ED 019 855   EM 006 161

COMPUTER FACILITIES FOR MATHEMATICS INSTRUCTION.

NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS INC.
EDRS PRICE MF-$0.25 HC NOT AVAILABLE FROM EDRS.   51P.

DESCRIPTORS- MATHEMATICS INSTRUCTION, *SECONDARY SCHOOL MATHEMATICS, EDUCATIONAL EQUIPMENT, *COST EFFECTIVENESS, *SYSTEMS APPROACH, *COMPUTER ASSISTED INSTRUCTION, *TIME SHARING, ALGORITHMS, COMPUTER SCIENCE, ELECTRONIC EQUIPMENT, COMPARATIVE ANALYSIS, ALGOL, FORTRAN, TELETYPE

THE EFFECTIVE AND ECONOMICAL USE OF COMPUTER FACILITIES AS AN AID IN TEACHING SECONDARY SCHOOL MATHEMATICS DEPENDS ON (1) ACCESS TO A TYPE OF FACILITY WHICH PERMITS PROBLEM-SOLVING PROCEDURES (I.E., ALGORITHMS DESIGNED BY A STUDENT) TO BE TESTED BY THE STUDENT AND (2) FACILITY RESPONSE TIME PER PROBLEM. THIS UNIT TIME VARIES WITH THE SIZE, TYPE, AND COST OF AVAILABLE COMPUTER FACILITIES. FOR VERY LOW COST AND SMALL SIZE FACILITIES, THE USE OF ALGORITHMIC LANGUAGES (E.G., ALGOL AND FORTRAN) IS SACRIFICED, RESULTING IN GADGET-ORIENTED, RATHER THAN MATHEMATICS-ORIENTED, STUDENTS. AT THE OTHER EXTREME, FACILITIES RICH IN SOFTWARE (I.E., INTERNAL CAPACITIES TO ENHANCE ALGORITHMIC COMMUNICATIONS) ARE ALSO EXPENSIVE AND OF LIMITED AVAILABILITY TO SECONDARY SCHOOL STUDENTS. TO OBTAIN SOME QUANTITATIVE INFORMATION ON THE EFFICIENCY OF COMPUTER FACILITIES, FIVE TYPICAL PROBLEMS WERE RUN ON SIX TYPES OF COMPUTER FACILITIES. THE TIME TO RUN THE SET OF FIVE PROBLEMS VARIED FROM ABOUT 90 MINUTES TO 3 MINUTES. THE 3-MINUTE SYSTEM WAS A LARGE-SCALE FACILITY LOCATED AT A UNIVERSITY. OF SIGNIFICANT INTEREST WAS THE 20-MINUTE SYSTEM, WHICH CONSISTED OF TELETYPE ACCESS TO A LARGE-SCALE, TIME-SHARING SYSTEM. AT THE PRESENT TIME, SOME FORM OF INDIRECT ACCESS IS THE LEAST EXPENSIVE AND MOST EFFICIENT WAY TO USE COMPUTERS AS INSTRUCTIONAL TOOLS IN TEACHING MATHEMATICS. THIS DOCUMENT IS AVAILABLE FOR $0.90 FROM NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS, 1201 SIXTEENTH ST., N.W., WASHINGTON, D.C. 20036. (CS)

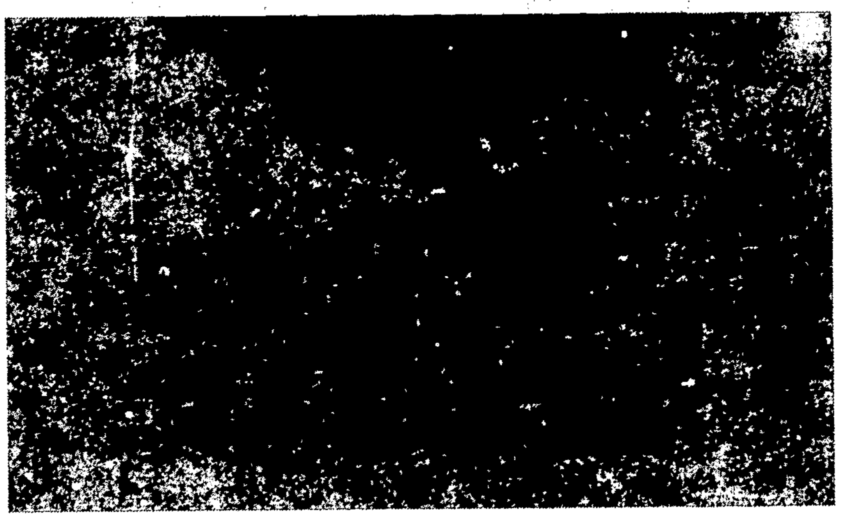
Level I

EM 006 161

COMPUTER FACILITIES FOR MATHEMATICS INSTRUCTION

NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS

MATHEMATICS INSTRUCTION.



Acknowledgments to
Tom Kurtz
Bruce Thompson

Library of Congress
Catalog Card Number
67-29896

Table of Contents

oduction

COMPUTER-ORIENTED educational programs are appearing with increasing frequency in secondary schools throughout the United States. Computers are used for instructional purposes, already, in hundreds of schools. Existing programs are growing rapidly, and many new programs are being initiated. All levels of secondary education are involved.

The rapid growth of computer education in the secondary schools has created a demand for information on educational uses of computers, instructional areas, types of computer systems, methods of starting and operating a computer-oriented educational program, and the costs of such a program.

This booklet is written to provide this type of information for the teacher or administrator who is considering methods of introducing computer topics or courses on computers into the secondary school curricula. It is assumed that the person who reads this booklet is *not* a complete novice in computer concepts and utilization. A sufficient background can be acquired by reading an introductory text on computer methods. In particular, the authors recommend the books *Computer Oriented Mathematics*, a publication of the National Council of Teachers of Mathematics, and *Information*, a Scientific American book published by W. H. Freeman and Company.

I[illegible]uctional Uses [illegible]omputers

COMPUTERS are at work in hundreds of school systems. The use of computers for administrative data processing, record keeping, counseling, and research is commonplace and well developed. Although the use of computers in the instructional area is not yet so widespread, it is expanding at an explosive rate.

Most of the computer-oriented educational programs that are currently under way fall into one of the following instructional categories.

1. Vocational education in computers

This is computer education, per se. The computer and its applications are the primary objects of instruction. At the post-high-school level, the intent of this type of education is to prepare people for professional or semiprofessional careers in the computer field. Secondary school courses are *not* adequate preparation for these careers; however, they can provide guidance and counseling information for students who are considering a post-high-school course of study in some area of vocational computer education. For example, prevocational courses may be offered in the secondary school for students interested in careers as computer maintenance technicians or as business data-processing programmers.

2. Topics in "computer culture"

There is a great deal of interest, curiosity, and superstition about computers among both schoolchildren and the general public. Hence it is appropriate to include in existing courses, at all grade levels, topics about computers and the impact of computers on our society. Short computer-oriented topics can be taught in existing courses in the secondary school and perhaps in the elementary school as well. These topics can be incorporated into many courses in a way that reinforces and enriches the subject matter of the course itself. Obvious areas are mathematics, the physical sciences, and business. Meaningful units could be developed also in the biological and social sciences.

3. Computer-assisted problem solving

One of the most widespread areas of computer application is the use of a computer as a problem-solving tool. This subject can be taught at all grade levels in the secondary school. It is frequently taught as an integral part of an existing course. Students use the computer to help solve problems pertinent to the subject matter of the course. This permits solving problems of greater scope and complexity than could be handled otherwise.

There is evidence that computer-assisted problem solving can be used effectively as an instructional aid in teaching mathematics, science, economics, business, and other noncomputer subjects. In mathematics, for example, a computer can be used to demonstrate concepts, provide a laboratory for mathematical experimentation and discovery, and provide an environment in which students can put into practice the mathematics taught in the classroom. It provides a practical means for testing problem-solving procedures (algorithms) designed by students.

4. Computers as instructional aids

A few experimental systems have been developed in which computers either are used directly as teaching machines or provide various types of direct assistance to the classroom teacher. Some of the ways in which systems of this type are used are listed below.

Preparation of instructional material

Reinforcement of instruction by means of simulation of physical, social, economic, political, and business systems (for example, by playing computer-based economic or political games)

Presentation of programmed instructional stimuli to students by means of terminals connected to, and controlled by, the computer

Acquisition and storage of responses made by students

Retrieval and presentation of information stored within the system upon request of the teacher or student

Today, systems of this type are highly experimental and very expensive. Furthermore, well-designed instructional material for computer-controlled presentation (the third item) is almost nonexistent. However, a great deal of developmental work is under way, and within a few years computers may become very economical and useful teaching assistants.

Although computer usage for instructional purposes is growing rapidly, very little research has been done to identify what to teach and how to teach it and to evaluate the impact of the computer on secondary education. Furthermore, many pilot projects have been hampered by lack of suitable instructional materials and by computer facilities that are inadequate and poorly designed for educational use. In this booklet our primary objectives will be the description of various types of computer facilities and the economic, efficient, and practical use of these facilities in operating a computer-oriented educational program.

Getting Started

Since so many secondary schools are responding to the impact of computers, the following general suggestions are offered.[1]

1. The most important consideration in developing a computer-oriented educational program is to be sure that one has an enthusiastic and knowledgeable person to direct the computer activity.

One of his first steps should be to interest teachers in many departments, as well as administrators, in their potential use of the computer facility. He should also offer to give in-service courses to train these persons. It is desirable for at least one person in each department to be familiar with the use of computational facilities.

2. The selection of an adequate computer facility is probably the next most important point to be considered in developing an academic program. This, of course, may be a function of the amount of money available. Under any circumstances, however, a *stored-program computer* is the only type that should be considered. It is desirable that the computer facility be able to process programs written in an algorithmic language such as ALGOL, BASIC, FORTRAN, or JOSS. Computer systems and languages are described in Chapter 3 of this booklet.

3. The person who directs the computer activities and those who work closely with him should make early detailed plans for the use of the

[1] Most of the information in this section is taken from the article "Computers for School Mathematics," by Walter Hoffman *et al., The Mathematics Teacher*, LVIII (May 1965), 393–401.

computer. This includes becoming familiar with the library of programs available for the computer as well as having a number of applications prepared ahead of time.

4. The computer facility should most certainly be available to all school departments and should be put to considerable use at least in mathematics, physics, chemistry, and business courses. There should be continued effort to develop computer applications that are nontrivial and meaningful in terms of the content of the school curriculum.

5. Another consideration is to be sure that the computer facility is easily accessible to all potential users. If it is on the school premises, it should be centrally located in the school complex. If the computer facility used is not at the school, the method of handling and transporting computer programs must be as simple and expeditious as possible. It is important to make it easy for the student to get his computer work done.

6. In the development of computer activity it is wise to seek the advice of knowledgeable people who have been in the computer field for a number of years. They can be helpful both in the selection of equipment and in offering suggestions about curriculum. Such advisers can usually be found in a nearby college or university or in successful industrial or governmental computer installations.

Computer Facilities for School Use

Some computer-oriented topics can be taught without the use of a computer. However, the real benefits of a school program of computer education can be obtained only if teachers and students have access to a computer facility. At present, computers and computer facilities that are well designed to meet the objectives of secondary education are extremely rare. In many existing programs, educational objectives have necessarily been modified in order to make use of computer facilities that are available.

Devices other than true computers are being used to teach computer-oriented concepts, computation skills, and problem solving. These include computer simulators, adding machines, calculators, logic trainers, digital trainers.

The above devices are of limited use (for example, none of them can process algorithmic-language programs) and are not discussed in detail in this booklet. Instead, we will concentrate on the use of small stored-program computers and centralized computer systems that provide for processing of student-written procedures.

A school that has decided to include computer instruction in its curriculum can arrange for computer processing of student programs by a number of different methods. We shall consider three ways in which this can be done:

Direct access is provided by transporting students to a computer center, "hands-on" execution of programs.

2. *Indirect access.*—Student programs are sent to a computer center for processing, and the results are returned to the student.

3. *Time-shared access.*—Students communicate with a time-sharing computer system by means of a terminal that operates over a telephone line.

Direct access is provided by transporting students to a computer center, by installing a computer in the school, or by bringing a small portable computer into the classroom.

