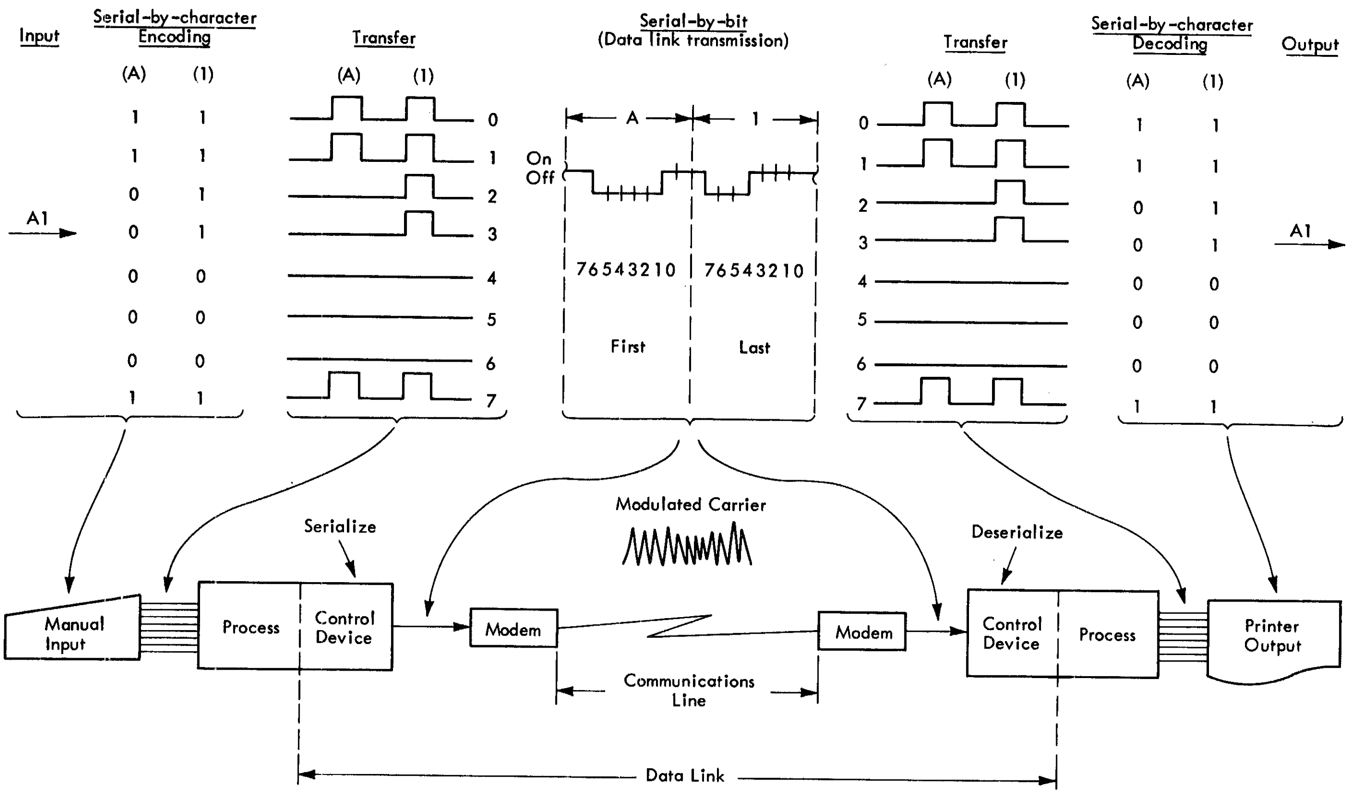


Figure 6-2 Data Conversion for Data Transmission

Modulation/Demodulation Devices

After data which is to be transmitted is serialized by the control device, the binary signals must be converted to audio-frequency signals (modulated) for transmission over communications lines and reconverted (demodulated) at the other end. A modulation/demodulation device or modem performs this function. One modem is required at each end of a data link (Figure 6-2). Data sets and line adapters have the same function as a modem.

Depending upon the type of communications lines and modem equipment, transmission of data can be voice grade (permits transmission of both data and human voice) or subvoice-grade (transmission of data only). A modem can be an integral part of a control device or terminal, or it can be an external unit.

Communications Lines

Communications lines are classified according to configuration, transmission direction, type, and transmission mode.

Configuration

Two basic communications line configurations are:

- Point-to-point (connects two terminals).
- Multipoint (connects multiple terminals). In a multipoint configuration, one terminal must always be designated as the primary (control) terminal, and all others are secondary (tributary) terminals.

Transmission Direction

A communications line that transmits data in either direction, but not simultaneously, is called *half-duplex*. A line that transmits in both directions at the same time is called duplex or *full duplex* (Figure 6-3).

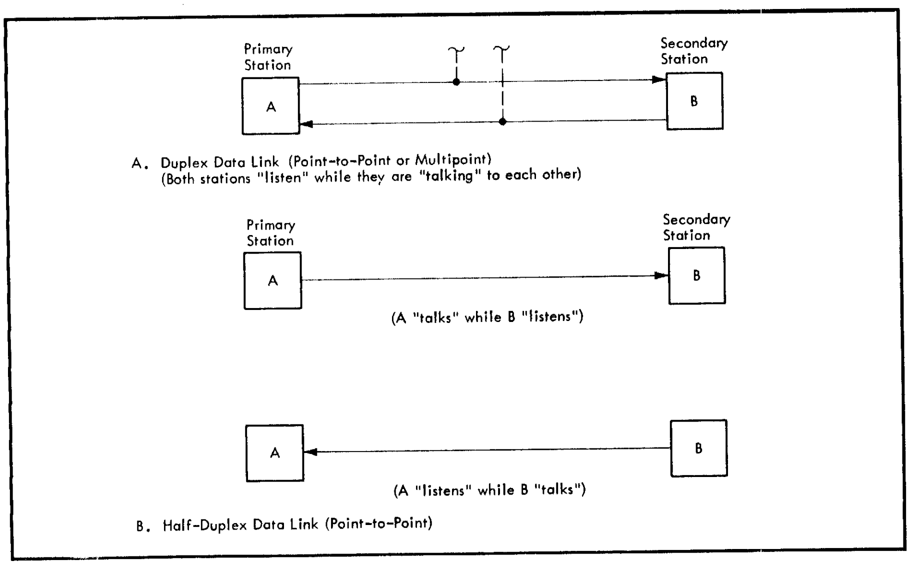


Figure 6-3 Communications Lines Configurations

Type

Basically, two types of communications lines are available: switched and nonswitched.

Switched lines (also called dial) connect terminals by means of common carrier exchange equipment. Dialing establishes a connection, which is maintained only while data is being transmitted. Switched lines are half-duplex only.

Nonswitched lines are available for use at any time, and dialing is not required to make a connection. Nonswitched lines may be either leased or private lines. Leased lines are leased from a communications common carrier and are usually telephone or telegraph lines. Private lines are privately owned and may be supplied by the teleprocessing network owner or by a communications equipment company.

Duplex transmission requires leased or private lines (nonswitched).

Transmission Mode

Communications lines can transmit in asynchronous mode (also called serial start-stop mode) or synchronous mode.

Asynchronous transmission requires the use of start and stop bits to designate the beginning and ending of transmission.

Synchronous transmission is transmission in which the sending and receiving terminals are operating continuously at substantially the same frequency: the receiving terminal on a communications line operates in step with the transmitting terminal through the recognition of a specific bit pattern (sync pattern) at the beginning of each transmission. Synchronous mode, therefore, eliminates the need for start and stop bits and permits continuous uninterrupted transmission, increasing transmission speed and reducing turnaround time.

Codes

A variety of codes can be used to represent data characters when transmitting over communications lines. Two of the most commonly used are ASCII (American National Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code).

Line Disciplines

A *line discipline* provides a set of rules for the orderly transfer of data from one location to another using communications facilities. Two line disciplines currently used are binary synchronous communications (BSC) and synchronous data link control (SDLC).

Binary Synchronous Communications

Binary synchronous communications (BSC) procedure provides for synchronous transmission of binary-coded data. BSC expands transmission capability in a teleprocessing network through its ability to accommodate three different transmission codes and a broad range of medium-and high-speed equipment. BSC offers intermix capabilities which allow different types of BSC terminals to communicate with the host processor (functioning as the control terminal) on a nonswitched multipoint network, or with the host processor (functioning as the central terminal) over a switched point-to-point network. Also available with BSC is a transparency feature which permits greater versatility in the range of coded data that can be transmitted. This versatility is achieved because all data including the normally restricted data link line control characters, is treated as specific "bit patterns" (data only) when transmitted in transparent mode. Thus, unrestricted coding of data is permitted for transparent mode operation. This is particularly useful for transmitting binary data, floating point numbers, packed decimal data, and so on.

Synchronous Data Link Control

Synchronous data link control (SDLC), a more sophisticated line discipline than BSC, provides for the efficient management of synchronous data transmission between buffered terminals using centralized control over communications lines.

SDLC can transmit over duplex or half-duplex, switched or nonswitched lines. Whereas SDLC configurations may be point-to-point, multipoint, or multi-multipoint (Figure 6-4), BSC uses point-to-point or multipoint configurations. Also, SDLC will accommodate any code while BSC accommodates three codes.

SDLC offers a fixed format for transmission which eliminates many of the control characters required for BSC transmission. When using SDLC, message delimiting is not required, error checking is automatic, previous transmissions can be confirmed at the same time that additional data is being transmitted, and any type of data, coded or noncoded, can be transmitted.

SDLC allows terminals with different characteristics to share a single communications line. Thus hard-copy (printer) terminals and video-display terminals can share the same communications lines.

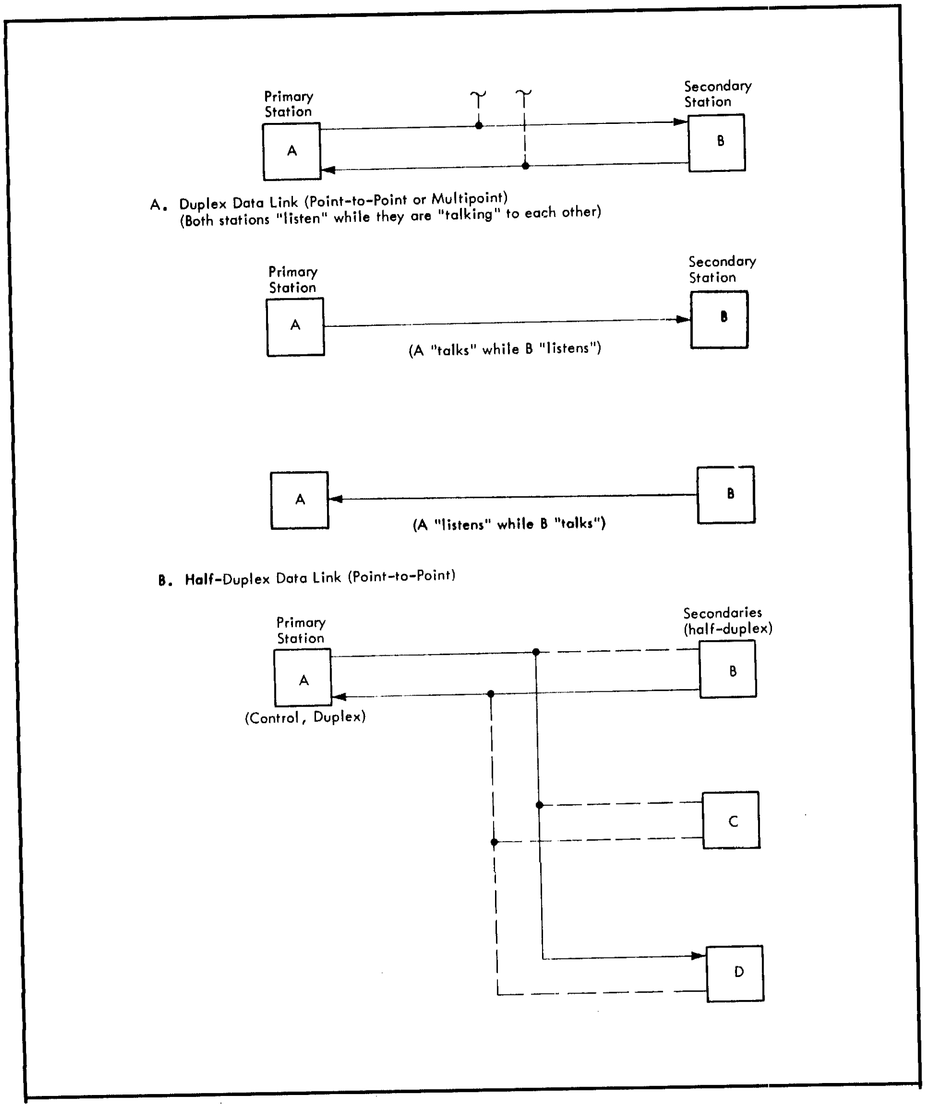


Figure 6-4 Data Link Configurations

Teleprocessing Applications

The types of applications which are provided by a teleprocessing network are many and varied. Some of the most widely used applications include:

- *Data Entry*. Entry of data from a remote terminal into a host processor via a communications link by a remote terminal.
- *Record Update*. Alteration, deletion, or addition of data contained on existing data files stored at the host processor site via a communications link from a remote terminal.
- *Remote Job Entry*. Entry of logic functions from a remote terminal to be executed at the host processor location via a communications link.
- *Message Switching*. The ability to relay a message from one remote terminal to one or more remote terminals via a host processor and a series of communications links.
- *Time Sharing*. The allocation of host processor resources so that many remote terminals may execute programs concurrently and may interact with the programs during execution.
- *Data Acquisition And Process Control*. A high-speed data acquisition system is designed to maintain constant communication with a process for such purposes as:

  1. Determining whether the process is operating within acceptable limits.
  2. Providing records for accounting or management decisions.
  3. Providing a record of data obtained during a research experiment.

  A process control system usually incorporates data acquisition facilities and has the additional capability of using the acquired data as a basis for supervising and controlling the process.

Section 7:

Stored Program Concepts

After data is transcribed to an input medium, the computer system can take over the complete processing and the preparation of results. However, the procedural steps that are to take place within the computer system must be defined precisely in terms of operations that the system can perform. Each step must be written as an instruction to the computer.

A series of instructions pertaining to an entire procedure is called a *program*. In modern data processing systems, the program is stored internally, and the system has access to the instructions at electronic speeds. Such programs are called stored programs.

Instructions

The computer is directed to perform each of its operations by an instruction - a unit of specific information located in main storage. This information is interpreted by the central processing unit as an operation to be performed.

If data is involved, the instruction directs the computer to the data. If some device is to be controlled - a magnetic tape unit, for example - the instruction specifies the device and the required operations.

Instructions may change the condition of an indicator; they may shift data from one location in storage to another; they may cause a tape unit to rewind; or they may change the contents of a counter. Some instructions arbitrarily, or as a result of some machine or data indication, can specify the storage location of the next instruction. In this way, it is possible to alter the sequence in which any instruction or block of instructions is followed.

An instruction (Figure 7-1) usually consists of at least two parts:

1. An operation part that designates read, write, add, subtract, compare, move data, and so on.
2. At least one operand to designate the location(s) of the information to be used for the specified operation.

Operation	Operand(S)
Start I/O	Channel 1, Device 191
Read	One Record into Storage Positions 1000 - 1050
Zero & Add	Quantity in Storage Location 1004 into Storage Location 2000
Subtract	Quantity in Storage Location 1005 from Contents of Register 10
Branch	To Instruction in Storage Location 5004

Figure 7-1 Instructions

During an instruction cycle, an instruction is selected from storage and analyzed by the central processing unit. The operation part indicates the operation to be performed. This information is coded to have a special meaning for the computer. For example, in a System/370 the hexadecimal representation "5A" is for a register add, "59" is for a compare, and "5C" is for a multiply. Other computers use different coding and numbers of characters or positions to define an operation.

The operand further defines or augments the function of the operation. For example, to perform arithmetic, the storage locations of the factors involved are indicated. For input or output devices, the unit to be used is specified. For reading or writing, the area of storage for input or output records is indicated or fixed by machine design.

Because all instructions use the same storage media as data, they must be represented in the same form of coding. In System/370, instructions may be any of three lengths: half-word (two bytes), whole word (four bytes), and word-and-a-half (six bytes), depending upon specific instruction operand requirements.

In general, no particular areas of storage are reserved for the instructions only. In most instances, they are grouped together and placed, in ascending sequential locations, in the normal order in which they are to be executed by the computer. However, the order of execution may be varied by special instruction, by recognition of a predetermined condition of data or devices within the system, by unpredictable interruptions from outside the system (teleprocessing input), by hardware conditions that require servicing from special set of programs, or by other programs that require unusual priority.

The normal sequence of computer operation in a complete program is as follows: The computer locates the first instruction either by looking in a predetermined location of storage assigned for this purpose or by manual reset. This first instruction is executed. The computer then locates the next instruction and executes it. This process continues automatically, instruction by instruction, until the program is completed or until the computer is instructed to stop execution of this program.

Two-Address Instructions

In computers, such as the System/370, instructions have two address portions. Depending on the function of the instruction, the two addresses can, for example, indicate a device to be used and the data to be operated on, or two factors of data to be processed. An output unit to be used could be indicated by one address, and the storage location from which information is to be written could be indicated by the other address. In arithmetic operations, the two addresses could specify two related factors of data, such as multiplier and multiplicand, divisor and dividend, or addend and augend.

Instructions and Data

The only distinction between instructions and data in main storage lies in the time they are brought into the central processing unit. If information is brought in during an instruction cycle, it is interpreted as an instruction; if brought in during any other type of cycle, it is considered to be data.

The computer can operate upon its own instructions, if those instructions are supplied as data. The computer can also be programmed to alter its own instructions according to conditions encountered during the handling of a procedure. It is this ability to process instructions that provides the almost unlimited flexibility and the so-called logical ability of the stored program system.

Developing a Program

In recent years, newer program development techniques have been developed. Some of these Improved Programming Technologies are intended to supplement or to replace the "traditional" methods described in this section. A brief overview of several of these techniques appears later in this publication.

To develop a program, the programmer must know (1) the number of different operations (and their functions) available in the system with which he has to work; (2) the procedure itself, which must be translated, step by step, into computer instructions; (3) the requirements to be met by the result of processing.

The first step in program preparation is a complete analysis of the application to be programmed, including existing and proposed procedures. This analysis is normally accomplished by developing flowcharts and block diagrams, because most data processing applications involve a large number of alternatives, choices, and exceptions.

It is difficult to state these possibilities verbally. Thus, the systems analyst finds use for many types of pictorial representations, including form layouts, control panel diagrams, manpower planning charts, and so on. The two representations to be discussed here are the system flowchart and the program flowchart.

The outstanding value of a flowchart is that it shows a lot at a glance. It graphically represents organized procedures and data flow so that broad essentials and many details, along with their relationships, are readily apparent. Such sequences and interrelationships are hard to extract from detailed paragraphs of text - and, for the program, next to impossible to determine without supporting documentation. In flowcharting, symbols and words support each other; identifications and descriptions, which may be obscure in text, take on more significance when placed in diagrammed sequence. The communication is further improved by consistent use of meaningful symbols. Figure 7-2 reproduces the IBM template envelope.

The template provides basically for two kinds of flowcharts - system and program. This distinction is not new. A system flowchart shows the flow of data through all parts of a data processing system. A program flowchart shows what takes place within a particular program in a data processing system.

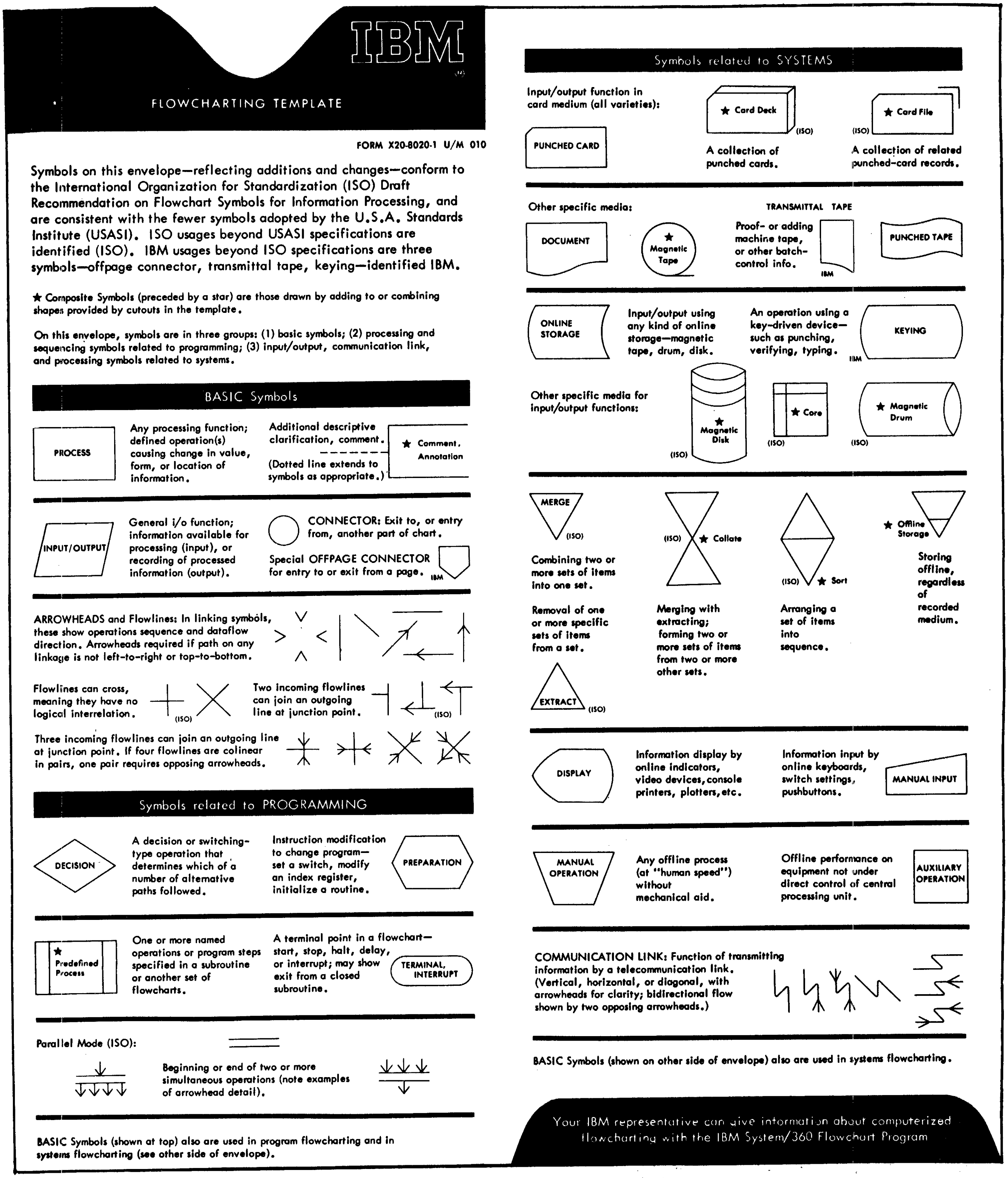


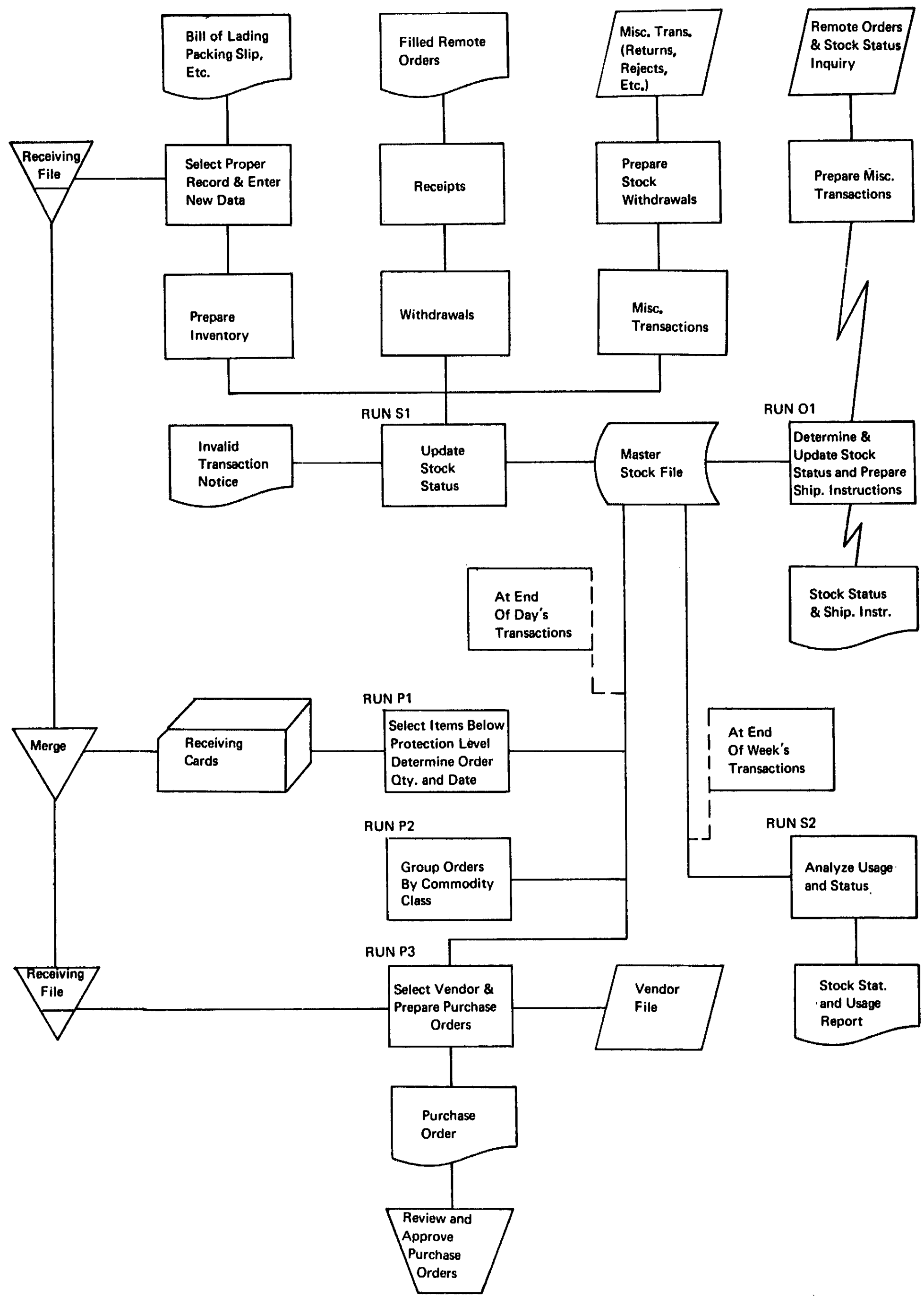
Figure 7-2 Program and System Flowchart Symbols

System Flowchart

A system flowchart represents an application in which data provided by source media is converted to final media. The emphasis is on the media involved and the work stations through which they pass. In a program flowchart, on the other hand, emphasis is on computer decisions and processing; the chart provides a picture of the problem solution, the program logic used for coding, and the processing sequences.

Because a fundamental program flowchart evolves from a system flowchart, the former inherently has more detail than the latter. Beyond that, a program flowchart is quite explicit, frequently "exploding" into a series of subsidiary flowcharts to an extremely high level of detail.

System flowcharts are likely to be simpler, less "formalized" and easier to draw than program flowcharts; they are also more flexible. A greater choice of symbols is available, as well as more latitude in their use. Examples of the symbols, conventions, and techniques used in a typical system flowchart are shown, with comments, in Figure 7-3.



A typical system-flowchart description of an inventory-control application, this chart uses specific symbols for certain processing functions and input/output. The application involves a multiple-warehouse system: items are stocked in a central warehouse for distribution to remote warehouses; all customer orders are received by remotely-located warehouses and transmitted by teletype (communication-link symbol) to the central data-processing installation. The system provides four major groups of operations:

1. Updating stock status [run S1], based on actual transactions.
2. Response to inquiries [run O1] from auxiliary warehouses and central warehouse.
3. Reorder analysis [runs P1, P2, P3], including purchase-order preparation.
4. Weekly analysis reports [run S2] to show slow-moving items, major changes in usage rates, behind-schedule deliveries, economic lot sizes, etc.

Figure 7-3 Flowchart for an Inventory Control System

Program Flowchart

Certain aspects of program flowcharts deserve emphasis by themselves, because they are so specialized and so thoroughly integrated into both the routine and the creative phases of programming. To the programmer, a program flowchart is a kind of all-purpose tool. It is the "blueprint" of a program. In program development, the programmer uses flowcharting in and through every part of his task: to marshal and organize his facts on paper; to outline problems, logic, and solutions; to deal systematically with the problem as a whole. He uses flowcharting to build, step by step, his own reference documentation and reminders.

