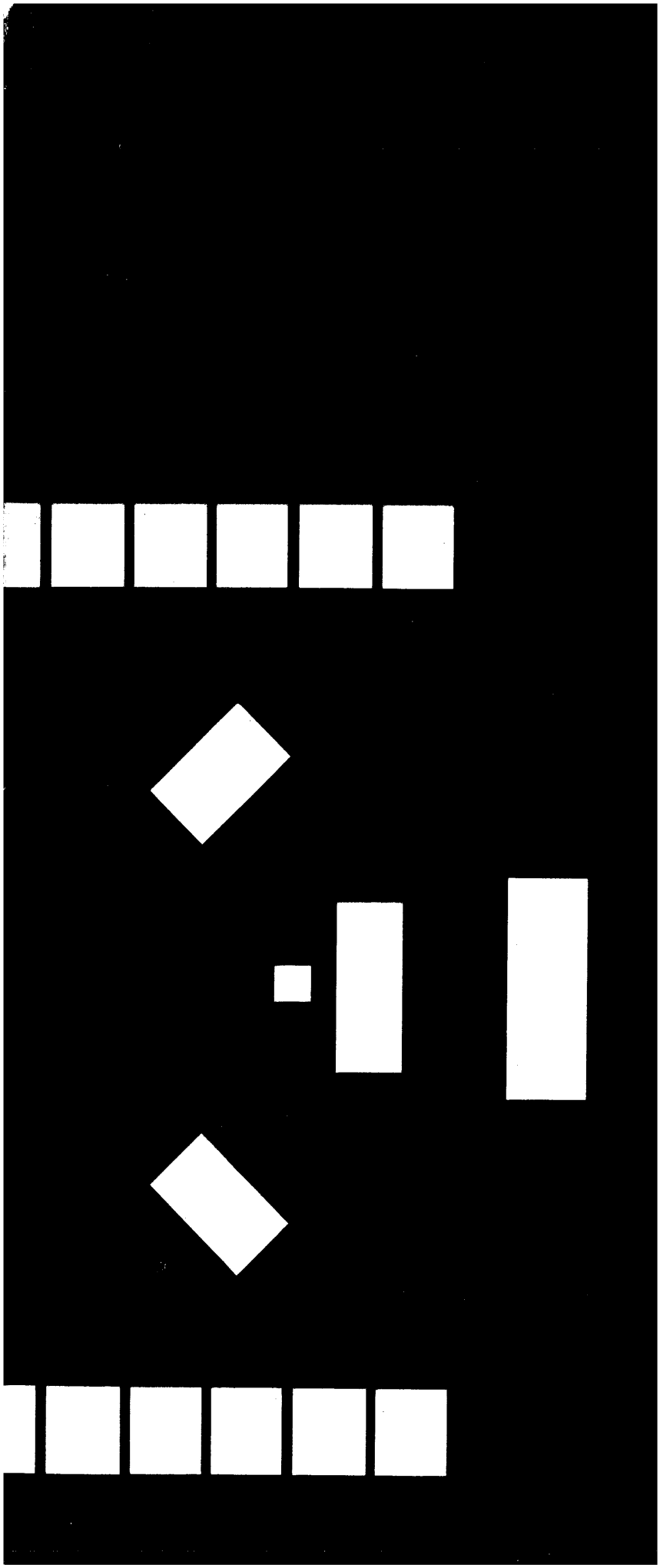


AN INTRODUCTION TO

DIGITAL COMPUTERS

Volume I

CONTROL DATA INSTITUTE
An Educational Service of CONTROL DATA CORPORATION

An Introduction to DIGITAL COMPUTERS

volume 1

FOR TRAINING PURPOSES ONLY

This manual was compiled and written by members of the instructional staff of

CONTROL DATA INSTITUTE
CONTROL DATA CORPORATION

CDI 60241600 (Formerly 081866A)
January 1967

Foreword

An Introduction to Digital Computers is published in three volumes as a primary text for anyone becomming acquainted with digital computers, or desiring a review of basic concepts. Volume I considers fundamental computer principles and basics of programming. Volume II acquaints the reader with the basic types of logic circuits interconnected to perform logical functions. Volume III contains information regarding the progress of a typical computer from start to delivery, a Glossary of computer-related terms and Appendixes to aid the student.

Throughout this manual, questions are interspersed with text material. Answers to questions appear at the end of each chapter. For the student, this provides a convenient way of checking progress through the text material.

A solid grasp of the subject matter contained within this manual will enable the reader to continue into more advanced areas of digital computers.

General Table of Contents

Contents

Introduction

The first observation of a computer system leaves the average person filled with mixed emotions about some of the strange events that transpire. Imagine yourself, on your first visit, alone in a room with a huge, complicated-looking computer. The indicator lights on the operator's console greet you with ominous winks and you subtly search for the door. The line printer blocks your escape route and seems to be rapidly predicting your fate. Punched cards are being consumed by the card reader at a rate comparable to the thoughts racing through your mind. A quartet of magnetic tape units with whirling reels is singing a dirge for your inevitable demise. Suddenly, everything stops and the typewriter deals the crushing blow as it rapidly types "THE END".

Concerned? Hardly the proper word to express your anxiety. However, your anxiety is due to lack of knowledge. Once properly introduced to each element of this mad machine, you discover that each has a congenial personality and will incessantly perform its assigned duties.

The objective of this manual is to introduce you to the concept of digital computers. Many of the mysteries will be solved as you develop an understanding of a digital computer system.

On your second visit to the computer room, the console lights wink a greeting, the line printer is publishing love songs, the card reader is "curled up before the fireplace" reading a good novel, and the tape unit quartet is singing "Home Sweet Home."

What a difference, as mysteries are replaced by knowledge.

chapter 1

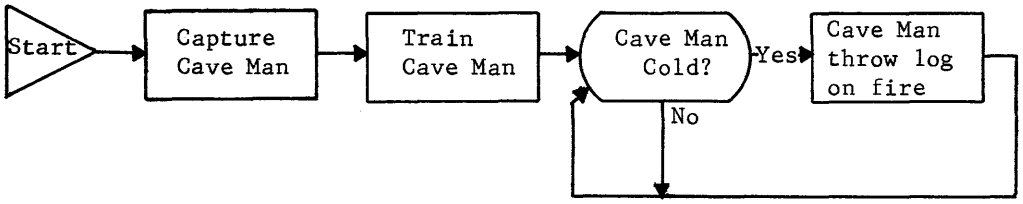
Computer Fundamentals

CHAPTER I

COMPUTER FUNDAMENTALS

Portions of the following were taken from H. D. Leeds' Computer Programming Fundamentals, Chapters 1 and 2, and from the Handbook of Automation and Control by M. E. Grabbe.

Mr. I. M. Cold, who was born before the advent of automation, solved his heating problem this way:



Automation means automatic action. Therefore, an automated device is one which performs its functions automatically. Automatic devices may be divided into two main categories, computing and controlling devices.

1. A computing device provides an automatic action after it performs a comparison or computation.

2. A controlling device is one which provides an action without a comparison or computation.

Which of the following are computing devices? Justify your answers.

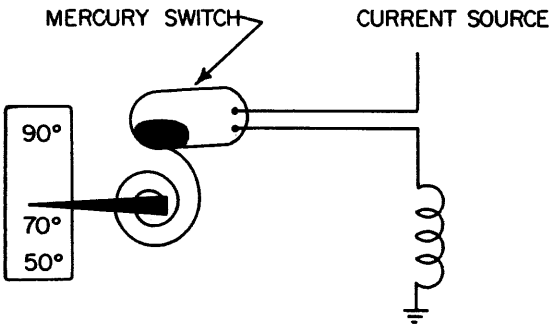
1. power-steering _____
2. thermostat _____
3. cash register _____
4. electric typewriter _____

Remember, a computing device provides an automatic action after it performs a comparison or computation. A controlling device provides an action without a comparison or computation. Power-steering is not a computing device because it does not calculate. A thermostat is a computing device because it compares the present position, or temperature, with a preset position to control the furnace. A cash register is a computer because it finds a sum or difference (computes), displays a number, and makes a record. An electric typewriter does not compare or compute.

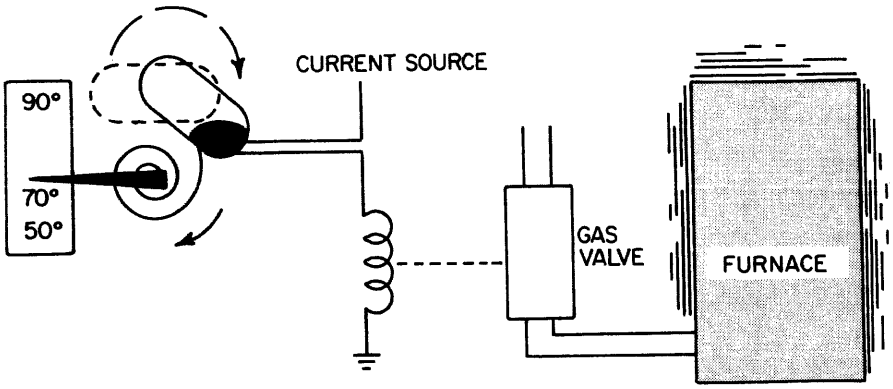
All computing devices can be divided into four basic sections of parts:

a) Input/Output - this section provides the communication with external devices. (Frequently referred to as simply "I/O".)
b) Memory (or Storage) - this section may permanently store commands or directions or temporarily store a partial sum.
c) Arithmetic - this section performs the comparisons or computations.
d) Control - this section coordinates and/or sequences all of the operations.

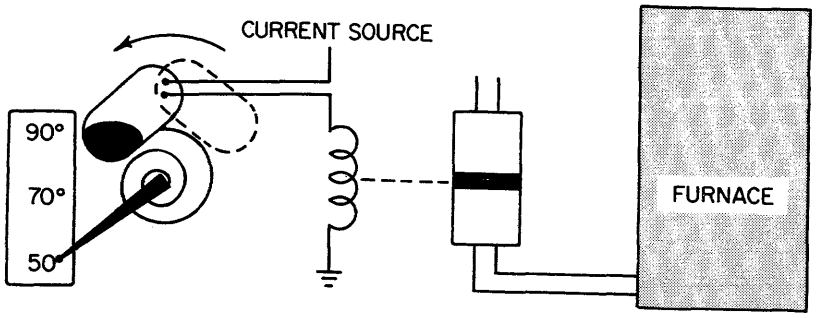
The following diagrams illustrate the function of a thermostat.



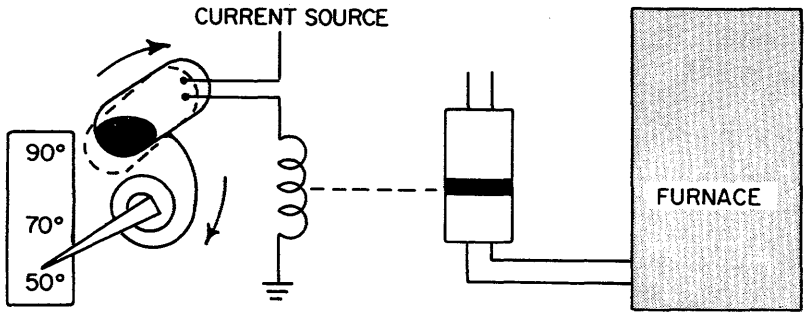
Temperature drops to 72° - coiled spring retracts - opens gas valve - furnace comes on - temperature rises.



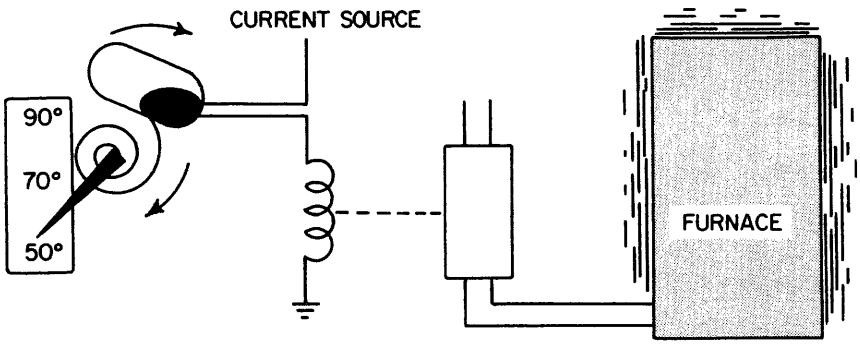
Now you manually readjust thermostat to 50° - switch opens - valve closes - furnace goes off.



Temperature again drops to 72° - spring retracts but not enough to close switch.



Temperature must now drop to 50° before switch closes.



Remember the four basic sections of a computer? List them!

1. _____
2. _____
3. _____
4. _____

Which of those basic sections could be subdivided into two major parts?

Right again - the I/O section.

Now list the <u>five</u> sections into which you have divided the computer.

1. _____
2. _____
3. _____
4. _____
5. _____

The terms associated with the thermostat are:

1. Room temperature
2. Coiled spring
3. Mercury switch
4. Manual setting of thermostst
5. Signal to control gas valve

We have already agreed that a thermostat is a type of computer. Our thermostat computer has five sections the same as our theoretical computer. Now list the five major sections of any computer and beside them list the thermostat associated term that represents a computer section.

1. _____ _____
2. _____ _____
3. _____ _____
4. _____ _____
5. _____ _____

Now turn to the next page to see if we agree on our answers.

1. Input - Room temperature
2. Arithmetic - Coiled spring
3. Memory - Manual setting
4. Control - Mercury switch
5. Output - Signal to control gas valve

Good! We did agree. You now understand how any computing device can be analyzed and compared to a basic computer with four sections (five if you consider I/O as two sections).

List five other computing devices and associate them with a basic computer. Why not start with one of the earliest computing devices - the abacus.

1. Abacus

   a. Input _____

   b. Arithmetic _____

   c. Memory _____

   d. Control _____

   e. Output _____

Tough, huh? There is a drawing of an abacus at the beginning of Chapter II, if you wish to look ahead. One quick look and then back here.