Since it is not really important to teach students to be skilled computer *operators*, the objectives of computer-oriented educational programs can frequently be achieved by providing the means for indirect use of a computer located at a computer center. Access to the computer can be by mail, by courier, or by transmission over a telephone line. In this type of access, an important consideration is *turn-around time;* this is the elapsed time between requesting the use of the computer and receiving the results.

Four types of computer centers are listed below.

1. *School district computer center.*—Many school districts already have computers for administrative data processing which could be used also for instructional purposes. In many installations, however, the administrative load expands to the maximum capacity of the computer system; hence, instructional use may be very limited.

2. *Commercial centers.*—Computer time can be purchased in small units from commercial computer centers.

3. *College and university centers.*—Several colleges and universities provide computer services for secondary school instructional use.

4. *Educational computer centers.*—The ideal facility for educational use is a center set up and designed specifically for educational purposes. Centers of this type are already in the planning stage and will probably be available soon in many areas.

In Chapter 3 we will describe several types of computer facilities and discuss methods of access.

II

s in Problem Solving

MORE THAN 45,000 computers are in use in the United States, dealing with problems encountered in almost every area of human endeavor. Computers range in size from small machines that can solve only a few small problems in a day to large-scale computer systems that can provide solutions to thousands of small problems or a few very large problems in a day.

Many methods of computer operation have been developed. On small computers, the methods of operation are limited by the relatively small capacity of the computer; on large-scale computers, powerful and efficient operating methods are available. In the educational use of computers, the type of computer and the method of operation must be matched to the educational objectives desired and to the economics of operating a program of computer-mediated education.

In order to evaluate the relative merits of various computers and methods of computer operation, it is necessary to understand the manner in which a problem is solved on a computer. The process of solving a problem on a computer should consist of the following steps:

1. Definition of the problem to be solved
2. Analysis of the problem
3. Development of a computer program
4. Input of the program into the computer
5. Execution of the program with test data
6. Interpretation of the trial execution

7. Corrections and revisions of the program
8. Documentation of the problem-solving procedure

These basic steps are outlined in Figure 2–1, on page 9. The diagram is an example of a *flow chart,* or *block diagram.*

Definition of the Problem

The problem to be solved must be clearly identified and precisely stated. This requirement seems obvious, but it is so often violated in practice that we emphasize it here. *The user must have a clear idea of the problem that he wants to solve and of the results he wants to obtain.*

For example, consider the following definition of a problem to be solved by use of a computer: "Compute the roots of the equation $ax^2 + bx + c = 0$, where a, b, and c are input data."

This is an example of an incomplete problem definition. Unanswered questions include the following:

1. How are a, b, and c defined? Are they integers? Rational numbers? Real numbers? Complex numbers?

2. If a, b, and c are, say, integers, are only integer roots to be computed? Approximations to real roots? Complex roots? If the solution set is empty ($a = b = 0$ and $c \neq 0$), how should this be indicated?

3. What should be done to analyze errors due to approximate arithmetic in the computer?

A more complete problem definition is this: "Let a, b, and c be integers in the range ($-1{,}000 \leqslant a, b, c \leqslant 1{,}000$). Develop a program to compute the rational roots of the equation $ax^2 + bx + c = 0$. If the solution set is not empty, print a, b, c, and the root or roots. If the solution set is empty, print a, b, and c."

Analysis of the Problem

Analysis of a problem consists of the development of a mathematical model for the solution. The analysis includes a consideration of what information is given, what information is desired, and what mathematical techniques are required to transform the known data into the desired results. In order to develop a general procedure, the analyst must consider the different circumstances and cases that might occur and provide for them. (For example, in the "general" solution of the equation $ax^2 + bx + c = 0$, he must provide for cases such as $a = 0$, $b^2 - 4ac < 0$, and so on.) It is essential that he understand the role of the computer in developing the solution. He should be aware of the limitations and pitfalls inherent in computer-assisted problem solving, as well as the advantages and conveniences.

There are usually several possible mathematical formulations of a given program. Hence part of the analysis may consist of deciding on a "best" formulation in order to achieve some definite objective such as minimum computer time, minimum memory space, maximum precision of results, or most straightforward (easiest to understand) solution.

Development of the Program

Programming is the development and precise statement of a step-by-step procedure (algorithm) for solving a problem on a computer. It is the process of explicitly stating each step of a problem-solving procedure. The complete set of steps is called a program.

The algorithm to solve a given problem must be designed in terms of operations that can be performed by the computer system to be used. The algorithm is frequently first prepared in the form of a flow chart that displays the overall structure of the procedure.

A flow chart is a pictorial representation of the program. Steps in the program are enclosed in rectangular boxes, diamonds, circles, and various other geometric figures. Each of these figures represents a step or set of steps in the solution of the problem. The boxes are connected by directed lines that specify the sequence or "flow" of the steps in the procedure. Flow-charting methods and conventions are described more fully in *Computer Oriented Mathematics* [1] and *Algorithms, Computation and Mathematics.*[2]

Since "reading" flow charts is not within the powers of present-day computers, the user must express his program in a form that can be utilized by the computer. Just as we communicate with other human beings through a language such as English, German, or French, we may communicate with a computer in a computer language. The language in which a program is expressed may be the built-in language of the computer (machine language) or a "problem-oriented" language that can be translated into machine language by the use of the computer itself. ALGOL (*ALGO*rithmic *L*anguage) and FORTRAN (*FOR*mula *TRAN*slator) are examples of problem-oriented languages, which will be discussed in Chapter 3.

The statement of a program in computer language is called a *coded*

[1] National Council of Teachers of Mathematics (Washington, D.C.: The Council, 1963).

[2] School Mathematics Study Group (Stanford, Calif.: Stanford University, 1965).

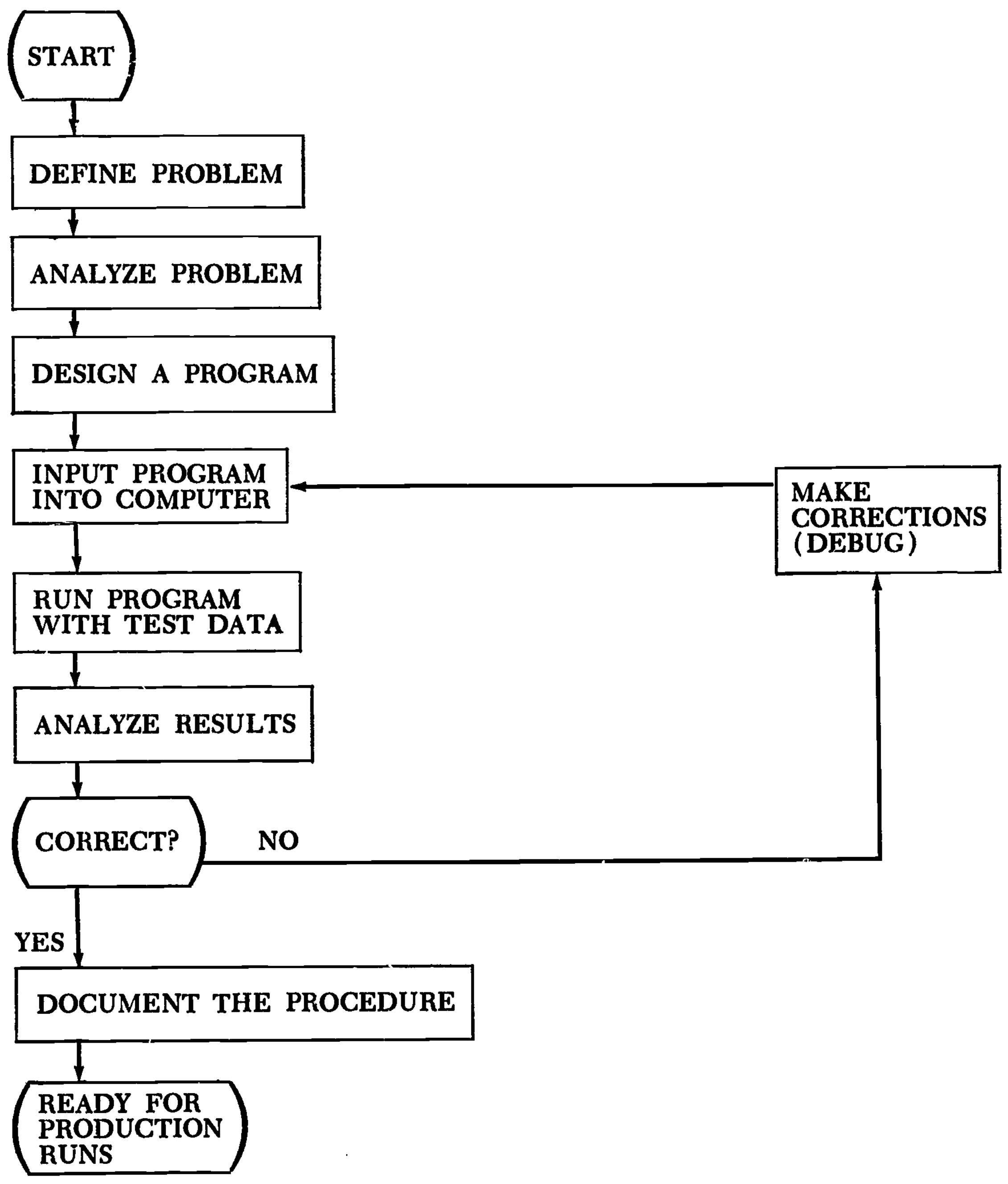


FIG. 2–1.—Flow Chart of Computer-Assisted Problem Solving

program, or *routine.* The next step is to test the coded program in the computer, using input data for which correct results have been determined in advance.

Input of the Program into the Computer

In order to run (or execute) the coded program on a computer, the user must first enter it into the computer. The code may be entered manually by means of a keyboard (e.g., typewriter) attached directly to the computer. This method is commonly used on small computers. A code to be entered into a larger computer is usually transcribed onto a computer input medium, such as punched cards or magnetic tape, and read in automatically.

If the program is written in a language such as ALGOL or FORTRAN, it must be translated into the language suitable for the computer that is being used. This is accomplished by the use of a special computer program, called a compiler, that is provided as part of the complete computer system.

Execution of the Program with Test Data

After the coded program has been entered into the computer, it is executed (run automatically), using input data for which the correct results are known. The input data may be entered by means of a keyboard or on punched cards, magnetic tape, or other input media, as determined by the program.

The results of the test run are made available on a computer output medium, such as a listing printed on a typewriter or printer, or on punched cards.

Interpretation of the Trial Run

The results produced by the computer during the trial run are examined to determine if the run was successful. That is, the results produced by the computer are compared with the results known to be correct for the data used in the trial run. If the computer results are correct (within acceptable tolerances) it may be assumed that the procedure is correct along the "paths" tested by the data.

Most computer procedures have one or more branch points at which an alternate route is selected through the procedure; hence, there may be two or more distinct subsequences of operations possible in solving the problem for different sets of input data. Care must be taken in selecting input data that all the possible paths through the program will be tested.

Debugging (Correcting Errors in a Program)

In using a computer, the new user is often shocked when programs do not work out correctly the first time. (Experienced users are shocked when the program *does* run correctly the first time!) A typing or punching error can cause the entire program to fail. Also, an error in the original analysis of the problem or the design of the program may result in the computer's producing answers that bear no relation to the original problem.

As the user becomes more experienced, the process of *debugging* (finding and correcting errors) becomes an easier task. The process tends to be highly individualistic, with some users being able to develop great skill in locating errors. The following general principles may be helpful.

1. Proofread the program carefully.
2. Perform hand calculations that follow the computer program.
3. Rerun the program with additional instructions that direct the computer to print the results of intermediate calculations.

Documentation of the Problem-Solving Procedure

Much of the problem-solving use of computers is "one-shot" (the procedure is used once and thrown away) or of interest only to a single person. Frequently, however, a problem-solving procedure is developed that is sufficiently useful that it will be used again. Occasionally a procedure is developed that could be of use to a great many people. In this event, the procedure should be documented so that it can be easily used again by both the originator and others who might find the procedure useful. A report should be written that describes the problem, the analysis of the problem, the problem-solving procedure, and the method of using the procedure on the computer, and includes the results of test runs that illustrate features or limitations of the procedure. This report should be clear, concise, and complete. When this has been done, the procedure is ready for production use by the originator and others who have the same type of problem to solve.