In the development stage of a program, a flowchart serves as a means of experimenting with various approaches - laying out the logic. The programmer starts with symbols representing major functions of a proposed program. He develops the overall logic by combining blocks to depict input/output functions, steps for identification and selection of records, and decision functions.

Once the programmer has tentatively established mainline logic, he usually extracts large segments and describes them in more detail on subsidiary charts. This is like drawing a set of increasingly detailed maps - starting with a general, all-inclusive map, then exploding sections of it on succeeding maps, each map showing greater detail. The technique is called modular program flowcharting. For thorough documentation on this basis, a typical, file maintenance routine could possibly require as many as 80 flowcharts.

When the programmer is satisfied that the procedure is sound, he uses the flowchart as his guide for coding. Sometimes, at this stage, program logic may have to be modified to agree with machine logic, and a chart may have to be redrawn and reverified. Modifications are even more likely during testing, installation, and future operation.

Flowcharting Worksheet

Because most program flowcharts are so detailed, it is a great advantage that they be drawn in a consistent, well organized format. Such "formalization" is best done on a regular form designed for that purpose. The IBM flowcharting worksheet form (GX20-8021) is shown in Figure 7-4 with a typical chart superimposed on it.

The 11 x 16 1/2" worksheet can be used for all kinds of flowcharts, but it is particularly useful in program flowcharting.

Essentially what the worksheet provides is an arrangement of 50 blocks with alphabetic and numeric coordinates. Ten horizontal rows are lettered from the top (A) to the bottom (K). Five-vertical rows are numbered from left to right - 1 through 5. These coordinates prove helpful in documenting and cross-referencing.

The worksheet has guidelines for the 50 positions, and crosslines indicating the horizontal and vertical centers of each position. These are simply aids for squaring up flow-lines, centering symbols in each position, and maintaining uniform spacing between symbols. The result is a neatly arranged, compact chart, but not too crowded. The worksheet itself is printed in light-blue ink so that its guidelines will disappear during photographic copying.

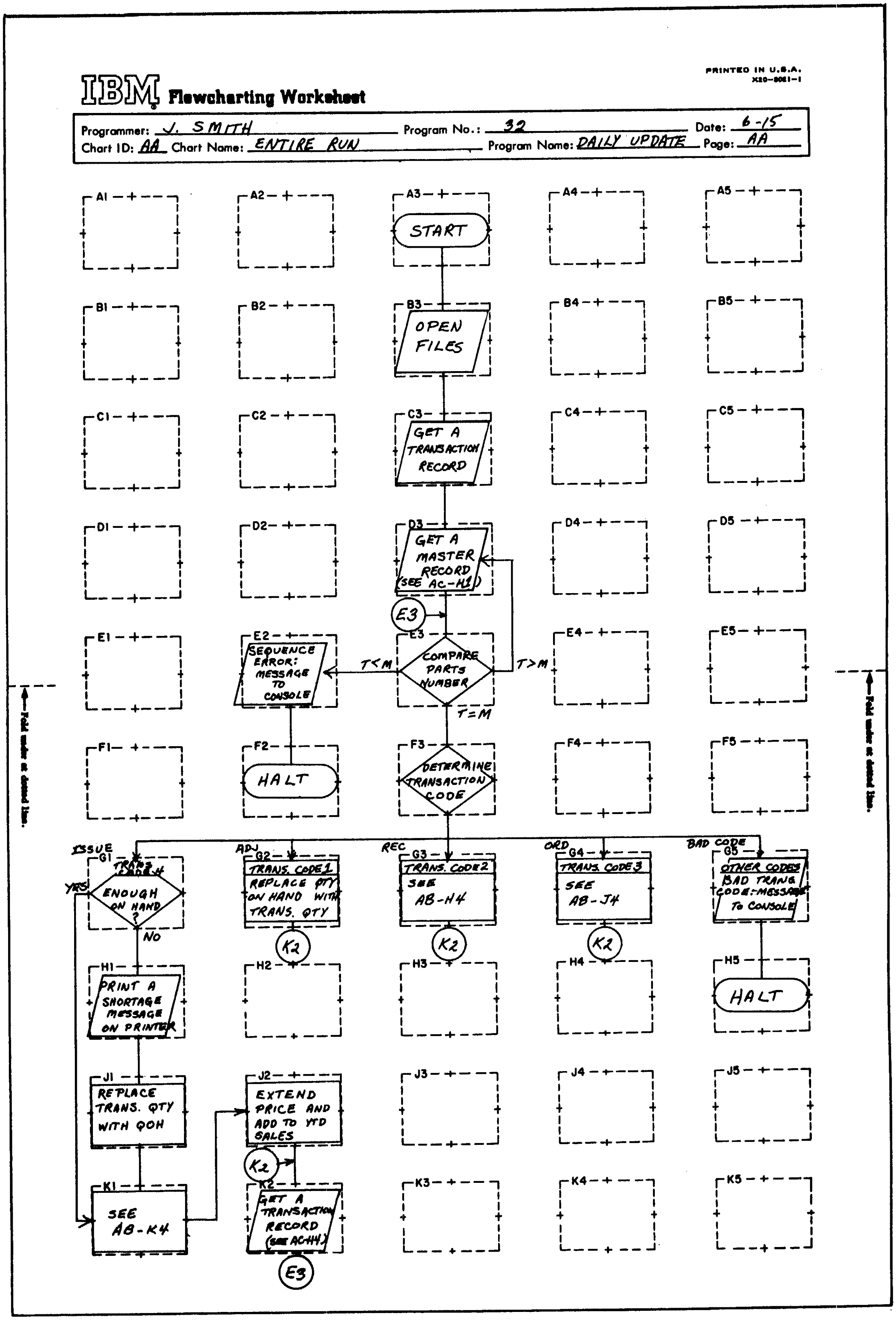


Figure 7-4 First Sheet of a Sample Program Flowchart for an Updating Run of Inventory Records

Miscellaneous Techniques

A few of the more detailed techniques in the drawing of program flowcharts are described briefly and illustrated:

Cross-referencing relates the program flowchart to source language programs (described later). One way is to locate an instruction either by its label or by the page and line number of the coding sheet on which it appears. The cross-reference can be placed above the upper left corner of a symbol, as shown in Figure 7-5.

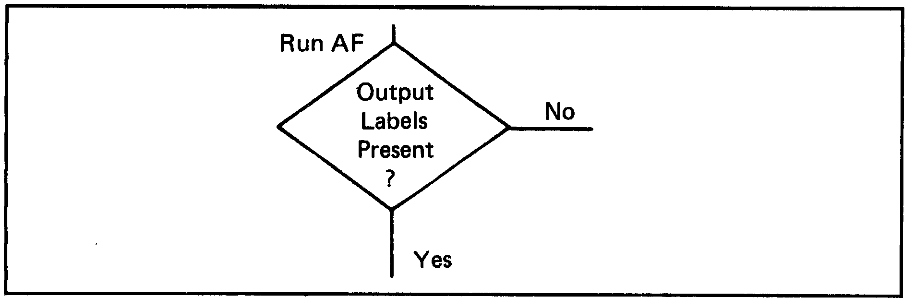


Figure 7-5 Example of Cross-Referencing

Composite symbols are a defined group. Although more than thirty symbols are described on the template envelope (Figure 7-2), about one-third fewer symbols are provided as cutouts in the template. The additional symbols are those drawn by adding to or by combining shapes appearing as template cutouts; these composite symbols are identified underneath the cutouts, their names are preceded by a star, and the symbols themselves are shown on the envelope. For example, merge is shown by an inverted triangle, by adding a horizontal stripe, the composite symbol signifies *off-line storage*.

Sort needs the addition of a horizontal line at its midpoint to conform to the International Standards Association's standard. Consequently, it is shown on the template as a starred composite symbol.

Where a more detailed flowchart of the program unit exists, the predefined process symbol (Figure 7-6) is used to indicate this.

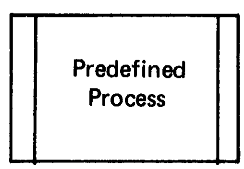


Figure 7-6 Predefined Process Symbol

Decision Techniques may be shown in several ways. See examples in Figure 7-7. These decisions will determine which action is to be performed next by the program (see branch operation).

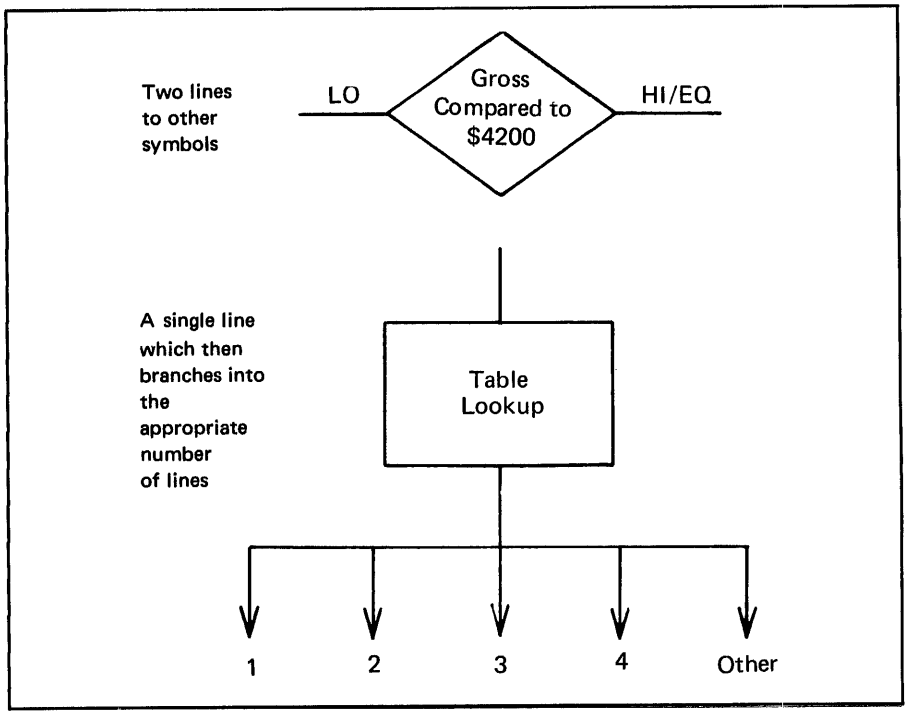


Figure 7-7 Examples of Decision Techniques

An example of a typical program flowchart is shown in Figure 7-4 (superimposed on the flowcharting worksheet form referred to earlier). The program flowchart, for a daily updating run of master inventory records, is based on the system flowchart. The system flowchart (Figure 7-8) indicates that the updating is from adjustments, receipts, orders, and issues for the day; in addition, a shortage-and-reorder listing is prepared. The system requires no additions or deletions involving the master file.

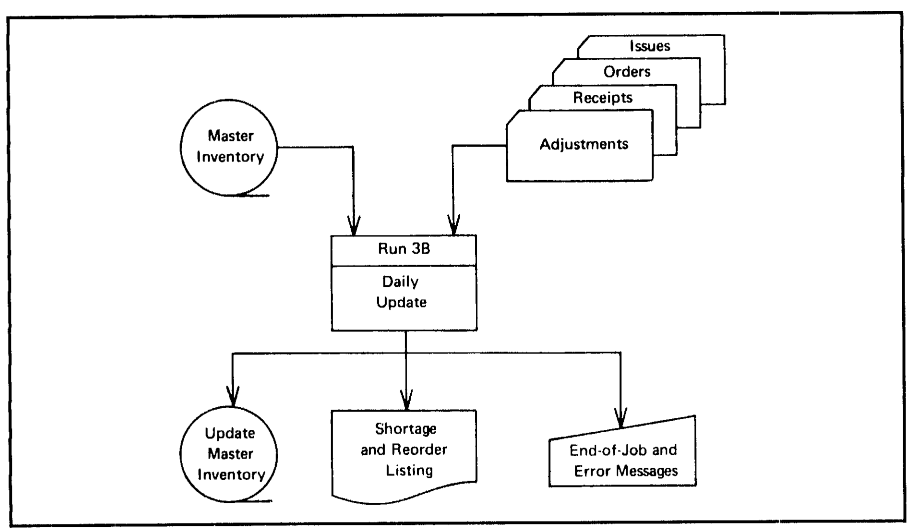


Figure 7-8 System Flowchart

Flowcharting by Computer

All of the foregoing has dealt with manual flowcharting - drawing the charts by hand. Much progress has been made, however, on the running of flowcharts on a computer. An IBM flowchart program for the IBM System/370 enables the computer to print out, from detailed instructions, a complete flowchart - including machine versions of the same symbols usually drawn by hand.

Mechanized flowcharting is particulary helpful in programming, where modifications of a program often require time-consuming redrawing of charts. With this program, flowcharts manually drawn in a prescribed manner (and coded) can be produced automatically; once produced, they can be modified and rerun with a minimum of time and effort. The flowcharting worksheet in Figure 7-4 ties in nicely with this implementation: the 50-position grid provided by the worksheet lends itself readily to mechanization.

Reading Data

All data entering the computer system must first be read by an input device and routed to main storage. Each input device is assigned a number to serve as its address in the same way that each storage position is also assigned a location address.

A data processing procedure is normally concerned with entire files (called *data sets*, in Operating System (OS) - described in a later section) of records on one or more of the input media. These files are either fed into the input device, where the computer has access to them, or read directly from a secondary storage unit. To read a record from a file, one or more instructions in a program activate the input device and place the record in main storage.

At this point, it must be determined exactly where in storage the incoming record is to be placed, and an instruction must direct the machine to send it to this predetermined locations. Also, in the plan of manipulation, it is necessary to know at all times where to find information as needed in the successive stages of processing.

These considerations involve the allocation of storage space for specific purposes in a logical and convenient manner. For example, particular fields or quantities may be used for computation. The instructions to be used later must specify the location in storage where the information from each record can be found.

When a data processing system includes an operating system of programs to take care of placement of incoming records and fetching of stored records as needed, the problem programmer is not concerned with the location details.

The reading operation performs the following distinct functions;

1. The input device is selected and made ready before actual reading begins. The device chosen is the one that has access to the proper file records as determined by the programmer. This device is selected by specifying its assigned code number or address.
2. The read instruction causes the previously selected unit to carry out the transfer of a record to the storage of the computer. The record is placed in a particular storage area reserved for this purpose and is then available for further processing. A number of input areas may be assigned to

handle several related records at once (for example, a master record and its corresponding transaction detail).

3. The order of read instructions in the program determines the sequence in which the files are read. Other instructions later compare records from separate files to determine the relationship of detail to master, detail to detail, and so on.

4. The number of records to be placed in storage at one time depends on the construction of the files, the type and length of records being handled, and the available storage capacity.

Calculating

Once data has been read into the computer system and placed in known locations of storage, calculation can begin. Each computer is capable of performing addition, subtraction, multiplication, and division, either as built-in operations or under program control. For most commercial applications, these operations are adequate. Even in many of the more advanced scientific procedures, the most complex equations can be reduced to steps of elementary arithmetic. However, many specialized operations can be performed by some systems to make the solving of mathematical problems easier.

In every operation of simple arithmetic, at least two factors are involved: multiplier and multiplicand, divisor and dividend, and so on. These factors are operated on by the arithmetic unit of the machine to produce a result, such as a product or a quotient. In every calculation, therefore, at least two storage locations are needed. One quantity is usually in main storage, the other in a register. In System/370, both quantities may be in registers.

A calculation can be started by placing one factor in the register and at the same time clearing this unit of any previous factors or results that may be contained there. The address part of the instruction specifies the storage location of the first factor; the register is implied by the operation. In some computers, more than one register is available for calculation. In this case, the address must also specify the register to be used.

When one of the factors is properly placed in a suitable register, the actual calculation is executed by an instruction whose operation part specifies the arithmetic to be performed and whose operand is the location of the second factor. The computer acts upon two factors, one in the register and the other in storage, and produces a result in either place, as directed. In System/370, using certain instructions, both factors may be in storage with the result of the calculation replacing one of the factors.

The result may be moved to a storage area, as a *field* in some record. A field is a related arrangement of characters or digits to represent a quantity, amount, name, identity, and so on.

Any practical number of calculations can take place on many factors in a single series of instructions. That is, a factor may be placed in a register and multiplied, and several other factors may be added to or subtracted from the product. Division can then be executed, and other operations of adding and subtracting can proceed using this quotient. Intermediate results can be stored at any time.

For example, a field containing employee hours worked can be placed in a register and multiplied by hourly rate to produce earnings. Piece work and bonus amounts may then be added to develop a total regular earnings amount, which is stored in the pay record. Total regular earnings may then be divided by hours to produce an average hourly rate. This rate is multiplied by 1.5 overtime hours to produce overtime earnings. Total gross pay is then calculated and stored. Taxes are computed using the calculated gross pay; other payroll data is accumulated using the amounts as they are calculated. Intermediate results of tax amounts and deductions, and, finally, net pay are all stored in the pay record.

Operations of shifting and rounding the contents of the register are also provided to adjust, lengthen, or shorten results. With these operations, decimal values may be handled and directions for placing of the decimal point may be given to the computer.

All calculations must take into account the algebraic sign of factors in storage or associated registers. Consequently, the computer system is equipped with some provision to store and recognize the sign of a factor.

If records are made up of fixed words of data, one position of the word is designated as a sign position and automatically accompanies the word. Registers also include either a special sign position of storage or a sign indicator that is available to the programmer. In this way, the sign of results can be determined, together with the effect, after calculations. The computer follows the rules of algebra in all basic arithmetic calculations.

The size of words, quantities, and values depends upon the design of each particular system. The exact rules governing the placement of factors, size of results, etc., vary somewhat from system to system. In all cases where a result is expected to exceed the capacity of the register, the programmer must arrange his data to produce partial results and then combine these for totals. Other operations of scaling may be executed so that very large or small values and fractions may be handled conveniently. Computers designed primarily for mathematical applications generally include a series of specialized arithmetic operations for this purpose. (See "Floating-Point Operation".)

Calculation is carried out in all computer systems at much higher rates of speed than input or output, because reading and writing require the use of mechanical devices and the movement of documents, while calculation is performed electronically. In many commercial applications, calculation is relatively simple, and the overall interleaving of the system is usually governed by the speed of the input/output units. In mathematical applications, the situation is reversed; calculation is complex and involved, and high calculating speeds are essential. The design of any particular system must achieve a realistic balance between calculating and record-handling ability.

Logical Operations

The sequence in which a stored program computer follows its instructions is determined in one of two ways: either it finds the instructions in consecutive storage locations, or the instruction operand also designates the location of each following instruction. If instructions could be followed sequentially only in a fixed pattern, a program would follow only a single path of operation without any possibility of dealing with exceptions to the procedure and without any ability to choose alternatives on the basis of special conditions encountered in processing data. Further, without some way of resetting the computer to repeat a given series of instructions, it would be necessary to have a complete program for each record in a file.

Consider the program illustrated by the flowchart in Figure 7-9.

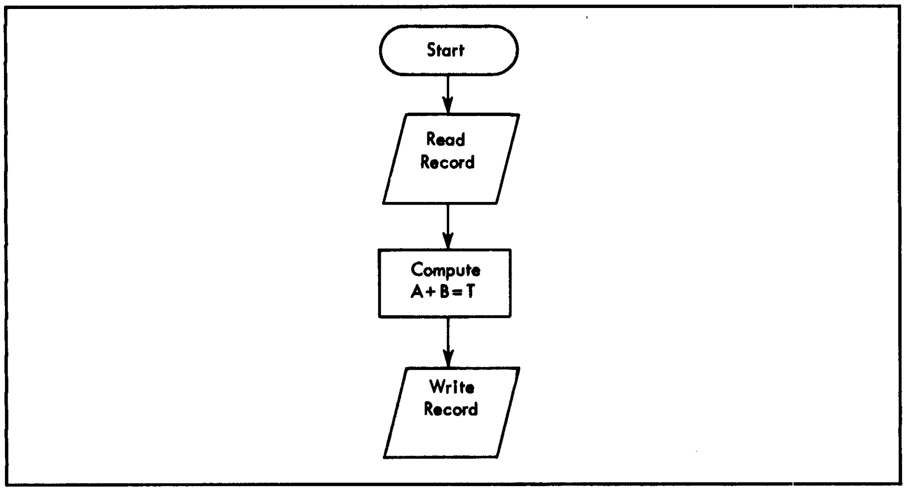


Figure 7-9 Program Flowchart A + B = T

These instructions taken alone compute T for only one record. But, by returning to the first instruction, any number of records can be processed, repeating the same program as a *loop*. For this purpose, another instruction is given to return to the first instruction (Figure 7-10). Once this program is initiated, it will continue to run until there are no more records to process. Program loops are common, and they can be terminated in many ways.

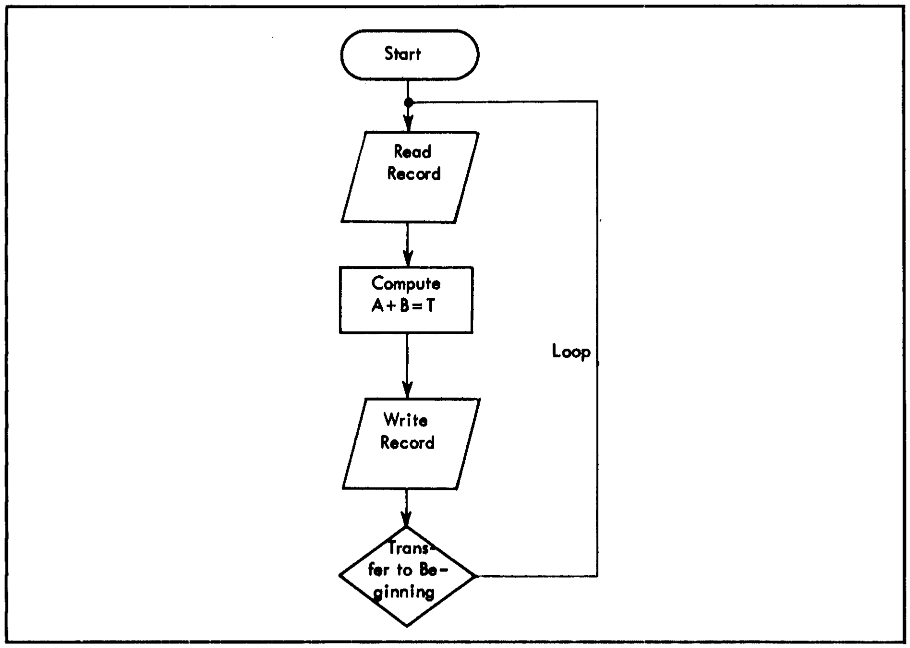


Figure 7-10 Program Loop

For example, the computer may be instructed to examine T each time it is computed and to go to a certain routine when the value of T becomes negative (Figure 7-11).

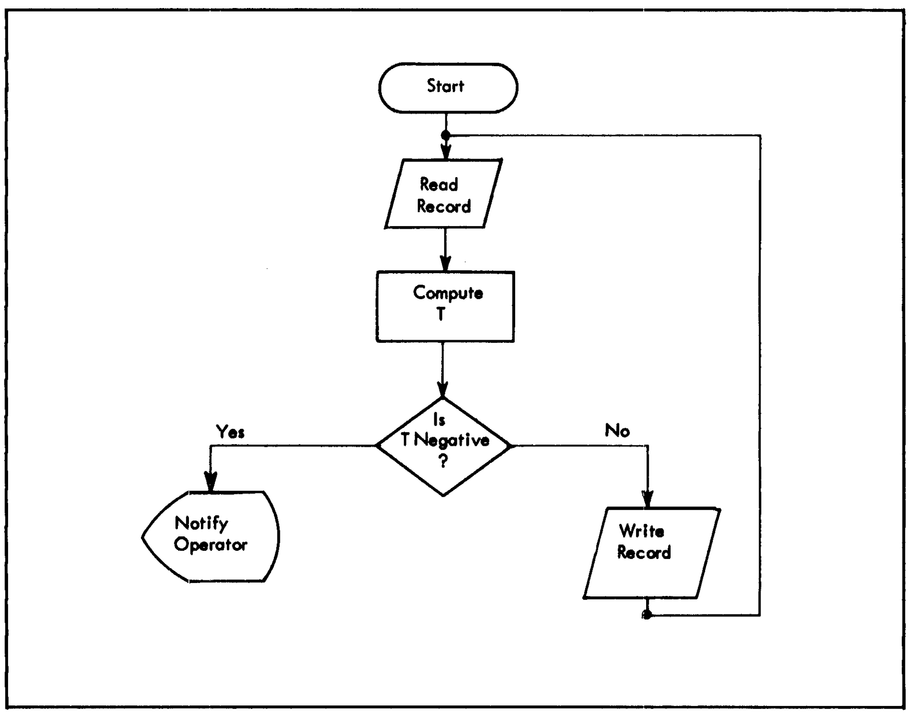


Figure 7-11 Conditional Transfer

In this case, the instruction becomes a conditional transfer or branch. The program is repeated only if some predetermined condition has been satisfied. The computer may also be instructed to execute the program for ten records and then go to a certain routine (Figure 7-12).

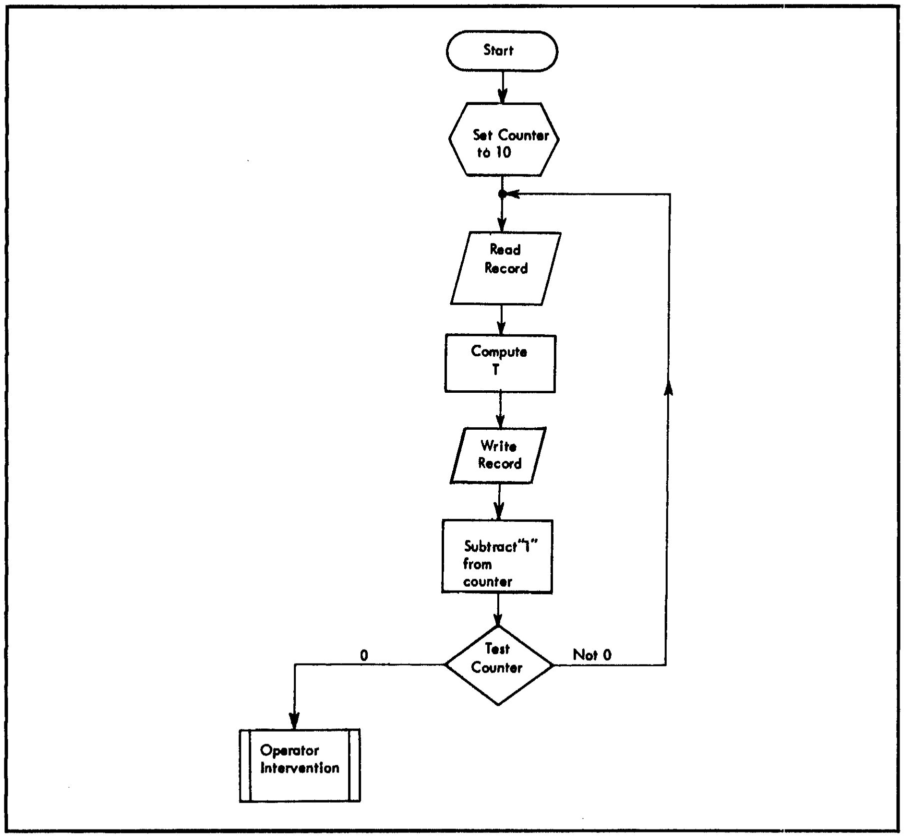


Figure 7-12 Record Count Conditional Transfer

It is assumed that the constants 10 and 1 are in the computer and that 1 is subtracted from 10 each time the loop is completed. After ten times around, a zero will be found in the location that contained 10. A transfer or branch then terminates the loop.

The conditional transfer or branch operation may be used to cause a special purpose subroutine to be executed outside the normal or straight-line path of the program. This subroutine is executed only when a predetermined exception or condition is noted by the machine.

One common example of the subroutine is checking the accuracy of records as they are read from, or written on, magnetic tape. As each record enters or leaves the central processing unit, a read/write error indicator is examined. If the indicator has been turned on, the computer is instructed to enter a subroutine of instructions that attempt to correct the error. The program logic for such an operation - the reading only - is shown in Figure 7-13. A similar loop would also be included for writing.