2. A voltmeter

   a. Input _____

   b. Arithmetic _____

   c. Memory _____

   d. Control _____

   e. Output _____

3. _____

   a. Input _____

   b. Arithmetic _____

   c. Memory _____

   d. Control _____

   e. Output _____

4. Gasoline Pump

   a. Input _____

   b. Arithmetic _____

   c. Memory _____

   d. Control _____

   e. Output _____

Would you consider the final price or gasoline to be the output?

5. _____

   a. Input _____

   b. Arithmetic _____

   c. Memory _____

   d. Control _____

   e. Output _____

Did you finish all five projects? Are you reasonably sure that you are correct? If the answer to both questions is yes, continue:

TYPES OF COMPUTERS

A thermostat is a simple ANALOG computer. Let us look at a DIGITAL computer (a cash register) and compare the two.

A cash register is fundamentally an adding machine. Each bit of data (the price) is entered when the operator presses the keys. Another key is pressed which causes the price to be added to a previous total. How does this adding take place? How is the total represented inside the machine? Actually, the operator does not have to know the internal operations in order to use the machine. The operator does not need to know the internal operations of a complex computer either, but it aids his understanding if he has a basic knowledge of its function and operation.

A computer is classified as analog or digital on the basis of how the input is transformed to the output. In the cash register, the transformation is accomplished through gears which can take only discrete and separate positions; it is therefore called digital. In the thermostat, the transformation is accomplished through the coiled spring which takes continuous (not separate) positions or states; it is therefore called analog.

Match the following areas of the cash register by drawing lines:

(A) Input	(1) Operator
(B) Output	(2) Internal gearing
(C) Arithmetic	(3) Keys
(D) Memory	(4) Window or tape
(E) Control	(5) Gears

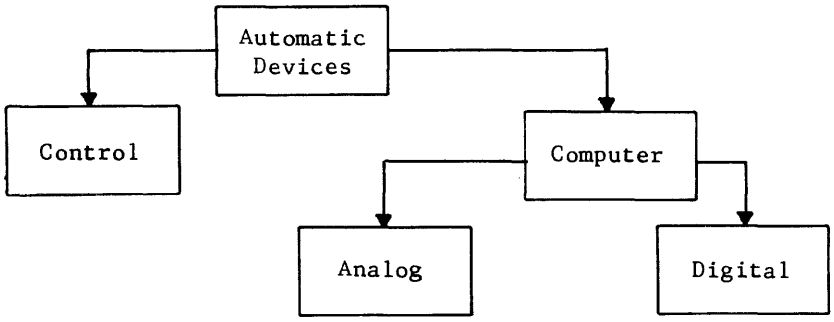
The temperature of the air is the input to the thermostat. The coiled spring is continually comparing the air temperature to the thermostat setting. The mercury switch tests the results of the comparison to control the whole operation. The thermostat remembers the specified setting and signals the results of the comparison to the furnace, the output.

Let's check your performance where you associated a cash register with the sections of a computer. The input to the cash register is entered by pressing the keys; the internal gearing performs the arithmetic. The positions of the gears provide a temporary memory, and the window functions as the output. All operations are under control of the operator. The tape output is the permanent record.

Which of the computing devices that you have listed are <u>digital</u> and which are <u>analog</u>?

1.	Abacus	_____
2.	Voltmeter	_____
3.	_____	_____
4.	Gas Pump	_____
5.	_____	_____

To summarize at this point, consider the following chart.



Because the object of this manual is to explain computing devices, the expansion on the left side of the chart is not shown.

Complete the following:

1. Define a computing device.
2. Define an analog computer.
3. Define a digital computer.
4. Explain why a cash register is a digital computer.
5. Explain why a thermostat is an analog computer.

ADVANTAGES OF A DIGITAL COMPUTER

Three main advantages of a digital computer are (1) speed, (2) accuracy and (3) cost. Let's examine each of these areas closely, using a cash register as an example.

1. Speed - The input keys and arithmetic portion of the cash register eliminate the need for the operator to write down a column of numbers and then add them. Few people with pencil and paper can keep up with an average checker at a cash register.

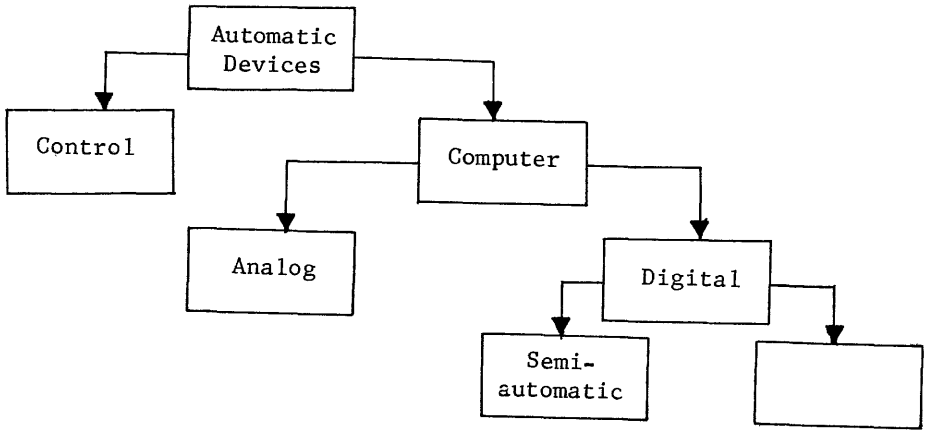
2. Accuracy - To analyze accuracy, consider the possible errors that could occur. The operator may enter a number incorrectly. The operator may incorrectly read the total. However, both of these errors could also occur during the pencil and paper computation; one cannot say that a cash register decreases errors in these areas. The arithmetic performed by the machine, however, is correct (unless machine failure occurs) far more often than pencil and paper addition and it is in this area that the accuracy is improved.

3. Cost - Imagine the amount of time saved by cash registers in a large supermarket. Multiply the time saved by the employee's hourly wage and it becomes evident that the cash register (assuming the volume of business justifies its purchase) more than pays for itself in a short time.

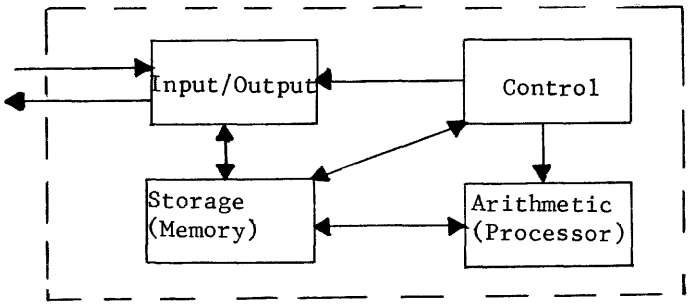
CLASSES OF COMPUTERS

Both analog and digital computers may be divided into two classes, automatic and semi-automatic. The cash register is an example of a semi-automatic digital computer because it needs the operator between steps. The thermostat is an automatic computer because each succeeding step (compare the temperature and the setting) is entered without an operator. If the full powers of machines, which are faster and more accurate than humans, are to be utilized, the control of these machines, as well as their calculating ability, must be made independent of the speed and accuracy of humans. When this occurs, the machines are called "automatic" computers. This type of computer usually has an operator who feeds the input to the machine, starts the computer, and watches for obvious malfunctioning. The computer control is built into the machine. The operator furnishes a set of instructions (the program), which the machine automatically executes.

Complete the following:



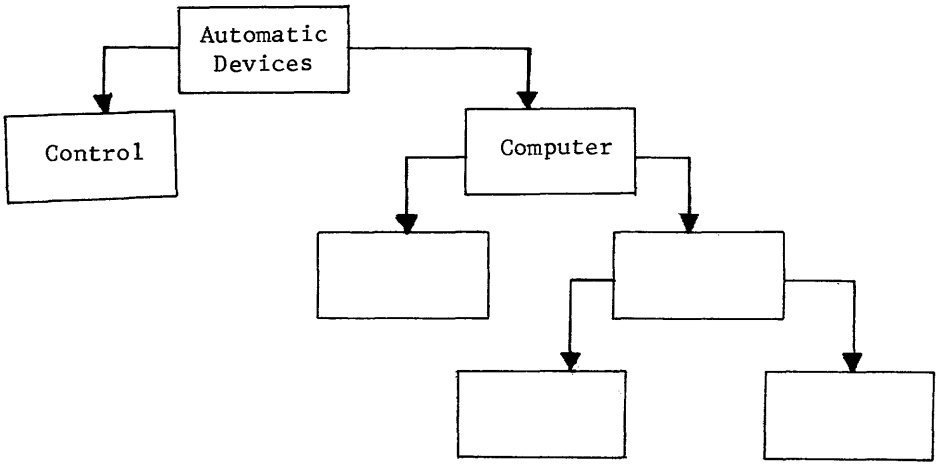
Using a diagram, let us look at a computer again.



By the previous definitions, it should be apparent that all four sections are necessary to an automatic computer. The dotted lines around the four sections are the outer dimensions of an automatic computer.

If a computer is automatic, the control is part of the machine. If the computer is semi-automatic, its operations have an external control.

In review, complete the following diagram.



A computer frequently checks the status of itself or associated equipment. You can also make a status check of your knowledge by completing the following.

1. Define automation.

2. List two categories that classify automatic devices.

   a. _____ b. _____

3. Explain the difference between a computing device and a controlling device.

4. List the four major sections of a computer and explain the function of each.

5. List the two types of computing devices and why each is unique.

6. List the three main advantages of computers.

   a. _____ b. _____ c. _____

7. Each type of computing device that you listed in question 5 can be divided into two classes. List the two classes.

   a. _____ b. _____

8. Explain the difference between an automatic computer and a semi-automatic computer.

If you have any blanks, either in the preceeding questions or in your own mind, get them filled now. Incidentally, the human brain is comparable to a computer section - MEMORY. Most of us have several million memory cells, or locations, that are unused. The answers to the preceding questions could be stored in some of these previously un-used locations for future reference. You may possibly need the information later on as you progress.

Did this chapter fulfill its objectives for you?

1. Do you know the definition of a computer?
2. Can you think of two types of computers?
3. How does a digital computer differ from an analog computer?
4. Is automation a synonym for computer technology?

If not, it's better to retreat to the beginning of the book and review until those objectives have been satisfied.

SUMMARY

This introduction has familiarized the reader with the different types of computers and with the items which are necessary to create a computing device. The purpose of this book is to show how each section of a typical computer operates and how each section must, to some extent, rely on each of the other sections.

The following chapters discuss those sections of a typical computer to further familiarize you with the programming and the internal workings of the machine.

Before you study the sections of a computer, it is necessary to become familiar with some fundamental concepts - number systems, programming, boolean algebra, and logic circuits. Chapters II through VI should provide you with those needed fundamentals.

chapter II

History and Applications

CHAPTER II

HISTORY AND APPLICATIONS

INTRODUCTION

Now that you are somewhat familiar with those devices that can be classified as computers, let's examine the evolution of the computing device from its origin to today's super computers.

There were fewer than 100 computers installed in the United States in 1951, after a development period of over 300 years. Fifteen years later, at the beginning of the year 1966, there were 30,000 computers in operation throughout the United States. These figures alone should give you some insight into the potential of the computer industry and your potential, should you decide to become proficient in the computer field.

This chapter should provide you with a background knowledge of computer development and a few of the thousands of applications for modern-day computers in a modern world. With that in mind, shall we proceed?

THE EVOLUTION OF COMPUTING MACHINES[1]

Some computation aids, like the abacus, are quite ancient (see figure 2-1).

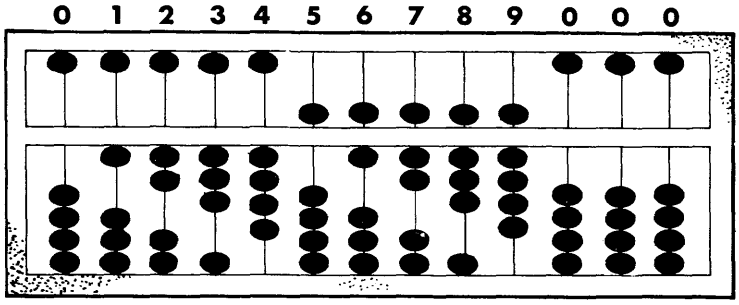


Figure 2-1. Abacus, and ancient counting device. The numbers show the count the beads represent in that position.

1. Abstracted from proceedings of the IRE, May 1962

The invention of the first mechanical device capable of addition and subtraction in a digital manner has been generally credited to Pascal, who built his first machine in 1642. Pascal, at the age of 19, wearied of adding long columns of figures in his father's tax office in Rouen, France. He made a number of calculators, some of which are still preserved in museums. One of his machines (figure 2-2) had number wheels with parallel, horizontal axes. The positions of these wheels could be observed and the digits read through windows in the cover. Numbers were entered by means of horizontal wheels which were coupled to the number wheels by pin gearing. Most of the number wheels were geared for decimal reckoning but the two wheels on the extreme right, had twenty and twelve divisions for compatability with two French coins, the sou and the denier. A carry ratchet coupled each wheel to the next higher place. Stylus-operated pocket adding machines now widely used, are descendants of Pascal's machine.

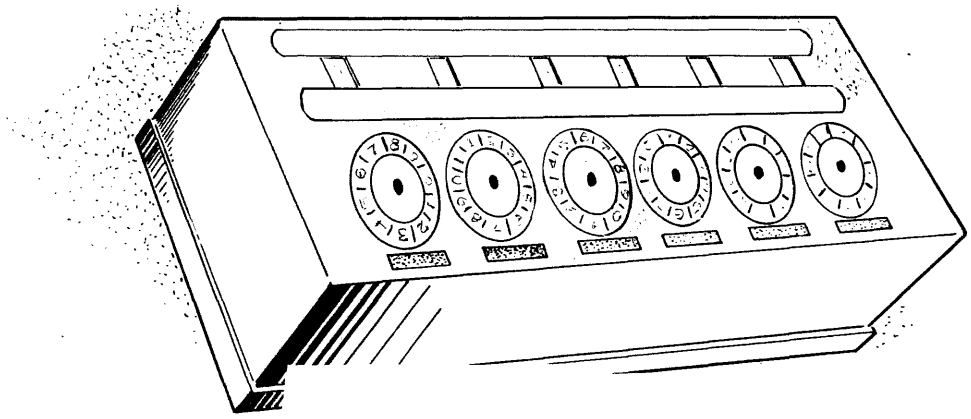


Figure 2-2. Pascal's Adding Machine

In 1671, Leibniz proposed a machine which could multiply by rapid, repeated addition. One was built in 1694, but was not reliable. The first machine to perform all four basic arithmetic operations well enough for commercial manufacture was the Arithmometer of C. S. Thomas, built in 1820. However, only a small number of Thomas's machines were constructed. Commercial exploitation of mechanical calculators did not take place until the last two decades of the nineteenth century.