III puter Systems

FREQUENTLY we tend to think of a computer system primarily in terms of the physical devices that comprise it: the transistors, magnetic cores, diodes, indicator lights, control levers, input units, and other items of *hardware*. We consider the size and speed of the memory, the number of operations per second, the facilities for input and output, and the richness of the repertory of instructions that the system can execute.

The hardware, however, is only one of several elements in a general-purpose computer system. An equally important element is the set of computer programs designed to make the computer do what we want it to do. These programs are collectively called *software*.

There are two basic types of software. One type is designed to make the computer system easier to use. It determines what programming languages and what methods of operation can be used on the system. This category includes *utility routines*, *interpreters*, *assemblers*, *compilers*, and *monitors*. These will be described later in this chapter.

The second category of software consists of programs designed to use the computer as a problem-solving tool in mathematics, science, and other areas of application. For example, in mathematics we might write programs to use a computer to evaluate functions, find roots of equations, generate sequences, invert matrices, and compute statistics. Some of these programs are used only once or twice and then discarded; others are of general utility and are used repeatedly. Some routines are used as building blocks, called *subroutines*, in larger programs. Routines that are used frequently are placed into the software *library* of the computer system. The library is the set of all programs that are always available to users of the system. The libraries of two computer systems may be different, even though both systems use the same kind of hardware.

A third element in a computer system is the set of *instructional materials* for using the system. This includes written procedures for using the hardware and software and the textbooks, workbooks, reference manuals, and other educational material for training people in use of the computer. This element is particularly important when a computer is used in the secondary school.

To repeat—a general purpose computer system consists of three elements:

1. Hardware
2. Software
3. Instructional materials

In Chapter 2 we listed and described eight steps in solving a problem on a computer. These steps are shown below in abbreviated form.

1. Define the problem.
2. Analyze the problem.
3. Develop a program.
4. Input the program into the computer.
5. Execute the program with test data.
6. Interpret the results.
7. If necessary, debug the program.
8. Document the procedure.

Consideration of the particular computer system to be used may be necessary as early as Step 2. For example, the analysis may indicate that the problem is too large or too complicated to be run on the computer system that is available. In this event, the user must reduce the size of the problem or try to obtain access to a larger computer system.

In Steps 3, 4, and 5, the system is clearly involved. The program must be written in a language acceptable to the system, then entered into the system and executed. Hence, the user must be aware of the programming language or languages available on the system and must conform to the conventions and procedures for using the system.

The proper interpretation of the results of the test run (Step 6) may depend in part on the user's knowledge of the computer system. This is particularly true if there are errors and debugging is required. Errors may be due to a faulty analysis, an incorrect program, or possibly a peculiarity of the computer system itself (for example, *round-off error* due to approximate arithmetic with numerals of fixed length). Occasionally errors occur because the user does not fully understand the characteristics of the programming language (e.g., FORTRAN) that he is using. If there are errors, the user may use the system again to assist him in debugging his program.

The rest of this chapter consists of a more detailed description of computer systems and methods of communication with computer systems. We assume that the reader is already acquainted with the general concept of a stored-program computer and with the writing of a program in some computer language. The intent of this chapter is to describe several types of computer systems and several methods of using computer systems as instructional tools in educational programs.

Minimum Computer System

The most basic computer configuration consists of a computer with manual input and output. The input device may be a set of push buttons on the control panel of the computer or a numeric keyboard similar to the keyboard on an adding machine. Output is obtained by stopping the computer and displaying results by lights or dials on the control panel. In order to obtain a permanent record of the results, the user must record the results by hand. Figure 3–1 is a diagram of this configuration.

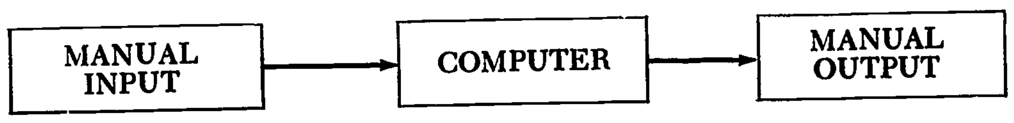


FIG. 3–1.—Minimum Computer Configuration

A system of this type is usually programmed directly in *machine language*. In developing a program, the user must prepare a list of instructions coded directly in the built-in language of the computer. Hence the program usually appears as a list of numerals, each numeral representing one instruction.

The coded program is entered into the computer by means of the manual input device. The user sits at the control panel and enters the instructions, one at a time, into the storage unit of the computer. This procedure is tedious and is prone to error. When the program has been entered, the user usually verifies that it has been entered correctly by reading it back, one instruction at a time, using the manual output (display).

After the program has been entered and verified, it is ready for execution. If data are required, they may be manually entered into the storage unit prior to execution or (as called for by the program) during execution. This, of course, depends on the manner in which the program directs the computer to request the input of data.

During execution, results may be displayed as they occur, or they may be stored in the storage unit for later inspection. In the first case, as each result is generated, the computer stops with the result displayed on the panel display; the operator records the result and then directs the computer to continue. In the second case, the computer stops only at the end of the program, with all the results stored in the storage unit. The operator then reads the results out, one at a time, and records them.

If an incorrect result occurs during a test execution of the program, the program must be debugged. One method of debugging is by *hand-executing* the program. This is done by following the program, step by step, exactly as the computer would do it. The user carries out each instruction in exactly the same way that the computer would and records the information generated at that step. This method is very time-consuming and is not practical for large programs. Another method is called *console debugging*. It consists of executing the program a step at a time, or a few steps at a time, on the computer itself. After each step or set of steps is executed, the user compares the information in the computer with previously calculated correct information. In this way the error can be localized, found, and corrected. This procedure ties up the computer for long periods of time and is usually used only as a last resort.

The minimum computer configuration may be useful for solving small problems or for training in computer fundamentals. Its utility is limited by the necessity to input programs by hand and by the fact that it can be programmed only in machine language. Machine language is difficult to learn and tedious to use. On many computers it is not practical to use machine language for programming student problems. Operating with such a system, the student would face a difficult and time-consuming task in organizing his program, setting up the arithmetic operations, and planning for input of data and output of answers.

Systems with Automatic Input and Output

The usefulness of the basic computer configuration can be greatly enhanced by the addition of automatic input and output equipment. Automatic input equipment in common use includes the following:

1. Paper-tape readers
2. Card readers

Automatic output devices include the following:

1. Typewriters
2. Line printers
3. Paper-tape punches
4. Card punches

Typewriters, paper-tape readers, and paper-tape punches are the least expensive of the above devices and are widely used with small, "low cost" computers. These devices, however, are also the slowest and least convenient automatic input and output devices. A typical small computer configuration is shown in Figure 3–2.

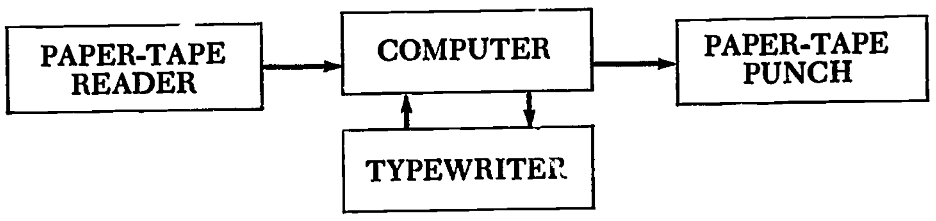


FIG. 3–2.—Typical Small Computer Configuration

The paper-tape reader usually provides automatic input at rates from 10 characters per second to 350 characters per second, depending on the computer and the type of reader. The reader may be mechanical or photoelectric.

Paper-tape punches usually range in speed from 10 characters per second to 60 characters per second. The typewriter provides for manual input (via its keyboard) and for output at the rate of 10 to 15 characters per second.

Another commonly used configuration, usually more expensive than the one described above, is shown in Figure 3–3.

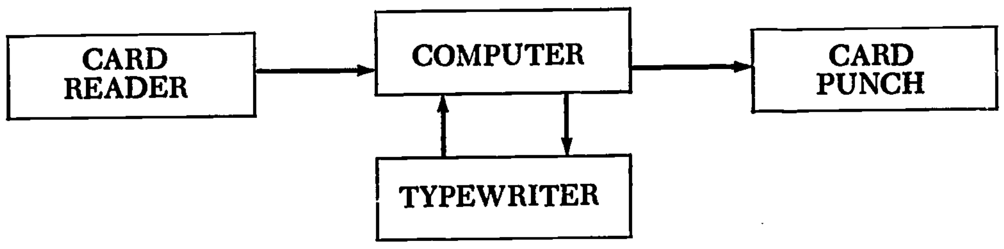


FIG. 3–3.—Typical Card I/O System

Punched-card readers operate at 100–1,500 cards per minute and card punches at 100–250 cards per minute. Each card can contain up to 80 columns of information. A configuration such as the one in Figure 3–3 will usually also have a line printer. Line printers will print information at rates from 100–1,200 lines per minute, with line widths of up to 132 columns. A line printer may operate *off-line* or *on-line*. If the printer is off-line, information is first punched out on cards. The cards are then placed on the printer and the information is printed. If on-line, the printer is attached to the computer and readable copy is produced directly on the printer. This configuration is shown in Figure 3–4.

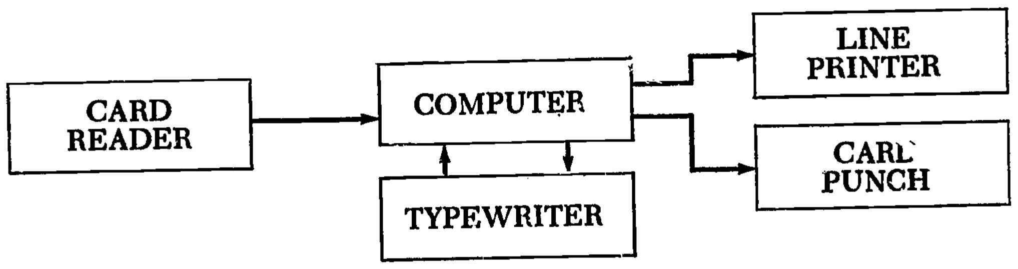


FIG. 3–4.—Card System with On-Line Printer

Addition of automatic input and output units tremendously increases the power of the computer. Some advantages are listed below.

1. Programs can be transcribed into paper tape or cards by inexpensive preparation devices, then read automatically into the computer.
2. If a program has been debugged by console debugging, the correct version can be *dumped* from the storage unit by punching it on paper tape or cards.
3. A library of routines can be stored permanently on paper tape or cards.
4. The use of more powerful programming languages becomes possible.
5. Data can be prepared off-line and entered quickly and automatically.
6. Results are available in permanent form.

On a system with automatic input and output, machine-language programming is usually done by means of a *symbolic assembly system*. An individual using a symbolic assembly system proceeds in the following way. First he writes his program in a symbolic representation of machine language: that is, he specifies an addition or subtraction operation with a mnemonic combination of letters such as ADD or SUB instead of with some nonsuggestive numeral. This symbolic version of the program must be *translated* (converted into machine language) before it can be performed. This translation is called *assembly* and is carried out by a program called an *assembler* or *assembly program*. (The assembler exists, of course, in machine language, for it has to be executed directly by the hardware equipment.)

The programmer is also assisted by previously prepared utility programs and routines. These may include routines for performing input and output, and for simulating floating-point arithmetic.[1] With these aids it is much less difficult to program student problems. However, symbolic language programming is still difficult, tedious, and prone to error. With practically all computers in current use, it is desirable to employ programming languages that are easier to learn and use.

On many computers, particularly the small ones, one frequently programs in a language that resembles machine language but is actually much easier to learn and use. A program written in such a language is ordinarily executed one instruction at a time by a special machine-language program called an *interpreter*. For instance, when an ADD instruction is encountered, the interpreter interprets it as a floating-point addition to be simulated within the interpreter, rather than as a machine-language addition. Such a language is called an *interpreter language* because programs written in it are executed with an interpreter.