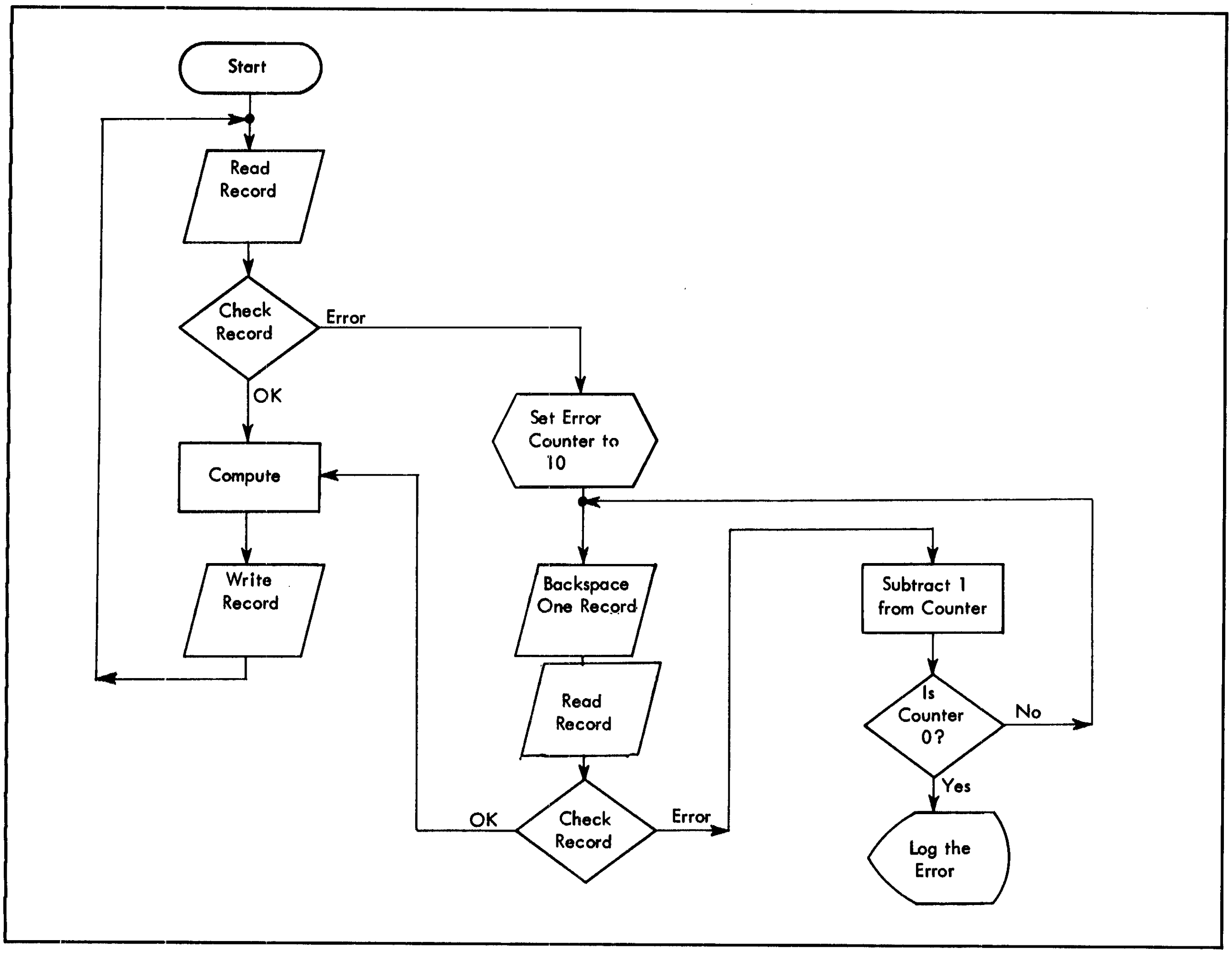


Figure 7-13 Read Error Loop

When a reading error is detected, a branch is effected to the error subroutine. A counter is reset to the quantity 10 to count the number of times a reread will be attempted. The tape is backspaced over the error, and a second read instruction is given. Another check is made to determine whether this operation is correct. If it is, a transfer returns to the main program, where computing continues.

If the error persists, 1 is subtracted from the counter, and the counter is tested for 0. The error loop is again entered and a second reread and check are executed. The machine can reread ten times. If the error is not corrected, the program transfers to another routine, where it goes through some sort of procedure to log the error record and then return to read in and process the next record. With IBM 3420 nine-track tape, however, the cyclic redundancy character feature corrects single-track tape errors automatically, thus eliminating the need for this type of program, in many instances. Also, input/output control systems and operating systems (programming systems supplied by IBM) are available for most data processing systems, eliminating the need for the problem program to include this type of checking.

A program can also be arranged so that the machine can recognize one or more types of records as they are processed from a single file. The method of computation can be varied according to the type of record in storage. This procedure is common when a number of types or classes of transactions are processed against a single master file (for example, in an application of file maintenance).

Assume that a file (data set) of master stock status records contains quantities that reflect the number of parts available for manufacturing planning. The records also have considerable other information pertaining to the status of inventory, but for purposes of illustration, this example is concerned only with those fields used to show availability.

These fields are:

Quantity in stock
Quantity on order
Quantity available

Transactions that affect the status of the parts availability originate daily. These transactions are punched in cards with an identifying digit code for each type of activity.

Codes are as follows:

Code 1 Receipts
Code 2 Orders
Code 3 Withdrawals
Code 4 Adjustments plus
Code 5 Adjustments minus

As each transaction is placed in storage, it is analyzed by code to determine the class to which it belongs (Figure 7-14). A branch instruction then transfers to the proper program subroutine to calculate availability and to adjust the corresponding master record. Reading and writing of the adjusted master record are not shown in the flow hart.

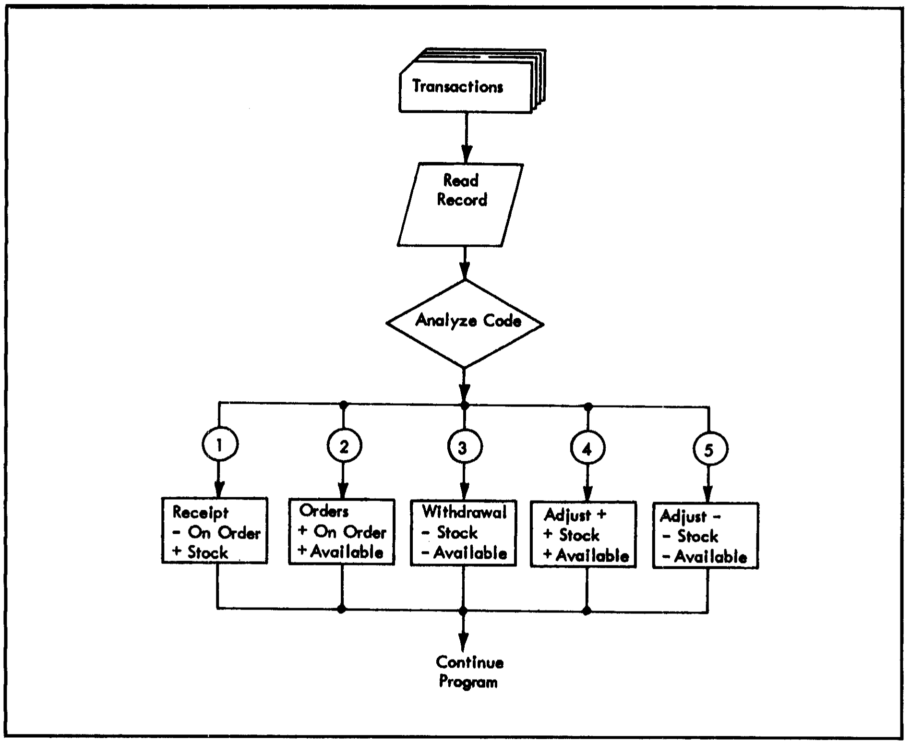


Figure 7-14 Branching by Code

Comparing

The ability of the computer to make limited decisions on the basis of programmed logic is substantially extended by operations of comparing. Such operations enable the computer to determine whether two data fields in storage match, or whether one is lower or higher than the other. The basis of comparison is set according to some predetermined sequence built into the circuitry.

The sequence may be considered to be a normal filing order of records of all types. For example, the familiar ascending sequence of the digits 0-9 assumes that the digit 9 is the highest digit of the series. In the same manner, the letter Z is assumed to be the highest letter of the alphabet. To the computer, therefore, as in any file, the number 162 is higher in sequence than the number 159, and the name Jones is lower than the name Smith. Special characters, such as /@*, or-, may also be included, because all computer data has a value that can be compared with any other value. This is known as the *collating sequence*.

Comparing operations are used to program the sequence checking of files, sorting procedures, or the rearrangement of records in some desired order. The comparison of an identifying field in one record with that of another enables the computer to handle a number of associated files in one processing procedure, provided that all files are in sequence by this common field.

One or both fields are placed in a storage register(s). The compare instruction then compares the first field against the second. (The second field, in System/370, may be in a second register; in other systems, it is in a main storage location). The results of comparison are registered as high, low, or equal, by indicators or triggers that may then be interrogated to determine their condition. If the indicator is on, a branch (transfer) instruction transfers the machine to a subroutine that will continue processing according to the result of the comparison.

Figure 7-15 shows a typical program arrangement for sequence checking a single file of records. All records in the file are assumed to be in ascending sequence by account number.

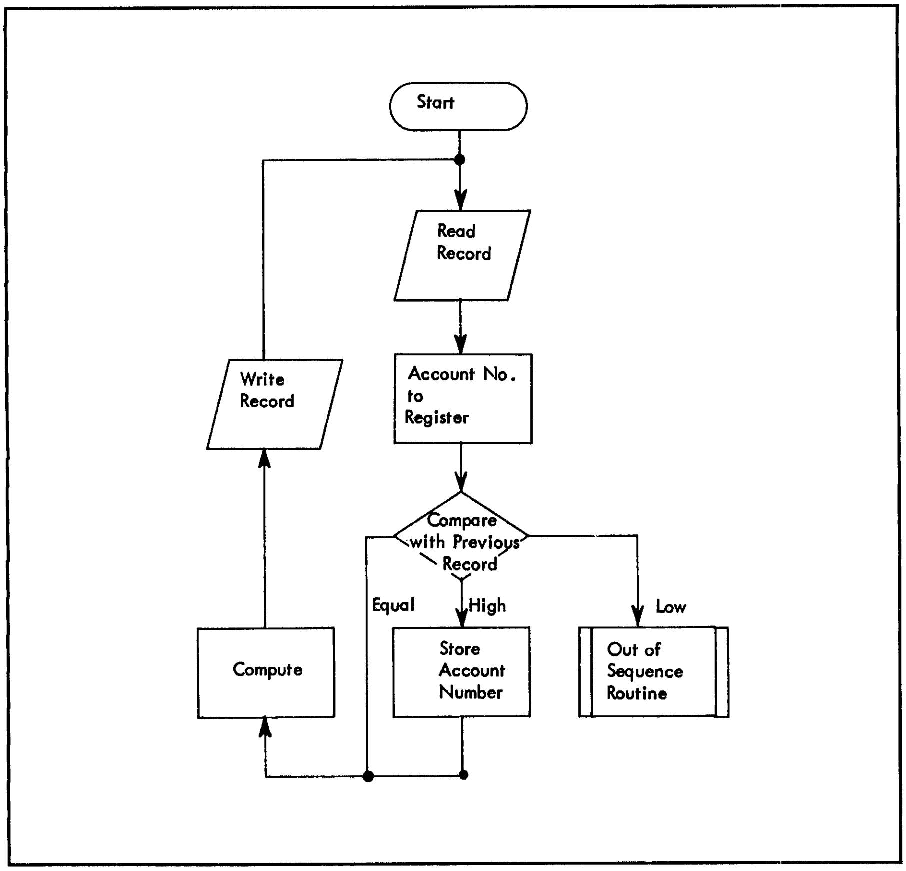


Figure 7-15 Sequence Checking

An input area is set aside in storage, where records are received, one at a time, from an input unit. A second area is also reserved in storage to store the account number from the preceding record. The purpose of this area is to allow comparison of the account number of the incoming record with the corresponding field of the previous record.

If the file is in ascending sequence, the incoming record should always be higher than the record that preceded it. When duplicate records are encountered, the incoming record is equal to the preceding one. If any incoming record is lower than the previous record, it is recognized as an out-of-sequence condition, and the program transfers to a subroutine to take corrective action. After each high comparison, the account field is placed in storage, where it may be compared with the next record.

Indexing

In most computers, the address operand(s) of an instruction can be modified by adding or subtracting variable quantities contained in one or more special purpose counters. The counter may be called an index register when it is set aside specifically for this purpose, or it may be a predetermined location in main storage called an index word. A computer may have several index registers or a number of storage locations for index words.

Computers with an indexing feature use an instruction format that allows a particular register or word to be specified as a part of the instruction operand.

Assume that 50 quantities are placed in ascending word positions of storage from locations 1001 to 1050 inclusive and that these quantities are to be added to the contents of a register. Without indexing or some other means of address modification, it is necessary to repeat an add instruction 50 times with the address of each instruction incremented by 1, as ADD 1001, ADD 1002, ADD 1003, and so on.

With indexing, the add instruction can be written as ADD 1000, with the address incremented by an index register containing the quantity 1. The address remains 1000, but the computer calculates an effective address of 1000 plus 1, or 1001. When the add instruction is executed, the contents of the index register are also incremented by 1, giving a total of 2. When the same add instruction is reexecuted and is again incremented by the contents of the same index register, the effective address os 1000 plus 2, or 1002. If a program loop is formed to repeat this process, the effective address of the add instruction is stepped up 1 each time it is executed (as the index register is stepped up). When the index register equals 51, all 50 quantities will have been added, and the loop is terminated. The computer has consequently performed 50 operations using the same instructions.

Figure 7-16 is a flowchart of the index loop . The first instruction places the quantity 1 in index register 4. An add instruction, with an address 1000 also specifies as part of its operand a designation that the given address is to be modified by the quantity contained in index register 4.

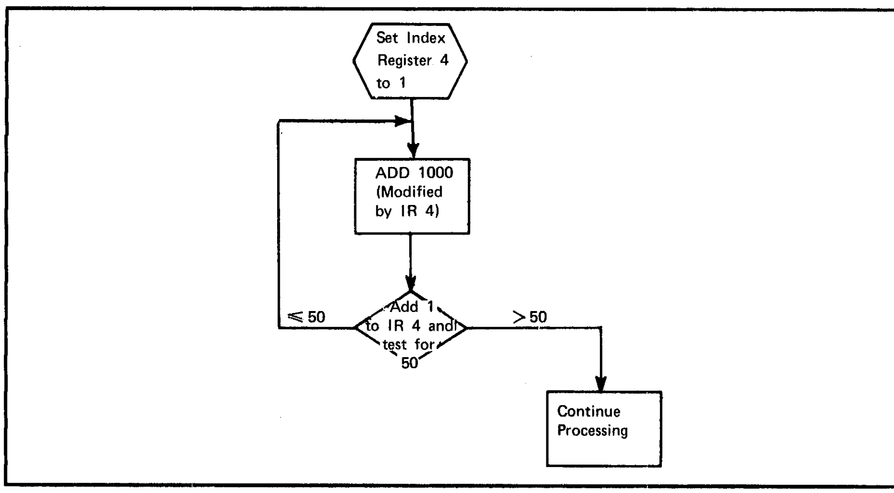


Figure 7-16 Index Loop

The next instruction is branch on index, which means increase the contents of the index register by a quantity contained in another register - in this case, the quantity is 1; if the contents of the index register are less than or equal to 50, branch to repeat the add instruction; if the contents of the index register exceed 50, continue to the next instruction in the program.

The indexing feature greatly simplifies programming of repetitious calculations or other operations and reduces the required number of instructions.

In System/370, the method used to address storage is known as *base displacement addressing*. A general register is used as a *base* register for this purpose. An operand of an instruction referencing a storage address does not contain the actual storage address. Instead, it contains a reference to the base register and a *displacement* value such that the sum of the contents of the base register and the displacement value equals the storage address. Furthermore, any instruction that makes reference to an index register will have its storage address modified each time the instruction is executed by adding the index value to the base displacement value. The base, displacement and index values are left unchanged as a result of executing the instruction.

Linking

Up to now, we have discussed conditional branches (transfers). In more modern computers, it is usually possible to *link*, with one instruction, to a subroutine so that the program can come back to its point of departure from the main program after finishing with the subroutine, and not have to store, unload, reload, and perform other housekeeping jobs. By using *indirect* addressing in the linking procedure, a programmer can independently write many subroutines; the link instruction then causes the computer to insert the desired effective return address in the appropriate instruction of the subroutine each time the subroutine is entered. We might imagine each of these subroutines as a data set (comparable to a book in a library with disk storage acting as a revolving bookcase, Figure 7-17). The exact linking method differs from computer to computer; a simplified procedure is outlined below.

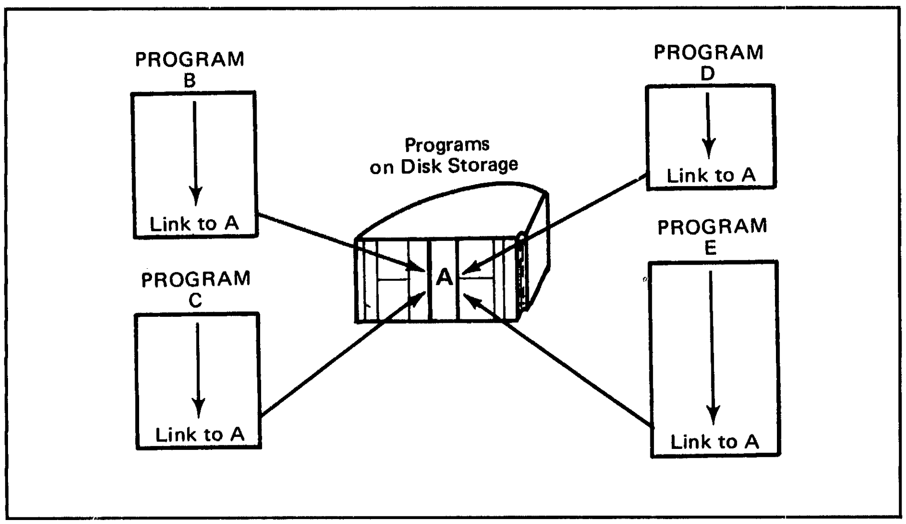


Figure 7-17 Modular Programs with Linking

Suppose that many different types of reports are to be printed at a remote (ASCII programmable) IBM 3770 Data Communication System and that the computer is also polling other remote terminals for input. Assume also that as soon as the internal processing of the information for each of these types of output reports is completed, the information is stored on disk files in standard EBCDIC format. Each time a message is taken from a disk file, it must be translated into ASCII code format. The program sequence might be:

```
INSTRUCTION NUMBER
    1000 Link to:  Read message 1 from disk
    1001 Link to:  ASCII translation routine
    1002 Link to:  Process input from line 1, if any
    1003 Link to:  Write message 1 to 3770 on line 2
    1004 Link to:  Process input from line 3, if any
    1005 Link to:  Read message 2 from disk
    1006 Link to:  ASCII translation routine
    1007 Link to:  Process input from line 4, if any
    1008 Link to:  Write message 2 to 3770 on line 2.
    .                          .
    .                          .
    .                          .
```

In this case, the first time the program links to the ASCII translation routine, either the programmer puts the address 1002 as the return address or the computer does it automatically for him. The second time, the return address is 1007. In step 1002, the program links to a routine to test for a certain condition. This means that the subroutine ends with another subroutine. If there is no input from line 1, as the result of some indicator being tested, the program branches to pick up the indirect address (stored previously by either the programmer's program or the computer). If there is input from line 1, the last instruction of the subroutine will have to have the indirect address that points the way back to 1003.

Factors in today's computers that make this type of programming indispensable are internal interrupt systems that permit processing to continue until some type of input/output activity is ready to start, or that permit a second program to be processed while a first one is waiting for an error condition to be rectified or an I/O operation to take place, and external interrupt systems that permit teleprocessing interrupts from remote terminals.

It is easy to see, from the simplified programming example described above, and from Figure 7-17, that programs that are subject to interrupts must consist of short subroutines with a hierarchy of linking, and, if the computer itself does not insert the return address in the "linked to" subroutine, the programs must build tables of indirect addresses.

Depending upon the design of the computer, more or less of this linking procedure can be done automatically. The programmer must store the subroutines and call for them when the computer does not do it automatically.

Interrupts that can occur as the result of a normal anticipated computer function, such as an interval timer timing out or the completion of an I/O operation, can cause an automatic link to an address that is in a fixed location in main storage. Other types of links that cannot be anticipated by machine design require that the program itself do the linking and maintenance of the indirect addresses.

Chaining

Chaining has several possible connotations in present-day IBM computers. In general purpose computers, such as System/370, chaining refers to I/O channel program-linking of commands or data addresses as *command chaining* or *data chaining*. Linking described a system for keeping track of the return addresses at the end of a program subroutine; chaining, on the other hand, is a similar technique of programming for a "subroutine" of input/output channel commands to be carried out, independently of CPU activity. This can take place in a computer system that permits the programmer to include, as part of one command, the address of the next command that the channel is to execute when it finishes its current input/output operation. The next command may be either to start a different input/output operation or to transfer data to or from a different location in main storage.

Another concept of chaining is one that exists in computers designed especially for communication control. Here, chaining is a system of automatic block allocation, whereby incoming data from each line is automatically stored wherever main storage space is available, with the computer automatically inserting the address of each new block assigned to a message in the last two characters of the previously assigned block, wherever it may be. On output messages, chaining is not quite so automatic. The program inserts the address of each new block (of 32 characters) in the last two characters of the previous block. This type of chaining eliminates the need for allocating ahead of time a static amount of storage space for each line, space that may either be insufficient or go to waste because of inactivity. Chaining, in this sense, is an efficient form of data buffering discussed earlier under "Input/Output Devices".

Section 8:

Programming Languages and Techniques

The capability of computers is expanding at a fantastic rate, and the technology of utilization and control is advancing at an equal pace. These improvements in techniques are as vitally important as the design of the data processing system itself. To a large extent, the future of computers depends not only on increases in speed, logical ability, and storage capacity, but also on the efficient use of these facilities as they are made available.

Programming languages and techniques have been developed by IBM and others to meet both present and future requirements of computer application.

Program Preparation

A computer program represents much more than a set of detailed instructions. It is the outcome of a programmer's applied knowledge of the problem and the operation of the computer system.

Problem definition, analysis, and program flowcharting (see preceding section) are the first steps in program preparation. They are usually carried out independently of the computer and the programming system.

Some or all of the following must be considered to prepare even the simplest program (without the aid of preprogrammed input/output and monitoring support, which will be discussed later):

1. Allocation of storage locations to data, instructions, and related information.
2. Conversion of original data to an input medium.
3. Availability of reference data, such as tables, files, or constant factors.
4. Requirements for accuracy, and methods of checking and auditing.
5. Ability to restart the system in case of unscheduled interruptions or error conditions.
6. Automatic monitoring of the system to ascertain that the required input and output devices are connected and available for operation.
7. Housekeeping procedures that preset indicators, switches, and registers; that type or display operator messages; and that check file labels.
8. Format of output data with provisions, if required, for later conversion to cards, printed reports, or displayed reports.
9. Availability of program routines that have been used and tested in other procedures and that may be used to advantage in the current procedure.
10. Conversion from the decimal number system to binary and from binary to decimal, plus whatever other conversions are necessary to the internal numbering system of the computer and codes used in input and output.
11. Editing of data with provision to record exceptions that cannot be processed.

Types of Programming Languages

The computer executes instructions that are presented to it in *machine language*. In System/370, machine language instructions are in binary in several formats. Even with the use of hexadecimal equivalents, writing (coding) a program in machine language is extremely difficult, costly, and time consuming. Furthermore, such a program is likely to contain numerous errors. Subsequent corrections may be practically impossible to make without introducing other errors. Instead of coding a program in machine language, the programmer writes the program in one of many languages that are more easily understood by the programmer and are more convenient means of specifying the problem. Since the expressions of the particular language cannot be executed directly by the computer, the expressions are translated into machine language by the computer using previously written language translator programs for the specific language. Depending upon the functions provided by these programs, they are referred to as assemblers, compilers, or interpreters.

Symbolic Language

Symbolic languages permit the programmer to write convenient equivalents of machine instructions using symbols (called mnemonics) to represent them. Symbolic instruction representations include the following: A for add, S for subtract, D for divide, ST for store, B for branch, and so on.

The first languages resulted in a one-for-one translation. That is, each instruction written in the programming language was translated into a single machine language instruction. For example;

```
A     REG1,184
```

where REG1 is one of the 16 general purpose registers in S/370, would produce the IBM S/370 machine language instruction:

```
01011010000100000000000010111000
```

Later, *macro instructions* were developed. That is, single programmer language instructions could be used to produce a whole series of machine instructions. This development greatly increased the power of programming languages.

The translation feature of the machine language program is perhaps the most important feature, but not the only one. The computer instructions needed to produce a given result must be executed in a given sequence. If an addition is to be performed, one of the values involved must be in an accumulator before the add instruction itself is executed. This is normally accomplished with an operation called L, "Load". After this operation is executed, the add operation may be executed. The two-instruction sequence is shown in both a machine language and a symbolic language in Figure 8-1.

Machine	Symbolic
01011000000100000000000110000000	L REG1, 384
01011010000100000000000110001000	A REG1, 392

Figure 8-1 Machine and Symbolic Codes

Each final machine language instruction must be assigned a particular location in main storage. For example, if the L instruction is to be assigned a location of 1000 (its precise location in main storage), and the add instruction is to immediately follow it, the location of the add instruction must be 1004 (since the load instruction is a four-byte instruction). Therefore, the location of each instruction must be known precisely. It is, in effect, the "name" of the instruction. If an additional instruction is to be inserted in a program of many instructions, every instruction from the point of insertion must have its previously assigned location changed. Since most programs undergo changing or updating, instruction location assignment becomes a tedious but necessary part of programming. The solution, of course, is to have the translating program do the actual assignment of instruction locations in addition to its translating function. The programmer need simply tell the translating program the desired location of the first instruction, and succeeding instructions are assigned sequentially ascending locations.

The advantage of expressing a problem in symbolic language over machine language should now be evident. This symbolism may be carried one step further by using symbolic data addresses as well as symbolic operation codes. The translating program can then be designed to translate and assign these symbols to actual main storage locations. Using the same instructions as before, assume that the two values to be added are expressed as values A and B. Of course, in both methods the values must have been previously placed in main storage, but the problem can now be stated as in Figure 8-2.

Instruction	
Operation Part	Address Part
L	REG1, A
A	REG1, B

Figure 8-2 Symbolic Operations and Addresses

If we now were to tell the translating/assigning program that we want the first instruction placed at main storage location 1000, the program shown in Figure 8-3 would result. (For better understanding, the program is expressed with symbolic operation codes and decimal addresses and locations, instead of machine language).

Instruction Location	Instruction	
	Operation Part	Address Part
1000	L	REG1, 4000
1004	A	REG1, 4004
.	.	.
.	.	.
4000	Value of A	
4004	Value of B	
.	.	.
.	.	.

Figure 8-3 Assigned Addresses and Locations

The translating/assigning program is called a *translator* or *assembler*. In normal operation, the translator is loaded into the computer system's main storage. Next, the instructions (prepared by the programmer to accomplish a particular job, as coded in his language) are entered into the computer. The computer than translates the programmer's instructions into machine language instructions, which use the order and logic set up by the programmer. The translated machine instructions are placed in main storage and form the actual program.

Language Translation

A programming language can be thought of in two parts: (1) the language itself, with associated rules of grammar, and (2) a machine language program (the translator), whose main function is to translate the language of the programmer into machine language.

The input to a translator is called the *source* program. This is written by the programmer in the language of the programming system (processor language). and states the requirements of the problem and the method of solution. Before the programmer writes his source program, he must have completely analyzed and defined the problem.

The output from the processor is the *object* program (or *object module*), the translation of the source program from the programmer's language to the language of the computer system on which the program will be used.

In some systems the object program may be executed. In others, the object program is not in executable format but must be processed by a program called the Linkage Editor which will produce a *load* program (or load module) which is an executable program. Subroutines (standard programs used with many problems), together with tables and other constant factors, may also be required within the computer to support the execution of the problem. Depending upon the processing mode, the input or data source may have to be made available before execution can begin.

After the problem is executed (solved) by the computer, the result (output) may be recorded on magnetic tape or disk for later printing or viewing on a display terminal. Results also may be printed directly from the computer.

A proven object program may be used time after time, with varying problem data, to produce periodic results (such as production type programs of payroll or inventory) or to produce different results to assist the designer seeking an optimum design (such as the best wing airfoil or the most efficient placement of steam pipes within a boiler), considering all variables for each application.