Two further inventions deserve mention as forerunners of automatic calculators. The first is the Jacquard loom punched-card system devised to control the automatic weaving of complex patterns. Jacquard's loom, which came into wide use during the decade following 1804, was the first successful application of the principles of punched tape and card control demonstrated originally between 1725 and 1745 by Bouchon, Falcon, and Jacques.

The second major invention was the difference engine, a device for automatically calculating mathematical tables of functions whose higher-order differences are constant. Such a machine required a register for each order of difference and a means of successfully adding the contents of each register to those of the next lower-order register. The difference engine was constructed by Charles Babbage, who, between 1812 and 1822 designed and built a small working model with three registers of six digits each (figure 2-3). The final machine was to have had seven 20-digit registers and printed output. With the backing of the Royal Society, he obtained government support and began work in 1813.

However, the engineering of the time was not up to such a machine and Babbage had to invent techniques for engineering drawing and for precision construction. The machine was only partially completed in 1822, when government support ended and work stopped. A five-register sixteen-digit machine was later built in Sweden by Scheutz and demonstrated in England in 1854.

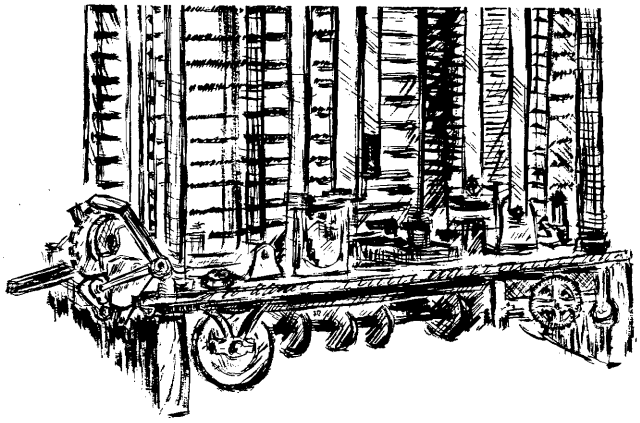


Figure 2-3. Babbage's Analytical Engine

In 1833, Babbage conceived his analytical engine, the first design for a universal automatic calculator. He worked on it with his own money until his death in 1871. Babbage's design had all the elements of a modern general-purpose,digital computer; it had units of memory, control, arithmetic, and input/output. The memory was to hold 1000 words of 50 digits each, all in counting wheels. Sequences of Jacquard punched cards were to serve as control. The very important ability to modify the course of a calculation according to the intermediate results obtained, now called conditional branching, was to be incorporated in the form of a procedure for skipping forward to backward a specified number of cards. As in modern, computer practice, the branch was determined by the algebraic sign of a designated number. The arithmetic unit, Babbage supposed, would perform addition or subtraction in one second while a 50 x 50 multiplication would take about one minute.

Babbage spent many years developing a mechanical method of achieving simultaneous propagation of carries during addition to eliminate the need for fifty successive carry cycles. Input to the machine was to be by individual punched cards and manual setting of the memory counters; output was to be punched cards, printed copy, or stereotype molds. When random access to tables of functions was required, the machine would ring a bell and display the identity of the card needed. Although Babbage prepared thousands of detailed drawings for his machine, only a few parts were ever completed.

The description of Babbage's ideas would not be adequate without a mention of Lady Ada Augusta, Countess of Lovelace, who was acquainted with Babbage and his work. Her writings have helped us understand his work and they contain the first descriptions of programming techniques.

Less than twenty years after Babbage's death, H. Hollerith conceived and developed the idea of machine-readable, unit-record documents. He introduced electro-mechanical sensing means and apparati for entering, classifying, distributing and recording data on punched cards. Hollerith's machines were used during the compilation of the 1890 census reports. The development of many types of electro-mechanical accounting machines and early computers was based on Hollerith's inventions.

The development by J. W. Bruce and his associates of devices and circuitry, which would transfer data between registers, or from registers to recording devices, was an important step in the evolution of electro-mechanical computers. One result was a machine developed by the International Business Machines Corporation for Columbia University during 1929, which solved mechanical problems that had thwarted Babbage.

Another significant series of developments were the Bell Telephone Laboratories relay computers, which were based initially on the work of G. Stibitz in 1938. These biquinary (using both the base 2 and the base 5, as does the abacus) and binary-coded-decimal machines included paper tape input, program control, branching, self checking and many other features later incorporated in electronic computers.

Within the past two decades, digital computers have progressed from the use of relays and rotating switches as computing elements, to vacuum tubes, and to semiconductor logic circuits; from electromechanical, delay line and cathode ray tube storage to magnetic drum, magnetic tape and magnetic core storage. They have progressed from input/output speeds of one or two dozen characters per second to tens of thousands of characters per second, and from computation speeds of a few operations per second to millions of operations per second. Through the use of self-checking codes and by other means, they reached high levels of reliability.

The progress of the electronic computer art could not be so rapid if it did not concern more than machines. By programming its operation in some new way, almost every digital computer built has been found capable of doing more than it was originally designed to do. This has often led to significant improvements of the computing machinery. The process was then repeated. Early digital computers were built to solve a few important, but relatively small, scientific problems. Electronic digital computing machines and systems now at work:

1. perform calculations for millions of pay checks, bank accounts, and insurance policies every day.

2. prepare weather forecasts.

3. solve scientific and engineering problems requiring from a few hundred to billions of arithmetic operations.

4. perform calculations for the design of almost every product of advanced technology.

5. translate English into Braille or other languages.

6. process data for the production, inventory control, and transportation of millions of products.

7. perform the calculations needed in many diverse fields for the effective defense of the nation.

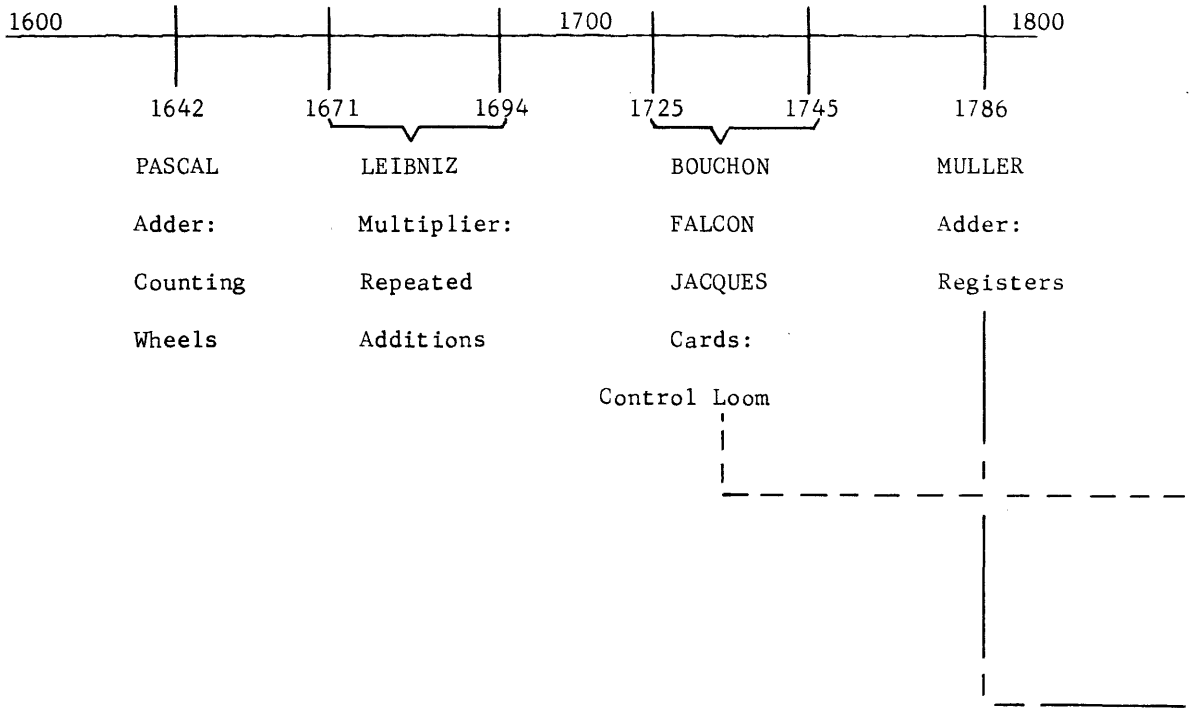
8. advance medical research by finding new patterns of diagnosis and by bringing new understanding of medical problems.

9. make commerical airline passenger reservation.

10. perform calculations for satellite launching orbiting and tracking, and constantly accomplish untold new tasks and demonstrate abundantly their usefulness to mankind.

The next portion of this chapter shows some acutal applications of present-day digital computers.

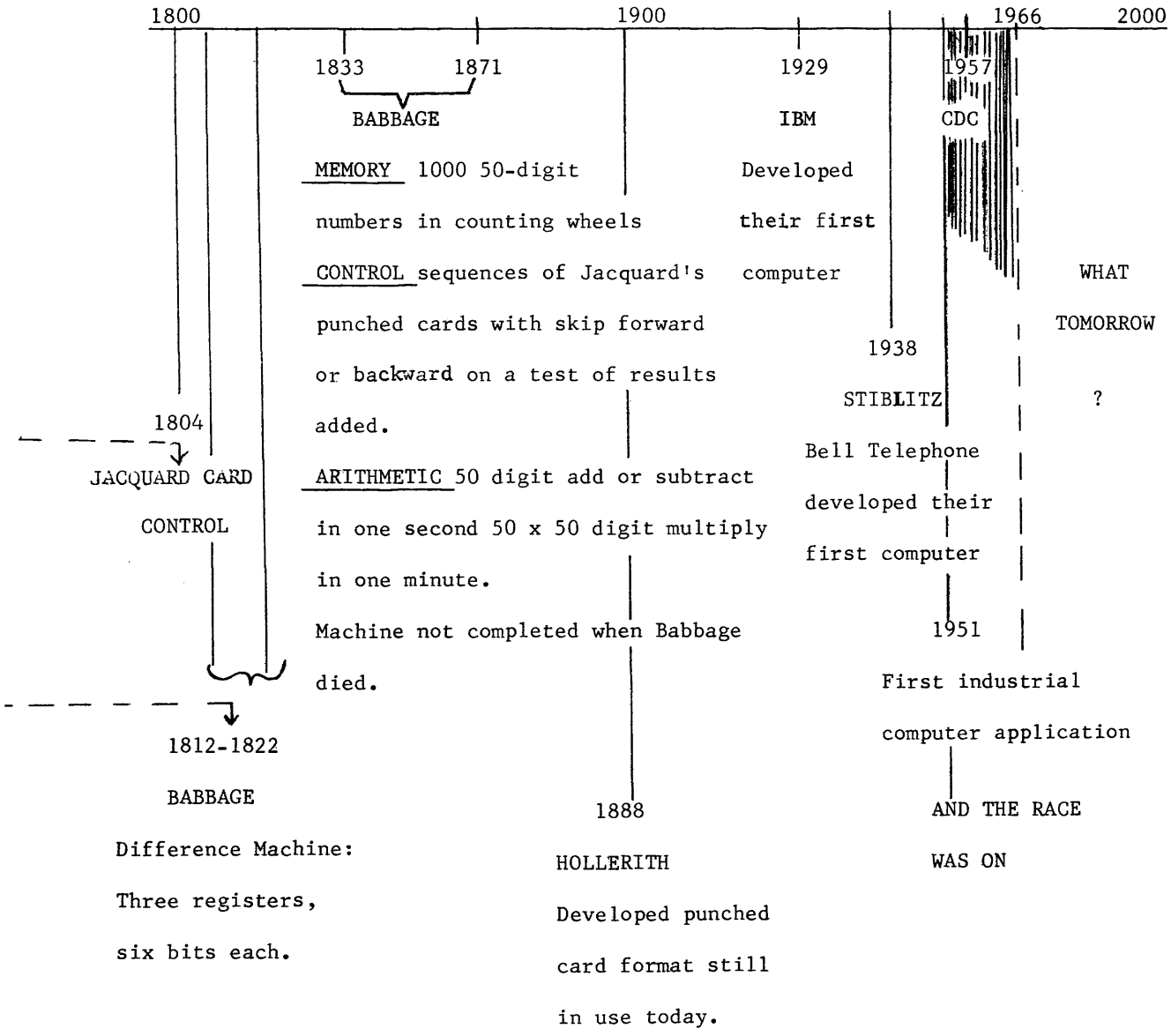
THE DEVELOPMENT



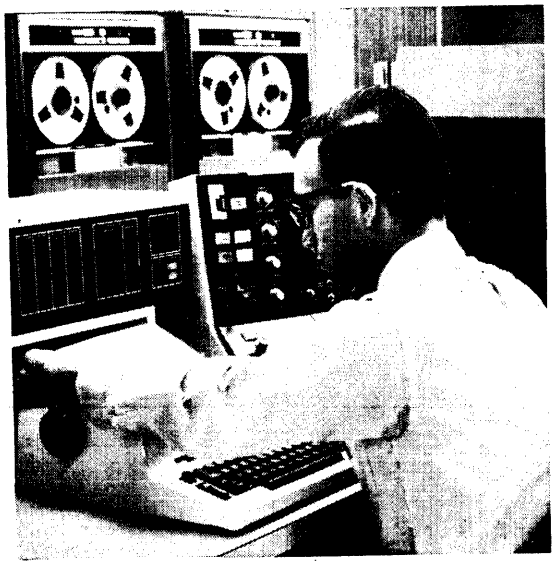
"I could not have seen so far if I had not stood on the shoulders of such giants".