While interpretive languages are relatively easy to use, and are not nearly so difficult to teach as are machine languages, nonetheless they are oriented more toward the computer than toward the user. That is, interpretive languages usually imitate machine languages in their general appearance. (Interpreters are discussed in the next section.)

Another class of languages are the *algorithmic* languages such as ALGOL and FORTRAN. The common characteristic of these *user-oriented* languages is that computational statements are written in a notation similar to ordinary algebra (hence also the term *algebraic* language).

Before the computer can execute a program written in an algorithmic language, the program must be translated into machine language. Since the typical algorithmic language is not at all like machine language, the translater programs must be much more powerful than the assembly programs and interpreter programs mentioned earlier. They are usually called *compilers*. The compiling or translating portion of a problem run can be a significant part of the total computer time used.

The use of interpreters and compilers on computer systems with automatic input and output is described in the next two sections.

[1] This is a form of arithmetic in which numbers are represented by numerals written in a special form of scientific notation. Scaling and keeping track of decimal points are done automatically. Floating-point arithmetic is usually a hardware feature of large computers, but it may not be a feature of a small computer.

Interpreters

Interpreters (also called *interpretive systems*) permit programming in a language that is easier to learn and use than machine language. The interpretive language is usually similar to a machine language, but it provides features and operations that are not inherent in the computer used for execution of the program. These usually include the following features and operations:

1. Conversion of input data from decimal to binary numerals and conversion of output results from binary to decimal numerals
2. Floating-point arithmetic
3. Standard functions such as square root, sine, cosine, exponential, logarithm, and inverse tangent
4. Operations for indexing and iteration
5. Debugging aids
6. Simplified operation of the computer

Interpreters that use a machine-language-like code are still widely used on older computers or on computers that are too slow to permit efficient use of algorithmic languages. They are most commonly used with small computers that have magnetic drum, magnetic disk, or delay-line memories and use typewriters and paper tape for input and output.

The interpreter routine is a machine-language program that must be entered into the memory of the computer and remain there during the execution of a program written in interpretive language. It is usually a lengthy program of several hundred or even several thousand instructions. Hence, it may occupy a major portion of the memory. The interpreter routine is usually stored permanently in the form of a roll of paper tape or a deck of punched cards.

In order to input and execute an interpretive-language program, a student follows a procedure similar to the following:

1. Load the interpretive system into the computer via the automatic input device.

2. Load the interpretive-language program into the computer. It may be entered by means of the typewriter, or it may be punched into paper tape or punched cards and entered by means of the automatic input device.

3. Execute the interpretive-language program.

If several interpretive-language programs are to be executed, one after the other, it is usually necessary to input the interpreter only at the beginning. Then the interpretive-language programs can be entered and executed, one at a time, without having to load the interpreter each time. However, for many small computers in current use, the interpretive system is very "touchy"; that is, an error made by a student can easily wipe out a portion of the interpreter in the computer's memory. In this event, the interpreter must be reloaded. The time required to load the interpreter is very important; this time may range from as little as a minute to an hour or more! Recently, some interpreters have been introduced that permit use of the algorithmic languages described in the next section. These interpretive systems are much easier to use than systems in which programs must be coded in a language that resembles machine language.

Algorithmic Languages

Most computer programs for the solution of mathematical problems are written in an algorithmic language: that is, in a language especially designed for expressing algorithms. The most commonly used algorithmic languages are ALGOL (ALGOrithmic Language) and FORTRAN (FORmula TRANslator). There are many subsets and dialects (variations) of ALGOL and FORTRAN in current use.

When using an algorithmic language, a student writes a program in a form that is very close to the mathematical description of the problem. For example, some expressions are shown below as they might be written in mathematical notation, in ALGOL, and in FORTRAN.

Mathematical Expression	ALGOL	FORTRAN
$b^2 - 4ac$	B↑2 − 4 × A × C	B**2 − −4.0*A*C
e^x	EXP (X)	EXPF (X)
$\sin (a + b)$	SIN (A + B)	SINF (A + B)
$\|X - 3\|$	ABS (X − 3)	ABSF (X − 3.0)
$\frac{ed - bf}{ad - bc}$	(E×D − B×F)/(A×D − B×C)	(E*D − B*F)/(A*D − B*C)

On present computers, it is not possible to run ALGOL or FORTRAN programs directly; they must first be translated into machine language. This translation is usually done by the computer itself under control of a machine language program called a *compiler*. After the algorithmic language program has been translated, it can be executed. The steps in writing, compiling, and executing an algorithmic-language program are outlined on the following page.

1. The student writes a program in the algorithmic language; this program is called the *source program.*

2. The compiler is loaded into the computer.

3. The source program is entered under control of the compiler and translated to an equivalent *object program* of machine-language instructions. The source program can be entered manually via the typewriter, or it may be punched into paper tape or cards and entered via the automatic input unit. As the object program is generated, it usually is punched out on the automatic output unit.

4. A tape or card deck of library routines is read in and the routines required by the object program (e.g., floating-point, arithmetic, sine, cosine, etc.) are punched into the object-program tape or card deck.

5. The object program is loaded into the computer (this usually wipes out the compiler) and executed.

The student must write a source program (Step 1), compile the source program to obtain an object program (Steps 2, 3, 4), and load and execute the object program (Step 5). The time required to write a source program is usually *much* less for algorithmic-language programming than for machine-language or interpreter-language programming. Unfortunately, on many small computers the *compile time* is so long that only a few programs can be compiled in a day. On some small computers with drum or disk memories, the compile time may be more than one hour even for short programs. On the other hand, many small computers with magnetic-core memories and fast paper-tape or card input and output units can compile short programs in a few minutes. Large-scale computers can compile short programs in a few seconds!

In order to increase the efficiency of processing algorithmic-language programs, a method called *batching,* or *batch processing,* is frequently used. With batch processing, the compiler is read into the computer only once. Then several source programs are compiled and object programs obtained for each one; they are not executed at this time. The object programs are separated so that data may be placed with each program. Then the object programs with data may be read into the machine and executed, one after another, without interruption. When the first program finishes, the second program and data are read into the computer automatically. When the second program finishes, the third program and data are read into the machine, and so on until all programs have been processed.

This method of operating in batches has the advantage of increasing the program-handling capacity of the system. However, there are also some disadvantages:

1. The object programs must be separated by hand.

2. If there are many programs, there may be a long wait between compilation and execution of a program.

3. The processing must be monitored (supervised) by a skilled operator who can restart the processing in case something goes wrong (for example, a program may *hang up* and stop the automatic processing).

If the computer has sufficient memory capacity, a *compile-and-go* system may be used. With this type of system, the compiler and the library are retained in a portion of the memory. The source program is read, and the object program is generated, stored in a different part of the memory, and immediately executed. The object program is not punched out. With a compile-and-go system, we might proceed as follows:

1. Load the compiler and the library into the computer.

2. Place the source program and the data in the input unit. Compile and execute the program.

Step 2 is repeated for each student's program. This type of system is well suited to situations in which many *small* programs are to be processed. With most small computers, the compiling phase can require minutes, tens of minutes, even an hour or more. Consequently, it may not be practical to use algorithmic languages on many currently available small computers. On very large computers the compiling phase may require only a second or less, and one finds algorithmic languages to be by far the most common programming languages for large computers.

Compilers for algorithmic languages automatically provide all the necessary functions such as input-output, and even floating-point simulation if it should be needed (a few quite large computers do not have floating-point hardware).

In addition, the compiler may provide for a library of special programs. For instance, there might be a readily available routine, prepared in advance, for solving linear equations. One dimension in evaluating a total computer system is the extensiveness of such programs in the library. Routines needed, but not provided in the library, must be prepared by the programmer himself.

In some recently developed systems, algorithmic-language programs are executed by an interpretive system instead of being first translated into an equivalent machine-language program. This approach is used in "time-sharing" systems (described later in this chapter) and in some very recently introduced small computers.

Systems with Auxiliary Storage

The power and flexibility of a computer system can be increased by adding additional storage capacity. This may be done by expanding the primary storage capacity of the computer or by adding auxiliary storage units. The auxiliary units usually provide much *greater capacity* but *slower access* to information than the storage unit on the computer itself. The cost-per-unit of stored information is usually much less for an auxiliary unit than for the primary computer storage. A computer with a few thousand characters of primary storage may have several million characters of auxiliary storage.

Increased storage capacity permits the use of more sophisticated software and makes it possible to solve larger problems on the system. Some of the uses of auxiliary units are listed below.

1. The library may be stored in the auxiliary unit. Hence, any program in the library can be read into the computer quickly and conveniently.

2. A program that is too large to fit into the storage unit of the computer itself can be read in from the auxiliary unit, a section at a time, and executed.

3. A problem may require repeated processing of a set of data that is too large to fit into the computer's storage unit. The data can be processed by placing them in the auxiliary unit.

A computer hardware system with auxiliary storage is shown in Figure 3–5.

We will briefly describe the following four types of auxiliary storage

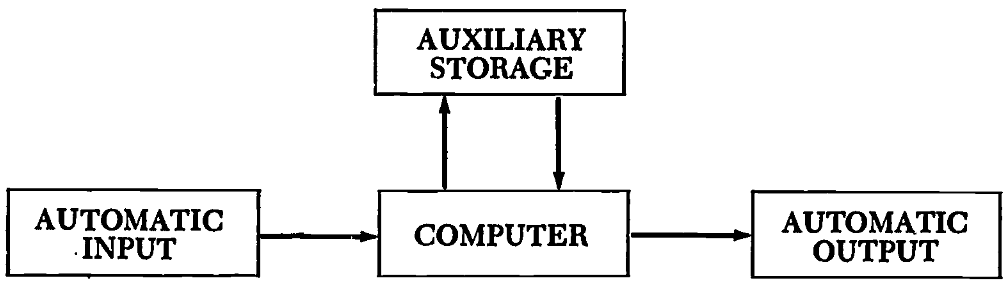


FIG. 3–5.—Computer Hardware System with Auxiliary Storage

units. For a detailed description of these and other auxiliary storage units, see Borko [2] or Chapin.[3]

1. Magnetic-tape units
2. Large-capacity magnetic drums
3. Disk files
4. Large-capacity magnetic-core units

The magnetic-tape unit is the most commonly used auxiliary storage device, especially on small and medium-size computers. On large computers, magnetic-tape units may serve all three functions: input, output, and auxiliary storage.

More than 10,000,000 characters of information can be stored on a reel of magnetic tape. In many cases, a single reel of tape can hold the entire software library of a computer system. Most systems that use magnetic tape have several tape units. Hence, several reels of tape (one reel per unit) can be on the system at any time in order to provide a large amount of auxiliary storage.

The primary disadvantage of magnetic tape is the slow *access* time. Since information is stored serially on a reel of tape, it may take several seconds or even several minutes for the tape unit to position its reel for the desired information.

Much faster access time is provided by magnetic drums and magnetic disk files. A drum may provide capacities ranging from a few tens of thousands to a few million characters of storage. Access to information stored on a drum is very fast, compared to magnetic-tape access. Typical access times are in the range of 10–50 milliseconds (1 millisecond = 0.001 second). However, large-capacity drums are very expensive and are usually found only on large, expensive systems.

A disk file provides very large storage capacities at less cost than a drum. A typical disk file may store 2,000,000 to 50,000,000 characters of information. Access to information in a disk file is usually slower than access to information stored on a drum. Typical access times are in the range of 50–500 milliseconds.

Recently, large-capacity magnetic-core units have appeared. These units provide very rapid access; information can be obtained from a core unit in a few microseconds (1 microsecond = 10^{-6} seconds). Mass core storage is very expensive and is usually used only on large systems.

[2] Harold Borko, *Computer Applications in the Behavioral Sciences* (Englewood Cliffs, N.J.: Prentice-Hall, 1962).

[3] Ned Chapin, *Introduction to Automatic Computers: A Systems Approach* (Princeton, N.J.: D. Van Nostrand Co., 1963).