Machine-Oriented Programming Languages

In a machine-oriented programming language, the programmer uses symbolic codes or names to designate operations that the computer is to perform. Symbols are also used to designate the location in main storage of data used with the operations. The translator then assembles the symbolic codes and translates them into machine language instructions with actual machine storage locations.

The System/370 Assembler Language is an example of a machine-oriented language. The programmer uses coding sheets (Figure 8-4) to write each instruction that the computer must perform. Each line of coding will be punched into one card. The vertical columns on the form correspond to card columns.

IBM — IBM System/360 Assembler Coding Form

PROGRAM PROGA
PROGRAMMER J.J. JONES

```
PROGA    START Ø
BEGIN    BALR  11,Ø
         USING *,11
         L     2,DATA            LOAD REGISTER 2
         A     2,CON             ADD 1Ø
         SLA   2,1               THIS HAS EFFECT OF MULTIPLYING BY 2
         S     2,DATA+4          NOTE RELATIVE ADDRESSING
         ST    2,RESULT
         L     6,BIN1
         A     6,BIN2
         CVD   6,DEC             CONVERT TO DECIMAL
         EOJ                     END OF JOB
DATA     DC    F'25'
         DC    F'15'
CON      DC    F'1Ø'
RESULT   DS    F
BIN1     DC    F'12'
BIN2     DC    F'78'
DEC      DS    D
         END   BEGIN
```

Figure 8-4 A Program to Illustrate Assembler Language Concepts

Space is provided on the form for program identification and instructions to keypunch operators. None of this information is punched into a card.

The body of the form is composed of two fields: the statement field (columns 1 through 71), and the identification-sequence field (columns 73 through 80).

Statements consist of from one to four entries in the statement field. From left to right, they are: name (eight characters), operation (five characters), operand and/or comments (56 characters).

The name entry is a symbol created by the programmer to identify a statement. A name entry is optional. It must consist of eight characters or fewer, and be entered with the first character appearing in column 1. No blanks may appear in the symbol.

The operation entry is the mnemonic operation code specifying the machine operation or assembler functions desired. An operation entry is mandatory. If there is no name entry, the operation entry may be placed anywhere to the right of column 1; if there is a name entry, at least one blank column must separate name from operation. (The same "free-form" rules apply to the other entries - that is, at least one blank must separate them, and they must be in the order described above, but otherwise they are not restricted in location.) To be able to see and comprehend easily what there is in the way of a program, it is best to follow the field column designations on the coding sheet and start the operation in column 10.

Operand entries are the coding specifying and describing data to be acted upon by the instruction, by indicating such things as storage locations, storage area lengths, or types of data. Depending on the needs of the instruction, one or more operands may be written. The operands must be separated by commas, and no blanks may occur between the operands and the commas that separate them. Operands of machine instructions generally represent such things as storage locations, general registers, immediate data, or constant values. Operands of assembler instructions provide the information needed by the assembler program to perform the designated operation.

Comments entries are descriptive items of information about the program, usually something to remind the programmer of (or direct another programmer to) the purpose of the program step (or related sequence of steps). An entire line may be used for a comment by placing an asterisk in column 1. Extensive comments may be written by placing an asterisk at the beginning of each line.

There are many more conventions to the System/370 assembler language, but knowing this much about the makeup of the four types of entries should provide a basic understanding of the principles of coding.

Macro Instructions

The next step to increasing the effectiveness of a machine-oriented language involves enlarging the functions of the translator.

The macro language provides the programmer with a convenient way to write a definition that can be used to generate a desired sequence of assembler language statements. Most macros are supplied by IBM. However, any user may develop his own to satisfy a special requirement.

The definition is written only once, and a single statement, the macro instruction statement, is written each time a programmer wants to generate the desired sequence of assembler language statements. This facility simplifies the coding of programs, reduces the chance of committing programming errors, and ensures that standard sequences of assembler language statements are used to accomplish desired functions.

The Macro Instruction Statement

A macro instruction statement (usually referred to simply as macro instruction) is a source program statement that can produce a variable number of machine instructions. Macros, just like assembler language statements, are source program statements that are processed by the assembler.

The assembler generates a sequence of assembler language statements for each occurrence of the same macro instruction. The generated statements are then processed like any other assembler language statement.

The Macro Definition

Before a macro instruction can be assembled, a macro definition must be available to the assembler. A macro definition is a set of statements that provide the assembler with (1) the mnemonic operation code and the format of the macro instruction, and (2) the sequence of statements that the assembler generates when the macro instruction appears in the source program.

The Macro Library

The same macro definition may be made available to more than one source program by placing the macro definition in the macro library. The macro library is a collection of macro definitions that can be used by all the assembler language programs in an installation.

Varying the Generated Statements

Each time a macro instruction appears in the source program, it is replaced by the same sequence of assembler language statements, unless one or more conditional assembly instructions appear in the macro definition. Conditional assembly instructions are used to vary the number and format of the generated statement.

Each problem-oriented language has its own method of writing macros. Some seem to be almost like writing plain English sentences. Problem-oriented languages include COBOL, FORTRAN, PL/I, APL, RPG, and BASIC. They are called *problem-oriented* because they are tailored to the problem rather than to any particular machine.

COBOL

With COBOL (Common Business-Oriented Language), the translator still must produce a machine language program before a problem can be solved. However, the language written by a COBOL programmer bears little resemblance to machine language, and the problem programmer has little direct concern with the method by which the COBOL language program is translated into machine language.

A simple example will best illustrate the basic principles of the problem-oriented type of programming system. Assume we wish to increase the value of an item called INCOME by the value of an item called DIVIDENDS. The COBOL language allows us to specify the addition by writing the following sentence:.

```
ADD DIVIDENDS TO INCOME.
```

Before the COBOL translator can interpret this sentence, however, it must be given certain information. For example, the programmer will have to write the names DIVIDENDS and INCOME in a special part of the program, called the *data division*, where facts about the data represented by those names (such as maximum size, how the data is expressed, etc.) are stated.

When the translator encounters the sentence, it has access to certain information that will aid it in translating the sentence. In addition, it will be able to obtain certain information "built into" the translator itself. (Note, however, that the exact procedure will vary from machine to machine and that, in any case, the problem programmer is not directly concerned with the details.)

First, the translator examines the word ADD. It consults a special list of words that have clearly defined meanings in the COBOL language. This list is a part of the translator. If ADD is one of these words, the translator interprets it to mean that it must insert into the object program the machine instruction (or instructions) necessary to perform an addition.

The translator then examines the word DIVIDENDS. Since it can obtain certain information about DIVIDENDS, it will know where and how this information is to be stored in the computer, and it will insert into the object program the instructions needed to locate and obtain the data.

When the translator encouters the word TO, it again consults the special word list. In this case, it finds that TO directs it to the value of INCOME, which is to be increased as a result of the addition.

The translator must now examine the word INCOME. Again it has access to certain information about this word, and, as a result, it is able to place in the object program the instructions necessary in locating and using INCOME data.

We have indicated that the programmer placed a period (.) after the word INCOME, just as he or she would in terminating an English language sentence. The effect of the period on the COBOL translator is quite similar. It tells the translator that it has reached the last word to which the verb ADD applies.

The previously described steps are performed by the translator in creating the object program. They might not always be performed in exactly this way or in the same sequence, because machines vary and because each translator is adapted to a particular machine. However, regardless of the machine, the same COBOL language sentence produces machine instructions that cause the object program to add together the values DIVIDENDS and INCOME.

FORTRAN

The FORTRAN (Formula Translation) language is very similar in concept to COBOL. One of the main differences is in the language the programmer uses to express the source program. Where business English is used by COBOL, mathematical language is used with FORTRAN. The effect of the COBOL sentence

```
ADD DIVIDENDS TO INCOME.
```

could be achieved by the FORTRAN *statement*

```
INCOME = DIVIDENDS + INCOME
```

However, FORTRAN translators for some machines might insist that the words be abbreviated to something like:

```
INCO = DIV + INCO
```

This would depend on the individual machine FORTRAN translator. The statement, in effect, tells the translator to insert the necessary instructions into the object program to make the INCOME data location equal to the DIVIDEND data added to the present INCOME data. Note that the computer is not merely instructed to find the value of INCOME, but is also told where to put the result of the addition after it is performed. If the original INCOME field (in core storage) contained 10000, and the DIVIDEND field contained 15, the original INCOME field would be replaced by 10015 after the operation had been executed.

If this result is not desired, the programmer could change the statement to:

```
INCOME1 = DIVIDENDS + INCOME
```

With this change, a new INCOME1 data field would be generated in main storage, the result of the addition would be placed there, and the original INCOME field would remain unchanged.

PL/I

The System/370 is able to handle problems from both the scientific and the business fields with equal facility. Neither FORTRAN (science-oriented) nor COBOL (business-oriented) allow access to the full capability of such a system as System/370.

Also, with the development of more and more decision-making, forecasting, and teleprocessing uses for computers, the business programmer requires frequent changes to his or her programs and a wider scope of computations. The scientific programmer, on the other hand, is faced with the task of handling problems with masses of data, with a wider variety of input/output requirements. The differences between scientific and business programming are thus becoming less distinct. PL/I has been developed to meet the need for a broad-base language that may be used for both business and scientific applications. Some general characteristics and features of the language are summarized in the example in Figure 8-5.

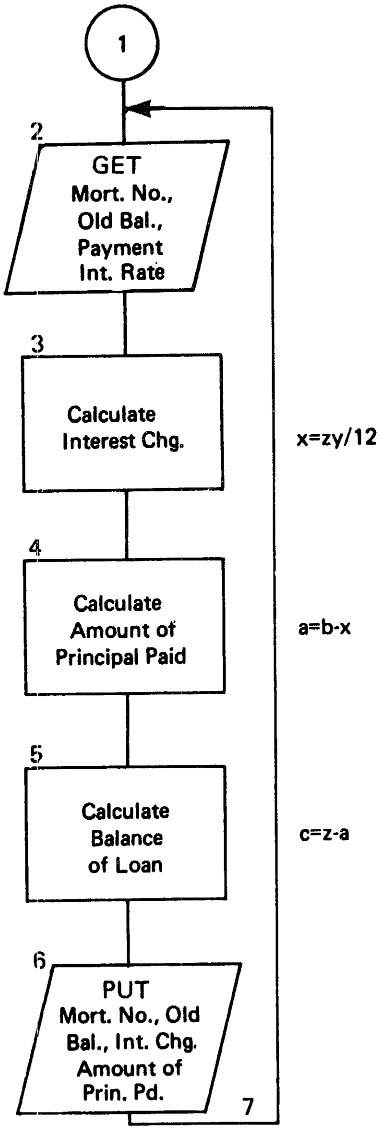
In PL/I as in FORTRAN, addition (or other arithmetic operations) can be specified by the standard oeprators +,-,/,*,and **. For example, INCOME = DIVIDENDS + INCOME; PL/I differs from FORTRAN only in the presence of the semicolon (;) which ends a statement, and in that this statement can be written in free form, as found convenient by the programmer. Addition can also be indicated by a statement which looks more like COBOL such as:

```
SUM=ADD(DIVIDENDS,INCOME,8,2);
```

This will put the result of adding DIVIDENDS to INCOME into an address named SUM which has an 8-position field, including two positions to the right of the decimal point.

ADD, DIVIDE, MULTIPLY, ATAN, COMPLEX, ERF, INDEX, MAX, MIN, ROUND, SUM, TAN, are a few of the built-in functions available to the PL/I programmer to operate on individual data items, on arrays (vectors or matrices), or on structures (tables) of data items. The keywords of PL/I are not reserved words, and program text may be written in free form. For example, the program (called PROCEDURE in PL/I) on lines 1-8 in Figure 8-5 is the same as that on lines 9-12.

Keywords such as PAGE, SKIP, LINE, COLUMN, PAGE SIZE, LINE SIZE, make it possible to specify the format of printed output in as much detail as desired.




```
BILLING:PROCEDURE OPTIONS (MAIN);
   NEXTCARD:GET DATA (MORNO,OBAL,PAYM,RATE);
          CHARGE= OBAL*RATE/12;
          PRINPAID= PAYM-CHARGE;
          BALANCE = OBAL-PRINPAID;
       PUT DATA (MORNO,OBAL,CHARGE,PRINPAID,BALANCE);
   GO TO NEXTCARD;
END BILLING;
```

```
BILLING:PROCEDURE OPTIONS (MAIN);NEXTCARD:GET DATA (MORNO,OBAL, PAYM,
RATE);CHARGE=OBAL*RATE/12;PRINPAID=PAYM-CHARGE;BALANCE=
OBAL-PRINPAID;PUT DATA (MORNO,OBAL,CHARGE,PRINPAID,
BALANCE) ;GO TO NEXTCARD;END BILLING;
```

```
BOB:PROCEDURE OPTIONS (MAIN); /*THIS IS A COMMENT*/
   D:GET DATA (N,Z,B,Y); /*COMMENTS MAY BE PUT*/
        X=Z*Y/12; /*WHEREVER BLANKS ARE ALLOWED*/
        A=B-X;
        C=Z-A;
     PUT DATA (N,Z,X,A,C);
   GO TO D;
END BOB;

BOB:PROCEDURE;D:GET DATA (N,Z,B,Y);X=Z*Y/12;A=B-X;
C=Z-A;PUT DATA (N,Z,X,A,C);GO TO D;END BOB;
```

Figure 8-5 Flowchart and Coding in PL/I of Part of Mortgage Processing

APL APL (A Programming Language) is a highly sophisticated problem-oriented language, having a large number of symbolic built-in functions (primitive functions) and operators that render it possible to write complicated programs within a simple syntax in a concise form. (In other languages, functions such as matrix inversion and output formatting are library routines, but in APL they are part of the primitive function set).

APL uses a special character set as shown in Figure 8-6. (Several APL characters such as ⍞ require overstriking). In System/370, most APL characters are assigned otherwise unused EBCDIC values.

There are several methods of executing programs written in a high-level language. The most widely used is to compile the programs into machine language. Another is to translate the programs into some intermediate form and then execute that form interpretively. (The intermediate form is not in machine language or object code. During execution of the program, each statement is interpreted, translated into machine instructions, and executed prior to proceeding to the next statement).

In many high-level languages, a compromise has been made between making the language easy to use and enabling it to be compiled. In APL, this compromise has not been made. APL is designed to be both powerful and easy to use: consequently APL programs are executed by interpreters, as it is difficult and perhaps impossible to compile APL into conventional machine language instructions.

In all APL systems, processing takes place in a piece of memory called a workspace, which contains APL programs, data, and control information, as well as some unused space. The user may further save those work spaces in a disk library and recall them at will, in this way giving to each user a flexible and easy procedure for building and using his/her application packages. Each user can have a library of many workspaces, but only one can be active at a time. When APL execution begins, the user is provided with a clear workspace - one containing some control information but no programs or data. The user can enter programs into the clear workspace, or replace it by loading a workspace from the user's library. When the system is ready for input, the user can type an APL command, define or edit an APL function, or type an APL statement. Statements, which are executed immediately, may initiate extensive calculations by calling previously defined APL functions.

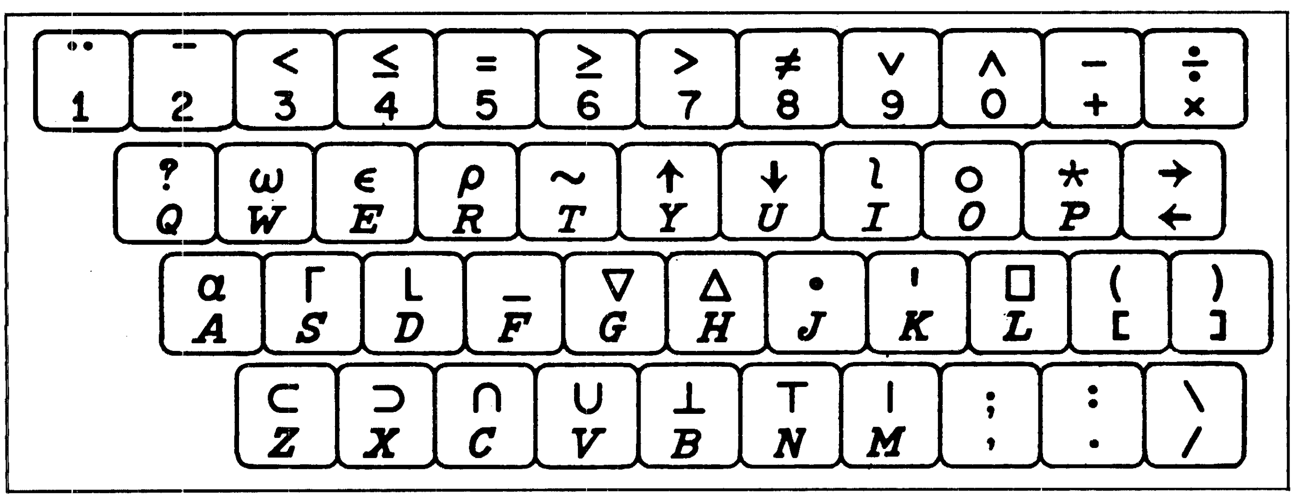


Figure 8-6 Typical APL Keyboard

The APL program shown in Figure 8-7 was entered interactively at a terminal. The examples shown were entered immediately thereafter.

```
          ∇AVERAGE
     [1]  'ENTER NUMBERS'
     [2]  X←⎕
     [3]  'THE AVERAGE IS ';+/X÷ρX
     [4]  ∇

          AVERAGE
     ENTER NUMBERS
     ⎕:
          2 4 6 8
     THE AVERAGE IS 5

          AVERAGE
     ENTER NUMBERS
     ⎕:
          1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
     THE AVERAGE IS 8

          AVERAGE
     ENTER NUMBERS
     ⎕:
          ι15
     THE AVERAGE IS 8
```

Figure 8-7 A Simple APL Program and Examples Using the Program

Report Program Generator

The Report Program Generator (RPG) language is a problem-oriented language. It is a very simple way of adapting an application to a computer. The file definition and input/output control considerations normally required of the programmer by other programming languages are reduced to the filling out of simplified control forms.

The input format specifications form is used to:

1. Specify the file or files to be read into the system.
2. Identify the different types of records contained in each file.
3. Describe the location of the data fields in each record.

The output format specifications form is used to:

1. Specify the kind of output files to be produced, printed reports, summary cards, etc.
2. Specify the location of the data fields of the reports or records.
3. Specify any headings or totals for printed reports.
4. Specify any editing or zero suppression needed for the printed reports or records.

The file description form provides additional information regarding the input/output files, such as specifying the input/output units used by the program and other features associated with input/output control.

The calculations form is the heart of the logic of the program. For calculation, RPG is a three-address language.

We can perform a mathematical operation (like add) on two addresses and store the result in a third address, all in the same statement.

The calculations form is divided into three categories, as follows:

1. Time to do the calculations.
2. Kind of calculations to be performed.
3. Tests to be made on the results.

Depending upon which programming system is used, there are 22-27 different operation codes. They include add, subtract, multiply, divide, table lookup, compare, move, branch, test zone, exit to a subroutine, etc.

Program Checkout

After successful translation of a source program, and then linkage editing in some systems, the next step in program preparation is to check the resultant machine language program by running it with test data. This is done to make sure that the program does not have logical errors and that it is capable of producing a right answer when using test data. Two results are possible. The first - and, hopefully, the only - result is that the problem (for which the program was written) can now be executed with real data. The second result - the test run does not function properly - may occur because of many things. The most frequent cause is that the source program has been improperly or incompletely stated.

Mistakes by the programmer are more difficult to avoid than might be expected. It is, in fact, a rare program that works correctly the first time it is tried with test data. In most cases, many test runs must be made before most or, hopefully, all mistakes are found and corrected. The translator itself finds most of the obvious mistakes during the translation run. Such things as calling for a storage location by a name when that name has not been defined, attempting to perform fixed-point arithmetic on floating-point data (or the reverse), lack of defined alternative paths on testing operations, and card-punching errors of all kinds are detected and noted during the translation run.

Computer mistakes are rare and usually obvious. Built-in detection circuits will normally reflect the kind of mistake the computer has made by turning on an indicator and stopping the computer. Detection and classification of the mistakes a programmer can make are, however, many times more complex.

Testing Techniques

As previously stated, a computer program may be expressed in machine, symbolic, or one of the problem-oriented languages, such as PL/I, FORTRAN, or COBOL.

Many techniques exist to assist the programmer during the checkout phase of his work. Each has its own advantages and disadvantages. The one to be used for a particular problem will depend upon the programmer's thoughts as to what area of the program is in error and how extensive the error is. Two common techniques are storage printout and tracing.

Storage Printout

This type of utility program (routine) is helpful because practically the entire contents of storage, plus the contents of working computer registers and the condition of indicators and switches, may be presented in printed form. Normally, the register contents and the condition of indicators and switches are printed first. The contents of storage are then printed. Each line of printing representing storage begins with the starting location of that line expressed in hexadecimal and/or mnemonic format. The print (dump) routine sometimes has provisions for dumping one or more selected blocks of storage instead of all of it.

Tracing

If visually checking a storage printout fails to reveal the program difficulty, a technique called tracing may be used. The trace technique usually involves an interpretive routine and, therefore, executes a number of instructions for each program instruction being traced. The printout received while tracing normally includes the location of the instruction being executed, the instruction being executed, and the contents of the working registers after the instruction has been executed. The printing of each instruction execution in a program would result in excessive machine time and should be used only when all other methods fail to reveal the program trouble.

The basic tracing technique may be revised so that only the contents of selected storage locations are printed when program execution reaches a specified point in the program. With this variation, a *snapshot* is obtained of a particular part of the program under particular conditions. For example, the trace and resultant printout may be specified to occur only when the program executes a transfer instruction. A whole series of snapshots then result showing the execution path through the program. Only the instructions that altered the normal execution path are recorded to show the exception paths the program has executed.

Improved Programming Technologies

Until several years ago, improvements in program development techniques did not keep pace with the significant improvements in hardware technology. Increased hardware speed and capacity, configuration flexibility, and programming system capability led to increased complexity of the application systems developed. The volume of completed programs used for production purposes in user installations increased along with the continued need to modify (and often correct) such programs. (This is usually referred to as *program maintenance*). With larger portions of the data processing department budget being devoted to programming and with "maintenance" requiring an increasing fraction of the programming budget, considerable efforts have been made to increase programmer productivity resulting in the development of the Improved Programming Technologies described below.

Structured Programming

Traditionally, individual programmers have applied their own sets of rules to the construction of the logic of their programs. Starting with this logic structure, as one encounters additional combinations of conditions to be met, programmers add them as afterthoughts rather than revising the logic of the program.

The resultant code tends to contain a large number of GO TO statements, and its logic may not be easy to follow. During subsequent unit and integration testing, disintegration of the programmer's original structure occurs as new constraints and conditions are imposed upon it. This leads to more GO TO statements, more labels, and a final program whose logic may be completely obscured. Reading and testing such programs is difficult The degree of confidence in their quality or correctness tends to be low. And, of course, such programs tend to be difficult to maintain and modify.

Research by computer scientists and mathematicians indicates that an alternative method of programming, known as *structured programming*, can help solve these problems. This technique involves coding programs using a limited number of control logic structures to form highly structured units of code that are more readable and, therefore, more easily tested, maintained, and modified.

Structured Programming Theory

Structured programming is based on a mathematically proven structure theorem which states that any program can be written using only the three control logic structures illustrated in Figure 8-8.

- Sequence of two or more operations (MOVE, ADD....)
- Conditional branch to one of two operations and return (the IF p THEN C ELSE D of Figure 8-8)
- Repetition of an operation while a condition is true (the DO E WHILE q of Figure 8-8)

Any program may be developed by the appropriate iteration and nesting of these three basic structures. Each of the three structures has only one entry and one exit. A program consisting solely of these structures is a *proper program*; that is, a program with one entry and one exit. It always proceeds from the beginning to the end without arbitrary branching. The logic is easier to follow, permitting functions to be isolated, understood, and tested. Proving the logical correctness of structured code becomes feasible.

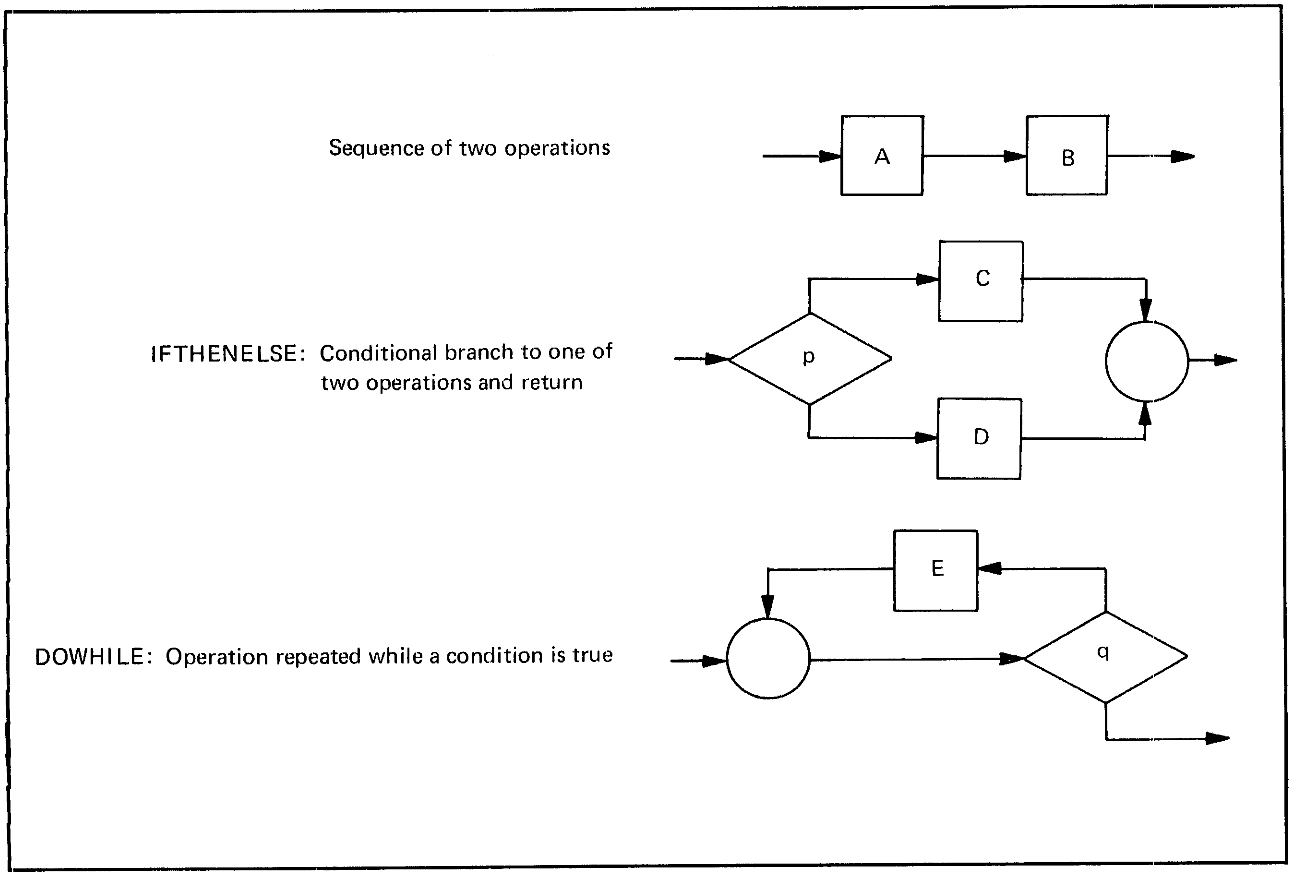


Figure 8-8 The Three Elemental Logic Structures of Structured Programming

Extensions to the three basic logic structures are permitted only when they retain the one-entry, one-exit property. An example of such an extension is the DOUNTIL structure (Figure 8-9) that provides for the execution of the function F until a condition is true.

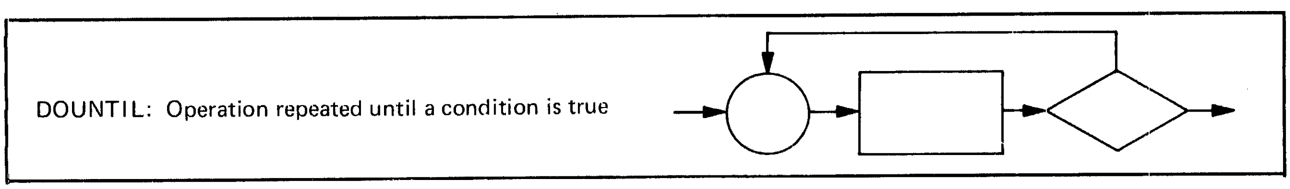


Figure 8-9 The DOUNTIL Structure

Top-Down Program Development

Traditional software development has often been approached as a bottom-up procedure, where the lowest level units are coded first, unit tested (the testing of each unit such as a module individually), and made ready for integration. Data definitions and interfaces between units tend to be simultaneously defined by each of the programmers, including those working on the lowest levels of code, and are often inconsistent. During integration, definitions and interface problems are recognized. Integration is delayed while the data definitions and interfaces are correctly defined and the units are reworked and unit tested to accommodate the changes. It is often difficult to isolate a problem during the traditional integration cycle because of the difficulty in identifying which of the many units combined during integration is the source of the problem. The resultant program, because of last-minute redesign, coding, and testing, is often lacking in quality. Superfluous code in the form of driver programs is needed to perform the unit testing and lower levels of integration testing. Management control is often ineffective during much of the traditional development cycle because there may be no coherent, visible product until final integration.