Albert Einstein

OF THE COMPUTER



At Missouri's Medical Center, lab results are sent to the computer.



Behind computer console are the magnetic tapes that serve as memory.

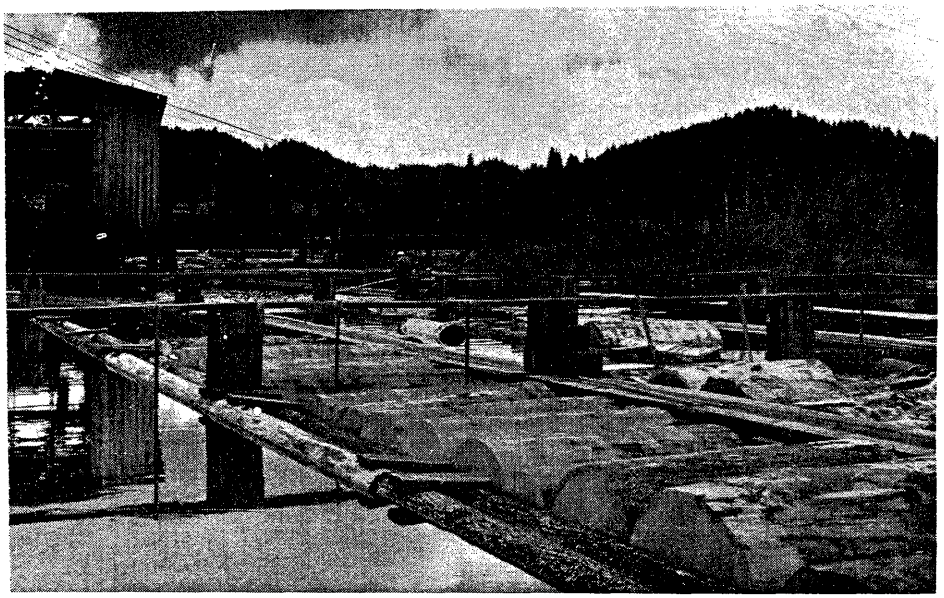
Rx for hospitals—computers

In Massachusetts and Missouri, they are taking over routine housekeeping chores, speeding communications, and making a wealth of data swiftly available for research

PRODUCTION

Finding new ways to make autos

Computers, numerically controlled tools, and other advanced techniques are clipping months off lead times to make cars. Besides economies, they will be a marketing boom



Dozens of grades and species of logs flow into lumber mill, where a computer takes over "direction" of the uses to which they will be put.

PRODUCTION

Computers point way to profits in lumber

Some of the largest forest products companies brought in computers to look over their operations. The results: In one instance, an additional $1-million in annual earnings

The tales told by American lumberjacks have always been as tall as the trees they cut—such as, for example, the logger who was so tough that when he needed a shave he'd knotty forest products business.

Complexity. Behind the interest in operations research is the growing complexity involved in allocation of the industry's raw material,

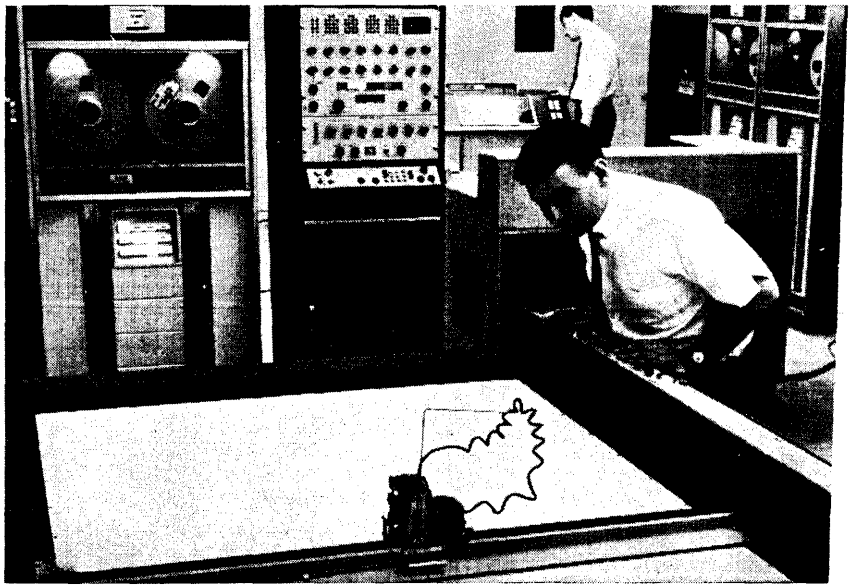
Can computers call the signals?

Worsening traffic problems worry local officials all over the nation. Working with manufacturers, they are conducting a number of tests and hope to find new answers

TRANSPORTATION

Doubling the freight car's workday

That's what computers may be able to do, by rationalizing the movement of freight, and thus vastly increasing railroad profits. They might even handle some operations



Dataplotter in Chicago receives information via telephone circuits from comp in Minneapolis. It can print lines at more than 200 in. a minute.

Draftsman with speed to spare

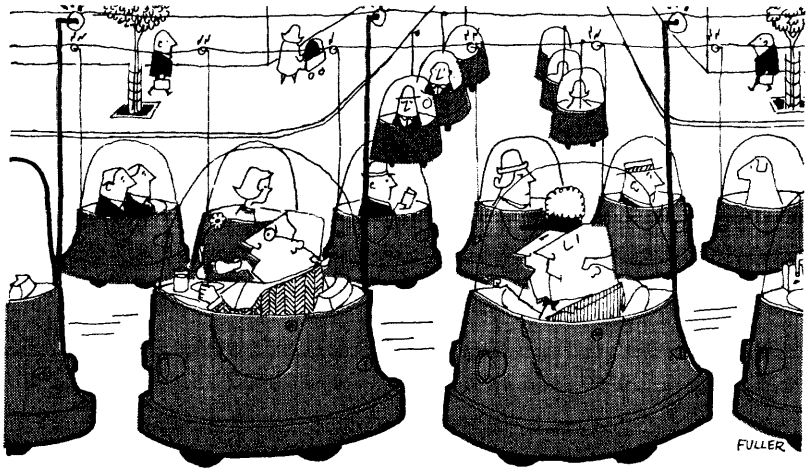
PRODUCTION

Keeping ahead on 'real time'

System provides up-to-the-minute production data, permits executives to act more quickly on problems. Companies hope to improve over-all efficiency, as well as on-time delivery

SPECIAL REPORT

Computers begin to solve the marketing puzzle

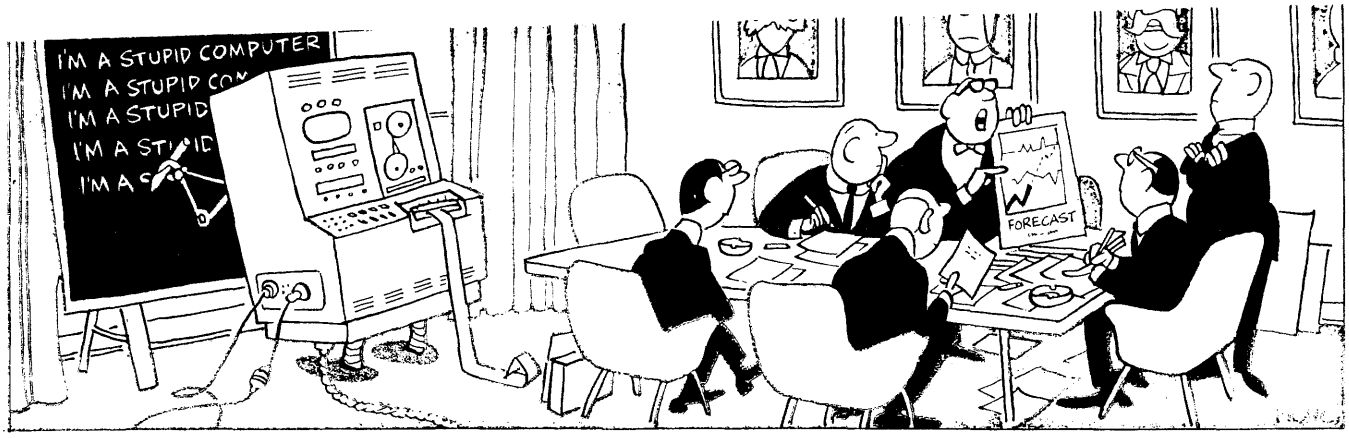


In 15 years, big-city commuters may own electric-powered Urbmobiles

TRANSPORTATION

Electronic roads for tomorrow's traffic

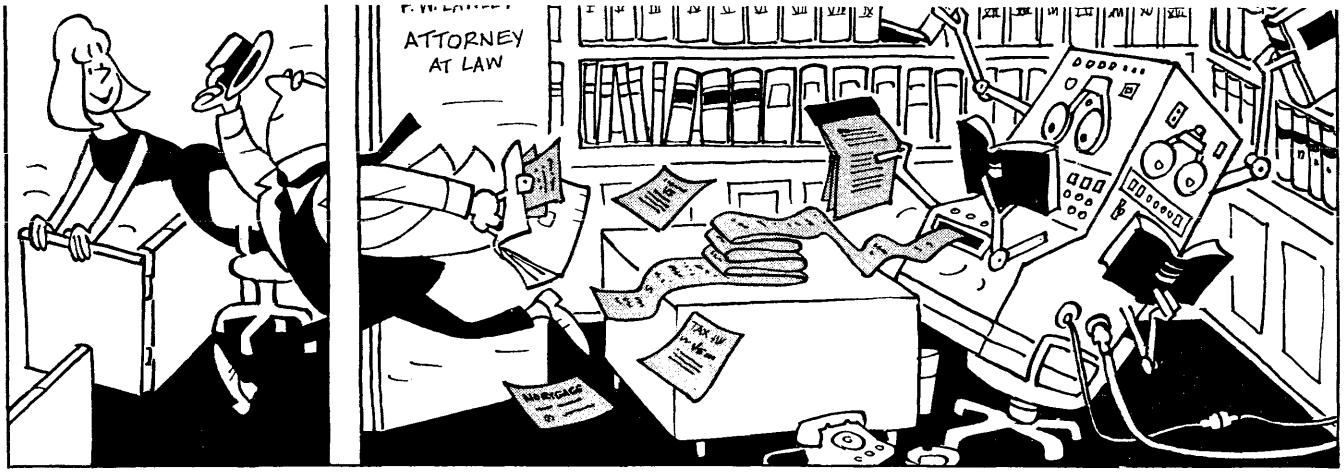
Cornell Lab study says mass transit isn't the solution for travel needs in Northeast. It proposes small electric autos, plus automation of long-haul traffic



The mindless computer, with nothing but the past to go on, runs a poor second to the economist in forecasting

ECONOMICS

How to rate the forecasters



Computers whiz through masses of data, release attorneys for more time with clients and in court

PRODUCTION

When computers do the digging

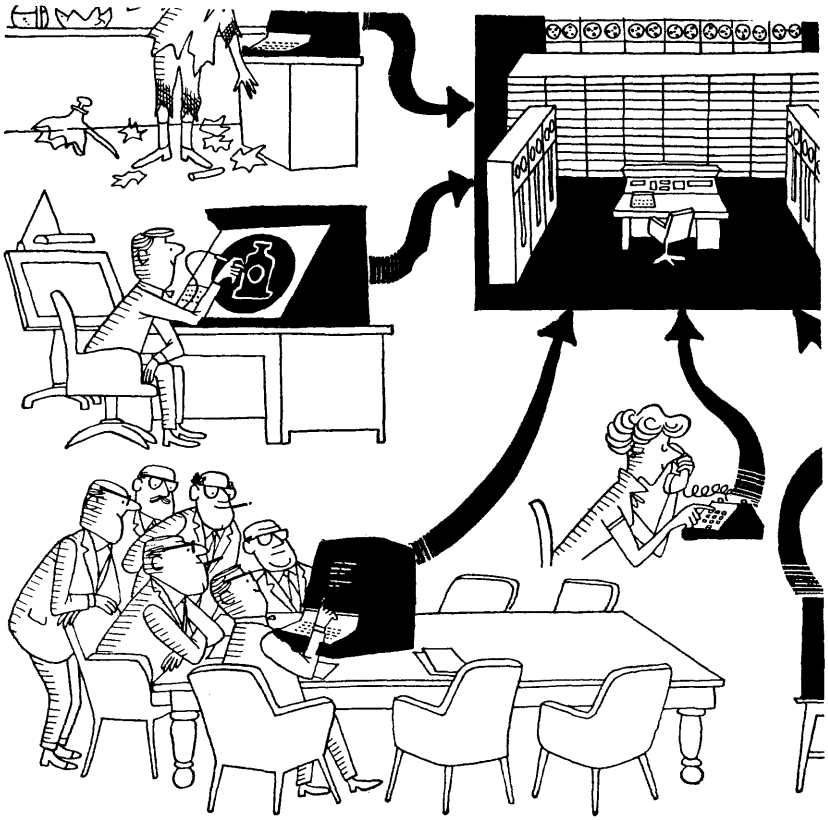
Lawyers are shifting much of their research burden to commercial computer services that speed up the hunt for statutes, key decisions, or previous rulings

Case research, the dusty and time-consuming chore necessary in most [illegible] in full-time business in N[illegible]

PRODUCTION

Cement strives to pour the proper profit mix

The drive to increase production and keep costs down has resulted in use of some startling new machinery. But it also has given industry an overcapacity headache



PRODUCTION

A lot of little users share a big computer

If you get enough small customers, you can keep even a giant computer busy through time sharing, selling service the way a utility sells power. Therein lies a new industry

BUSINESS WEEK August 7, 1965

Pillsbury Co.'s information system headed by James Rude, simulate markets, aids in scheduling crops and processing.