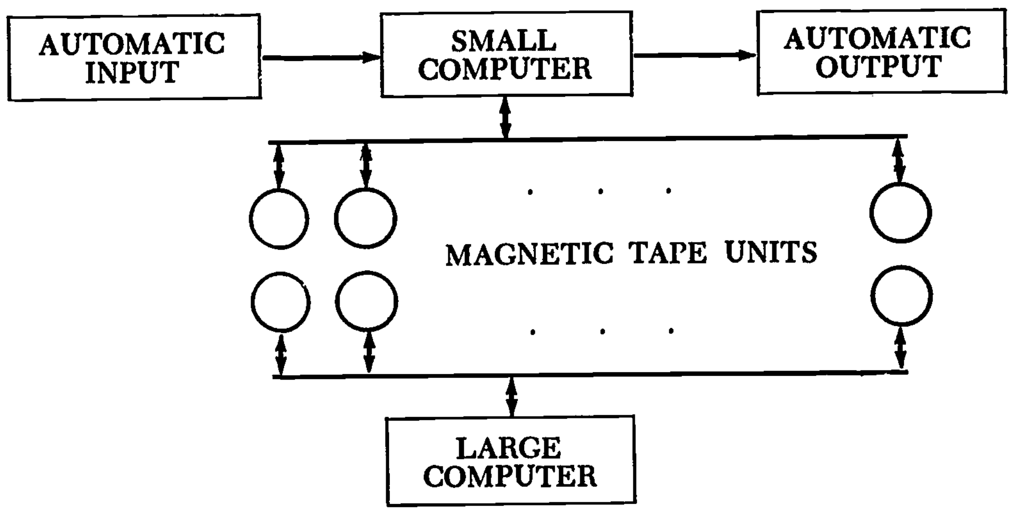


FIG. 3-6.—Large-Scale Computer System

On a large-scale computer system, magnetic tape is used for both auxiliary storage and input-output. In fact, the only input-output devices attached to the large computer may be magnetic-tape units. In this event, a small computer is used to transcribe information from paper tape or cards to magnetic tape. The magnetic tape is then used as input to the large computer. Similarly, the large computer sends information out to magnetic tape; the tape is then placed on the small computer system and the information punched or printed.

In this system the small computer acts as a buffer between the slow input-output equipment and the large computer. Many small computers in current use can operate several input-output devices and several tape units simultaneously. Once the information is recorded on magnetic tape, it can be read into the large computer at rates up to 240,000 characters per second.

Auxiliary storage permits the use of sophisticated *operating systems*, even on smaller computers. For example, on a system with auxiliary storage the steps in compiling and executing an algorithmic language are greatly simplified. These steps are described below, with the assumption that the source program has been written and punched into cards for automatic input. We assume also that the compiler is stored in the auxiliary storage unit, as are all library routines that might be required by a source program. The steps in compiling and executing the program are outlined below:

1. Place the source program and data on the card reader.
2. Call in the compiler from the auxiliary storage unit and start it.

The compiler now assumes control; the source program is read and translated. As the object program is generated, it is sent to the auxiliary storage unit. If library routines are required, they are automatically read from the auxiliary storage unit and copied into the object program. When the compilation is complete, the object program is automatically read back into the computer and executed.

An even more powerful operating system, called a *monitor*, is described in the next section.

Monitors

In the operating procedures described previously we have assumed that the processing of every program is initiated and supervised by a human operator. For example, suppose that five students wish to use the computer, for the following reasons:

1. Student No. 1 wishes to compile and execute an algorithmic language program.
2. Student No. 2 wishes to retrieve and execute a program that is in the library. He has a deck of cards containing his data.
3. Student No. 3 wishes to compile and execute an algorithmic-language program.
4. Student No. 4 wishes to assemble and run a program written in a symbolic form of machine language.
5. Student No. 5 wishes to use an interpretive system to run a program written in interpretive language.

Each problem is initiated by a manual operation to call in the appropriate system from the auxiliary storage unit. The set-up time may be appreciable. In fact, on a fast computer, the set-up time may be longer than that required for the actual processing of a program.

The operation of the computer must be closely *monitored* (supervised) by a human operator so that when a run is completed the next one can be initiated with minimum delay. The operator may also intervene if a run appears to hang up—that is, if it seems to require too much time and no results are printed.

Most of the responsibilities of a human operator can be taken over by the computer itself under control of a software system called a *monitor*, or *executive system*. A monitor is a machine-language program that resides in the computer storage unit at all times and exercises supervisory control over all computer system activities.

When a monitor is used, each job (program, problem, etc.) to be run is preceded by a *control card* that tells the monitor what is to be done.

The monitor calls in the appropriate software from the auxiliary storage unit, runs the job, and then reads the next control card. As new jobs come in, they are added to the back of the stack; the monitor proceeds automatically from job to job until all of the jobs have been completed. Manual set-up time is eliminated.

The monitor also provides many other conveniences. For example, if the running time for a job exceeds a predetermined limit (indicated in the control card), the monitor bumps the job and reads in the next control card. By means of control cards, the monitor can be directed to add a program to the library; to delete, modify, or list a program; or to print a catalog of the programs that are currently in the library.

Practically all large-scale computers operate under control of a monitor.

Time-Sharing Systems

A powerful new type of computer system has been developed—one that permits many users to share the simultaneous use of a large, fast system in a convenient and practical manner. This type of system is called a *time-sharing* system.

Each user communicates with the computer system by means of a terminal such as a Teletype®, an electric typewriter, or a keyboard for input and a small TV screen for output. Terminals are connected directly to the system by coaxial cable or indirectly by means of ordinary telephone lines.

If the connection is by means of telephone lines, a terminal can be geographically remote from the computer system. It can be in a different city or even in a different state.

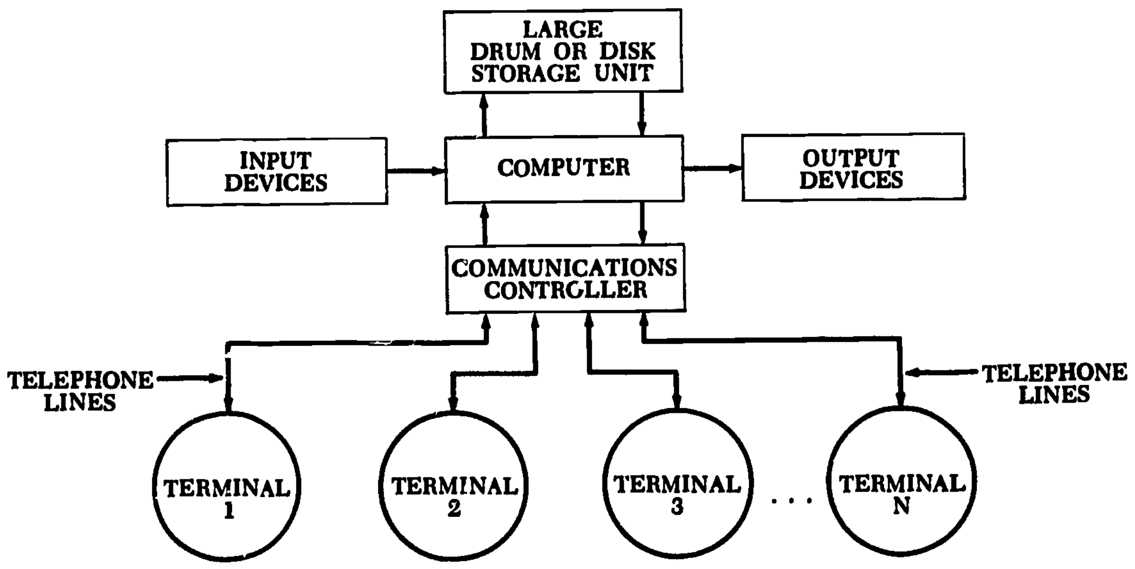


Fig. 3-7.—Time-Sharing System

Several time-sharing systems are already in everyday use. Typical systems service from 20 to 200 terminals per system. Many of these systems include features designed specifically for educational use, and terminals are in use at secondary schools, colleges, and universities throughout the United States. Some of these systems operate nearly 24 hours a day, every day.

A time-sharing system requires a very fast computer with a large drum or disk auxiliary storage unit. In addition, it must have hardware devices that permit many terminals to be attached to the system. Figure 3–7 diagrams a time-sharing system.

The time-sharing system operates under control of a very powerful set of software. All of the functions normally performed by the monitor in a monitor-controlled system, plus many of the administrative functions normally performed by computation-center personnel, are controlled by the time-sharing software. These administrative functions include scheduling the use of the computer (determining which user gets to run his problem and when), recording automatically the computer time used (for charging purposes), and managing the library of programs.

An understanding of the use of the system can perhaps best be gained by examining the procedure for solving a problem on the system, using one of the terminals. We will assume that the user (a teacher or student) has written a program in an algorithmic language and now wishes to process his program on the system. We also assume that he is using a terminal that is connected to the system through an ordinary dial telephone.

First, he gets the attention of the system by dialing. He is then able to communicate with the system by means of the terminal.

He enters the program by typing it on the keyboard of the terminal. As the program is typed, it is stored on the drum or disk storage unit.

Now suppose that the user, having completed his typing, requests the computer to run his program (which exists now on the disk unit). If the computer is not being used at the moment by any other user, other than for routine input and output, our user's program will be running in a matter of a fraction of a second. If his problem requires only a second or so to complete (the case with many student exercises), it will be run to completion and then the output printed on the terminal. This output may consist of the desired results or may be one or more error messages pointing out typing or other errors in his program.

If the computer *is* being used when the run request is made. the request is held in a list with a certain priority. The time-sharing software contains a scheduling portion that determines, among all those programs requesting computer time, which program is to be run next and how much time is to be allocated. Eventually (in a few seconds) our user is scheduled to be next. The software system causes the computer to cease running the program then being worked on (even if it is not finished), causes it to be written off onto the disk unit, brings in our user's program from the disk, and starts running it.

If our user's program requires longer than a second or so, his problem may in turn be interrupted, written off on the disk, and held in abeyance for a short time while other users are serviced. Later (again, in a matter of seconds) his program will be brought back from the disk and continued from the exact place where it was interrupted. This process is replicated until the program is completed and all the results have been printed.

The time-sharing system is especially well suited to situations in which many short programs are to be processed. This, of course, is the type of situation encountered when a computer is used as an instructional tool in teaching secondary mathematics.

It should be noted also that the type of computer used in a time-sharing system is usually 100 to 1,000 times as fast as small drum- or disk-memory computers that might be acquired by a school for classroom use. Hence, a program that might run for several minutes on a small drum-memory computer would be executed in a few seconds on the time-sharing system.

Student Access to a Computer System

In some programs of computer education, students are permitted direct access to the computer for hands-on execution of programs. Each student enters his own program into the computer and operates the computer during the execution of the program. This type of operation is usually possible only with "low cost" computers. In many cases, these are machines that have been technologically replaced by more modern equipment.

Most of the small, obsolete computers in current use for student access do *not* have efficient systems for processing algorithmic language programs. Hence they are usually programmed in machine language or in an interpretive language that resembles machine language. Wherever this is the case, it will be difficult to use new textbooks developed for using computers in secondary mathematics (e.g., the SMSG text, *Algorithms, Computation and Mathematics*). Furthermore, the time spent in the mere *mechanics* of using the computer may exceed the time spent in the underlying mathematical analysis and development of problem-solving concepts. Students may tend to become gadget-oriented rather than problem-oriented and mathematics-oriented.

Hands-on use of a computer has at least one advantage over some other types of access. If a program has an error, it may be possible for the student to correct the error and run the program while he is still at the computer initially, or within a few minutes in the event that he has to relinquish control of the computer to another student. In fact, each student may be on and off the computer several times during the period in which the computer is available. There may also be an additional motivational benefit (for some students) with hands-on use.

There are several disadvantages to hands-on use. Operation of the computer by a student, compared with operation by a trained, skilled operator (for example, a vocational trainee), is relatively slow and inefficient. This tends to reduce the number of student programs that can be processed in a given length of time. This is especially true if programs must be typed in by a student who cannot type except by the hunt-and-peck method! Similarly, if programs are first prepared on paper tape or cards, the available preparation units may quickly be overloaded by slow, error-prone, unskilled students. The direct-access approach can drastically reduce the number of students able to participate in the computer-oriented educational program.

The advantages and disadvantages of actually having a computer in the school must be related to the costs of existing small computers. With older, obsolete computers, the cost-per-student in the program is quite high. However, new equipment is being introduced that is much more useful and considerably less expensive than the small drum-memory computers available during the last few years. Also, some of the new computers are small enough that they can be brought directly into the classroom when needed.