Top-down program development is designed to reduce these problems by reordering the sequence in which units of code are written. A program unit is coded only after the unit that invokes it has been coded and tested. Therefore, top-down program development both assumes and is patterned after a program structure of hierarchical form as illustrated in Figure 8-10.

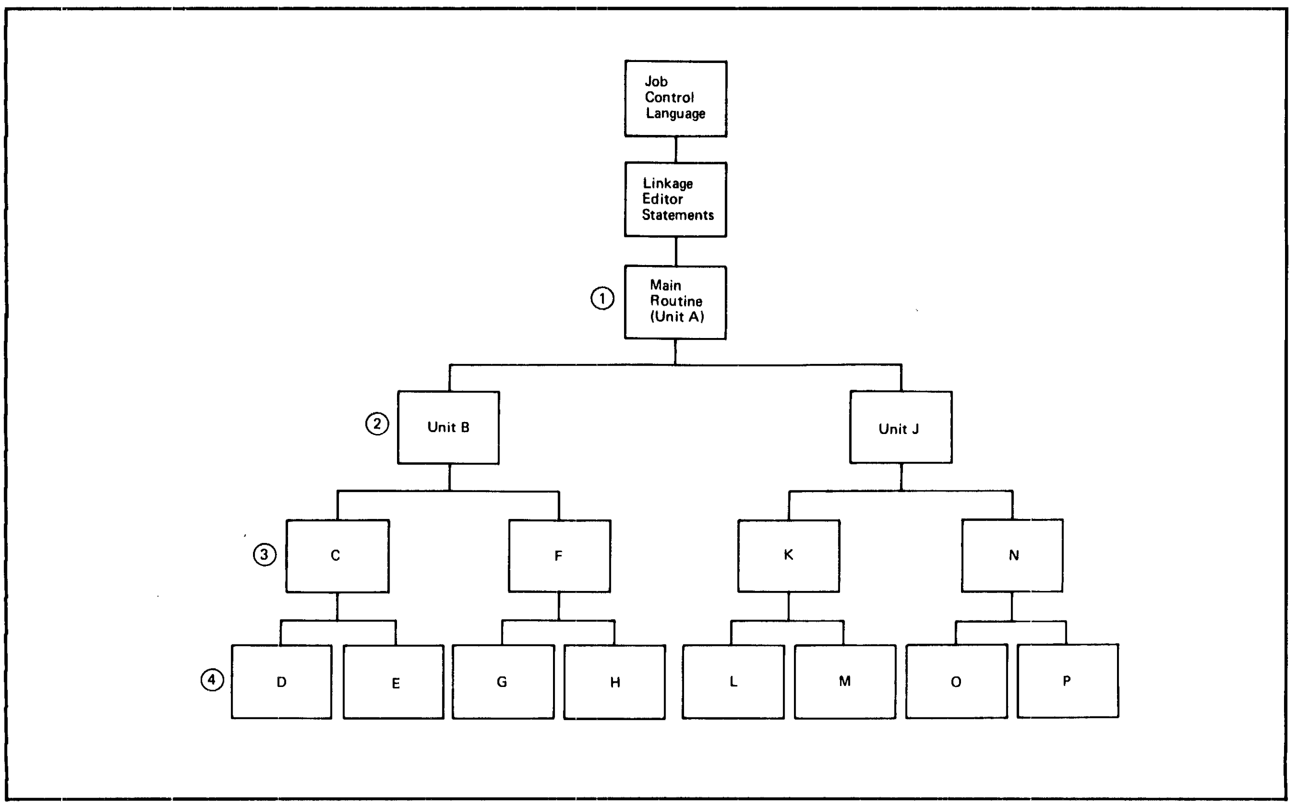


Figure 8-10 A Hierarchical Program Structure

Chief Programmer Teams

The chief programmer team is an organizational technique designed to coordinate the efforts of programming specialists while retaining the responsiveness and integrity of design expected of a skilled individual. A chief programmer team is a small group under the direction of a senior level programmer called the chief programmer. The team normally consists of three to five programmers, and other specialists as appropriate. A chief programmer team represents an opportunity to improve both the manageability and the productivity of programming by moving the program development process from private art to public practice through organizational techniques. These techniques include:

- Restructuring the work of program development into specialized jobs
- Recognizing the need for technical expertise in the leadership of the team effort
- Recognizing the need for leadership in the training and career development of the team's personnel.
- Defining relationships among specialists
- Using disciplines to help team members communicate effectively with one another.
- Working effectively with a developing, always visible, project

The chief programmer is responsible for program design of the system, and is vested with complete technical responsiblity for the project. The chief writes the mainline routines, the critical code, and the operating system interfaces. The chief defines the modules to be coded by other team members and is responsible for specifying the interfaces between modules and for the data definitions. The chief reviews code written by other team members, oversees the testing and integration of all code, informs management of the project status, and arranges for additional team members when necessary.

HIPO

Application function is often documented in prose toward the end of a project. This creates a two-fold problem: (1) descriptions of functions may be incomplete because of the difficulty of extracting functions from the bit manipulation performed by the programs, and (2) prose descriptions of functions are often voluminous, ambiguous, and difficult to relate systematically to the program modules performing the function. Hierarchy plus Input-Process-Output (HIPO) helps solve these problems by providing the programmer with a graphic technique for documenting function from the beginning, before programming starts and while the function is clear in the designer's mind. HIPO also reduces the amount and ambiguity of the prose required to document function and provides a systematic means of identifying all the functions to be performed and the modules that perform them.

In describing the functions to be performed, HIPO diagrams progress from a generalized functional description to greater levels of detail. The functions themselves are described in terms of the process that occurs, coupled with the necessary input and resultant output. Such diagrams are a logical means of representing functions identified in a top-down design effort. Specifically, a typical HIPO package consists of a hierarchy chart, or diagrammatic visual table of contents, one or more overview diagrams, and detail diagrams.

The visual table of contents (see Figure 8-11) identifies all the overview and detail diagrams in the package, shows their hierarchical relationships, and permits the reader to locate quickly a particular level of information or a specific diagram.

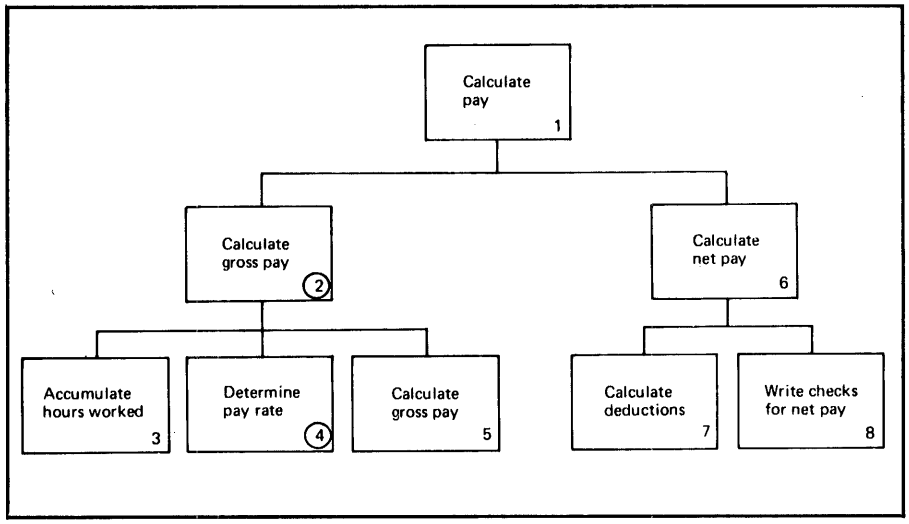


Figure 8-11 Sample HIPO Visual Table of Contents

The overview and detail diagrams describe functions graphically. Each diagram consists of three parts: (1) input - the input to the function (files, records, fields, control blocks, etc.), (2) process - the process steps that support the function being described, and (3) output - the output of the process (files, records, control blocks, etc.).

The overview diagram describes one or more functions in general; the detail diagrams expand on this description. Figures 8-12 and 8-13 illustrate an overview diagram and a detail diagram, respectively.

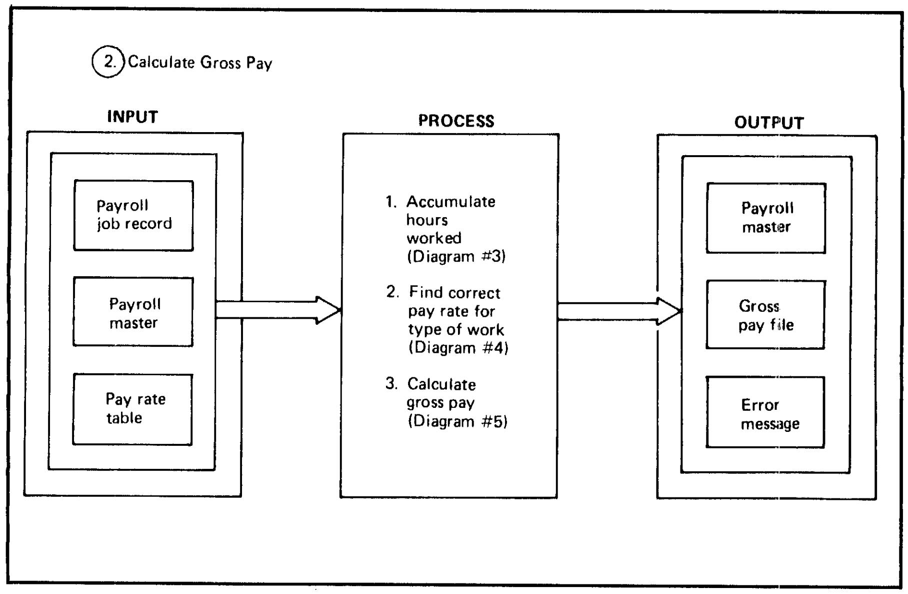
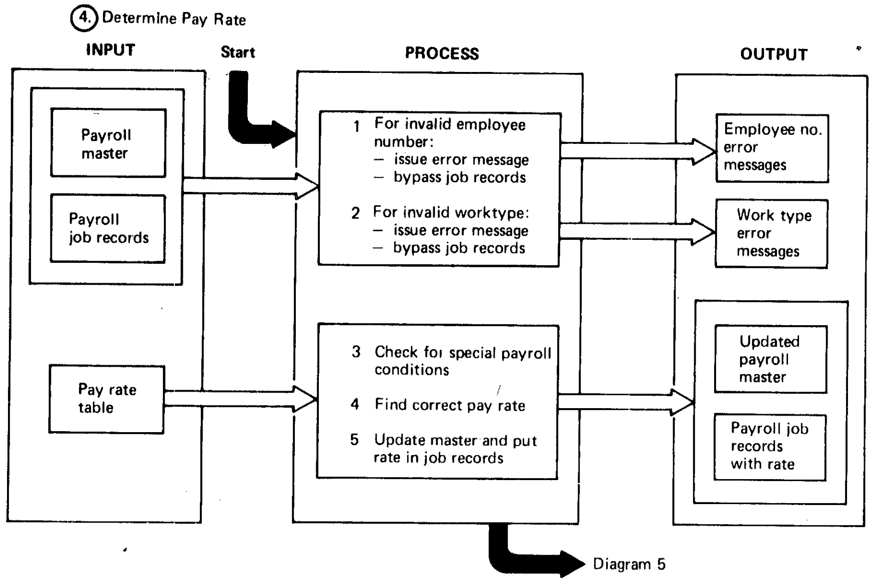


Figure 8-12 Sample HIPO Overview Diagram

In addition to the input, process, and output sections, each detail diagram may include an extended description section, keyed by number to the process section. This section can be used to describe each numbered process in more detail and point to the program modules in which the process is implemented and to the modules calling the process (see Figure 8-13.) An extended description section can also be used in an overview diagram to further describe each process and point to detail diagrams in which the numbered processes are further expanded.



	Extended Description	Routine	Label
1.	The program checks for valid employee number. If valid, job records for that number are bypassed and an error message is printed.	IODNA	DETR
2.	A check is made for correct type of work. If invalid, bypass job records & print error message.		
3.	Special conditions such as overtime, shift pay, vacation pay, or holiday pay are checked to help determine correct rate.		
4.	The master record, job records, & pay rate table are all referenced to determine correct pay rate.		
5.	When all conditions are checked, payroll job records are rewritten with proper rate, payroll master updated.		

Figure 8-13 Sample HIPO Detail Diagram

Structured Walk-Throughs

Sometimes program errors result from the lack of experience of the designer or programmer (developer). Probably, more often, they result from the lack of perspective of the developer, who has been too close to a program for too long and finds it difficult to see any errors in it. In fairness to the programmer, what is needed is an objective check on a program's overall logic and completeness by someone who has not been immersed in its details. It is important to detect and remove errors as early in the cycle as possible, when the cost of correcting them is lowest and their impact is smallest. The structured walk-through is designed to detect and remove errors as early as possible in the cycle in a problem solving and non-fault-finding atmosphere in which everyone, and especially the developer, is eager to find any errors in the work product being reviewed.

A structured walk-through is a review of a developer's work (program design, code, documentation, etc.) by fellow project members invited by the developer. It is conducted by the developer without the manager's presence. Such reviews help the developer find errors earlier in the development cycle. In addition, they give reviewers an opportunity to learn new approaches and techniques. Structured walk-throughs also help the participants communicate the characteristics of their work to one another.

Inspections

Inspections are similar to structured walk-throughs but provide additional advantages. In brief, inspections are a more formal, disciplined form of review through which program developers can improve their development skills and become more confident of their ability to meet schedule commitments.

Checklists are used by developers during the inspection meetings to assure that as many errors as possible are discovered prior to test runs. Exit criteria are established which are used to judge whether a function or segment is of high enough quality for work to commence on the next phase of its development. Data is accumulated regarding the amount of time consumed in the inspection process and the value of early discovery of errors, so that the value of inspections can be quantified. Moderators, who are responsible for conducting the inspections efficiently, are specially trained and do not moderate inspections of materials developed by themselves or their own development teams.

Section 9:

Programming Systems

In the early computing systems, the relatively slow I/O operations and the much faster data processing operations of the CPU could not be performed at the same time. Therefore, the CPU was idle much of the time waiting for the completion of data transfers between I/O devices and main storage. To reduce this idle time, computing systems were soon developed that could perform input, output and data processing operations all at the same time. This represented a significant improvement in the performance of computing systems as described above under the heading of Data Buffering. However, to take advantage of the improvement, the programmer had to make sure that the I/O operations were synchronized with the processing of data; otherwise, the CPU might attempt to process input data before it arrived in main storage or destroy output data before it was transferred to an output device. Therefore, input/output control systems were developed to automatically synchronize I/O operations with data processing.

Input/Output Control Systems

An input/output control system (IOCS) consisted of an interrelated group of programs that was loaded into main storage along with the processing programs. Using such a system, a programmer merely had to issue a "READ" instruction to obtain the next block of data from an input device or a "WRITE" instruction to send a block of data to an output device. The input/output control system picked up and interpreted the instruction and then initiated and controlled the necessary transfer of data to or from main storage. In the meantime, the CPU could continue processing data.

If each block of input data contained more than one record the programmer merely issued a "GET" instruction to get the next record in sequence. The input/output control system automatically controlled the transfer and storage of data blocks and parcelled out records one at a time from the blocks as they were requested by the processing program. Similarly, to transfer an output record, the programmer merely issued a "PUT" instruction. The input/output control system then picked up and consolidated records into a block before transferring the block to an output device.

Input/output control systems assisted programmers in other significant ways. For example, if an error was detected during the input/output operation, the system automatically retried the operation and attempted to recover from the error condition. It also checked labels at the beginning of magnetic tape reels to ensure, among other things, that the correct reel was mounted on the right tape unit. Input/output control systems, as a whole, represented an important step in the evolution of operating systems.

Although input/output control systems helped improve the efficiency of computer usage, continued increases in the volume of data processing jobs and significant increases in computer processing speeds helped focus attention on several other areas requiring improvement. One of these areas involved the wasted time between computer jobs. Often, the computer would be idle while the instructions for processing the next job would be read by the operator, while the operator would mount several tape reels, while the operator would place punched cards in the card reader, and so on. For installations having many small jobs, this *set-up* time was especially wasteful.

Another area of concern related to the *mix* of the jobs themselves. Certain programs may require an enormous number of calculations with very little input and/or output. Such programs tend to make maximum utilization of the CPU while the I/O resources are hardly used. These programs have often been called computer-bound programs or number crunchers. On the other hand, other programs may require an enormous amount of I/O activity with very little computation. Such programs tend to underutilize the CPU allowing it to remain idle (in a *wait* state) most of the time. Such programs have often been called I/O-bound or high I/O programs. It seemed quite reasonable that if some way could be found to interweave the low I/O programs with the high I/O programs, significantly better computer utilization would result. Conceivably, it should be possible for more than two programs to reside in the computer at the same time although the CPU would only be processing one at a time for short intervals depending upon I/O needs. However, in order to accomplish this even with two programs, some means had to exist to supervise the process including preventing one program from destroying another. Another frequently underutilized (but comparatively expensive) resource was the main memory of the computer since many programs would require relatively small portions of memory. (Frequently, however, in the same installation, the main memory size would serve as a constraint for other applications).

Various hardware and software solutions have evolved to help solve these problems. Such items as the operating systems, multiprogramming, and virtual storage have enabled users to make more efficient utilization of their computer systems.

Operating Systems

A program, or set of programs, that directs a computing system to perform such operations as managing computer resources, scheduling and supervising work and operating and controlling mechanical devices is called an *operating system*. An operating system is really a system application of a computing system in the form of organized collections of programs and data. Like other system applications it is designed to handle complex activities, but it differs in the kind of activity it supports. Most other system applications support specialized activities outside of the data processing installation, such as banking, process control, or missile design. An operating system is designed to support the activities of the data processing installation itself. In short, an operating system is an application of a computing system in the form of program and data resources, that is specifically designed for use in creating and controlling the performance of other applications. Its prime objective is to improve the performance of a data processing system and increase facility -- the ease with which the system can be used.

Brief descriptions of the various operating systems supplied by IBM for System/370 appear later in this section.

Virtual Storage

The rapid growth in the number and types of data processing applications has led to an increasing demand for freedom in designing applications without being functionally constrained by the physical characteristics -- system architecture, I/O device types, and CPU space -- of a particular computer system.

While System/360 operating systems already allowed the programmer a certain degree of device independence, the need for making programs fit into the available real storage still existed. Since available real storage was usually divided into at least three partitions (or regions), this often required *overlay* techniques to make the program fit into a partition or region. Structuring these overlays added to the complexity of solving a problem. With the increased use of high-level languages, multiprogramming, expanded system control programs, and applications that require relatively larger amounts of real storage (teleprocessing, data base, etc.), the need for more real storage space and a more dynamic use of it is still growing.

To meet this need, the System 370 models provide significantly more real storage capacity than the comparable System/360 models. The availability of more storage though, did not relieve all the constraints associated with this storage. It did not eliminate the waste of storage resources through, for example, dormant code, as might be the case with an inactive or low activity teleprocessing network, or through storage fragmentation as a result of programs running in bigger partitions than required for their execution. The system had no means of dynamically utilizing the fragments of free storage space. Consider also the following situations:

1. An application is designed to operate in a 50K real storage area, which is adequate to handle current processing needs and provides room for some expansion. Some time after the application is installed, maintenance changes and the addition of new function cause one of the programs in the application to require 51K and another to require 52K. Installation of the next real storage increment cannot be justified on the basis of these two programs, so time must be spent restructuring the programs to fit within 50K.

2. An existing application has programs with an overlay structure. The volume of transactions processed by these programs has doubled. Additional processor storage is installed. However, the overlay programs cannot automatically use the additional storage. Therefore, reworking of the overlay structure programs is required to make them non-overlay and, thereby, achieve the better performance desired.

3. A simple, low-volume, terminal-oriented inquiry program that will operate for three hours a day is to be installed. If the program is written without any overlay structure, it will require 60K of real storage to handle all the various types of inquiries. However, because of a low inquiry rate, only 8K to 12K of the total program is active at any given time. The inquiry program is designed to operate in 12K with a dynamic overlay structure in order to justify its operational cost.

4. A series of new applications are to be installed that require additional compute speed and twice the amount of real storage available on the existing system. The new application programs have been designed and are being tested on the currently installed system until the new one is delivered. However, because many of the new application programs have storage design points that are larger than those of existing applications, testing has to be limited to those times when the required amount of real storage can be made available. Although another smaller scale model is also installed that has time available for program testing, it

cannot be used because it does not have the amount of real storage required by the new application programs.

5. A large terminal-oriented application is to be operative during one entire shift. During times of peak activity, four times more real storage is required than during low activity periods. Peak activity is experienced about 20 percent of the time and low activity about 40 percent. The rest of the time activity ranges from low to peak. Allocation of the peak activity storage requirement for the entire shift cannot be justified and a smaller design point is chosen. As a result, a dynamic program structure must be used, certain desired functions are not included in the program, and response during peak and near peak activity periods is affected.

In the situations described, real storage is the constraining factor. However, even if more real storage were added to a system as needed, the system could not automatically make use of it. Applications would still have to be redesigned, and the waste of storage through fragmentation and dormant code would still exist.

To assist in solving these problems, the System/370 virtual storage concept offers a means of dynamically and automatically using real storage resources, storage fragments as well as storage space added to the system at later times. With virtual storage support, programs are no longer restricted to the address space available to their partition in real storage. They may exceed this limit to a certain extent and still get the necessary real storage as it is needed for the execution of each section of the program.

The time required for any program to execute under any operating system has always been and still is dependent on such factors as the mix of programs executing concurrently, their relative priorities, system and application file placement, and in some cases on the particular data being processed. Under DOS/VS, for example, program performance is also highly dependent on such factors as the amount of real storage overcommitment, the storage reference patterns of the program, and the speed of the paging device. The performance of each program must be evaluated in the light of at least these factors. For on-line or real time systems with specific performance or response requirements, particular attention must be given to assuring that adequate resources (real storage, CPU time, channels, disk arms, etc). are available. In some cases it may be necessary to test the program using the specific user workload and configuration to verify what system resources are necessary to give adequate performance.

Further details about virtual storage appear later in this section.

The IBM-supplied programming systems that support the System/370 include:

- DOS/VS (Disk Operating System/Virtual Storage)
- OS/VS1 (Operating System/Virtual Storage 1)
- OS/VS2 (Operating System/Virtual Storage 2)
  SVS (Single Virtual Storage) and MVS (Multiple Virtual Storage)
- VM/370 (Virtual Machine Facility/370)

In general, DOS/VS supports the smaller and middle sized models (up to 158). OS/VS1 supports Models 135 to 158. VM/370 supports Model 135 and up. OS/VS2 supports Model 145 and up.

Since these programming systems have evolved from prior systems, variations exist in the manner in which similar functions have been implemented. Terminology variations also exist. In addition, available functions and facilities are somewhat different. Brief descriptions of some of these programming systems appear below.

DOS/VS

DOS/VS is a disk-resident comprehensive collection of programs designed to make full use of the resources of the various models of System/370 that it supports.

Component Programs

The component programs that make up DOS/VS may be divided into:

1. Control programs
2. Processing programs
3. Data management routines

These programs and routines combine many data processing functions into a programming package that is designed to make maximum use of a hardware system and to relieve programmers and operators of a great deal of manual work.

For execution, the components of DOS/VS are stored on-line (that is, immediately and directly accessible whenever required) in areas on magnetic disk, called *libraries*. This allows fast loading of any program or routine into storage whenever its function is needed.

Control Programs

As their name implies, the control programs control the execution of all processing programs, IBM-supplied as well as user-written. DOS/VS control programs comprise the initial program loader, the supervisor, and the job control program.

The *inital program loader* is used to start operation with the system. It loads the supervisor into storage.

The *supervisor* controls overall system operations and provides general functions required by the job control program and all processing programs. It resides in the lowest area of storage, called the supervisor area, throughout system operation.

The *job control program* is loaded by the supervisor to initiate the execution of each new program and to establish which system facilities are to be invoked while that program is running.

Processing Programs

Processing programs are classified as all programs whose execution is initiated by the job control program and controlled by the supervisor. Processing programs can be divided into the three categories: language translators, service programs, and application programs.

Language translators translate source programs written in the various programming languages supported by DOS/VS into machine (or object) language.

Service programs assist with the use of the computing system and in the successful execution of problem programs, without contributing directly to the control of the system or the production of results. Among the most important service programs are the linkage editor, which converts the output of language translators into executable object programs; the librarian, which performs service and maintenance functions for the libraries that are on disk and contain the programs and routines that make up DOS/VS as well as user-written programs and control information; POWER/VS, which provides spooling support for unit record input/output (see description of spooling functions of JES under OS/VS1 appearing later in this section) and also offers the remote job entry (RJE) capability (a teleprocessing feature that allows jobs to be entered simultaneously into the system from up to twenty-five remote terminals); and emulators. (An *emulator* is a combination of programming and special machine features that permits a computing system to execute programs written for another kind of system. For programs that were written to run on 1401/1440/1460, 1410/7010, or System/360 Model 20 computers, combinations of machine features and system programming are provided to allow these programs to run under DOS/VS. Integrated emulation offered with DOS/VS allows the user to emulate a number of non-DOS/VS programs on the System/370 concurrently with the execution of normal DOS/VS programs.)

Application programs include user-written and, in some cases, IBM-supplied commercial and scientific programs.

Data Management

A third important class of components of DOS/VS are its data management routines. These are available for inclusion in problem programs to relieve the programmer of the detailed programming associated with the transfer of data between auxiliary storage and programs as well as the organization of such data. The services of data management are invoked by all system and user-written programs whenever they require the execution of input or output operations.

Data Organization refers to the techniques used in placing records on an auxiliary storage device such as cards, magnetic tape or disk. It involves such considerations as:

- The choice of *storage media* best suited to the processing requirements of the data.
- The *sequence* of the individual records in the file. For example, the file could be sorted on one control field or on several, in a prescribed hierarchy; the file could be in ascending or descending order.
- The *length of records*. Records in a file can be of a fixed length or of variable length.
- The *blocking factor* for the file. This determines how many logical records constitute a physical record, and is an important factor in storage media utilization and in processing efficiency.

- The use of *indexes* with a file on a direct access device to provide an efficient means of randomly selecting specific records.
- The use of programmed *addressing techniques* to determine where a record is stored on a direct access device and the location from which it can subsequently be retrieved for processing.
- The type of *data additions* to an already existing file. It is of major importance whether many or few data additions and updates are made to a file, whether additions and updates are made in sequential or random order, whether a file is accessed by more than one program in different partitions, and whether it is a read-only file.

The routines which assist the programmer in transferring records in a particular data organization format between storage and an I/O device are referred to as *access methods*. It is important to understand the relationship between data organization and access methods. Broadly speaking, how data is organized and the type of device that it is stored on largely determine the access methods that can subsequently be used to retrieve it. DOS/VS provides several methods of data organization. For each of these there is an access method which allows one or more techniques of file creation and retrieval. A brief description of several access methods appears later in this section.

Multiprogramming

DOS/VS allows the user to divide the problem-program area (which consists of all of real storage not used by the supervisor) into as many as five areas, called *partitions*.

Each partition can contain a separate program, which allows concurrent execution of multiple programs. This is called multiprogramming. Each program is logically independent, but it takes turns with the other programs in using the CPU facilities, thus reducing the time that the multiprogramming system is in the unproductive *wait state*.

The user specifies the number and size of partitions at *system generation* time. (System generation is the creation of an installation-tailored operating system based on the general system distributed by IBM. The general system is capable of immediate operation. However, in order to ensure optimum efficiency, most users generate supervisors that are adjusted to meet the configuration of their installation.) The number of partitions cannot be changed during system operation, but the partition sizes can be modified between jobs or job steps by means of an operator command. The number and size of partitions, which best meet the needs of an installation, depend upon such factors as the total amount of storage available, the size and structural characteristics of the processing programs, their balance among job streams, and the operating environment. The user may choose to vary the multiprogramming capability of his system at different intervals during the day or shift operation.