PRODUCTION

IBM buys its own sales pitch

It has fitted out one of the world's most highly automated design and production lines to make huge quantities of semiconductors for its System 360 computers in six plants

n job specifications becoming more and more precise, recruiters turn to computers to put a man in the management seat

ANAGEMENT

Picking top men–by electronics

Electronic data processing equipment is being used more and more in executive recruitment. New systems can keep up-to-date records on both jobs and applicants

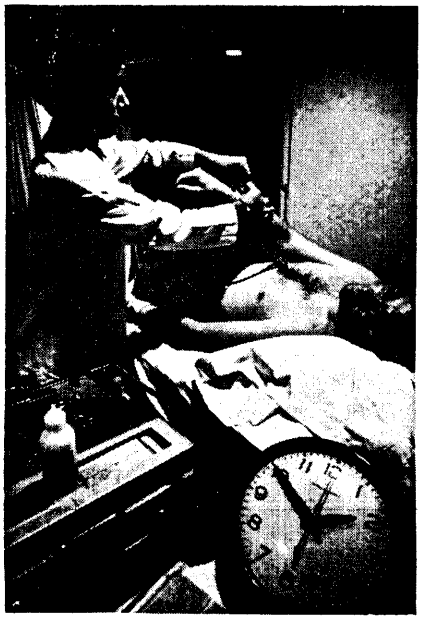
How Iris finds a job

Computers move up in personnel ranks

IBM location needs a scientist or professional

Job requirements listed on requisition sheet

IBM's new Recruitment Information System—Iris—is most sophisticated electronic machine yet for matching men and jobs. The key is its 12-page Data-Pak application form



Clock reads 1:50 as a wrist electrode is attached for the electrocardiogram and telephone relay to Washington . . .

Diagnosis by computer speeds heart checkup

A new electronics system for analyzing electrocardiograms in minutes uses a central information bank that will some day help doctors anywhere to spot heart attacks in time

Doctors have one device that's a two-edged sword in the fight against heart attacks. The electrocardiogram helps them (1) spot and [illegible] conditions that are likely to

ically in New York. The computer also sent back the average number of heart beats per min., and gave a brief written analysis of the data.

Lowering costs. The applications

Optical readers turn a fresh page

Electronic processing of vast quantities of records is passing from first to second generation machines that can read 1,200 characters a sec.; recognize more than 100 type faces

chapter III

Computer Mathematics

CHAPTER III

COMPUTER MATHEMATICS

INTRODUCTION

Before we begin our study of the digital computer, there are several other subjects that are worthy of note and, quite frankly, necessary in our understanding of a computer.

One of these subjects is the communications medium between the user and the device. Even if the computer could react to voice commands, the language that it uses must be understood by the programmer, the operator, and the engineer responsible for its maintenance.

The language of the computer consists entirely of numbers or numerical symbols. Because the operation of one computer may be based on one number system and a different computer on a different number system, it is necessary for us to be familiar with number systems in general.

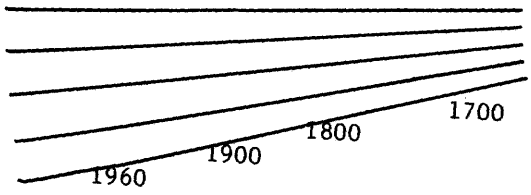
The objectives of this chapter are to teach you:

1) the origin of number systems.
2) the difference between positional and non-positional number systems.
3) what is meant by the radix of a number system.
4) what is meant by the modulus of a device.
5) positional values of numbers with different radices.
6) conversion procedures between two numbers of different radices.
7) arithmetic operations upon numbers of a given radix.
8) what is meant by complement and how to find the complement of any number.
9) what happens when the results of arithmetic operations exceed the modulus of the device.

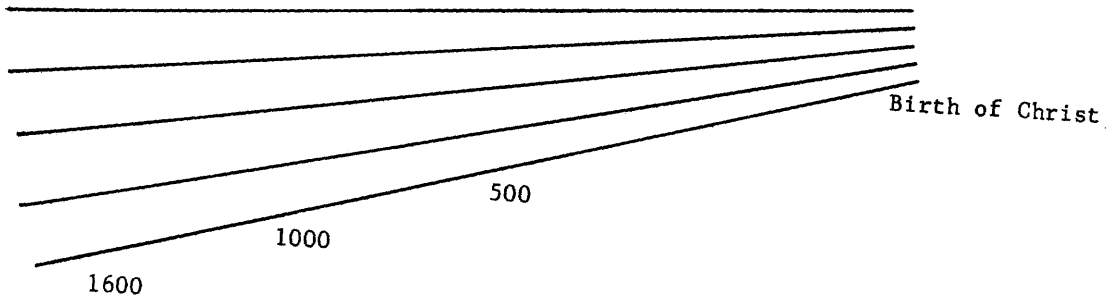
Well, it looks as though we have a job on our hands. You will be informed as you progress through the chapter when each of these objectives should have been satisfied. If you do not fully understand that topic, retreat and try again.

ORIGIN OF NUMBER SYSTEMS

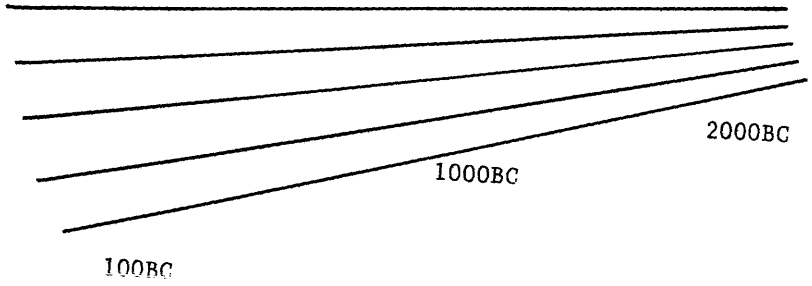
It is difficult to date the origin of a given number system, but let's go back in time.



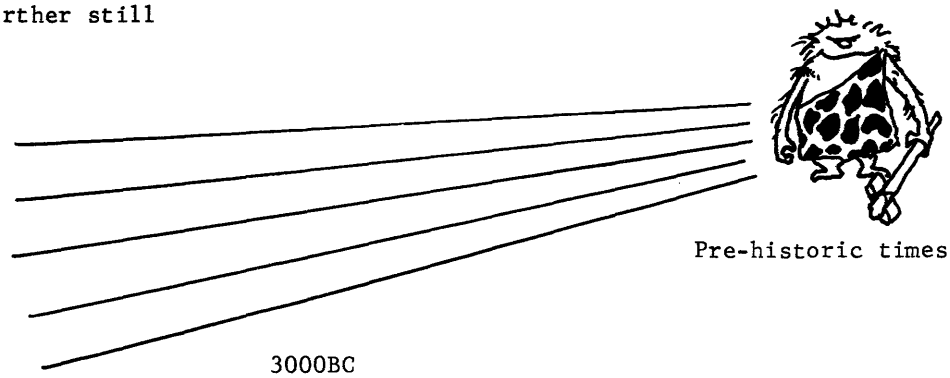
and back



further still



and further still



Pre-historic times

3000BC
Early Egyptians

to Charlie the Caveman.

Charlie's life was simple -- and so were his mathematical problems. However, he was a good hunter and was able to provide his family with an ample supply of dinosaur steaks and filet of mastodon. He kept track of his prowess as a hunter on his fingers, or digits.

One day, while Charlie was swinging his stone axe in close battle with an overly-ferocious dinosaur, a neighbor from several caves down the gully yelled a query as to how many dinosaurs that would make for the season. Charlie held up seven fingers, lost his grip on his axe, and nearly became hors d'oeuvres for a dinosaur dinner.

One close call taught Charlie a lesson -- there must be a better way. He tried keeping count by using a white rock for each dinosaur and a black rock for each mastodon that he killed. That system worked better but every passing beast kicked his rock pile and made Charlie lose count. In anger, Charlie threw his stone axe at the side of his cave. Much to his surprise, it left a permanent mark on the wall of his dwelling. Charlie had a system.

When Charlie passed on to that happy hunting ground, the inside of his cave looked like the fossilized rib cage of a boa constrictor.

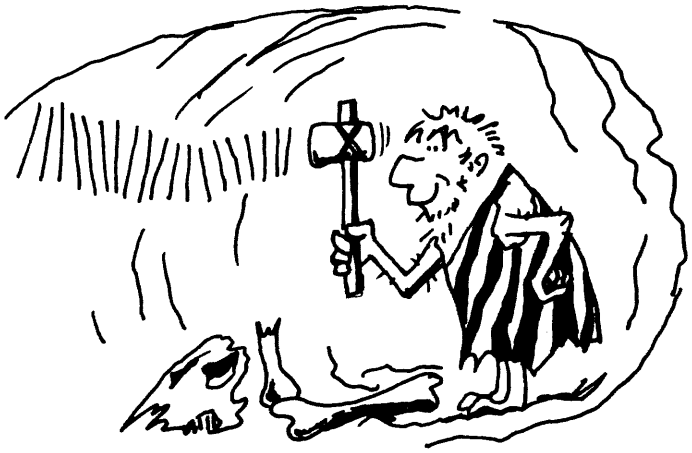


Figure 3-2. This may have been the first event of recording numbers

Let's advance with time to 3000BC and the early Egyptians. They had a number system (figure 3-3) which allowed them to express their large quantities of soldiers or cattle without recording a mark for each item. About the same time (give or take a thousand years), the Chinese were developing their own number system, also quite unique.

When Christ was born, the Roman Empire was gaining strength and the Romans devised their own number system, that of the familiar Roman numerals. This system is still in use today, but is limited in application.

The Mayan Indians of the Americas were discovered around 1500AD. They were highly civilized when discovered and had <u>their</u> own number system.

An example of numerical symbols for several civilizations is shown below. Each civilization apparently developed its own system, either due to lack of communication or because a different system failed to satisfy their need.

	1	2	3	4	5	6	7	8	9	10	100	1000
EGYPTIAN	𓏺	𓏺𓏺	𓏺𓏺𓏺	𓏺𓏺𓏺𓏺	𓏺𓏺𓏺𓏺𓏺	𓏺𓏺𓏺𓏺𓏺𓏺	𓏺𓏺𓏺𓏺𓏺𓏺𓏺	𓏺𓏺𓏺𓏺𓏺𓏺𓏺𓏺	𓏺𓏺𓏺𓏺𓏺𓏺𓏺𓏺𓏺	𓎆	𓍢	𓆼
HINDU	१	२	३	४	५	६	७	८	९	१०		
CHINESE	一	二	三	四	五	六	七	八	九	十	百	千
MAYAN	•	••	•••	••••	—	•—	••—	•••—	••••—	——		
ROMAN	I	II	III	IV	V	VI	VII	VIII	IX	X	C	M

Figure 3-3. Number Systems

It's difficult to pinpoint the origin of our decimal number system, but several authorities claim that it is traceable to Sanskrit, an ancient Indic language used by the Hindus. It consisted of nine basic characters which expanded to the present ten when the Arabic civilization added their own character for zero. This modified system was generally accepted throughout Europe in the seventh century and was fully developed by the time civilization expanded to the new world.

This development of a number system indicates the probable origin of our present-day decimal number system.

POSITIONAL AND NON-POSITIONAL NUMBER SYSTEMS

We have already discussed several types of number systems including the early Egyptian and the Roman numeral system. The Roman numeral is an example of a non-positional number system because the value of a symbol is always the same, independent of where it falls in the number. For example:

I always means one
V always means five
X always means ten

Therefore, XVI means 10 + 5 + 1 or 16. Combinations of symbols have separate meanings: IV means 4, while VI means 6. Construction dates of various historical buildings across the country are still recorded in Roman numerals. One of the main disadvantages of non-positional numbers is that they do not lend themselves to easy arithmetic operations.

For example:

```
Add      MXCVII
         MM
```

Of course the answer is MMMXCVII but how did you get it? You probably converted MXCVII to 1097 and MM to 2000, then came up with 3097 which, reconverted to Roman numerals, becomes MMMXCVII.

We stated that the Roman numeral character I always has a value of one, regardless where it appears in the number. This means that III equals 1 + 1 + 1, and VI equals 5+1.

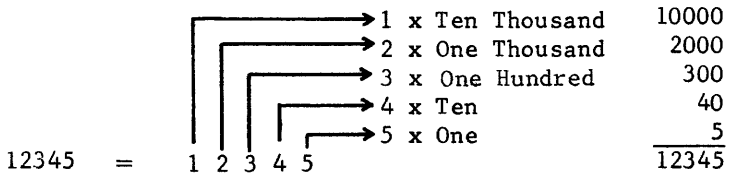
With that in mind, try another example:

```
Record the value of 29 in Roman numerals          XXIX
     Subtract 1 (I in Roman Numerals)              I
                          Leaves                  XXX - which is equal
                                                        to 30 in decimal.
```

If you subtract 1 from 29 in decimal, you don't get 30! Possibly the operations were slightly "slight of hand" but at least, you got the idea.

We're not interested in dating buildings, nor in non-positional number systems for that matter. However, we do know what a non-positional number system is, and also know its drawbacks.

Let's now look at the other side of the picture -- the positional number system. With this system, the position in which a digit of a number falls determines the value of the number. Each succeeding position to the left of a given digit is some multiple of the lower digit. In our familiar decimal number system, this multiple would be 10. The first column to the left of the (decimal) point is the ones column, the next is the tens column, etc. For example:



Moving the (decimal) point to the right one position would increase the value of the number by a factor of 10 and to the left would decrease the value of that number by a factor of 10.

123450. and 1234.5 respectively.

This type of number system does lend itself to easy arithmetic operations and will simplify our job of trying to make a number system compatible with a digital computer.

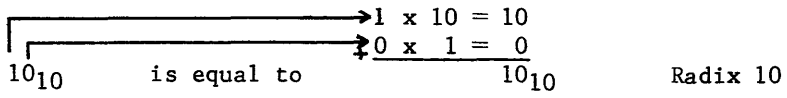
Now that we know the difference between a positional and a non-positional system, we can proceed to bigger and better things. Incidentally, was the Chinese number system positional or non-positional? No wonder they came up with the abacus.

RADIX OF A NUMBER SYSTEM

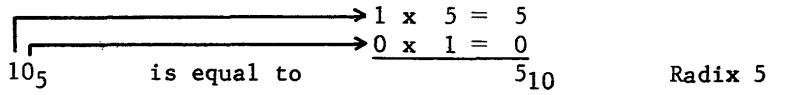
Any positional number system has a **radix** (or base) associated with it. By definition, radix means a number that is arbitrarily made the fundamental number of a number system. A more suitable explanation may be: The number of distinct values that may be expressed in any given position, starting with 0.