It must be remembered that when a computer is used as an instructional tool in teaching mathematics, it is not really important to teach a student to be a skilled computer *operator*. Instead, the important goals are for him to learn the effective use of a computer as a problem-solving tool and to gain an appreciation of the relationship between mathematics, computers, and problem solving. These objectives can be achieved by providing an efficient means for processing programs written by students. This can be done by indirect access, in which student programs are sent to a computer center by mail, by courier, or by transmission over a telephone system. In this type of access, an important consideration is the *turn-around time;* this is the elapsed time between sending the program and receiving the results.

A courier service provides an efficient and low-cost method of transmitting student programs. Programs are sent by messenger to a computer center for execution, and the results are returned in the same way. Programs may be sent in handwritten or typed form to the center to be transcribed to punched cards or magnetic tape. (As an alternative, equipment for preparation of programs into computer-readable form can be installed in the school. An exciting possibility for the near future is the use of optical scanners and optical character readers for this purpose.) Turn-around time may range from a few hours to a few days, depending on the proximity of the computer center to the school, the means of transporting information, and the efficiency of the computer center. Some school districts already have computers for administrative data processing that could also serve the needs of students. Another possibility is leasing computer time from commercial service bureaus. Still another is that the user of a computer in the vicinity of the school may agree to provide the service. An outstanding choice is a university center; in fact, several universities now provide free computer services to schools in their geographical areas. In some parts of the country, educational data centers are being established to provide a variety of computer services to participating schools. In some cases, these services include processing of student programs.

If a school district is geographically remote from a computer center, indirect access can be achieved by sending and receiving programs by mail. The main disadvantage of this method of operation is the longer turn-around time.

The availability of low-cost communication equipment permits direct telephonic transmission of student programs to a computer center for fast turn-around processing. The school installs a paper-tape or punched-card device that is directly attached to a telephone line for the communication of information to the computer center. When the device is not being used for actual transmission, it may be used by students for the preparation and verification of tapes or cards for later transmission. The program is reproduced on tapes, cards, or magnetic tapes at the computer center; it is processed, and the results are transmitted back to the student. Student programs might be sent in the evening, with the results available at the beginning of the next school day.

At the present time, some form of indirect access to computers is the least expensive and most efficient way to use computers as instructional tools in teaching mathematics. No capital investment is required by the school, and advantage can be taken of algorithmic-language processing on modern computers. It is especially recommended for schools that are just getting started in this field.

Although time-sharing is relatively new, several secondary schools are already using this approach. A school district can install a terminal by paying a flat monthly charge plus additional charges that depend on how much the terminal is used. In this way, the cost can be geared directly to the number of students participating in the educational program and the manner in which the program is conducted.

Terminals may be installed in individual schools or in a central location (e.g., the administration building). Students may have direct access or indirect access to the terminals, or a combination of both. For example, if a terminal is installed in a school, students may be given limited access during the school day; then, at night, student programs can be run so that results will be available the next morning. In many cases, the terminal itself can double as a paper-tape preparation device. Programs can be prepared off-line (that is, with no connection to the computer) and transmitted at a later time. This results in reduced operating costs.

Projects are under way to evaluate the several ways of providing student access to computers at a reasonable per-student cost. It may happen that some blend of two or more approaches will be the best way.

IV ple Problems

IN ORDER to obtain some quantitative information on the efficiency of the types of computer facilities described in the previous chapters, we set up five sample problems and ran them on several currently available computer systems. These samples are typical of the elementary problems that students in secondary schools program and execute on a computer. The results will give some indication of the number of student programs that can be processed, per day, on the types of computer systems we have selected and used.

First, we will describe the sample problems. Then we will describe the computer systems on which the problems were solved and analyze the capabilities of the systems.

Sample Problems

Sample problem 1

We wish to develop a procedure to direct a computer to compute the roots (real or complex) of the equation

$$ax^2 + bx + c = 0,$$

or—switching now to one kind of computer typography—

$$AX^2 + BX + C = 0,$$

where A, B, and C are rational numbers expressed in a form suitable for direct input into a computer. The following equations are already in suitable form for the computers that we will consider:

1. $3X^2 + 4X + 5 = 0.$

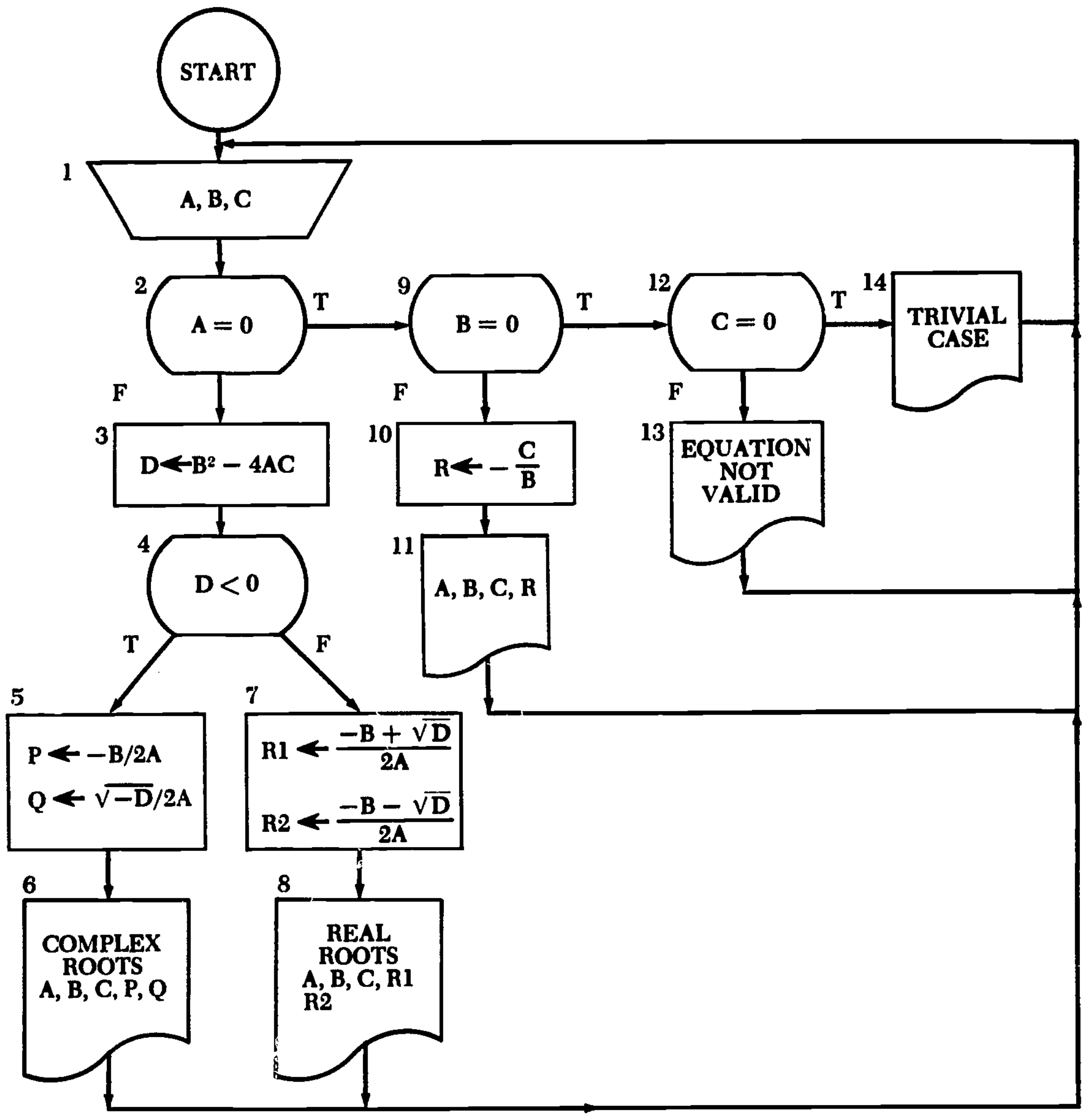


FIG. 4–1.—Flow Chart for Sample Problem 1

2. $0.7X^2 + 2.3X - 3.6 = 0.$
3. $9.8X^2 + 7.3 = 0.$

However, the following are not in proper form:

4. $\frac{2}{3}X^2 + \frac{4}{7}X - \frac{5}{6} = 0.$
5. $X^2 + \sqrt{2X} - 3 = 0.$

We can rewrite (4) in either of the following ways:

6. $0.66667X^2 + 0.57143X - 0.83333 = 0.$
7. $28X^2 + 24X - 35 = 0.$

In (6), we have replaced each rational number by a five-decimal-digit rational approximation (hence, we raise the question of whether the roots will be good approximations to the roots of the original equation).

In (7), we have multiplied both sides of Equation 5 by 42 and obtained an equivalent equation in a form suitable for computer input.

In Equation 5, we must replace $\sqrt{2}$ by a suitable approximation, say 1.4142, getting the following (nonequivalent) equation:

8. $X^2 + 1.4142\,X - 3 = 0.$

Shown in Figure 4–1 is a flow chart of our problem-solving procedure. Note that we have made provision for input of data such as the following.

A	B	C	Remarks
1	1	−2	Two real roots
1	1	1	Two complex roots
0	1	2	One real root
0	0	0	Trivial case
0	0	1	Not a valid equation

Our flow chart conventions conform to those in the SMSG text, *Algorithms, Computation and Mathematics.*

Sample problem 2

The second benchmark problem is a procedure to compute the mean, variance, and standard deviation of the array

$$X_1, X_2, \ldots, X_n.$$

This type of problem is usually characterized as having a lot of input data and very little output data. We have used the following formulas:

$$\text{MEAN} = A = \frac{\sum_{j=1}^{N} X_j}{N}.$$

$$\text{VARIANCE} = V = \frac{N\sum_{j=1}^{N} X^2 - (\sum_{j=1}^{N} X_j)^2}{N(N-1)}.$$

$$\text{STANDARD DEVIATION} = S = \sqrt{V}.$$

We have used the following variables for intermediate results:

$$S1 = \sum_{j=1}^{N} X_j.$$

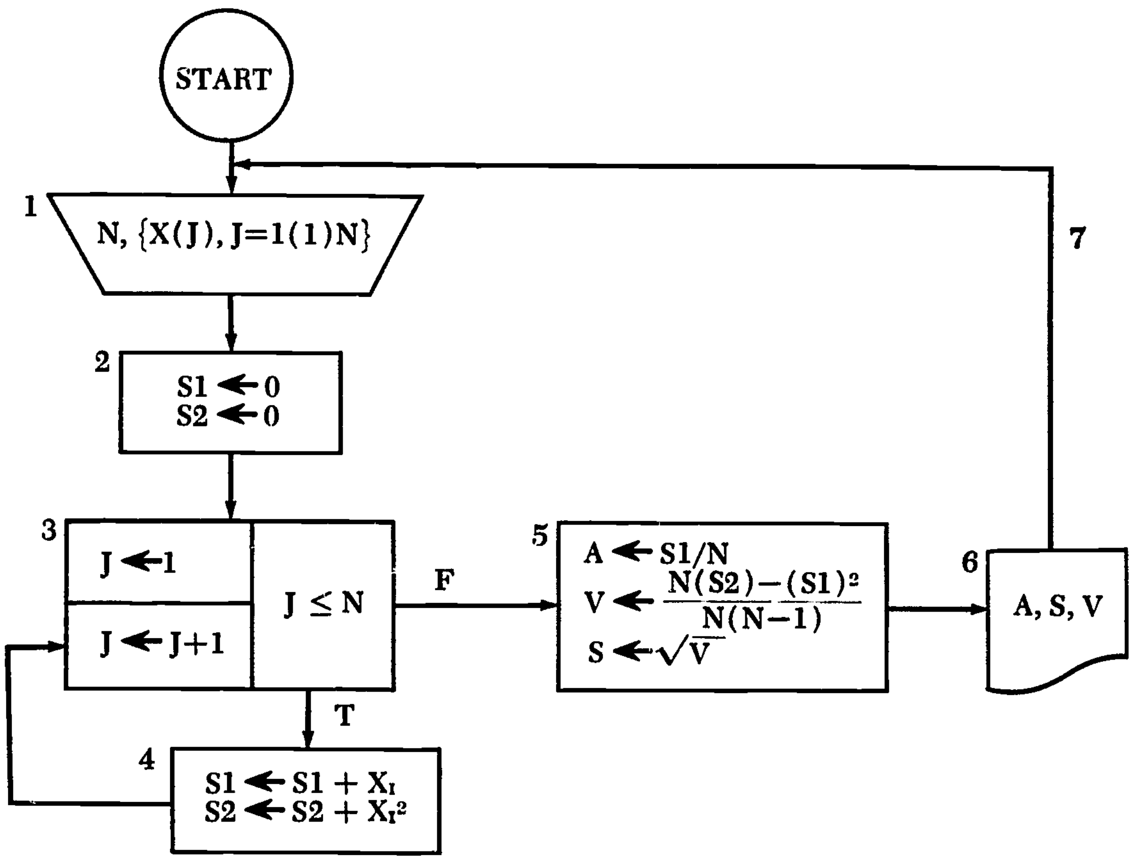


FIG. 4–2.—Flow Chart for Sample Problem 2

$$S2 = \sum_{j=1}^{N} X_j^2.$$

In terms of the variables S1 and S2, our equations for A and V are—

$$A = S1/N.$$

$$V = \frac{N(S2) - (S1)^2}{N(N-1)}.$$

Our procedure is shown in Figure 4–2. This procedure directs the computer to do the following:

1. Read the value of N and then the N values $X_1, X_2, \ldots, X_n$. (Flow Chart Box 1.)
2. Compute the values of S1 and S2. (Flow Chart Boxes 2, 3, 4.)
3. Compute the values of A, V, and S. (Flow Chart Box 5.)
4. Print the values of A, V, and S. (Flow Chart Box 6.)
5. Return to the beginning. (Return Line, labeled "7.")