Virtual Storage Support

Through a combination of System/370 hardware design and programming system support, DOS/VS has an address space, called *virtual storage,* that starts at zero and can extend to the maximum allowed by the system's addressing scheme. A System/370 address consists of 24 bits, providing for up to 16,777,216 bytes of address space. How much of this address space will be used in a particular system depends upon a number of factors: the size of the computer's real storage, the number of partitions, their size, the size of the SVA (shared virtual area), and the characteristics of the installation's programs and operating environment.

Based on these factors, trade-offs are made to arrive at an optimal virtual storage size for the requirements of the particular installation. A tailored system is then generated to this size, which will contain a virtual storage smaller than the maximum limit of 16,777,216 bytes, and normally larger than the real storage installed in the system.

That part of the installation's virtual storage which can be directly equated to real storage is called the real address area. That part beyond the end of the real address area, up to the limit of the system's generated virtual storage, is the installation's virtual address area. The relationship can be represented as shown in Figure 9-1.

Through the availability of the virtual address area, the constraint imposed by real storage on program size is no longer absolute. The virtual address area, as well as the real address area, is available for DOS/VS partitions and the SVA. Since the maximum size of virtual storage is very large, such partitions and the programs in them can also, theoretically, be of similar magnitude. In practice there are limitations on these sizes. The user must consider such factors as the amount of real storage available, the size and structural characteristics of the programs in the virtual address area, and make trade-offs between program size limitations and the efficiency of program execution.

Paging

DOS/VS must have a means of physically representing and containing the programs which at any instant are running in the virtual partitions. For this purpose, the user establishes an area of disk storage that is equivalent in capacity to the virtual address area allocated to the system. The disk area is called the *page data set* and it is used by DOS/VS to contain programs or parts of programs currently running in virtual mode for which there is no real storage available.

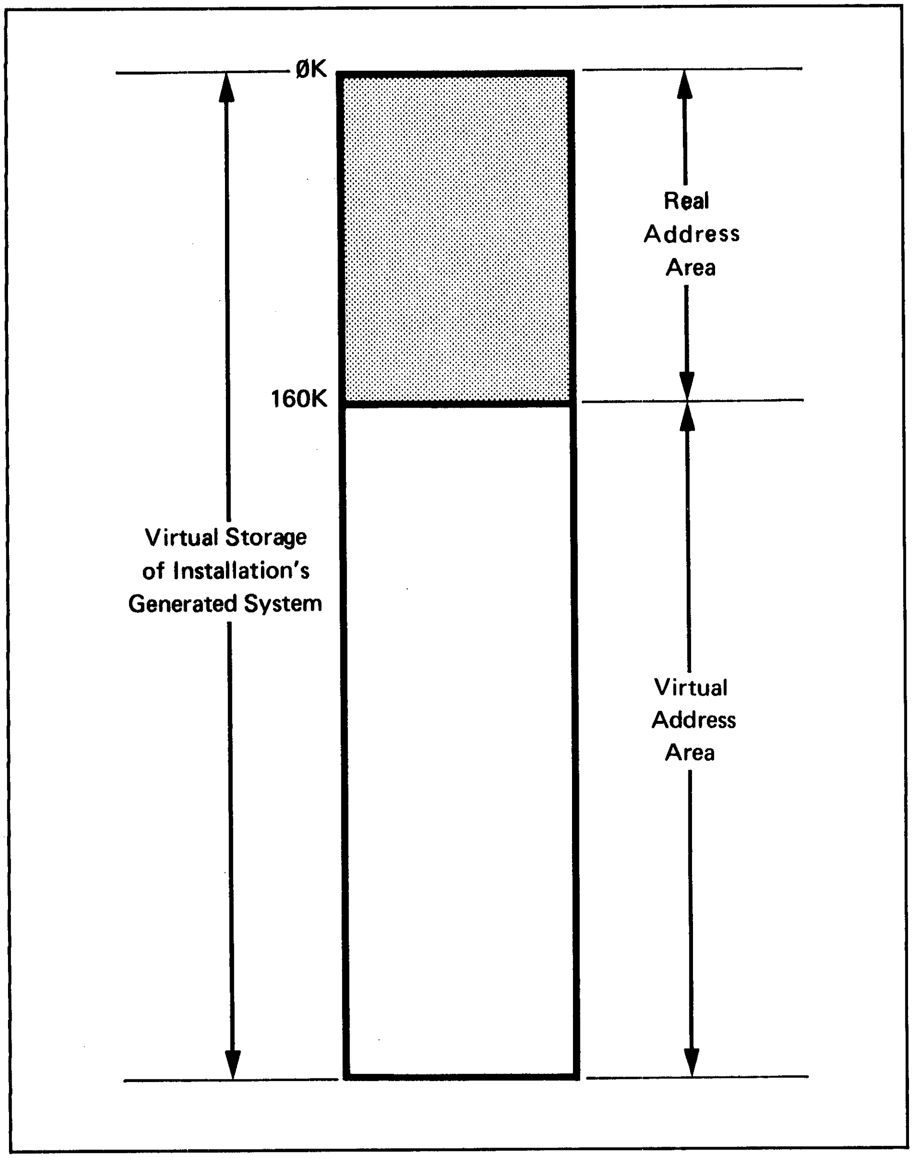


Figure 9-1 Virtual Storage in a DOS/VS System is Made Up of the Real Address Area and the Virtual Address Area

A part of real storage has to be kept available to contain programs running in virtual mode. When the limitations of this real storage prevent all programs running in virtual mode from being simultaneously present in real storage, DOS/VS exchanges sections of programs between the page data set and real storage, as they are required for execution. The program sections are called *pages*, each 2K bytes in length. The area of real storage into which the system loads a page is called a *page frame.* All the real storage page frames into which pages from any program running in virtual mode may be brought for execution make up the page pool. The page pool can dynamically change in size as the system runs. (For example, if a 40K byte program is running in real mode in the 54K real address area of a partition, the surplus 14K bytes can be made available to the page pool.) The pages are not necessarily contiguous in real storage.

When a program is running in virtual mode, all code required for execution may not be in real storage. When a program tries to refer to a storage address within a page which is not in real storage, a *page fault* occurs. The DOS/VS supervisor then performs a *page in* operation, locating the page containing the required code in the page data set, and bringing it into a page frame in the page pool. The interrupted program can then continue its execution. If all page frames are occupied, the system tries to locate a page frame not recently referenced and makes it available for the incoming page, after first *paging out* the contents of that page frame, if necessary.

OS/VS1

The purpose of a data processing installation is to do work. OS/VS1 enables the user to concentrate on this goal by performing many routine, and in some cases complicated, data processing operations. The programs that perform these operations are grouped and classified as a *system control program.*

OS/VS1 is a system control program (SCP) that makes possible the concurrent execution of as many as 15 separate jobs within a single computing system having only one central processor. The system control program of VS1 has four major functions. They are:

1. *Job management* - To accept and schedule jobs in a continuous flow.
2. *Task management* - To supervise, on a sequential or priority basis, each unit of work to be done.
3. *Data management* - To simplify storage, retrieval, and maintenance of all data, regardless of the way it is organized and stored.
4. *Recovery management* - To reduce the damaging effects that a computer, channel, or I/O device malfunction might otherwise have on a program in process.

In addition to these management functions, certain programs are included to complete the family of functions performed by the system control program. These programs include utilities, a language processor, service programs (linkage editor and loader), and service aid programs.

Job Management

Job management, or job scheduling services, performs basic functions in VS1. These include:

1. Analysis of the input stream: scanning the input data to identify control statements; interpreting and analyzing the control statements; preparing the necessary control tables that describe each job to the system.
2. Allocation of I/O devices: ensuring that all necessary I/O devices are allocated; ensuring that direct access storage space is allocated as required; ensuring that the operator has mounted any required tape and direct access volumes.
3. Overall scheduling: selecting jobs for execution, by class and priority within a class.
4. Transcription of input data units, and user output from a direct access device.
5. Communication between the operator and the system.

6. System restart capabilities

Task Management

Task management controls the allocation and use of the CPU, virtual storage, real storage, and programming resources. Task management has seven major functions or routines. The routines are collectively referred to as the *supervisor:*

1. Interruption supervisor analyzes interruptions to determine what supervisor processing is required.
2. Task supervisor records which tasks are currently in the system, their status, priorities, the programs they require, and the order in which these tasks are to be performed.
3. Virtual storage supervisor allocates and frees virtual storage, and records what use is being made of any portion of virtual storage.
4. Contents supervisor loads programs into virtual storage, and records what programs are currently in virtual storage and what characteristics these programs possess.
5. Timer supervisor sets and maintains the timers from information provided in timer macro instructions.
6. Input/output supervisor controls the reading of data from, and the writing of data to, physical devices. The I/O supervisor also provides the necessary translation for channel programs requiring a change from their virtual to real storage addresses for execution. During this translation, the I/O supervisor takes into account non-contiguous pages in real storage and fixes all required pages in real storage for the duration of the I/O operation.
7. Page supervisor allocates and releases real storage space for pages, and transfers pages between real storage and external page storage.

Data Management

Data management's objective is to achieve maximum efficiency in managing the mass of data associated with the many programs that are processed at an installation. Data management routines control all operations associated with input/output devices: allocating space on volumes, channel scheduling, storing, naming, and cataloging data sets, moving data between real and auxiliary storage, and handling errors that occur during input/output operations.

Recovery Management

A failure of the system, whether during the development of new programs or while processing jobs, can result in a loss of productivity. To protect against or at least to diminish the effects of a failure, RAS (reliability, availability, and serviceability) facilities interact with the control program. RAS facilities attempt to retry or bypass machine malfunctions that result in system failure.

Recovery management is a RAS service in VS1 that reduces the damaging effects that a computer, channel, or I/O device malfunction might otherwise have on a program in process.

Service Aids

A variety of programs that diagnose system or application program failures is available with VS1. Service aids that are offered as standard features with VS1 include:

GTF (Generalized Trace Facility)

GTF is a testing tool that can be used to trace software behavior (system or problem program).

GTF lets the user single out those programming activities to be traced within the system, including such things as I/O interrupts for all or specific devices, all program interrupts or only specific program interrupts, and all or only specific supervisor call interrupts.

HMASPZAP Program

This service aid is a problem program that allows the user to inspect and modify data at the time a problem is diagnosed.

Virtual Storage Support

The VS1 implementation of virtual storage is quite similar to (but not exactly the same as) that described above under DOS/VS.

JES (Job Entry Subsystem)

JES is a control-program facility that provides streamlined job processing in VS1. It spools and schedules primary input and output streams.

JES performs two spooling functions:

1. It reads all primary input streams, including JCL and data, from the input device, and stores them on a direct access storage device in a format convenient for later processing by the system and by the user's programs.
2. It similarly stores system (and selected user) print and punch output on a direct access storage device until a convenient time for printing or punching.

Spooling provides the following advantages:

- Nonsharable devices, generally unit record devices, are used at full rated speed if enough buffers are available.
- Nonsharable devices are used only for the time required to read, print, or punch the data.

Without spooling, the device is occupied for the entire time that a job is reading input or writing output. Thus, the device runs only as fast as the job can accept or generate data.

If system resources are the objects of contention (for example, buffer assignment), JES schedules the contending activities to assure the highest degree of system availability. Because data is stored on a direct access storage device, jobs or their output can be processed in a different order from that in which they were submitted. This ability to control system work is called *job queuing*. Jobs can be scheduled by class, and by priority within class.

Compatibility

VS1 is upward compatible to VS2. This compatibility includes source program code, object program code, job control language, and conventions and standards.

Integrated emulator programs, used with a compatibility feature, allow object programs written for one system to be executed on another system with little or no reprogramming. The compatibility feature consists of hardware and microprogrammed routines that aid emulation. The emulator programs are executed as problem programs under the VS1 system control program.

Depending upon the model of System/370, for programs that were written to run on 1401/1440/1460, 1410/7010, 7070/7074, 7080, 709/7090/7094II, or System/360 Model 20 computers or under OS/DOS on other System/360 models, combinations of machine features and system programming are provided to allow these programs to run under VS1.

Other Features

VS1 offers numerous standard and optional features. Brief descriptions of several of these appear below:

Shared DASD (Direct Access Storage Device)

The shared DASD option allows one or more direct access devices to be shared between two or more CPUs when the drives are connected to a control unit that has a path to each CPU. This feature allows access to the devices through separate channels connected to separate CPUs.

Systems can share common data and consolidate data when necessary; no change to existing records, data sets, or volumes is necessary.

SMF (System Management Facilities)

System management facilities (SMF) collects and, optionally, records system, job management, and data management information on a DASD file. The information obtained can be used in management information reports that describe system efficiency, performance, and usage. The SMF records collects such data as: system configuration, job and job step termination, CPU wait time, CPU and input/output device usage, temporary and nontemporary data set usage and status, virtual and real storage usage, status of removable direct access volumes, and paging statistics.

SMF provides exits to installation-supplied routines that can monitor the operation of a job or job step and generate the installation's own SMF records. The exit routines can cancel jobs, write records to the SMF data set, open and close user-defined data sets, suppress the writing of certain SMF records, and enforce installation standards (such as identification of users).

Checkpoint/Restart

The checkpoint/restart facility provides an opportunity to restart a job that terminates abnormally due to a hardware, programming, or system error. The restart is permitted either at the beginning of a job or at a checkpoint within a job step. In either case the restart can be automatic or you can defer it until the job is resubmitted.

MCS (Multiple Console Support)

MCS enables an installation to use one primary (or master) console and multiple secondary consoles. These secondary consoles can be dedicated to one or more system functions such as a tape library, disk library, or teleprocessing control. MCS services all consoles concurrently, creating an environment for operator-system interaction that gives each console the appearance of being the only console on the system. Each console operator receives only those messages from the system that are related to the commands that he enters and to his assigned functions.

Time-Slicing Facility

The time-slicing facility permits each task of a specified priority to have control of the CPU for a given time. Normally, a task maintains control either until it is complete, until a higher-priority task becomes ready, or until it must wait for some event (such as an I/O operation). With time-slicing, a group of tasks share the CPU, each for the same fixed time. As soon as one task has used its allotted time, control is passed to the next-ready task in the time-slice group. (This of course, is contingent on no other task outside of the time-slice group having a higher priority and being ready for execution.)

When a time-sliced task loses control before the expiration of its time (either because it must wait or because a higher-priority task acquires the CPU), the remainder of the time is not saved. When control is returned to the time-slice group, the next task is dispatched, not the task that lost control.

OS/VS2

OS/VS2 is an operating system, upward compatible from OS/VS1. OS/VS2 Release 1 (SVS) makes available a single space of 16 megabytes for programs and data. In MVS (Release 2 and subsequent releases), each job or time-sharing user is provided with individual 16 megabyte virtual address space. The 16 megabytes include the space required for system code and tables.

OS/VS2 supports the System/370 Models 145, 155II, 158, 165II, 168 and 3033. In addition, MVS supports the 158MP and 168MP systems.

In VS2, virtual storage is organized in 4K byte pages which are loaded from direct access storage into real storage for execution and written out to direct access storage when not actively being used, the space they occupy is needed and an exact copy does not exist on the paging device.

Job Management

Some of the major facilities provided by job management are multiple console support, system log, hardcopy log, checkpoint/restart, and system management facilities. (Some of these VS2 standard features are optional in VS1. A feature is considered as standard if it is automatically resident in the system after system generation).

Multiple Console Support - MCS

Multiple console support (MCS) allows one operating system to use many operator consoles. Each console in a multiple console configuration is defined by specifying the operator commands the system will accept from that console, a console to act as an alternate if a failure occurs and the types of messages the console will receive. In such a system, one console acts as the master console and the rest are secondary consoles.

System Log

The system log consists of data sets on which the communication between problem programs, operators, and the system is recorded. It may contain the following kinds of information:

- Operating data entered by problem programs using a write-to-log (WTL) macro instruction.
- Descriptions of unusual events that occurred during a shift.
- Write-to-operator (WTO) and write-to-operator with reply (WTOR) messages.
- Accepted replies to WTOR messages
- Commands issued through operator's consoles and the input stream, and commands issued by the operating system.

Hardcopy Log

The hardcopy log is a permanent record of system activity that is mandatory for systems with an active graphic console or multiple active consoles; for other systems, the primary console device serves as the hardcopy log. The hardcopy log is kept on another, non-graphic, console device or can also be kept on the system log.

Since multiple console support allows more than one console in a system, an installation might find it helpful to record all the messages issued by and to a system. The hardcopy log is a place to collect these messages, and therefore, an installation can review system activity by reviewing message activity.

Checkpoint/Restart

If a job step is terminated before successful completion, checkpoint/restart can make it possible to resume execution from the beginning of the step or from a place within the step. Either way, the restart can be made to occur automatically when the failure occurs.

The CHKPT macro instruction is coded in the user's program at a checkpoint to be taken. A checkpoint is the point at which information about the status of a job can be recorded so that the job step can be later restarted.

Checkpoint/restart includes a checkpoint routine and several restart routines.

The checkpoint routine gathers and records on a checkpoint data set enough information about the status of the job step to allow a restart from the place where the checkpoint is taken.

The restart routines can be invoked when a job step is resubmitted for restart, or they can be invoked automatically when a failure occurs.

In MVS, restarted jobs are processed by the job entry subsystem, JES, which returns them to its job execution queue for subsequent initiation based upon priority and resource availability.

System Management Facilities - SMF

System Management Facilities (SMF) collect and record system information similar to that described above under VS1. SMF records in VS2 contain additional accounting information to reflect new system environmental characteristics.

Other Features

The major new or redesigned features available in MVS include:

SRM (System Resources Manager) monitors a wide range of data about the condition of the system. It analyzes system-wide CPU and I/O load, monitors storage utilization, and requests that address spaces be swapped into or out of real storage in an attempt to keep the utilization of each of the resources within an acceptable range. In addition, SRM monitors the rate of usage of system resources by individual users, compares this rate to a target usage specified by the installation, and attempts to maintain this target resource usage rate by making the appropriate swapping decisions.

MF/1 (System Activity Measurement Facility) collects information about system activities and produces trace records and reports. MF/1 can monitor the following classes of system activity: CPU, paging, work load, channel and I/O device. The measurement data and the reports produced by MF/1 can aid in improving system performance, analysis of system trends, and evaluating future system requirements.

VIO (virtual I/O) handles temporary data sets in MVS and provides the following advantages:

- Elimination of some of the usual I/O device allocation and data management overhead for temporary data sets.
- Generally, more efficient use of direct access storage space.
- Use of the I/O balancing capability of the paging mechanism.

System Integrity

System Integrity is the ability of the system to protect itself against unauthorized user access; that is, an unauthorized program using any system interface should be unable to:

- Bypass store or fetch protection (read from or write into another user's area)
- Bypass password checking (access password protected data for which a password has not been supplied)
- Obtain control in an authorized state.

(An authorized program in MVS is one that executes in a system key (Key 0-7), in supervisor state, or is authorized via the authorized program facility (APF). In MVS, all known integrity exposures have been removed.

Time Sharing Option - TSO

An extension which provides OS/VS2 users general purpose time sharing capability in a compatible OS/VS2 environment. Terminal users share remote access to the powerful facilities of the OS/VS2 for conversational interaction -- preparation, syntax checking, execution, updating of programs and data -- concurrently with normal background OS/VS2 operations. A comprehensive easy-to-use conversational command language is provided for the terminal user to communicate with the system. TSO provides conversational remote access to the OS/VS2 environment for both the experienced professional programmer and the individual with little or no experience with computers.

Features and advantages of TSO include:

- General purpose time sharing capability operating concurrently with OS/VS2 background operation within one operating system.
- Data sets (and devices, in MVS) can be dynamically allocated in the time sharing region.
- Real storage utilization reflects the actual requirements to execute the program in the time shared region as compared to a fixed requirement in OS/MVT.
- In SVS, multiple Time Sharing users share a time sharing region and their active page (working set) are swapped (block paged) to the paging data sets.

  In MVS, each time-sharing user is assigned to an individual virtual address space.
- Time sharing provides an environment for creating and executing conversational programs.
- Programming languages and data management are compatible between conventional (batch) programs and programs developed at the terminal. Batch or terminal-developed programs can be stored, retrieved and executed locally (at the computer center) or from the remote terminal allowing the use of data sets by time shared or other regions/address spaces.
- The debugging command, TEST, allows system programmers and assembler language programmers to control the execution of a program, interrupting it at dynamically specified points.
- In MVS, the installation may specify a time interval which establishes a period that will permit a time-sharing user to reconnect to the system in the event of a line disconnect. Should the interval lapse prior to the user reconnecting to the system, then the system will automatically save any data set which the user was in the process of editing.
- TSO offers comprehensive language support for on-line development, debugging and execution of programs in COBOL, FORTRAN, PL/I, BASIC, and assembler.

  Language facilities available to the terminal user include: compilation, usually invoked with a single command; linkage editing or loading; program execution with terminal I/O capabilities for interactive application; and interactive debugging, using the data names and labels of the source program, of a program in execution for rapid program checkout.

Virtual Machine Facility/370 (VM/370)

VM/370 is a system control program (SCP) that manages a real computing system so that all of its resources -- CPU, storage, and input/output devices -- are available to many users at the same time. Each user has at his disposal the functional equivalent of a real, dedicated computing system. Because this functional equivalent is simulated for the user by VM/370 and does not really exist, it is called a *virtual machine.*

VM/370 is designed for IBM System/370 Models 135, 138, 145, 148, 155II, 158, 165II, 168, and 3033. The real System/370 must have the dynamic address translation (DAT) feature, a hardware facility that translates virtual storage addresses to real storage addresses, and the System Timing Facility. Also, it must operate in extended control mode, a mode in which all the features of a System/370, including dynamic address translation, are operational.

VM/370 provides:

- Virtual machines and virtual storage
- The ability to run multiple operating systems concurrently
- A conversational, time-sharing system
- A remote spooling communications subsystem (SCS).

Elements of VM/370

VM/370 has three major elements:

1. The control program (CP), which controls the resources of the real computer to provide multiple virtual machines. Executing a program on a virtual machine produces exactly the same output as executing that program on a real machine.

   When a user logs onto VM/370, CP creates a virtual machine for that user based on information stored in the VM/370 directory. The VM/370 directory contains one entry for each user identification. Each entry includes: the password associated with the userid; a description of the virtual input/output devices associated with this virtual machine; its normal and maximum virtual storage sizes; the user's CP command privilege class(es); and optional virtual machine characteristics, such as extended control mode.

   CP controls the resources of the real computer to provide multiple virtual machines. CP intercepts, translates, and also schedules all real input/output operations of the virtual machine. All virtual machines execute in problem state, and the control program traps and processes all interrupts and privileged instructions. Only CP executes in supervisor state.

2. The conversational monitor system (CMS) is the major subsystem of VM/370. Together with the control program of VM/370, it provides a time-sharing system suitable for direct problem solving and program development. CMS is an operating system that runs only in a VM/370 virtual machine.

CMS is a conversational, single-user system. The user's interface to CMS is the virtual operator's console, that is, the terminal used to gain access to VM/370.

CMS has no multiprogramming capabilities, as it is designed to run in a VM/370 virtual machine. CP provides the time-sharing environment; CMS provides the conversational user interface. Using CMS, the user can write programs to run under CMS or under another virtual machine operating system.

3. The VM/370 remote spooling communications subsystem (RSCS) provides the spooling of files between remote stations and virtual machines at the VM/370 installation. (Remote stations are configurations of I/O devices attached to the VM/370 computer by binary synchronous communications (BSC) switched or nonswitched lines.)

   The VM/370 computer is the functional center of communications in the RSCS teleprocessing network. The operator of the RSCS virtual machine controls the network by issuing RSCS commands at the RSCS virtual machine console.

   The facilities of RSCS are selected and controlled by means of commands and control cards. Connections between geographically remote locations are made by the operator of the RSCS virtual machine.

   Each location in the RSCS network is assigned a location identifier, which RSCS uses to find a link, or path, to the remote location.

Virtual Machine Operating Systems

While the control program of VM/370 manages the concurrent execution of the virtual machines, it is also necessary to have an operating system managing the work flow within each virtual machine. Because each virtual machine executes independently of other virtual machines, each one can use a different operating system or different releases of the same operating system.

CP provides each operating system with virtual device support and virtual storage. The operating systems themselves execute as though they were controlling real devices and real storage, but they must not violate any of the VM/370 restrictions.

VM/370 Applications

VM/370 assists an installation to perform its work more efficiently and easily. Virtual machine applications aid programmers, operations personnel, and interactive users.

Programming

Programming is facilitated in the following ways:

1. Programs being developed need not conform to the real storage size of the real computer.
2. Virtual machines make program testing more flexible. Subject to available resources, a virtual machine can be made active whenever needed, thus relaxing normally tight or inflexible testing schedules and allowing programmers more compilations and tests per day.
3. JCL (job control language) usually is not needed when compiling, assembling, and/or testing under CMS.

4. Users can test privileged code in their own virtual machines.
5. Programmers can use debugging aids at their terminal that parallel those of an operator at a system console: displaying and storing into the general or floating-point registers or into virtual storage, instruction address stopping, and altering the normal flow of execution. Which of these functions each user is allowed to perform are defined by the privilege class(es) assigned to that user.
6. CMS simplifies the creation and manipulation of source programs on disk and allows the user to examine selected portions of program listings and storage dumps at the user's terminal.
7. RSCS allows users to transmit files to, and receive files from, users at other remote locations.
8. The VM/370 data privacy, security, and user-isolation features protect each user's data, programs, and disk files from access or destruction by other users.
9. Many System/360 and System/370 programs can be compiled under control of CMS; within certain restrictions these programs can also be tested under CMS. DOS assembler language programs can be compiled under CMS if the installation adds the appropriate DOS macros to the CMS system. Problem programs using DOS macros can be conversationally developed under control of CMS; then control of the virtual machine is passed to DOS, and the programs are compiled and tested. The user specifies which operating system is to control the user's virtual machine by means of the IPL command of CP.

Operations

The virtual machine environment relieves certain problems of scheduling, updating programs and backup, and expedites production in the following ways:

1. System generation, updating, and system testing, as well as operating system conversion and testing, can be done without a dedicated real machine, concurrently with normal production work. This reduces errors that might otherwise be caused by using a system that has not been fully tested, and it also reduces the possibility of abnormal terminations of the system. For example, a program temporary fix (PTF) applied to a copy of an IBM operating system volume can be tested concurrently with the production execution of that same operating system in another virtual machine, provided sufficient direct-access storage resources are available.
2. VM/370 allows DOS and OS, including virtual storage (VS) versions, to run concurrently on the same System/370. Multiple copies of the same operating system can also run concurrently in separate virtual machines.
3. Many types of batch applications can be run, either in an individual user's virtual machine or in a virtual machine dedicated to running batch, with no change to the batch program.

Interactive Use

Two kinds of interactive systems run under VM/370: multiple-access and single-user.

1. Multiple-access systems like APL\DOS-360 run in one virtual machine and directly service many interactive terminals. A user of a multiple-access system dials the system instead of logging on to connect his/her terminal with the virtual machine running the multiple-access system he/she wishes to use. Once the terminal is connected, the user issues statements in the command language associated with the multiple-access system only.

   For example, dialing APL could connect the user's terminal with an APL\DOS-360 system running in a virtual machine under VM/370. Once connected, the user communicates only with APL commands and cannot use any CP commands.

2. Systems that a single user can run interactively include the conversational monitor system (CMS) and any operating system that can run on a virtual machine. A time-sharing environment is created when VM/370 creates multiple virtual machines, each controlled by a copy of CMS. These systems operate concurrently with each other as well as with other conversational or batch systems. CMS is useful for program development and problem solving.

   The CMS command language provides each user with a wide range of capabilities at the user's remote terminal, such as:

   - Creating source programs, data, and text files directly on disk.
   - Adding, deleting, modifying, rearranging, extracting, or merging files and/or portions of files.
   - Compiling, testing, and debugging some types of OS problem programs under CMS.
   - Creating complete job streams to be passed to batch processing systems such as DOS or OS for compilation and/or execution. The resultant output can be printed on a high-speed printer or directed back to CMS for analysis and correction by the user.
   - Submitting jobs to a background CMS batch facility.
   - Extending CMS facilities to suit the user's requirements such as creating additional commands or developing command procedures.

Access Methods

As stated previously, *access methods* refer to the routines which assist the programmer in transferring records in a particular data organization format between storage and an I/O device. Broadly speaking, how data is organized and the type of device that it is stored on largely determine the access methods that can subsequently be used to retrieve it. Each VS operating system provides several methods of data organization. For each of these there is an access method which allows one or more techniques of file creation and retrieval. In addition, depending upon the operating system and the data organization, data management programs provide two general techniques known as access techniques for moving data: the queued technique and the basic technique. The queued technique offers the maximum amount of automatic I/O facilities. The basic technique places some of the responsibility for data handling on the programmer, but gives the programmer more direct control of I/O operations.