For example: In the ones column of a decimal number, we could express any value 0-9. These are the 10 decimal digits so the radix of a decimal number is 10. If this is true, what is the largest number that could be expressed in the ones position if the number had a radix of 5? Starting with 0 we would also include 1, 2, 3, and 4; five distinct values according to our definition. The radix of a number system is usually indicated by a subscript following the number. Example: 12345_{10}, where the subscript 10 indicates the radix of the number 12345. The decimal system is so commonplace we do not usually bother to indicate the subscript 10; however, from this point on, the radix should not be assumed and, therefore, should be indicated.

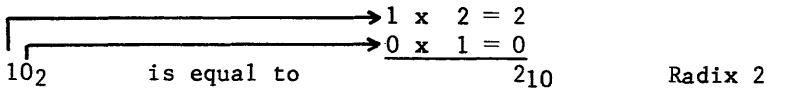
Another interesting fact is that the tens column of a radix 10 number suddenly becomes the fives column of a radix 5 number and the twos column of a radix 2 number. The symbol 10 (one-zero, not ten) always represents the radix of its own system. This is true because the radix is one unit larger than the largest character. For example:



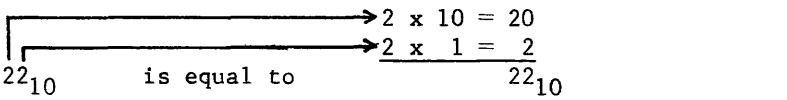
whereas



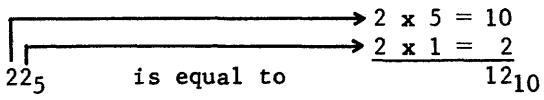
and



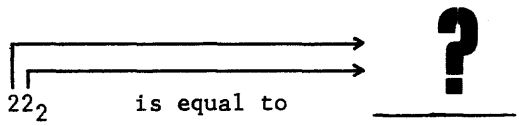
Now, let's try the same thing with another number.



whereas



and



Now we have a problem and also an impossibility. We are trying to express the decimal equivalent of a number with a radix of 2. According to our definition of radix, we could only express the digits 0 and 1 in the radix 2 number system. The number 22 would require a radix of <u>at least</u> 3. The number 22_2, therefore, is an impossibility.

How many of the following numbers are also impossibilities?

1. 1101101_2
2. 1234567_7
3. 98_9
4. 1000_{10}
5. 77_8
6. 70707_7
7. 1234_9
8. 43210_7
9. 00010_8
10. 10000_2

Review the definition of radix and re-check your answers. Only examples 2, 3, and 6, are impossible numbers. If you do not agree, you have misinterpreted the definition of radix. If so, turn back a few pages to the beginning of this section and start over. If we agree, you now understand the definition of radix and you deserve a 50 nanosecond rest (radix 10).

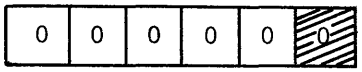
Know how long that is? Your rest period is 50 one-billionths of a second. We'll soon discover that to be a substantial period of time, when talking about some computer operations.

Rest period is over so let's continue.

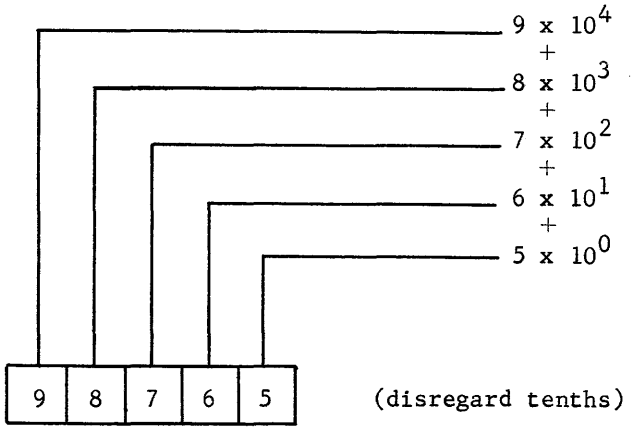
MODULUS OF A DEVICE

The modulus of a device can be defined as the number of unique quantities the device can express. The modulus is expressed as the radix of a number system raised to some exponent.

The odometer (mileage indicator) of an automobile looks something like this:



The following odometer reading could be expressed in radix 10 as



The largest number that could be expressed would then be

$$99999_{10}.$$

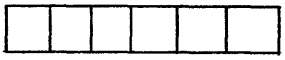
If we include the value 00000, we would have $100{,}000_{10}$ distinct values that could be expressed by the odometer. The number $100{,}000_{10}$ could be expressed

as

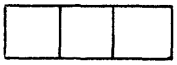
$$\begin{array}{r} 1 \times 10^5 \\ + \\ 0 \times 10^4 \\ + \\ 0 \times 10^3 \\ + \\ 0 \times 10^2 \\ + \\ 0 \times 10^1 \\ + \\ \underline{0 \times 10^0} \\ 1 \times 10^5 \end{array}$$

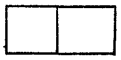
100, 000.

or more simply as 10^5, the <u>modulus</u> of the odometer. Notice that the exponent 5 is also the number of discrete positions available in our odometer. How would you express the modulus of the following odometers?

1.  radix 3

2.  radix 9

3.  radix 2

4.  radix 10

5.  radix 5

Back to the definition of modulus and how the modulus of a device is expressed: The radix of a number system raised to some exponent.

In example 1, the radix is 3 raised to the exponent 6. The modulus, therefore, can be defined as 3^6.

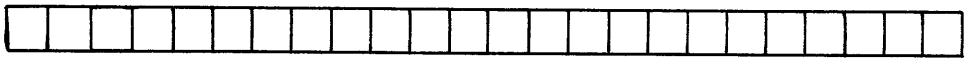
Example 2 has a radix of 9 with the possibility of expressing 9^4 different quantities. Hence, the modulus 9^4.

Example 3 -- modulus 2^3

Example 4 -- modulus 10^2

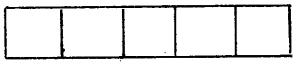
Example 5 -- modulus 5^{16}

With all this in mind, how would you express the modulus of the following?

 Radix 2

If you answered this one as modulus 2^{24}, you are correct. Keep this example in mind for later use. It will become very important in further study of number systems associated with a digital computer.

Now that we completely understand modulus and radix, let's look at some other examples.

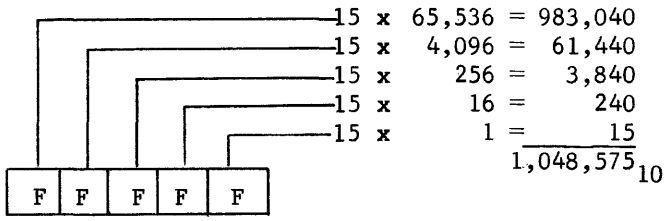
In another device,  assume that

each position can express distinct values of 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F. What is the radix of our new number system? Well, how many distinct values can be expressed in a given position? The correct answer is 16, making the radix of our device, by definition, also 16. The modulus of the device having five positions is expressed as 16^5.

Certain equalities exist between radix 16 and radix 10. Let's compare them.

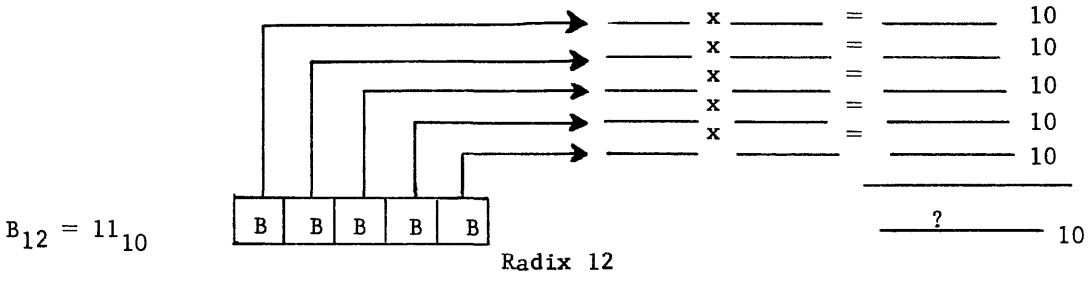
$0_{16} = 0_{10}$
$1_{16} = 1_{10}$
$2_{16} = 2_{10}$
$3_{16} = 3_{10}$
$4_{16} = 4_{10}$
$5_{16} = 5_{10}$
$6_{16} = 6_{10}$
$7_{16} = 7_{10}$
$8_{16} = 8_{10}$
$9_{16} = 9_{10}$
$A_{16} = 10_{10}$
$B_{16} = 11_{10}$
$C_{16} = 12_{10}$
$D_{16} = 13_{10}$
$E_{16} = 14_{10}$
$F_{16} = 15_{10}$

Now, let's examine our radix 16 device with the following contents. What would be the decimal equivalent?

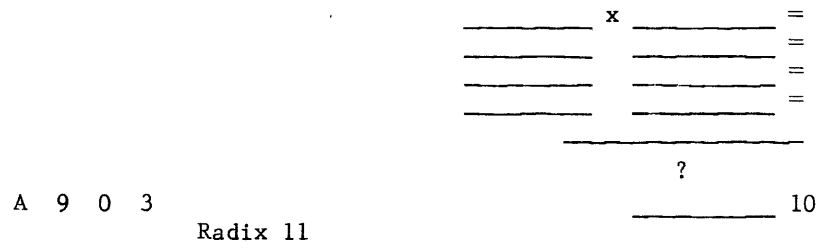


Remember, the radix of the device was 16, so the value of each positional character is 16 times the value of the character immediately to its right.

Just for practice, let's do a few more.



$B_{12} = 11_{10}$



Your solution to the radix 12 problem should look something like this:

$12^4 = 20{,}736$ so $11 \times 20{,}736 = 228{,}096$
$12^3 = 1{,}728$ $11 \times 1{,}728 = 19{,}008$
$12^2 = 144$ $11 \times 144 = 1{,}584$
$12^1 = 12$ $11 \times 12 = 132$
$12^0 = 1$ $11 \times 1 = 11$
Therefore $BBBBB_{12} = 248{,}831_{10}$

And your solution to the radix 11 problem should look something like

$A_{11} = 10_{10}$ and $11^3 = 1{,}331$

$9_{11} = 9_{10}$ and $11^2 = 121$

$0_{11} = 0_{10}$ and $11^1 = 11$

$3_{11} = 3_{10}$ and $11^0 = 1$

SO

Thus A x 1,331 = 13,310

9 x 121 = 1,089

0 x 11 = 0

3 x 1 = 3

Therefore $A903_{11} = 14{,}402_{10}$

If our answers agree and you completely understand what you have done, we have satisfied the objectives through number 5. With your solid understanding of modulus, radix, and positional values of numbers, you are now ready to proceed. If your understanding is not solid, review this section.

SELECTION OF A SUITABLE NUMBER SYSTEM

If you were to design a modern digital computer, you would first decide upon a number system, or radix, on which to base your design criterion. Assume that you decided to use the familiar decimal number system and you are now ready to begin. Remember that the decimal system has a radix of 10 which means that 10 distinct values could be expressed in each position. We have already talked about a similar device, the odometer which looked something like this:

5	6	7	8	9

The number 9 in the units position means that we must also be able to express all values below it including 0. A total of ten different digits could appear in that position. Each of these positions will be represented by an electronic circuit in the actual computer. If a transistorized circuit is used to represent that position, the circuit must be capable of conducting at ten different levels, each level representing a decimal digit. As a transistor started to weaken, a voltage level originally intended to represent a 9_{10} could then be misinterpreted to represent 8_{10} or 7_{10} or some other value, depending upon how much it weakened. This would result in gross calculation errors and transform the computer into a useless heap of electronic components.

One of the more common applications of a transistor is that of a switch, (not the willow type you recall from childhood, but the electrical type, similar to the wall switch). We realize that that type of switch has two positions, or states, either "off" or "on." A switching transistor also has two states -- off or on, conducting or non-conducting, saturation or cutoff, high level or low level, yes or no. What number system has only two discrete values comparable to the off or on of a transistor?_____ The radix 10 system has 10 values, the radix 5 system has 5 values, so it must hold true that a radix 2 system would have two values. Remember the

definition of radix? What would those two values be? Don't forget that we always start with 0. You're right! The two values would be 0 and 1. Henceforth, we will refer to the radix 2 system as the binary (bi meaning two) number system.

The majority of digital computer systems are designed around the binary number system. This means that if we wish to add

$10{,}000_{10}$ = $010\ 011\ 100\ 010\ 000_2$

and $5{,}000_{10}$ = $001\ 001\ 110\ 001\ 000_2$

We come up with $011\ 101\ 010\ 011\ 000_2$

Which is equal to $15{,}000_{10}$

That looks like a lot of ones and zeros with not much real meaning, doesn't it? You're right, but consider for a moment a long-hand multiplication problem:

```
          1 234 567
X         7 654 321
          ---------
          1 234 567
         24 691 34
        370 370 1
      4 938 268
     61 728 35
    470 740 2
  8 641 969
  ---------------
  9 449 772 114 007
```

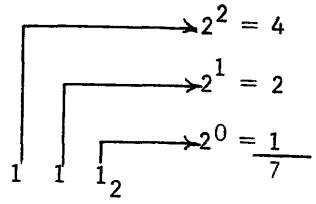
What were we really interested in as we worked the problem? Right! The solution. We were interested in the intermediate steps only because they were necessary to achieve a result. Likewise, our primary interest in the computer is that it will provide us with a result, an answer, or a solution to a problem. From a mathematician's point of view, he couldn't care less what happens within the computer. Operations will be taking place in billionths of a second, at a speed almost beyond comprehension. When the computer stops, we will be interested in the result.

That result was formed by combinations of ones and zeros. If we could find a simpler method of displaying the results of calculations other than with only ones and zeros, it would simplify the interpretation of results achieved by the computer.