This procedure was run for N=30. The data are shown on the following page.

```
30                          }  Value of N
53, 62, 29, 70, 85, 93   )
31, 49, 79, 66, 60, 45   |
83, 75, 59, 79, 76, 89   >  30 Values of X_I
66, 39, 76, 89, 92, 59   |
75, 46, 86, 83, 79, 67   )
```

Sample problem 3

Now let's look at a problem with a lot of input, a lot of output, and a great deal of shuffling of data within the computer. This procedure

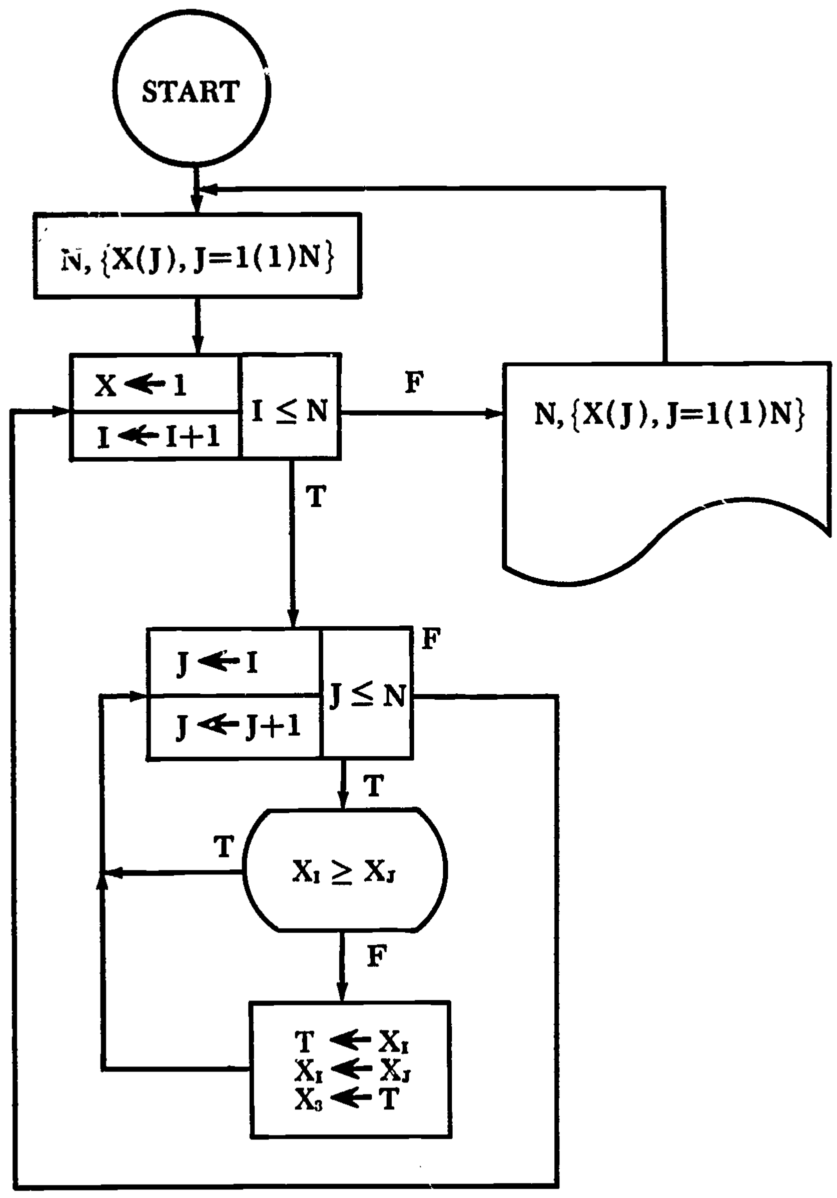


FIG. 4-3.—Flow Chart of Procedure for Sample Problem 3

directs the computer to permit input of an array, sort the array into descending numerical order, and output the sorted array.

EXAMPLE

Input Data	Output Data
53	91
65	81
29	76
37	73
81	65
76	65
65	53
73	46
91	37
46	29

A flow chart of our procedure is shown in Figure 4–3 on the preceding page.

This procedure was run for the set of data shown in the second sample problem, page 35.

Sample problem 4

This procedure directs the computer to tabulate the function $F(X) = \sqrt{X}$ for $X = 1, 2, 3, \ldots, N$. That is, the computer will generate and output a table of X and $\sqrt{X}$. There will be N lines in the table. For example, if $N = 10$, the table might appear as follows:

```
 X   SQUARE ROOT OF X
 1   1.00000
 2   1.41421
 3   1.73205
 4   2.00000
 5   2.23607
 6   2.44949
 7   2.64575
 8   2.82843
 9   3.00000
10   3.16228
```

This problem is characterized by having very little input (only the value of N) and quite a lot of output. A flow chart of the procedure is shown in Figure 4–4.

If 30 is the value of N, the output table will consist of 30 lines, with two numerals on each line.

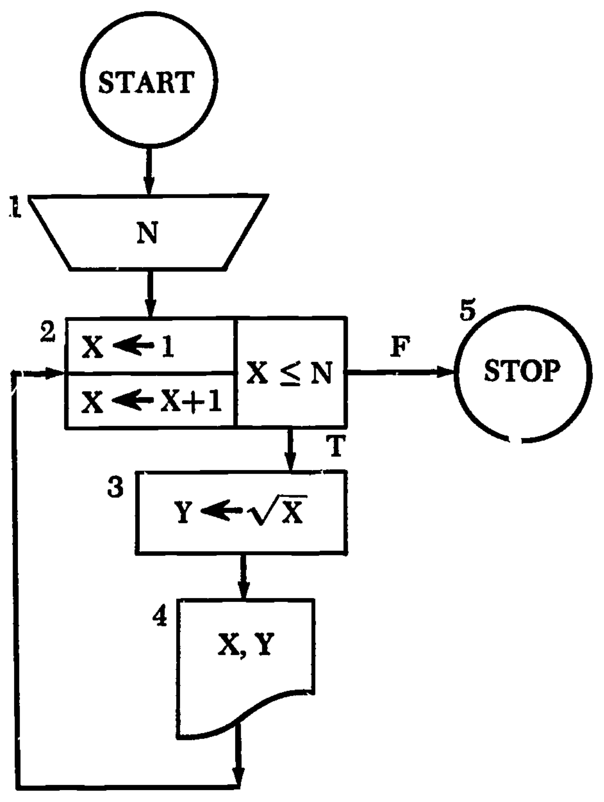


FIG. 4–4.—Flow Chart for Sample Problem 4

Sample problem 5

The final sample problem is a procedure to determine if a number, M, is a prime number. If M is a large integer, then this procedure might be characterized as having very little input (only M), very little output (a short message), and a *lot* of computation in between. A flow chart of the procedure is shown in Figure 4–5. It is one of many possible procedures and is not necessarily the "best" procedure. In the flow chart—

INT(M/N) = integer part of M ÷ N.
Q = integer quotient of M ÷ N.
R = remainder on dividing M by N.

This procedure directs the computer first to test M to determine if it is an integer ≥ 2. If so, it causes M to be divided by the trial divisors, 2, 3, 4, . . . , $\sqrt{M}$. (If M = 2, this will not happen.) After each division, the remainder is computed and tested. If the remainder is zero, the process terminates and the message "M IS NOT A PRIME NUMBER. N IS A DIVISOR" is printed. If the remainder is nonzero for N = 2, 3, . . . , $\sqrt{M}$, the computer prints the message "M IS A PRIME NUMBER."

The procedure was run for the following values of M: —5; 6.3; 1; 2; 13; 259; 1,517; 8,797; 30,031; 510,510.

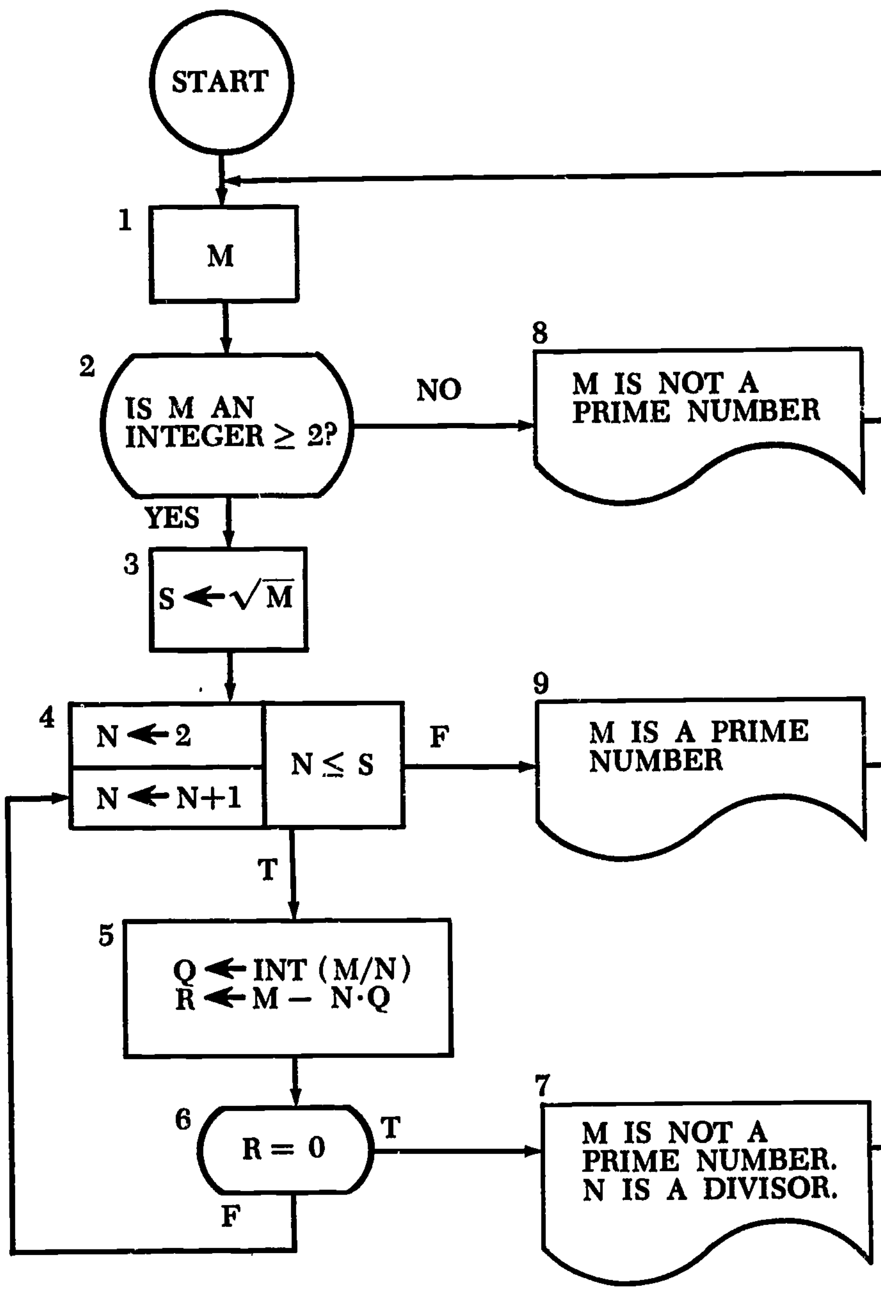


FIG. 4-5.—Flow Chart for Sample Problem 5

Computer Facilities

The sample problems were programmed and executed on several computer systems, ranging from small desk-sized computers to modern large-scale systems. Whenever possible, the programs were written in an algorithmic language.