Queued Access Technique: When using the queued access technique, you can concentrate on data processing alone; the data management routines handle most I/O considerations. For example, I/O is automatically synchronized with processing. When you issue a GET macro instruction, the desired record is already in an input buffer, so processing can continue without delay. When a buffer is empty, the data management routines automatically refill it. The same principle applies for output records (PUT macro). They are collected in an output buffer and written when the buffer is full. When operating under a priority scheduling system, and if output is directed to a system output class (a class of system output units shared by all jobs), data is first written on a direct access device. When scheduling permits, the writer transfers the data to the proper device.

Because the queued access technique brings records into virtual storage before they are actually needed, the data management programs need a method of anticipating the user's demands. Therefore, the queued access technique can be used to retrieve only records in a sequential order, for example, records on magnetic tape.

Basic Access Technique provides the READ and WRITE macro instructions for input and output. These instructions move blocks, not records. As with the queued access technique, actual transmission to a specified device may be deferred and done by the writer when working under a priority system and output is going to a system output class.

Unlike the queued technique, the basic technique does not provide automatic synchronization of program processing and I/O. When issuing a READ, you cannot assume that the record is in virtual storage as you can assume with GET. You must determine that the I/O operation has been completed before attempting to use the desired record. Data management provides macro instruction facilities to check for successful completion of I/O operations and, if necessary, to wait for their completion.

The basic access methods are used for all data organizations while the queued access methods apply only to sequential and indexed sequential data sets. The access method, VSAM, employs a modified basic and queued access technique and applies to direct and sequential data sets.

Brief descriptions of several data access methods follow.

Sequential Access Method - SAM

Sequential organization means that records physically follow one another in a sequence usually determined by one or more control fields within each record. Examples of control fields are name or man-number in a personnel file, or catalog number or part number in an inventory file.

Sequential organization is still the most widely used method of data organization and is supported for all device types except teleprocessing terminals. Card files, print files, diskette unit files, and magnetic tape files are always organized sequentially, simply because the physical characteristics of those devices require the reading or writing of one record after another. Data files on disk are also frequently organized sequentially, in control number sequence.

If required, records are sorted into their prescribed sequence prior to, or as a part of, creating a sequentially organized file. The Sequential Access Method (SAM) can create a sequential file from the sorted records presented to it and subsequently retrieve those records for sequential processing. In addition, by utilizing certain macros, sequential files on disk or tape may be positioned to specific physical blocks prior to reading or writing. Records from sequential disk files may be *updated*, meaning that each record may be written back onto its original physical location after having been changed by the program.

Sequential organization and access methods are used for some files in most data processing installations since the requirements of many applications are still met by batch processing. It should be noted that even teleprocessing applications often include a sequential data set such as a time-sequenced log of transactions received and/or processed.

Basic Direct Access Method - BDAM

In the Basic Direct Access Method (BDAM), records within a data set are organized on direct access volumes in any manner chosen by the programmer. Storage and retrieval of a record is by actual or relative address within the data set. This address can be that of the desired record or a starting point within the data set where a search for the record, based on a key furnished by the programmer, begins. Addresses are also used by BDAM as a starting point for searching for available space for new records.

Indexed Sequential Access Method - ISAM

An indexed sequential file is made up of (1) records in logical sequence by control field (or key) and (2) an index, which is built when the file is created. The index itself is structured in two or three levels. Each index entry is composed of the key of a data record, or a lower level index entry, and the physical address at which the record, or lower level index entry, is located on the disk. The programmer may process indexed sequential files *sequentially* when the definition of the programming application requires this approach, or process them *randomly*, retrieving a particular relevant record from the entire file, if this approach is better adapted to the requirements of the job. Both retrieval methods are easy from the programmer's viewpoint:

GET and PUT macro instructions are used to process the file in sequential order, the user supplying control information for the first record to be processed. READ and WRITE macro instructions are used to process the file in random order, the user supplying control information for each record to be processed.

ISAM also provides the routines for creating a file from sorted input, building the index, and adding records to an existing file. For the insertion of records into an existing file, additional disk space, called *overflow area* is reserved. On sequential retrieval of a file that has data in the overflow area, records are retrieved in logically sequential order.

Virtual Storage Access Method - VSAM

VSAM is an access method designed to operate with direct access devices and to support both direct and sequential processing by means of either an index key (keyed accessing) or by means of relative byte address (addressed accessing). Both fixed length and variable length records are supported.

Three types of data sets are provided: key-sequenced data sets, which are ordered by a key field in the data record, entry-sequenced data sets, which are ordered by the sequence in which the records were loaded, and relative record data sets which are ordered by record number. Keyed accessing is used to access key-sequenced or relative record data sets, and addressed accessing is used to access both key-sequenced and entry-sequenced data sets. Key-sequenced and entry-sequenced data sets may be either fixed or variable length records, relative record data sets are fixed length records only.

VSAM is composed of two major elements; a data organization which minimizes data movement and is suitable for data base applications; and routines for creating data sets in the VSAM organization, adding and deleting records, and performing other data management functions.

The data organization of ISAM is based on the physical units of disk cylinder and disk track, while the data organization of VSAM is based on logical units called control intervals and control areas. A *control interval* is the unit of direct-access storage that is transferred to and from virtual storage. It can contain one or more records in one or more blocks. A percentage of each control interval can be free space and some control intervals can be entirely free space. A *control area* is a group of control intervals. Thus, VSAM data organization provides for physical characteristics of the data and the index.

In general, the VSAM user can expect to see performance improvements relative to OS/VS ISAM and DOS/VS ISAM. Performance gains with VSAM can become increasingly significant as the number of insertions to the data set rises. This is due to the elimination of the *chained record overflow* concept employed by ISAM. VSAM will effectively maintain its sequential, non-inserted performance as records are added to the data set. Also, VSAM requires less time to perform a record insert than does ISAM. These factors, coupled with the efficient VSAM index structure and with the VSAM performance options, offer the potential of performance improvements relative to ISAM.

A significant feature of VSAM is that of data set and volume portability between DOS/VS and OS/VS systems. Portability of data sets and volumes is made possible by the user catalogs and the multi-function service program, Access Method Services.

Access to data via VSAM may be protected by multiple levels of passwords. In addition, VSAM provides an exit for users to impose their own security routines.

Access to VSAM is controlled by macro instructions. Most programs written to access ISAM data sets may be used with VSAM data sets via the ISAM interface.

VSAM offers a multi-function service program (Access Method Services) to facilitate overall management of data. Such services as defining data sets initially, deleting VSAM data sets from the VSAM catalog, printing and copying data, listing the VSAM catalog, and providing backup and portability features are controlled by this multi-function program. Converting data sets from the ISAM or SAM format to the VSAM format is another important function of this program.

Optional features include:

Alternate Indexes

This feature permits application programs to access the records of a VSAM entry or key sequenced data set on the basis of keys other than the prime key. These alternate keys may be non-unique and must be contained in the base data record. Once an Alternate Index has been constructed by using Access Method Services, it may optionally be automatically updated whenever a data record is changed in the base data set to which it relates.

Relative Record Data Set

With this feature the data set is viewed as a numbered sequence of fixed length slots. Records may be inserted, updated, read, or erased in these slots using VSAM keyed processing, with the slot (i.e., record) number as the key. No index is used since each record's physical location is calculated directly by VSAM from its record number and the characteristics of the data set.

Get Previous

This feature permits retrieval and update processing on the basis of descending key values, relative record numbers, or relative byte addresses. Processing may begin either within or at the end of the data set.

Telecommunications Access Methods

Included in the data management function of the virtual storage programming systems is teleprocessing device support which provides a link between the application program and remotely connected teleprocessing terminals via the communications control device. This support is divided into three access methods which aid the host processor application programs in obtaining data from remote terminals.

Basic or Virtual Telecommunications Access Method (BTAM/VTAM) directs the transmission of data between the host processor application programs and terminals. BTAM/VTAM provides basic capabilities to receive and send messages, dial and answer (switched lines), chain input buffers, detect and correct errors, and perform code translation. Stated simply, BTAM/VTAM controls terminal input/output operations. Further details concerning BTAM and VTAM appear later in this section.

If queued control is required in an asynchronous transmission mode network, *Queued Telecommunications Access Method* (QTAM) is available. QTAM includes BTAM capabilities as well as message queuing, routing, and logging. If queued control is required in a synchronous transmission mode network, *Telecommunciations Access Method* (TCAM) is available. For example, data can be directed to an inactive terminal and held in queue until that terminal is activated and connected.

TCAM and QTAM are not available with all of the virtual storage programming systems.

Basic Telecommunications Access Method - BTAM

The facilities of the Basic Telecommunications Access Method (BTAM) are designed chiefly to provide the basic tool required to write a telecommunications program. These include facilities for creating terminal lists and for performing the following operations:

- Initiating and answering calls to and from terminals on switched networks.
- Polling and addressing terminals on non-switched multi-point lines.
- Changing the status of terminal lists.
- Transmitting and receiving messages.
- Code translation.
- Retransmitting messages which are received with detected error.
- Providing on-line terminal test facilities.
- Keeping error statistics.

The support of Binary Synchronous Communications combined with that of the various start/stop devices gives BTAM a wide range of applicability and flexibility. BTAM supports low, medium, and high speed devices.

All terminals (except Binary Synchronous Communication) on a multi-point non-switched line must be the same type. Terminals may be mixed within the same problem program.

The BTAM facilities of DOS/VS may be used in all or any of the system partitions in either virtual or real mode, or may be used to design a dedicated telecommunications system in a system with a single partition.

Virtual Telecommunications Access Method - VTAM

VTAM is a direct-control access method that enables application programs to control VTAM terminals without concern for intermediate connections, such as control units and telecommunication lines. It is designed to use advanced hardware and software including System/370 virtual storage, the IBM 3704 and 3705 Communications Controllers (Figure 9-2), the virtual storage operating systems, and the teleprocessing subsystems that use the SDLC (synchronous data link control) line discipline.

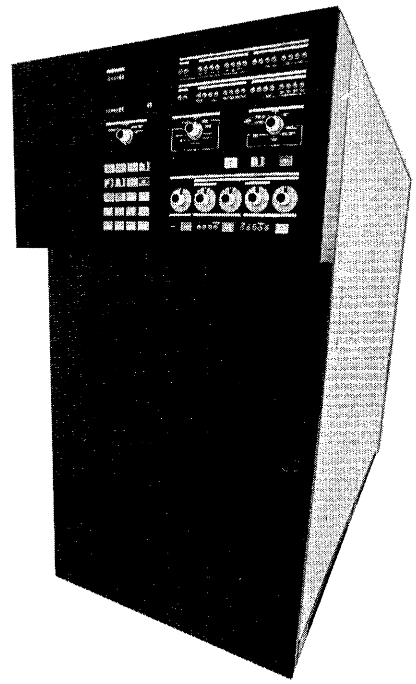


Figure 9-2 IBM 3705 Communications Controller

With VSAM (virtual storage access method), VTAM can be used to provide a complementary data base/data communication facility. In addition to its primary role of data transmission, VTAM has features that establish it as a base for building small to large telecommunication systems. These features include:

- Sharing of network resources, which reduces line costs and makes more efficient use of the network.
- Concurrent execution of TCAM and VTAM application programs using the same telecommunication network.
- Services required for interactive applications such as online inquiries and updates.
- Operation with the IBM 3704 and 3705 Communications Controllers to reduce the number of functions performed in the central computer for remote devices.
- Generation options for tailoring the telecommunication system to the user's needs.
- Support for industry-oriented teleprocessing subsystems such as the IBM 3600 Finance Communication System.
- RAS (reliability, availability, and serviceability) aids to assist in reducing both the incidence of errors in the telecommunication system and the

impact of errors that do occur, and in maintaining the telecommunication system.

When an IBM 3704 or 3705 Communications Controller is installed as the control device, VTAM allocates some of the network management responsibilities to this unit. The *Network Control Program - Virtual Storage* (NCP/VS) operates with the IBM 3704 or 3705 to route data through the network. Through commands sent from VTAM, the NCP/VS assumes much of the responsibility for controlling communications lines (Figure 9-3). Thus, valuable host processor space is freed for higher, application-related functions. Also, because some control is exercised locally (in the 3704 and 3705), line traffic is reduced and line costs are lowered.

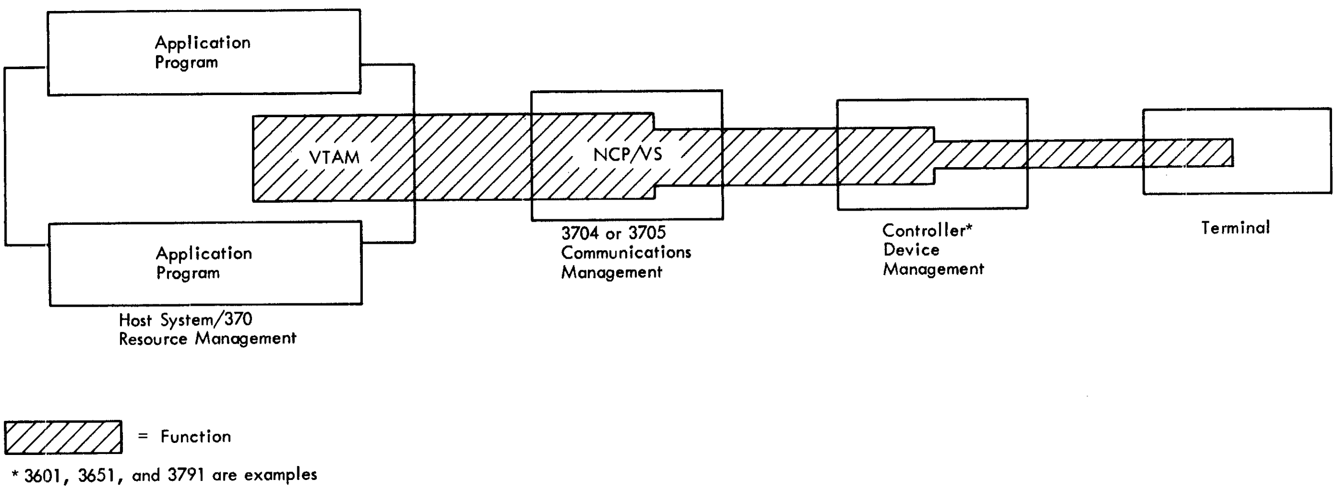


Figure 9-3. Concept of Function Distribution Under Programming Control

Telecommunications Access Method - TCAM

TCAM is a generalized IOCS (input/output control system) that extends the techniques of logical IOCS to the telecommunication environment. Data sets accessed by the problem program are queues of messages coming in from, or going out to, remote terminals via communication lines.

TCAM furnishes far more than the control for I/O operations. In addition to supporting the transfer of messages between the terminal and user-written application programs, TCAM provides a high-level, flexible, message control language. (Data enters a telecommunication system in the form of messages.) You can use TCAM macro instructions to construct an installation-oriented message control program that controls the flow of message traffic from one remote terminal to another (message switching application), and between remote terminals and any application programs (message processing applications).

A telecommunication control system created through the use of the TCAM message control language:

- Establishes contact and controls message traffic between the computer and terminals.

- Deletes and inserts line-control characters automatically, thereby removing line-control from the user's domain.
- Dynamically assigns and uses buffers as required.
- Edits incoming and outgoing messages (for example, code translation and insertion of new fields in message headers).
- Forwards messages to destination terminals and application programs.
- Takes corrective action and provides special handling for messages containing errors.
- Maintains statistical information about message traffic and system components.

Section 10:

Data Security and Integrity

With the continued expansion in the use of the computer to encompass more and more activities of individuals, business enterprises, private and public organizations, governments, and so on, concern over the possibility and the effect of data processing errors, misuse of information maintained within the computer system, destruction of files, and so on, whether intentionally or accidentally, takes on a significantly more serious tone. Furthermore, as procedures once done manually in an organization are automated, the potential for loss associated with the use of the computer increases.

In many cases, even an interruption in the availability of data or processing capability can have catastrophic effects on organizations highly dependent upon the availability of such data or processing.

Security involves protecting both the data and the equipment needed to maintain and process the data. The number of possible hazards is quite large. Physical hazards such as fire, water, and malicious visible acts are well-known. Others such as those considered below may not be but all require adequate security measures.

Often not considered a threat to the security of data, program errors and operator accidents can cause more damage than most other hazards. Seemingly minor accidents can have severe consequences. Damage can often be minimized by proper training and alertness of operating personnel. In one incident, a single disk pack was dropped; the warped pack was mounted on a drive, thus damaging the access mechanism. The pack was moved to another drive, and a different pack was tried on the first drive. The result was several damaged drives and unusable packs – a simple accident compounded by poor judgment. If one of the packs had been the working copy of a file and another the single backup copy being readied for emergency storage, the file would have been unrecoverable. In most backup systems, vulnerability is greatest while preparing the backup copy and getting it safely into storage.

The hazard of accidental modifications is exemplified by such everyday occurrences as keypunching or other key entry errors. Intentional modification can be a vehicle for internal fraud and embezzlement; its significance may be greater than that of destruction.

Malicious damage or destruction can be equally severe, whether accomplished by an employee or by an outsider. According to published reports, vindictive employees have inserted routines in programs to disrupt or destroy applications after they have left a company.

The hazard of disclosure can be obvious (a tape reel is removed from its slot in the library), or it can be concealed (a discarded report is retrieved by an outsider or a disgruntled employee). In certain situations disclosure can also be extremely harmful.

Concern for losses due to errors and omissions, inadequate systems design, and improper controls tends to result in an expressed need for better controls on the processing procedure and data processing personnel as the means to reduce such losses. However, it should be recognized that it is practically impossible to provide complete security for all possible hazards. Nevertheless, many controls exist or can be put into place to help minimize security risks. Some of these are described below.

Backup Facilities

A key element of a security program is backup, the means to recovery. The ability to recover protects against the widest set of hazards, including many that cannot be anticipated or prevented. The resources that must be provided to permit resumption of operations within a reasonable length of time after various kinds of disasters is the first consideration. Planning the use of alternate resources should be on a job-by-job basis, having first determined priorities for the jobs in the order of their relative urgency. A backup plan considers the people, plant, and records needed to resume operations.

A backup computer installation should be far enough away from the primary facility so it is not subject to the same hazards, but it should be accessible and close enough to serve its customer organizations. In a data communications system, accessiblility may mean simply being near communications lines rather than near other company plants or offices. But any system needs people to operate it people who would have to be moved if the backup facility were very far from the facility being backed up. Most systems have a great interdependence on the resources of the parent organizations; therefore, if the backup is to be operational quickly, and if it is to provide service for an extended period, then proximity to the primary facility is important.

Physical separation ensures that both locations will not be crippled by a single fire, explosion, airplane crash, or any other single accident; and it reduces the risk of simultaneous damage from weather hazards. The benefits of physical separation can be further enhanced by (1) obtaining electric service from separate substations so that, a single power outage would not disable both locations, and (2) avoiding use of common communications trunks or exchanges in communications-oriented systems.

Companies having multiple installations containing similar data processing systems have what may be the ideal backup situation, particularly if the separate installations are geographically close or have high-speed intercommunications. The advantages of this kind of backup are that it is available immediately, it is already under the company's control, trained personnel are already available, and each installation already has acceptable storage areas for supplies and data. It may be necessary to provide excess capacity and availability in some of the systems in order to take on the extra workload in a backup situation. This excess capacity can be in the form of unused shifts and personnel trained in multiple skills, as well as larger data processing systems than needed for normal operation. Those companies that already have multiple installations may find that few changes are needed to permit any installation to adequately back up another.

Those having a single data processing center may want to arrange for mutual backup with other organizations having similar systems. Arrangements of this type should be reviewed periodically to minimize risks such as these: the configurations can become dissimilar, the backup installation can become incapable of absorbing the added workload, or any of a number of other factors may render the alternative unavailable or unsuitable.

Security will be required in a backup location; however, the measures employed to obtain it may be different, and the level of risk acceptable may be higher. For example, the existence and location of backup may be kept reasonably confidential, probably more effectively than that of the primary site.

Data Controls

The purpose of data controls in any system, whether it be manual or automated, is to prevent or promptly detect undesirable events, such as loss of data, errors in processing, fraud, inconsistency of data, or any other form of loss.

An important control mechanism is the *audit trail.* Ideally, an audit trail should indicate how every record in the system got to its present state, what it looked like at any point in the past, when and by whom each change was made, as well as the pertinent circumstances; the audit trail should make it possible to reconstruct the record. The audit trail enables the system to fix accountability, and thus serves to maintain the accuracy and integrity of the data.

There are four basic types of data controls: input, processing, output, and procedural. Combined with audit trails, they give reasonable assurance that computer-generated transactions are accurate, reliable, and complete.

Input Controls

These are the steps taken to ensure that input transactions are complete and accurate before data is processed by the computer programs.

One of the most common examples is keypunch verification. It costs more to verify every field on the input card, but it may be necessary. In many applications, however, only the important control fields are verified. One person should key-enter the document and a second should key-verify it, thus separating the duties and reducing the possibility of the same error being made twice.

Manual verification is never needed for fields which contain a self-checking digit. Any numeric field which can be generated in advance, such as invoice number, customer number, or part number can be a self-redundant or self-checking number. These can be prepared in advance by computer, and can be authenticated on all successive passes through a computer. While the redundancy involved will not totally eliminate all possible key-entry errors, the probability of such errors entering the system becomes extremely small as the redundancy increases.

A second method of input control is to balance to predetermined totals. For example, a total is taken of the amounts of all invoices. After punching and processing, the predetermined total is checked with the computer-generated total to assure agreement. Differences indicate data may have been incorrectly punched or processed.

Another common method applies to computer-generated input. For example, in a purchasing/accounts payable/accounts receivable system, for each new purchase order placed on the master file there will ultimately be a receipt of goods. The system can be programmed to punch out a receiving card with unchanging information, such as purchase order number, part number, and so on. The card is then sent to the receiving department to await the parts. Upon receipt, the stock clerk posts the "quantity received" to the card.

Computer-generated input such as this reduces human posting errors, since much recording is eliminated.

Edit routines are another method. These are steps written in the program to ensure that all information required for processing is actually in the input, that it has a value within a reasonable range, or that it is consistent with other data. Using the example above, assume each purchase order placed on file has a six-digit control number. The computer checks each input card for six digits in the appropriate columns. If not found, the card is rejected.

It is necessary to ensure not only that input is processed correctly but that all the documents sent to the computer room are actually received. Batches of documents can be lost, and the loss be undetected. To eliminate this possibility, a control clerk should record each batch of documents sent to the computer room, and should follow up to ensure that they are processed.

A method of input control used in realtime teleprocessing systems is message verification. Because transactions are often processed through a terminal as they occur, rather than in groups or batches, some control should be established to ensure that the terminal message is received, and received correctly, by the computer. Normally, this can be accomplished by designing the system to respond to the data received. For example, the system could respond with part description to the receipt of a part number. Visual tests can then be made to determine the accuracy of the transmission. Realtime teleprocessing controls are discussed in greater detail later and are mentioned here only to demonstrate that, under any processing mode, input controls are possible and necessary.

Processing Controls

Processing controls are normally those controls written into the computer program itself to ensure that something abnormal or undesirable does not occur during processing.

Record Count

A record count is a tally of the number of records in a data set. The count is normally established when the data set is assembled.

The total number of records is carried as a control total at the end (or the beginning) of the file (data set) and is adjusted whenever records are added or deleted. Each time the data set is processed the records are recounted, and the quantity is balanced against the original or adjusted total. If the recount agrees with the control total, it is accepted as proof that all records have been run.

Record counts may also be established by batches. This is desirable when source data is to be put into the procedure for the first time.

Although the record count is useful as a proof of processing, it is difficult to determine the cause of error if the controls are out of balance. A failure to balance does not help to locate a missing record, nor does it indicate which record has been processed more than once. Therefore, some provision must be made to check the data set against the source records, a duplicate data set, or a listing known to contain the proper number of records.

An incorrect record count, often indicates a machine failure when records are being processed because, once written on the tape (or disk, etc.) correctly, records cannot be misplaced or lost. In this case, the doubtful portion of the data set should be rerun for correction.

Limit Check or Reasonableness Check

A limit check is a test of record fields or programmed totals to establish whether certain predetermined limits have been exceeded. For example, if transaction codes for certain records are known to cover only the digits 0 through 5, a check can be programmed to see that no code exceeds the limit of 5.

Reasonableness tests are processing controls written into important programs to determine if something highly unlikely or logically inconsistent has occurred. Payroll procedures often contain many limiting factors that can be checked by the program. The upper limit of gross pay is usually determined by the type of payroll: hourly, salary, piece rate, incentive, and so on. Hourly rates must fall within established wage scales. The total number of hours worked per employee is also subject to certain limits. For example, a weekly paycheck would not normally be $1000 or more for an hourly-rate employee. Therefore, the program would print out such *unreasonable* information on a special report. The occurrence may or may not be legitimate, but it would at least be available for review.

Limit checks may also be used in table lookup procedures. If an item is known to be in a given table in storage, the modified table address may be checked against the address of the upper table limit to verify correctness of the search. If the search begins to exceed the limits of the table, an error has occurred, and corrective action is required.

In many mathematical problems, the range of the final calculation can generally be estimated. If a result falls outside this reasonable range, it may be assumed that some error condition is present, either in the data, in the program, or in the calculation. Departures from normal trends may also indicate faulty procedures. The simple application of a limit check in such problems may save much detailed checking, with consequent simplification of the program.

Tape and Disk Labels

Identification information recorded at the beginning of a reel of magnetic tape is called the header label; identification information recorded at the end of a reel is called the trailer label. The label may specify file identification, data of last processing, number of reel, and so on. A label may also be placed at the end of the file. Standard label checking is automatic in programming systems.

While tapes may have both a header and a trailer label located physically before and after the data, respectively, disk labels can appear physically anywhere on the volume, as long as the user specifies where it is located.

The labels are read into storage at the beginning and at the end of the program as an added control to ensure that the proper records have been processed. The label may also ensure a true end-of-file or end-of-job condition and, in addition, include a record count.

The various programming systems supporting System/360 and System/370 have a definite format for "volume" labels and checking.

Housekeeping Checks

The first instructions of nearly every program are intended to perform functions of housekeeping in preparation for processing. These instructions may set program switches, clear registers, set up print areas, move constants, and so on. In addition, housekeeping instructions may perform systems checks by testing to determine whether all input/ output units required by the main program are attached to the system and ready for operation. File labels may be checked and updated, constant factors may be calculated, and other information pertinent to the proper operation of the system may be called to the operator's attention by programmed instructions. Programming systems provide many of these checking procedures.

Sequence Check

Sequence checking is another processing control that is used to determine if records have been sorted properly. For example, an inventory of production parts is kept on magentic tape in sequence by part number. Withdrawals and stock receipt transaction cards are sorted in part number sequence before they can be used to update balances on this file. During the update run, tests are made to determine that each record read on both files is equal to or greater than the previous one. Without tests, an undetected out-of-sequence condition could seriously impair the accuracy of the update.

Output Controls

Output controls report the resolts of computer processing - especially in conjunction with input and processing controls.

Control Total

The control total may be made up from amount or quantity fields in a group of records. It is accumulated manually or by machine when the data set is originated or when a quantity is first calculated. The control total can be either a grand total or more convenient intermediate or minor totals.

When the data set or group of records is processed, the fields are again accumulated and balanced against the control total. If the total is in balance, it serves as proof that all records have been processed correctly.

The control total is an efficient systems check when it can be used to predetermine the results of calculation or the updating of some record. For example, when preparing to process a payroll, the total number of hours worked by all employees is preestablished from clock or job-card records. This figure then becomes the control total for payroll hours for all subsequent reports. Totals may be broken down by group or department. The sum of all totals must balance back to the complete original total.

Control totals are normally established for batches of convenient size, such as department, location, account, or division. By this method, each group of records may be balanced as it is processed. Exception reports list the transactions that were not accepted by the system because they did not pass some input or processing control point. Corrective action, if needed, is limited to small, easily checked groups rather than to one grand total.

Console Error Messages

Console error messages are also important. They are normally used for errors significant enough to halt processing. For example, under most circumstances, processing terminates when records are found to be in improper sequence during an update. The console error message prints or displays the type of error encountered, the records that were out of sequence, and the action the operator should take, such as notifying the proper persons.

Checkpoint and Restart

A checkpoint procedure is a programmed checking routine performed at specific processing intervals or checkpoints as discussed in prior sections. Its purpose is to determine that processing has been performed correctly up to some designated point. If processing is correct, the status of the machine is recorded, usually by writing this informaition on a tape or disk. The normal procedure is then continued until the next checkpoint is reached.