This is exactly what is done and is also why we label a computer as an "octal" or a "hexadecimal" machine. Octal is derived from the Greek prefix, okt -- meaning eight; hexadecimal is derived from two Greek prefixes, hex -- meaning six and dek -- meaning ten (16). The label "octal" or "hexadecimal" indicates that the computer's results are displayed in octal or hexadecimal although internal operations are in binary.

Octal and hexadecimal displays, readouts, or printouts are quite useful and quite easily adapted to a binary system because both eight and sixteen

are integral powers of two. One octal (radix 8) digit may be expressed by three bits (8 = 2x2x2) and one hexadecimal digit may be expressed by four bits (16 = 2x2x2x2). The term BIT is derived from BInary digiT. In a radix 2 number system, each succeeding position to the left is equal to twice that of its predecessor. For example:



A 1 in the 2^2 position would have a decimal equivalent of 2^2 or 4, whereas a 1 in the 2^1 position would have half that value (2^1 or 2).

A 0 in any position gives that position a value of 0. This means that the 2^2 position could indicate a value of 4 or a value of 0, but nothing else. The 2^1 position would have a value of either 2 or 0 and the 2^0 position would indicate a value of either 1 or 0.

Let's adopt a set of rules for counting that will remain valid for any radix number system using positional notation.

1. Starting with zero, add one to the least significant digit until all basic characters have been used.

Radix 2	Radix 8	Radix 10	Radix 16
0	0	0	0
1	1	1	1
	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
		8	8
		9	9
			A
			B
			C
			D
			E
			F

2. Since we have already expressed the largest value for that position, a larger number would require two digits. Always start the series of two digit numbers with 10 (one, zero).

Radix 2	Radix 8	Radix 10	Radix 16
10	10	10	10

3. Whenever any digit reaches its maximum value, replace it with zero and add one to its next more significant digit.

Radix 2	Radix 8	Radix 10	Radix 16
0	0	0	0
1	1	1	1
→10	2	2	2
	3	3	3
	4	4	4
	5	5	5
	6	6	6
	7	7	7
	→10	8	8
		9	9
		→10	A
			B
			C
			D
			E
			F
			→10

4. When two or more consecutive digits reach their maximum value, replace them both with zeros and add one to the next more significant digit.

Radix 2	Radix 8	Radix 10	Radix 16
.	.	.	.
.	.	.	.
.	.	.	.
10	76	98	FE
11	77	99	FF
→100	→100	→100	→100
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
110	276	398	CFE
111	277	399	CFF
→1000	→300	→400	→D00

To re-state a previous point:
The symbol 10 (one, zero) always represents the radix of its own system. This is true because the radix is one unit larger than the system's largest

character, and according to rule 3.

Binary $10_{\swarrow 2^1 = 2}$

Decimal $10_{\swarrow 10^1 = 10}$

Octal $10_{\swarrow 8^1 = 8}$

Hexadecimal $10_{\swarrow 16^1 = 16}$

Let's try counting ...

in binary,	in octal,	in decimal,	and in hexadecimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11
10010	22	18	12
10011	23	19	13
10100	24	20	14

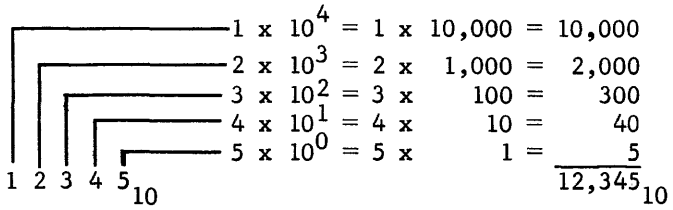
It looks as though 10100_2, 24_8, 20_{10}, and 14_{16}, should all equal the same quantity. Let's try to prove it.

CONVERSION PROCEDURES

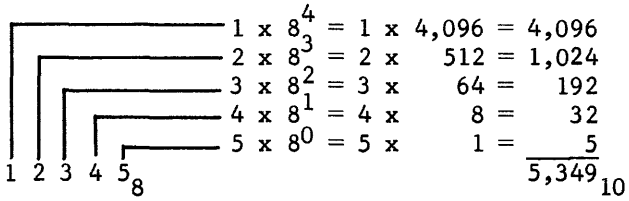
Three new number systems have been introduced; binary, octal, and hexadecimal. Many others exist and are sometimes used in computing devices. Each number system is unique and has distinct advantages not common to other systems. The objective of this section is to teach you how to determine equalities between numbers of different radices. You should then be able to associate any given number system with any other number system.

One procedure, polynomial expansion, can be used for all conversions. In some cases, a simpler method will be illustrated. If you forget that simpler method, you can always revert back to polynomial expansion.

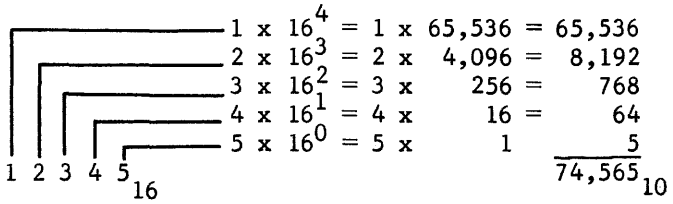
An example of the polynomial expansion of a decimal number is shown below.



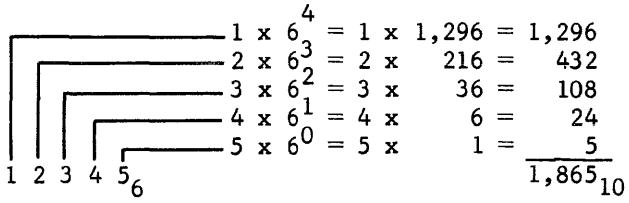
This time, use the same characters but a different radix. Find the decimal equivalent of 12345 octal. An exponential powers table is located at the end of this chapter.



Now try the same example in hexadecimal (radix 16)



and again using radix 6



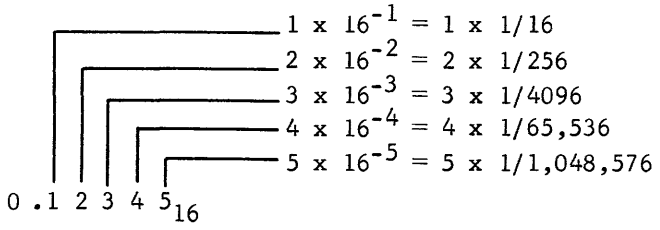
Find the decimal equivalent of 12345_{14} and 12345_{12} to complete the table.

$12345_{16} = 74,565_{10}$
$12345_{14} =$ _____$_{10}$
$12345_{12} =$ _____$_{10}$
$12345_{10} = 12,345_{10}$
$12345_{8} = 5,349_{10}$
$12345_{6} = 1,865_{10}$

Could you find the decimal equivalent of 12345_4 and 12345_2? Explain your answer.

The same procedure remains valid for fractions. The only difference is that we are now working with negative exponents.

Example 1 $0.12345_{16} =$._____$_{10}$

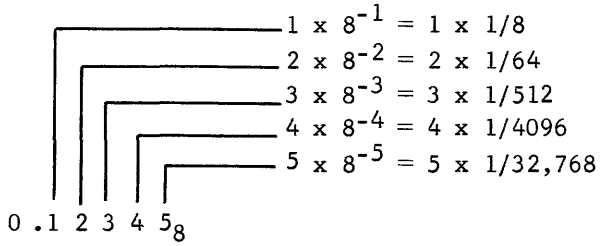


The least common denominator is 1,048,576 which renders this equality

$$
\begin{aligned}
1 \times 65{,}536/1{,}048{,}576 &= 65{,}536/1{,}048{,}576 \\
2 \times \ 4{,}096/1{,}048{,}576 &= \ 8{,}192/1{,}048{,}576 \\
3 \times \ \ 256/1{,}048{,}576 &= \ \ 768/1{,}048{,}576 \\
4 \times \ \ 16/1{,}048{,}576 &= \ \ 64/1{,}048{,}576 \\
5 \times \ \ 1/1{,}048{,}576 &= \ \ \underline{5/1{,}048{,}576} \\
&\quad 74{,}565/1{,}048{,}576
\end{aligned}
$$

and by long division = 0.07206_{10}

Example 2 $0.12345_8 =$._____$_{10}$



which is equal to

$$
\begin{aligned}
1 \times 4{,}096/32{,}768 &= 4{,}096/32{,}768 \\
2 \times \ 512/32{,}768 &= 1{,}024/32{,}768 \\
3 \times \ \ 64/32{,}768 &= \ \ 192/32{,}768 \\
4 \times \ \ 8/32{,}768 &= \ \ 32/32{,}768 \\
5 \times \ \ 1/32{,}768 &= \ \ \underline{5/32{,}768} \\
&\quad 5{,}349/32{,}768 = 0.16323_{10}
\end{aligned}
$$

Example 3 $0.12345_6 = .\_\_\_\_\_{}_{10}$

$1 \times 6^{-1} = 1 \times 1/6 = 1{,}296/7{,}776$
$2 \times 6^{-2} = 2 \times 1/36 = 432/7{,}776$
$3 \times 6^{-3} = 3 \times 1/216 = 108/7{,}776$
$4 \times 6^{-4} = 4 \times 1/1296 = 24/7{,}776$
$5 \times 6^{-5} = 5 \times 1/7776 = 5/7{,}776$

$0.1\ 2\ 3\ 4\ 5$

$1{,}865/7{,}776 = .23984_{10}$

Find the decimal equivalents of 0.12345_{14} and 0.12345_{12} to complete the following chart.

$0.12345_{16} = 0.07111_{10}$
$0.12345_{14} = 0.\_\_\_\_\_{}_{10}$
$0.12345_{12} = 0.\_\_\_\_\_{}_{10}$
$0.12345_{10} = 0.12345_{10}$
$0.12345_{8} = 0.16323_{10}$
$0.12345_{6} = 0.23984_{10}$

Again, let's examine the answers. Do your solutions agree with the progression (non-linear) already established? They should!

Remember the counting table back on page 3-17? The bottom line of that table indicated that 10100_2, 24_8, 20_{10}, and 14_{16} were all equalities. We should now be able to prove those equalities with the expansion process.

Problem 1. $10100_2 = \_\_\_\_\_{}_{10}$

$1 \times 2^4 = 16$
$1 \times 2^2 = 4$

10100_2

20_{10}

Problem 2 $24_8 = \_\_\_\_\_{}_{10}$

$2 \times 8^1 = 16$
$4 \times 8^0 = 4$

24_8

20_{10}

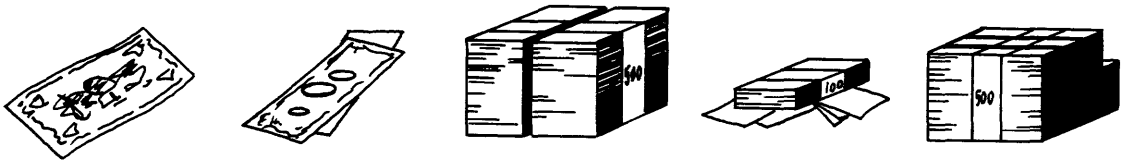
Problem 3 $14_{16} = \_\_\_\_\_{}_{10}$

$1 \times 16^1 = 16$
$4 \times 16^0 = 4$

14_{16}

20_{10}

Well, that's a relief. They are all equalities. Compare them to



1	=	$2.80	=	1000	=	112	=	1750
English Pound		American Money		Japanese Yen		Taiwan Yuan (N.T.Dollars)		Italian Lira

(legal exchange rate as of late 1965)

Each is a different amount in a given (monetary) system yet each would purchase approximately the same amount of goods in this country. Therefore, they are considered to be equalities.

Before going on to other methods, let's check our progress to this point by working a few practice problems. You may check your answers with those listed on page 3-76 at the end of this chapter. A table of exponential powers is also listed at the end of this chapter on page 3-75.

Practice Problems (Polynomial expansion method)

1. 10101010_2 = _____$_{10}$
2. 7034_8 = _____$_{10}$
3. $ABCBA_{16}$ = _____$_{10}$
4. $111\ 000\ 111_2$ = _____$_{10}$
5. 6789_8 = _____$_{10}$
6. $F4240_{16}$ = _____$_{10}$
7. 0.110011_2 = ._____$_{10}$
8. 0.1234_8 = ._____$_{10}$
9. $0.ABAB_{16}$ = ._____$_{10}$
10. 0.00001_2 = ._____$_{10}$
11. 0.4000_8 = ._____$_{10}$
12. $0.FEED_{16}$ = ._____$_{10}$

If your answers check with those at the back of the chapter, congratulations!

If they don't, back you go.

Perhaps you have already noticed that all of the conversions to this point have been to radix 10. Also, all of the mathematical operations have been performed in decimal -- the base to which we were going. A logical assumption would be that conversions to radix 8 would require octal arithmetic; conversions to radix 2 would require binary arithmetic; to radix 16, hexadecimal arithmetic; etc. Well, the assumption is valid and the procedure would work. However, we do not presently know how to perform arithmetic operations using binary, octal, and hexadecimal arithmetic. The next section of this chapter will deal with these operations.

It was previously stated that the binary, octal, and hexadecimal number systems are quite compatable because they are all integral powers of two (2^1, 2^3, 2^4). It's readily apparent that $2^1 = 2$, $2^3 = 8$, and $2^4 = 16$ (the radices of the binary, octal, and hexadecimal number systems). Conversions between these bases should be quite simple.

Problem 1 Convert $ABCD._{16}$ to _____8

Step 1 Record the binary equivalent of each of the hexadecimal characters.

A	B	C	D
1010	1011	1100	1101.

Step 2. Re-group the binary digits in groups of three, starting at the binary point. Complete the high-order group with zeros.

001 010 101 111 001 101_2

Step 3. Convert each group of three digits to its octal equivalent

001	010	101	111	001	101
1	2	5	7	1	5

$ABCD_{16} = 125715_8$

Problem 2 Convert 70707_8 to _____16

Step 1 Record the binary equivalent of each octal character

7	0	7	0	7
111	000	111	000	$111._2$

Step 2 Re-group the binary digits in groups of four, starting at the binary point. Complete the high-order group with zeros.