These systems are described in the next six sections, along with the results obtained in running the sample problems. The results are then summarized. The computers used are identified by letters, A through F.

Computer A

The sample problems were programmed and executed on a small computer that is in use in secondary school educational programs. This computer has a magnetic-drum memory, paper-tape input/output, and typewriter input/output.

In an educational environment, this computer is usually programmed by means of an interpretive system, using a language that resembles a machine language. Programs written in this language cannot be run on any other computer.

The five sample problems were programmed in interpretive language and executed on the machine according to the following procedure:

1. The interpretive system was entered by means of the *paper-tape reader.*

2. The interpretive language program was entered *manually* by means of the typewriter.

3. The machine was directed to execute the program. If data were required, they were entered by means of the typewriter. (The data to be used are shown in the sample problem description.)

The programs were run again, using the procedure shown below:

1. The interpretive system was entered by means of the paper-tape reader, as in Step 1 of the previous procedure.

2. The program and data were punched into paper tape *off-line.*

3. The program tape was entered by means of the tape reader.

4. The machine was directed to execute the program. Data were read in by means of the paper-tape reader.

The interpretive system was entered *once* at the beginning of the test runs. Then the five programs were entered and executed, one after the other. Times (in seconds) are given in Table 1, as information is available.

In using the table, the following points should be kept in mind.

1. The programs were completely coded prior to approaching the computer.

2. The programs were entered by a skilled operator without any errors being made. For manual input by students, the input time could vary tremendously from the figures shown.

An advertised feature of Computer A is the existence of a FORTRAN system. In order to judge the practicality of algorithmic-language

TABLE 1
TIMES FOR COMPUTER A, WITH INTERPRETIVE LANGUAGE

Task	Manual Input	Paper-Tape Input
Load Interpreter	210[1]	210[1]
Enter Program 1	420	
Execute Program 1	35	
Enter Program 2	180	
Execute Program 2	95	
Enter Program 3	180	(Times not available)
Execute Program 3	250	
Enter Program 4	90	
Execute Program 4	90	
Enter Program 5	360	
Execute Program 5	60	
Totals (Secs.)	1,970	
Totals (Mins.)	32.8	

[1] In both cases, the interpretive system was read in on paper tape.

processing on a small machine, the sample problems were programmed in FORTRAN, compiled, and executed. Two procedures were used. The first is described below.

1. Prepare a FORTRAN source-language tape on-line. (Note that the computer is tied up.)
2. Compile the source-language program. An object-language program is punched out on paper tape.
3. Enter the object program into the computer.
4. Execute the object-language program.

Results are shown in Table 2. As can be seen, total computer time used to run the five programs was 5,200 seconds = 86.7 minutes.

If a paper-tape preparation unit is available, the source program can be prepared off-line. In this event, the total *computer* time used would be 3,460 seconds = 57.7 minutes.

TABLE 2
TIMES FOR COMPUTER A, WITH FORTRAN

Problem	Prepare Source Tape	Compile and Enter Object Program	Execute
Sample Problem 1	600	1,020	35
Sample Problem 2	250	418	95
Sample Problem 3	250	418	250
Sample Problem 4	130	204	90
Sample Problem 5	510	870	60
Totals (Secs.)	1,740	2,930	530

Computer B

Computer B is a magnetic-core-memory, solid-state machine that was announced in late 1966. A minimum configuration consists of the computer with Teletype® input/output, a paper-tape reader, and a paper-tape punch. The computer is very small and can be moved from place to place.

The problems were programmed in an algorithmic language very similar to the new languages being developed for use on large time-sharing systems. Programs written in this language are executed by means of an interpretive system. However, the language itself is much more powerful than the machinelike interpretive language used in Computer A. An additional feature of Computer B is that it can be controlled from a remote location by a Teletype® connected to it over a telephone line. Hence, two or more schools in a district can have convenient access to this machine on a one-at-a-time schedule.

The five sample problems were programmed in interpretive language and executed on Computer B according to the two procedures used for Computer A, and the five programs were entered and executed, one after the other. Results, with the times given in seconds, are given in Table 3.

The times for manual input of programs include correcting several errors. Furthermore, only two programs were written beforehand. The others were composed directly from the flow charts while on-line to the computer. This was possible because of the similarity of the language that was used to mathematical notation and because of the ease in finding and correcting mistakes when using this type of system.

A FORTRAN system is also available for this computer, but it was not tested.

TABLE 3
TIMES FOR COMPUTER B, WITH INTERPRETIVE LANGUAGE

Task	Manual Input	Paper-Tape Input
Load Interpreter	600	600
Enter Program 1	540	90
Execute Program 1	65	60
Enter Program 2	340	47
Execute Program 2	110	88
Enter Program 3	153	36
Execute Program 3	360	360
Enter Program 4	60	17
Execute Program 4	140	90
Enter Program 5	180	44
Execute Program 5	184	170
Totals (Secs.)	2,734	1,512
Totals (Mins.)	45.6	25.2

Computer C

The sample problems were also run on a small core-memory computer with punched-card input/output. This computer is in very common use for administrative data processing in a number of school districts. The programs for this computer were written in FORTRAN. The source programs were punched into cards (off-line) and then compiled and executed. Data were also entered by means of punched cards. Compile-and-execute times are given in Table 4, in seconds. These times do not include the time to punch the source programs and data into cards for entry into the computer. The total time to run the five problems was 2,761 seconds = 46.0 minutes.

TABLE 4
TIMES FOR COMPUTER C, WITH FORTRAN

Problem	Compile	Execute
Sample Problem 1	546	45
Sample Problem 2	542	40
Sample Problem 3	510	65
Sample Problem 4	420	63
Sample Problem 5	480	50
Totals (Secs.)	2,498	263
Totals (Mins.)	41.6	4.4

Computer D

This system has core memory, punched-card I/O, an on-line printer in addition to a typewriter, and magnetic-disk auxiliary memory. The algorithmic language is compile-and-go, with the compiler as well as the generated object program residing on the disk. Thus the only cards which need be entered for a run are system request cards (2), compiler control card (1), the source program, and the data. Jobs are stacked one after the other in this fashion.

Compile-and-execute times are shown in Table 5. Compile time shown is that between the entrance of the system request cards and the computer's indication that compilation is complete. Execution time is that required to read the program off the disk and execute it before the next system request card is read. The entire operation is handled by the supervisor program (i.e., automatically). The total time for the five problems was 405 seconds = 6.8 minutes.

Computer D is actually an expanded and upgraded version of Computer C. Addition of auxiliary-disk storage is the primary reason for the decrease in compile time. The decrease in execute time is caused by improvements in the central computer and improved software to execute the compiled program.

TABLE 5
TIMES FOR COMPUTER D

Problem	Compile	Execute
Sample Problem 1	80	25
Sample Problem 2	45	20
Sample Problem 3	45	55
Sample Problem 4	30	25
Sample Problem 5	55	5
Totals (Secs.)	255	150
Totals (Mins.)	4.3	2.5

Computer E

Computer E is a large-scale computer located at a university. The sample problems were coded in two different algorithmic languages, FORTRAN and MAD. The programs were sent to the computer center in the afternoon after school, and results were available when school began the next morning (overnight turn-around).

For both FORTRAN and MAD a compile-and-go system, operating under monitor control, was used. The times shown in Table 6 are combined compile-and-execute times. These are given in minutes (since that is the form in which they are printed on the output sheets—the computer system itself times each run).

TABLE 6
COMPUTER E, COMBINED COMPILE-AND-EXECUTE TIMES

Problem	FORTRAN	MAD
Sample Problem 1	0.40	0.28
Sample Problem 2	0.39	0.25
Sample Problem 3	0.39	0.26
Sample Problem 4	0.40	0.24
Sample Problem 5	0.39	0.26
Totals (Mins.)	1.97	1.29

Computer F

Computer F is actually a Teletype® connected by phone line to a time-sharing system. The problems were run twice on the time-sharing system.

First they were run by hands-on use of the system: that is, a telephone connection was established with a time-sharing computer system, and the programs and data were transmitted manually, by typing them on the Teletype® terminal. Next the programs and data were punched into paper tape, off-line from the system (that is, without a telephone connection to the computer). Then the connection was established, and the programs and data were transmitted by means of the paper-tape reader on the Teletype® terminal. Times are given in Table 7. In each case, the time is the total time that the *Teletype® terminal* was in use during the entry and execution of the program. Times are given in seconds and in minutes.

TABLE 7
COMPUTER F, TOTAL TIMES TERMINAL IS IN USE

Problem	Manual Input	Paper-Tape Input
Sample Problem 1	300	125
Sample Problem 2	240	58
Sample Problem 3	240	99
Sample Problem 4	240	101
Sample Problem 5	250	108
Totals (Secs.)	1,270	491
Totals (Mins.)	21.2	8.2

For the time-sharing system, it must be emphasized that the times shown are *not* the computer times. Since the school's access to the computer is the Teletype® (or other terminal), the significant time is the Teletype® time. Remember, the computer is servicing *many* such Teletypes® simultaneously.

Summary

Table 8 shows a summary of the *total* time to input and execute the five sample programs for each combination of computer facility and method of operation discussed in this chapter.

We will not attempt to make a cost analysis for the facilities described, since there are too many variables. Some of the points to consider in making such an analysis are listed below.

1. How many students are to be involved?

2. The sample problems are simple problems which students work on as they are *beginning* to learn to use computers. As their skill increases, they will write programs that require much more time. (On Computers A, B, and C, it will be impossible or impractical to handle many of these problems.)

TABLE 8
TOTAL TIMES FOR INPUT AND EXECUTION

Computer	Method of Operation	Time (Mins.)
A	MLT[1] Interpreter—Manual Input	32.8
A	MLT[1] Interpreter—Paper-Tape Input	N.A.
A	FORTRAN—On-Line Source Preparation	86.7
A	FORTRAN—Off-Line Source Preparation	57.7
B	Algorithmic Language Interpretive System—Manual Input	45.6
B	Algorithmic Language Interpretive System—Paper Tape Input	25.2
C	FORTRAN—Card Input	46.0
D	FORTRAN—Monitor Control	6.8
E	FORTRAN—Compile-and-Go	2.0
E	MAD—Compile-and-Go	1.3
F	Algorithmic, Manual Input	21.2
F	Algorithmic, Paper-Tape Input	8.2

[1] Machine-Language Type

3. How many programs can be run in a given period of time (class period, day, week)?

4. What is the cost-per-problem?

5. What is the cost-per-student over a school year?

6. What is the turn-around time for each run? That is, how long a time elapses after a student submits his problem before he gets the results back? (Note: The actual computer time used is not by itself an important consideration—the cost-per-problem *is* important, and the turn-around time is important.)

7. How sophisticated is the software available? What languages are provided? How easy will it be for the student to use the system?

8. What is the teaching and training time before a student can write a program and use the computer? Mainly, this means: How easy is it to teach the main language and the operating procedures?

9. Can two or more approaches—for example, a time-sharing terminal for small problems plus overnight turn-around to a computer for large problems—be used together to provide more efficient, lower cost operation than a single system?

The most important thing to do in choosing a computer facility is to try out each one yourself!