Checkpoint procedures have the effect of breaking up a long job into a series of small ones. Each portion of the work is run as a separate and independent part, and each part is checked after it is completed. If the check is correct, enough information is written out to make it possible to return to this last point automatically. If not, the portion of work just completed incorrectly is discarded, and the system restarts from the last point at which the work is known to be correct.

A restart procedure (1) backs up the entire computer system to the specified point in the procedure, usually a checkpoint (tape files are backspaced or rewound; card units and printers are adjusted manually; disk updates are *backed out* using a transaction log tape or the disks rewritten using data saved at checkpoint); (2) restores the storage of the computer to its status at the preceding checkpoint (this may include the adjustment of accumulated totals, reloading the program itself, reestablishing switches and counters, restoring constant factors, and so on).

The proper use of checkpoint and restart procedures in a program contributes to the overall operating efficiency of a computer system. If power failure or serious machine malfunction occurs, these procedures provide a means of rerunning only a small part of a job without having to rework an entire job. This may mean a saving of many hours of machine time.

Restart procedures also allow interruption of a given job for the scheduling of other jobs that need immediate or emergency attention. Thus, any procedure may be interrupted intentionally by the operator and replaced with another job when necessary. Provision for restart is also convenient at the end of a shift or other work period when the operation of a job must be terminated without loss of production time. Finally, restart procedures provide interruption of machine operation for emergency repairs or unscheduled maintenance.

Procedural Controls

Procedural controls are maintained by the people who use the system. Three of the most important procedures are discussed below.

Separation of Responsibility

Organizational controls are those which protect an organization from fraud, error, waste, and so forth. They establish that one department's (or individual's) duties interact with another's so that errors or thefts made by one will be promptly detected by the other. No one person has complete control of any part of the business.

For example, the two main functions in an accounts receivable department are (1) recording a sale as an open receivable in the customer's account, and (2) receiving and recording the customer's payment. If one person is allowed to perform both functions, he or she could, intentionally or inadvertently fail to record the open receivable in the customer's account and keep the payment. Since there is no open invoice for the customer, there is little chance of detection.

If one person performs only one of these functions, however, it is much more difficult for either person to successfully effect a fraud for any length of time. The person making the accounts receivable entry to the customer's account would have no reason to intentionally omit an entry, since he would never receive a customer check. The cash receipts clerk might still be able to misappropriate a check; however, the clerk would not be able to remove the open receivable from the records. When it became delinquent and was investigated, the misappropriation would be detected.

This concept can be applied as readily to computer systems as to manual ones. However, computers are indiscriminate and will accept input from anyone, provided it meets all the criteria for acceptance. This is a different ground rule. In manual systems everything had to be passed through people. If a cash receipts clerk asked the ledger accountant to make an adjustment eliminating a receivable, it might raise some questions. However, unless procedural controls prevented it, a computer would not question an entry made by a cash receipts clerk to an accounts receivable master file.

If punched card input is used, persons receiving cash receipts and posting them to keypunch sheets should not have access to keypunch sheets used for other accounting entries. If on-line systems are used and entries are made through terminals, means of identification and authorization should prevent cash receipts clerks from entering other accounting data through the terminal.

While separation of responsibility is often maintained for the initial accounting entries in an operation, this control is lost if persons are given blanket approval to make corrections or adjustments. For example, in an accounts payable operation, it is important to ensure that receiving entries are made only by the receiving department, purchasing entries are made only by the purchasing department, and invoices are processed only by the accounts payable department. It is not unusual, however, to find that persons in each of these departments can correct or adjust any of this information. Corrections should be controlled as closely as original entries; otherwise, a theft or fraud could be easily concealed. Basically, controls should prevent someone from successfully perpetrating a fraud by himself or herself.

While accounts receivable and accounts payable operations illustrate the concept of separation of responsibility, the concept applies to all areas of business, especially those in which assets could be misappropriated. The more widely responsibilities are separated, the more people must be involved (that is, colluded with or duped) to perpetrate a fraud. As the number of people who must be involved increases, the probability of discovery increases, and the risk that fraud will be attempted decreases.

Not only does separation of responsibility prevent or lead to prompt detection of fraud, but it is equally important as a tool for detecting errors. For example, any error made by a cash receipts clerk in posting a receipt would normally be caught by an alert accountant upon noticing that the entry did not agree with the open receivable.

Resolving Exceptions

Exceptions are transactions rejected by the system because they did not pass one or more control points. Immediately, this indicates that the transactions are potential problems, and every effort should be made to correct and resolve them. It is not unusual, however, to find exceptions underestimated; many times they are put aside because of heavy workloads, or they may be ignored or deliberately overridden.

Some companies pay invoices when they are due whether the parts ordered have or have not been received. The system may then print out a listing of goods invoiced but not received. This listing is often put aside so that current invoice processing demands can be met.

Many times exceptions are cleared without adequate investigation to determine the exact reasons for the problem. Before they are resolved, they are at least in a pending state. Afterwards, they are normally no longer available for review.

Controls should ensure that rejections are corrected and reentered into the system. Sometimes rejection reports, cards, etc., are lost, and the corrected data is never reentered. Better systems keep open files of rejections and clear them only when they are reentered correctly. Also it usually is best to have rejections printed on separate reports rather than on reports containing other information. This makes them easier to isolate and correct.

Overriding Programmed Controls

Special efforts should be made to ensure that important controls written into computer programs cannot be indiscriminately overridden. For example, in most accounts payable systems, audit programs ensure that for each invoice entered, there is a corresponding authorized purchase order on file. In addition, the audit programs test whether purchase order unit price agrees with invoice quantity, etc. Override routines are written in the programs which allow an invoice to be paid without going through the audit tests. The special override codes are used to avoid investigating errors; as a result, an organization may lose control. Exception procedures should receive the closest scrutiny.

Programming Controls

Many of the controls formerly provided by people in manual systems are now provided by computer programs. When controls are maintained manually, it is relatively easy to divide responsibility for a complete transaction among a number of people. However, with critical controls often imbedded in the computer programs, a new internal control consideration arises. A programmer, through changes to program instructions only, can effect a successful fraud unless suitable safeguards are provided. Consequently, it is desirable that the responsibilities of writing, authorizing, modifying, and running programs be separated.

An accounts payable operation is again used as an example. Normally, in a manual system, an administrative person in the accounts payable department, before approving an invoice for payment, ensures that a corresponding purchase order and receiving ticket support it. He also verifies that information on all three documents agrees. In addition, the check signer, normally another person, inspects all three documents for agreement before signing the check.

Finally, an accouts payable auditor often reviews the whole package before the check is released--including the payee's name and address to verify that he is an authorized supplier. Three key control functions, performed by three different people, significantly limit the exposure to fraud.

However, in most automated accounts payable systems, one person seldom reviews all source documents. Purchasing and receiving documents are no longer sent to the accounts payable department. Instead, purchasing and receiving information is entered directly into the computer system and stored for future use. The invoice, upon receipt, is merely prepared for entry into the system by accounts payable. No longer does the payables department make a test for supporting purchase orders and receiving tickets. Instead, all control tests are made within the system by a computer program which compares invoice information to the previously stored purchasing and receiving information.

In companies where thousands of invoices are processed weekly, there is very little chance that an occasional fraudulent one will be detected visually. It is precisely these large-volume operations that expose an organization to programmer fraud--especially since they are the ones that are profitable to automate.

It is not unusual to find that one programmer or systems analyst has complete control and responsibility for a given system. He has easy access to programs, documentation, and master files and can make changes to programs without going through any approval procedures. In addition, a history is rarely kept of programming changes. It is this type of situation that creates vulnerability to fraud.

Establishing Programming Controls

The following steps could be taken to establish controls:

1. Determine which application could be fraudulently manipulated.
2. Identify critical programs within each of these applications.
3. Divide responsibility, where possible, for maintenance of these programs between two or more persons.
4. Review and approve new programs and changes to existing programs. Include review of the final program listing.
5. Establish controls to ensure that review and approval procedures are not bypassed. Following are some techniques that could be used:
    a. Control final program assembly so that only the approved program is installed.

b. Write a program to compare controlled duplicate object program decks to those in the computer room.

c. Establish predetermined hash totals for critical programs. These totals ensure that the program has not been changed.

d. If programs are kept on disk files, periodically compare them to a control copy. Differences may flag a possible unauthorized program change.

e. Include with output a listing of job control language to ensure that an unauthorized program has not been executed.

6. Maintain an audit trail of changes to programs.

Operations Control Groups

Some organizations have data processing control groups whose main function is to assure an efficient and effective workflow into and out of the computer room. These groups normally report to the operations manager. Their responsibilities include checking control totals before a job is returned to the user, scheduling jobs, resolving machine or operator errors, ensuring that new programs or program changes are approved by management, ensuring that job run documentation is complete, expediting important jobs, and so on. They provide a service to users, and exercise control for management.

Controlling Programs

Highlighted below are a few suggested considerations for controlling programs and data.

1. Hash totals, record counts, system summary totals, and control totals should all be monitored by someone other than the originator.

2. Test program decks, test documentation, and sample output from tests should be treated as securely as the program itself.

3. Management should screen requests for new application programs to determine their legitimacy. Management should also determine if the cumulative effect of multiple uses of the system by a given user yields more information than the user is entitled to have.

4. Documentation of programs should include a written record of all changes, reasons for the changes, dates they were made, authorizations, and cross references to other programs that might be affected by the changes. This record should be initiated by the person making each change and should be reviewed by management.

5. A review procedure should be established for monitoring the operation and input/output for programs that have been patched. The review procedure should take place before the program is authorized to reenter the production cycle.

6. An inventory of all tapes, disk files, programs, and supporting documents should be maintained. It should be reviewed and updated periodically or when specific changes occur. A physical audit of the inventory should be held periodically.

7. Production programs should be tested to verify that programs operate as specified.

8. Periodic dumps, traces, and transaction journals help provide an audit trail which, when couples with good physical security, defends against data loss by providing recent backups and helping to pinpoint the people or programs who had access to the data.

Network Security

Effective physical security can be established for the central facility, and, in most cases, an adequate degree of physical security needed for terminals can be accomplished. How to achieve a comparable level of protection for the common-carrier portion of the system remains a problem for system managers. One of the reasons some people think of data security only in relation to communications-oriented systems is that much of such a system cannot be under the physical control of the owner of the system.

The teleprocessing environment does introduce new risks to the system. Among them are exposure of capital equipment (terminals and terminal equipment) to loss, damage, misuse, or unauthorized use; unauthorized use of system time and resources from authorized terminals; unauthorized accumulation of communications line charges; exposure of the system to foreign (non-system) terminals dialing in; exposure of terminal messages to disclosure through wiretapping; or exposure of system data to alteration or manipulation by foreign terminals connected through wiretaps. Although the first hazards listed are contained by controlling security at the terminals themselves, protection against foreign terminals dialing into the system and against wiretapping is clearly associated with the communications network. Dialing in and using a system through any compatible terminal (located almost anywhere in the world) is wholly feasible, requiring no inconvenient, complicated, or illegal acts to connect into the system, as does wiretapping. Therefore, any system using a switched network and permitting the terminal to initiate communications should consider foreign terminals a valid threat and should protect against them.

In applications where there is active participation of an operator, identification of the operator may suffice, and determining which particular terminal is used may be of no consequence. Where intangible operator identification is used, such as a password which can be given to another person by telephone, being able to identify the terminal can offer additional protection. In applications where there is no interaction with an operator, such as an RJE terminal which is processing batched jobs, identification of the terminal identifies the valid system user.

The frequency with which terminals should be identified to the system is related to the job, the risks, the probability of substitution of another terminal, and the limitations of the terminals themselves. Some terminals will supply their identity codes only at sign-on; others will supply them whenever queried by the system. Terminals should be reidentified when the system detects, as it can, that communications have been interrupted. There is an unavoidable aberration of telephone switching which permits a new terminal to replace the previously-connected one at such times.

Wiretapping of data communications systems is technically feasible and therefore possible, but it is illegal and troublesome to accomplish. The opportunities for a passive wiretapper to merely monitor the line in the hope that something useful will go by seem poor enough in most applications.

There is a means of thwarting both the passive and the active wiretapper through cryptography or scrambling. Scramblers are available for high- or low-speed communications lines, and there are programmed algorithms for use when there is processing capability at both ends of the communications line. If scrambling is needed at all, the scrambling should offer reasonable secrecy. Rather than use a trivial or weak scheme, it is far better not to use scrambling at all, for then management will not be depending on it.

Terminal Security

Users of the terminals should be identified for every task or job performed. The minimum requirement is for accountability, but the intent is for positive identification of the operator to be used in the audit or authorization function that is employed. Signing on and identifying the job to be performed doesn't satisfy this need. A password or secrecy code meets this requirement only if it is a surrogate for the user; if it is a secret name for some system resource (for example, a data set), it cannot be used to establish user accountability or identity. The intent is to provide a datum equivalent for the user's signature, with the same affirmation by the signatory, and with some reasonable protection against duplication or forgery in the signature itself.

The ideal of using some personal characteristic is not economically feasible (authenticity of signature is proved by the way a person writes, not the particular sequence of characters).

If passwords are used, they should be assigned for the users and generated, recorded, and distributed in a controlled environment. A three- or four-character or a four-to-six-digit number should yield very ample protection against guessing or random generation, particularly if it must be entered in combination with an employee number (or other names or numbers which can be correlated with the password by the system or on subsequent audit). It is of questionable value to add the overhead of long security codes so long as the memory and discipline of users represent a much greater risk than the size of the set of all passwords.

System Controls and Procedures

The previous sections have inferred or stated several functions needed to provide network or terminal security. Among these are some that can be implemented only by the central facility:

- Identification of terminals for system and network integrity (and, with some terminals, for accountability as well)
- Scrambling programs, where needed
- Assignment and control of user identification
- Authorization and audit recording based on this identification

In addition, many of the controls discussed previously in connection with nonteleprocessing systems still apply, beginning with key verification of input. Record counts can be taken on entries made through a terminal. However, other methods just as effectively ensure that all transactions are received and processed by the computer. Message verification is one; sequential numbering of input transactions is another.

On-line systems involving money transactions can verify that dollar amounts are entered correctly by displaying them back to the terminal operator. For example, an accounts receivable system enters all customer remittances through terminals. After 50 checks are entered, the central computer prints each one on a terminal printer with the total dollar amount. The checks are matched manually to the listings to ensure correct entry. Edit, audit, and reasonableness tests can be made on terminal input just as on punched cards. An on-line system can facilitate correction of errors.

A teleprocessing system can store each transaction exactly as it is entered through the terminal on a tape or disk for future reference. In addition to including all the input data, the record could also include the time, date, and identification of the person entering the transactions (transaction log). Review of source transactions would then simply require printing out selected records from these tapes.

In a teleprocessing system the transaction log may also be used for recovery purposes in case a master file is destroyed and has to be reconstructed. It is usually easier and faster to reconstruct than to require personnel to reenter the transactions through terminals. Often, audit trail requirements can be included in these fallback and recovery tapes with minimal cost.

Index

A

access mechanisms 3-5
Access Method Services 9-24
access methods 9-22
access time 3-3
adder 4-5
address 1-13
alphameric 2-6
APL 2-33, 8-7, 8-10, 9-21
arithmetic/logical (unit) 1-12
ASCII 2-13, 6-5
ASCII-8 2-13
assembler 8-4
Assembler 9-18
assembler language 8-6
assemblers 8-1
asynchronous mode 6-5
audio response 1-3
audit trail 10-3
auxiliary (storage) 1-13, 3-2

B

backup 10-2
base displacement addressing 7-22
base register 7-22
BASIC 8-7
basic access technique 9-22
Basic Direct Access Method 9-23
Basic Telecommunications Access Method 9-26
batch 1-6, 3-11, 9-17, 9-23
batching 1-6
BCD 2-6, 2-8, 2-27, 5-12
BDAM 9-23
belt printer 5-22
binary 2-4, 2-5, 2-13
binary coded decimal 2-6, 2-8, 2-27
See also BCD
binary number system 2-5, 2-13
binary synchronous communications 1-17, 5-30, 6-5, 9-26
bisync see binary synchronous communications
bits 2-5
block 3-12
blocking factor 9-6
branch 4-7
BSC 6-5
BTAM 9-26
buffer 5-31, 6-1
byte 2-13
bytes 1-13

C

calculating 7-12
card code 2-21
card punch 2-21, 5-5
card reader 2-21, 5-4
cartridge storage cells 3-9
central processing unit (CPU) 1-12, 4-1
chaining 7-24
channel 5-1, 5-3
character reading 2-30
character recognition input units 5-24
check bit 2-8
See also parity check
checkpoint (procedure) 1-7
See also checkpoint/restart
checkpoint/restart 9-14, 9-16
See also checkpoint (procedure), restart (procedure)
chief programmer teams 8-18
CMS 9-19
COBOL 2-10, 8-7, 9-18
code 2-2, 2-5, 2-6
code checking 2-6
code translation 6-3
collating sequence 2-9, 7-19
command chaining 7-24
commands 5-3
communications control devices 6-2
communications line configuration 6-4
comparing 7-19
compatibility 9-13
compilers 8-2
conditional branch (structure) 8-15
console 1-16, 5-28
control area 9-24
control interval 9-24
control program 1-11, 9-5, 9-10
control section (CPU) 1-12, 4-1
control storage 1-13
control total 10-6
control unit 5-1, 5-2
See also control section (CPU)
controlling programs 10-11
controls 10-3
conversational 9-19
See also interactive
core 1-6, 3-2
See also magnetic core
counter 4-5
CPU see central processing unit
cylinder 3-5

D

DASD 2-1, 5-16
See also direct access
data acquisition 6-7
data base 1-8, 9-3
data buffering 5-30
data cartridges 3-9
data chaining 7-24
data communication 1-8, 2-26
data controls 10-3
data entry 5-28, 6-7
data link 6-2
data link control 6-3
data management 9-6, 9-11
data module 3-7
data organization 9-6
data recording media 2-21
data representation 2-1
data security 10-1

data sets (files) 3-11
data sets (communications) 6-3
data transmission multiplexers 6-2
debugging 9-18
See also program checkout, test, testing technique
decision techniques 7-9
demodulated 6-2, 6-3
deserialized 6-3
detail diagram (HIPO) 8-20
direct access 1-7, 2-1, 2-2, 3-2, 3-5, 3-11, 5-16
disk (storage) 1-7, 1-8, 1-17, 1-18, 2-3, 3-1, 3-2, 3-5
Disk Operating System/Virtual Storage see DOS/VS
diskette 1-18, 5-6
displacement 7-22
display 1-16, 1-18, 2-1, 2-3, 2-32
distributed function 1-8
See also function distribution
DOS/VS 9-5, 9-26
DOUNTIL (structure) 8-16
DOWHILE (structure) 8-15
dual-density 2-27
dump 8-14
duplex transmission 6-5

E

EBCDIC 2-11, 2-20, 2-29, 5-11, 6-5
edit routines 10-4
eight-bit (alphameric) code 2-11, 2-13, 2-28, 2-29
eight-channel code 2-24
electrographic printer 5-18
emulator 4-1, 9-13
end-of-block 5-13
end-of-file 5-14
end-of-reel 5-15
excess-64 notation 4-10
execution cycle 4-7
extended binary coded decimal interchange code 2-11
See also EBCDIC
extended description section (HIPO) 8-20

F

field 7-12
file protection 5-16
five-channel code 2-25
fixed head storage 3-7
fixed-count check 2-7
fixed-length words 4-9
floating-point (operation__ 4-9
flowchart 7-3
flowcharting worksheet 7-7
FORTRAN 2-10, 8-7, 8-8, 9-18
fraction conversion 2-17
full duplex 6-4
function distribution 9-28
See also distributed function

G

graphic display 5-28
See display
graphic subset 2-10

H

half-duplex 6-4
half-word 7-2
hexadecimal (number system) 2-13
hexadecimal floating-point see floating-point
HIPO 8-18
host processor 6-2
housekeeping checks 10-6

I

IFTHENELSE (structure) 8-16
image processing 1-3
impact printer 5-21
improved programming technologies 8-14
Indexed Sequential Access Method 9-25
indexing 7-20
indicators 5-4
input 1-9
input controls 10-3
input device 1-10, 1-13, 2-2, 5-2
input/output 2-2
input/output control systems 9-1
See also IOCS
input/output devices 1-14, 1-16, 2-3, 2-11, 5-2
See also input device
inspections 8-22
instruction cycle 4-6, 7-1
instructions 5-3, 7-1
integer conversion 2-14
integrated communications adapter 6-2
integration (test) 8-17
interactive 1-18, 9-27
interblock gap 2-27, 5-13
interpreters 8-1, 8-11
interrupt 5-3
IOCS 9-1, 9-28
ISAM 9-24

J

JCL 9-20
JES 9-12
Job Entry Subsystem 9-12
job management 9-10, 9-15

K

keys 5-4

L

labels 10-5
language translation 8-4
See also language translator programs
language translator programs 8-1
See also language translation
libraries 9-5
light pen 1-8, 1-13
limit check 10-5
line adapters 6-3
line discipline 6-5
on-line (program) development 9-18

line printer 5-21
linkage editor 8-4, 9-10
linking 7-23
load module see load program
load program 8-4
load-point 5-15
local 2-33
logical operations 4-1, 7-14
loop 7-14, 7-21

M

machine cycles 4-5
machine language 8-2
machine-oriented programming languages 8-5
macro definition 8-7
macro instructions 8-2, 8-6
macro library 8-7
magnetic bubbles 1-8
magnetic character 2-2
  See also magnetic ink characters
magnetic character reader 5-25
magnetic core 1-6, 2-3, 4-2
  See also DASD, direct access, disk (storage)
magnetic disk storage 3-5
magnetic drum storage 1-6
magnetic ink characters 2-1, 2-3, 2-30
  See also magnetic character
magnetic tape 1-5, 1-13, 2-1, 2-2, 2-3, 2-26, 2-27, 2-28, 5-8
main storage 1-13, 3-2, 3-4
maintenance see program maintenance
mass storage 3-10
MCS 9-14
  See also multiple console support
memory 3-4
  See also main storage
message switching 6-7
MF/1 9-16
microfilm 5-2
microprograms 1-13
microsecond 4-5
millisecond 45
mnemonics 8-2
modems 6-2
  See also modulation/demodulation (devices)
modulated 6-3
modulation/demodulation (devices) 6-3
module 8-4
monitor program see control program
monolithic 3-4
multi-multipoint 6-6
multiple console support 9-14, 9-15
multipoint 6-4
multiprocessing 1-11
multiprogramming 1-11, 9-7
MVS 9-16

N

nanosecond 4-5
NCP/VS 9-28
network control program 9-28
network security 10-12
network 2-1, 6-1
network(s) 1-8
nine-track magnetic tape 2-28
nonimpact printer 5-21
nonswitched lines 6-4
normalized floating-point number 4-10
NRZI 5-12
number systems 2-13
numeric bits 2-7

O

object module see object program
object program 8-4
off-line 5-8, 5-17
on-line 1-8, 1-11, 9-4, 9-5
on-line (program) development 9-18
operand 4-6, 7-2
operating systems 9-2
operation 4-6
operations control 10-11
optical reader 2-31, 5-27
organizational controls 10-8
OS/VS1 9-10
OS/VS2 9-14
output device 1-14, 2-3
  See also input/output devices
overflow 4-3
overview diagram (HIPO) 8-19

P

pack 1-13
  See also packed decimal
pack, disk see disk
packed decimal 2-6
page data set 9-8
page fault 9-10
page frame 9-9
page in 9-10
page supervisor 9-11
paging 9-8
paging out 9-10
paper tape 1-5, 1-14, 2-1, 2-2, 2-3, 2-24, 5-17
paper tape punch 2-24, 5-18
paper tape reader 2-24, 5-17
parallel operation 4-8
parity check 2-7, 2-28
partition 9-7
password 9-17, 10-13
phase encoding 5-12
PL/I 8-7, 8-9, 9-18
point-to-point 6-4
POWER/VS 9-6
primitive functions 8-10
printer 1-16, 2-3, 5-18
printing subsystem 5-18
problem programs 1-11
problem-oriented languages 8-7
procedural controls 10-7
process control 6-7
processing controls 10-4
processing programs 1-11, 9-5
program 1-10, 7-1
program checkout 8-13, 9-18
  See also test
program flowchart 7-7
program loop 7-15
  See also loop
program maintenance 8-14
program module see module
program preparation 7-3, 8-1

programming controls 10-4
programming languages 8-1, 9-18
programming systems 9-1
punched card 1-4, 2-1, 2-21, 5-4

Q

punched paper tape see paper tape
QTAM 9-26
queued access technique 9-22
Queued Telecommunications Access Method 9-26

R

random access 1-6, 3-4
See also direct access
RAS 9-11, 9-27
read 1-13
read-only storage 4-1
read/write head 2-26, 3-5, 5-10
real address area 9-9
real storage 1-13, 9-3
reasonable check 10-5
record count 10-4
recovery 10-2
recovery management 9-10
region 9-3
register 4-2
remote job entry 6-7, 9-6
repetition of an operation while (DOWHILE structure) 8-15
report program generator see RPG
restart (procedure) 10-7
See also checkpoint/restart
RJE 9-6
See also remote job entry
rotational position sensing 3-6
RPG 8-7, 8-12

S

SAM 9-23
scanner 2-31
SCP 9-10
See also DOS/VS, MVS, OS/VS1, SVS, VM/370
SDLC 6-5, 9-26
secondary storage see auxiliary storage
See also synchronous data link control
security 10-2
semiconductor 1-7, 3-3
sequence checking 7-20, 10-6
sequence (structure) 8-15
sequential 3-2
sequential access method 9-23
serial operation 4-8
serial start-stop mode see start-stop mode
serial wire matrix printer 5-23
serialized 6-2
service aids 9-12
seven-track 5-11
seven-track magnetic tape 2-27
shared DASD 9-13
shared virtual area 9-7
shifting 4-3
six-bit alphameric code 2-7
SMF 9-14, 9-16
snapshot 8-14
solid logic technology 1-7
source program 8-4
spooling 9-12
staging 3-9
start-stop mode 6-5
start/stop terminals 5-30
storage 1-6, 1-8, 1-10, 1-12
storage devices 3-1
storage printout 8-14
stored program see program
structured programming 8-14
structured walk-through 8-21
subroutine 5-3
subvoice-grade 6-4
supervisor 9-11
supervisory program see control program
SVA 9-8
SVS 9-15
switched lines 6-4
switches 5-4
symbolic languages 8-2
synchronization 6-3
synchronous data link control 5-30, 6-5
See also SDLC
synchronous mode 6-5
system control program 9-10
See also DOS/VS, MVS, OS/VS1, SVS, VM/370
system controls 10-13
system flowchart 7-5
system generation 9-7, 9-21
system integrity 9-16
system/3 1-17
system/3 card 2-22
system/32 1-18
System Management Facilities 9-16
See also SMF
system resources manager 9-16

T

tape 2-3, 3-2
See also magnetic tape
tape punch see paper tape punch
tapemark 5-13
See also end-of-file
task management 9-11
TCAM 9-28
Telecommunications Access Methods (TCAM) 9-25, 9-28
teleprocessing 1-1, 1-14, 2-7, 5-30, 6-1, 9-3, 9-6, 9-23, 10-12
template 7-3
terminal security 10-13
terminals 1-8, 5-30
test 9-18, 10-12
testing techniques 8-13
See also debugging, program checkout, test
throughput 5-3
thruput see throughput
time sharing 1-11, 6-7
Time Sharing Option 9-17
time-slicing 9-14
top-down program development 8-17
tracing 8-14
tracks 3-5
train printer 5-22
transter see branch
transistors 3-4, 4-2

translator 8-4
See also language translator programs
transmission direction 6-4
transmission mode 6-5
transparency 6-5
TSO 9-17
turnaround documents 2-31

U

unit record 2-21
unit test 8-17
universal product code 2-2, 2-32

V

validity checks 5-4
variable-length words 4-9
virtual I/O 9-16
Virtual Machine Facility/370 9-18
Virtual Machine Operating System 9-19
virtual mode 9-10
virtual storage 9-8, 9-11, 9-12
Virtual Storage Access Method 9-24
See also VSAM
Virtual Telecommunications Access Method 9-26
visual display 2-32, 5-29
See also display
visual output 2-32
See also display
visual table of contents (HIPO) 8-19
VM/370 9-18
voice grade 6-4
VSAM 9-27
VS1
VTAM 9-26

W

wait state 9-2
word 4-9, 7-2
workspace 8-11
write 1-14

Z

zone 2-21, 2-23
zone bits 2-7

5100 Portable Computer 1-18
96-column card 2-22

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)

GC20-1684-4

Introduction to IBM Data Processing Systems Printed in USA GC20-1684-4