0111 0001 1100 0111

Step 3 Convert each group of four binary digits to its hexadecimal equivalent.

$$\begin{array}{cccc} 0111 & 0001 & 1100 & 0111 \\ 7 & 1 & C & 7_{16} \end{array}$$

$$70707_8 = 71C7_{16}$$

Using the procedures explained in problem 1 and 2, you should be able to convert any number from radix 2, 8, or 16 to the other two radices. Let's try a few more practice problems. You'll find the answers on page 3-76, at the end of the chapter.

Practice Problems

13. $10101_2 =$ _____$_8$
14. $10101_2 =$ _____$_{16}$
15. $111000_2 =$ _____$_8$
16. $111000_2 =$ _____$_{16}$
17. $7036_8 =$ _____$_2$
18. $7036_8 =$ _____$_{16}$
19. $5252_8 =$ _____$_2$
20. $5252_8 =$ _____$_{16}$
21. $BEAD_{16} =$ _____$_2$
22. $BEAD_{16} =$ _____$_8$
23. $DEED_{16} =$ _____$_2$
24. $DEED_{16} =$ _____$_8$
25. $0.1010_2 =$ _____$_8$
26. $0.1010_2 =$ _____$_{16}$
27. $0.000\ 000\ 000\ 100_2 =$ _____$_8$
28. $0.000\ 000\ 000\ 100_2 =$ _____$_{16}$
29. $0.00123_8 =$ _____$_2$
30. $0.00123_8 =$ _____$_{16}$

31. 0.7654_8 = _____ 2

32. 0.7654_8 = _____ 16

33. $0.123ABC_{16}$ = _____ 2

34. $0.123ABC_{16}$ = _____ 8

35. $0.FFF_{16}$ = _____ 2

36. $0.FFF_{16}$ = _____ 8

The only conversions yet to be performed are those from radix 10 to other radices. As previously stated, the polynomial expansion process is still valid <u>if</u> the mathematical operations are in the radix to which you are going. Let's learn other methods that will allow us to still use decimal arithmetic. When you have learned how to perform mathematical operations in binary, octal, and hexadecimal, you may then choose the method easiest for you.

DIVISION (WHOLE NUMBERS)

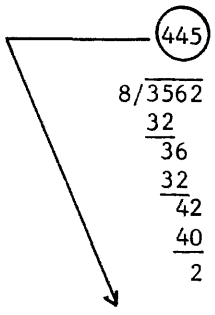
Example 1: Decimal to Octal

3562 (10) _____ (8)

/3562

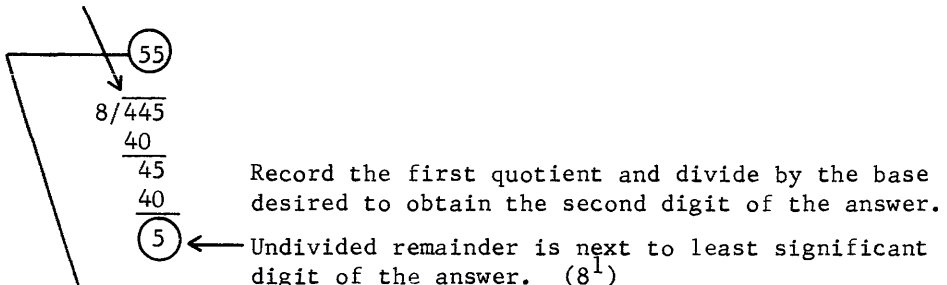
Record the number to be converted and divide by the base desired (could be 8, 5, 7, 3, 2, etc.)

Step 1.

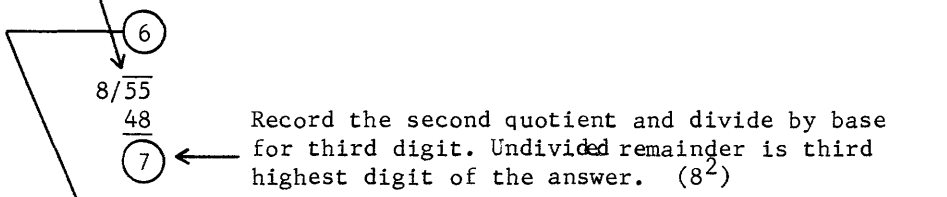


← Undivided remainder is least significant digit of the answer. (8^0)

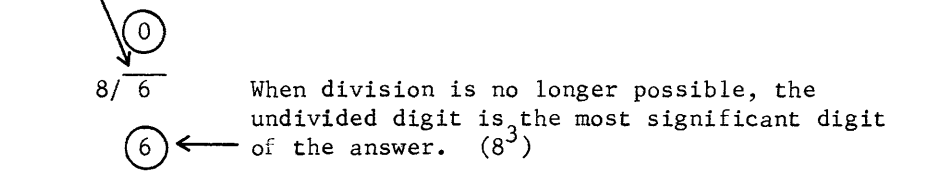
Step 2.



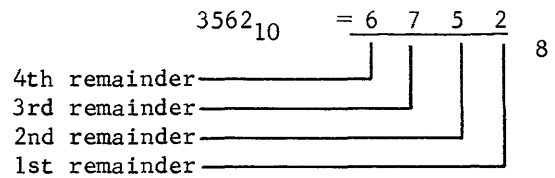
Step 3.



Step 4.



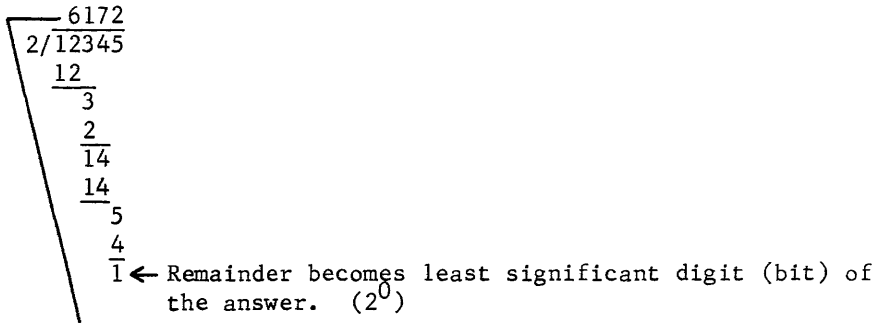
Then by recording the undivided remainders it can be seen that:

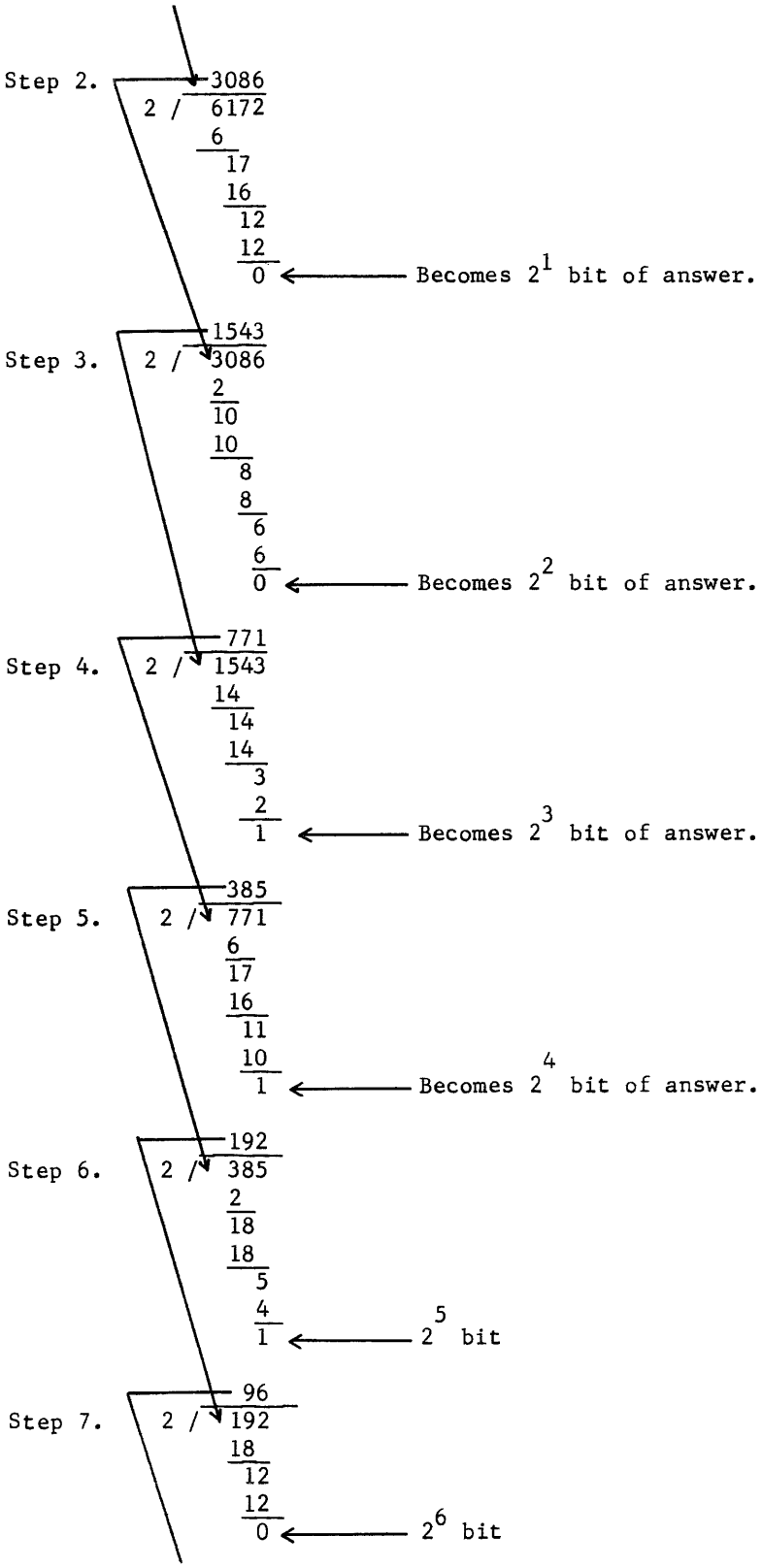


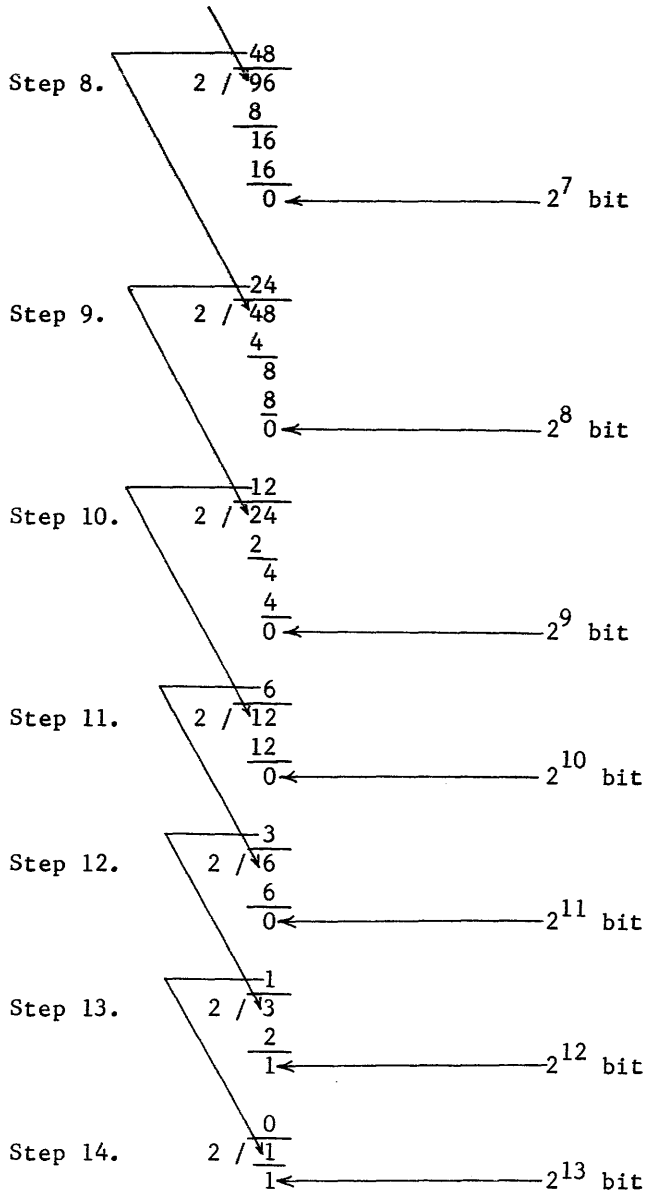
Example 2 Decimal to binary

12345_{10} = _____ 2

Step 1.




Step 2.
3086
2 / 6172
6
17
16
12
12
0
Becomes 2^1 bit of answer.
Step 3.
1543
2 / 3086
2
10
10
8
8
6
6
0
Becomes 2^2 bit of answer.
Step 4.
771
2 / 1543
14
14
14
3
2
1
Becomes 2^3 bit of answer.
Step 5.
385
2 / 771
6
17
16
11
10
1
Becomes 2^4 bit of answer.
Step 6.
192
2 / 385
2
18
18
5
4
1
2^5 bit
Step 7.
96
2 / 192
18
12
12
0
2^6 bit



Therefore $12345_{10} = \underline{11\ 000\ 000\ 111\ 001}_2$

Now that that is over, let's try the same thing an easier way.

Convert 12345_{10} to radix 8 (30071), then, by inspection, convert 30071_8 to $011\ 000\ 000\ 111\ 001_2$.

EXAMPLE 3. Decimal to Hexadecimal

Convert 12345_{10} to _____16

Step 1.
```
      771
16 / 12345
     112
     114
     112
      25
      16
       9
```
$= 9_{16}$ becomes 16^0 digit of answer.

Step 2.
```
      48
16 / 771
     64
     131
     128
       3
```
$= 3_{16}$ becomes 16^1 digit of answer.

Step 3.
```
       3
16 /  48
      48
       0
```
$= 0_{16}$ becomes 16^2 digit of answer.

Step 4.
```
       0
16 /   3
       0
       3
```
$= 3_{16}$ becomes 16^3 digit of answer.

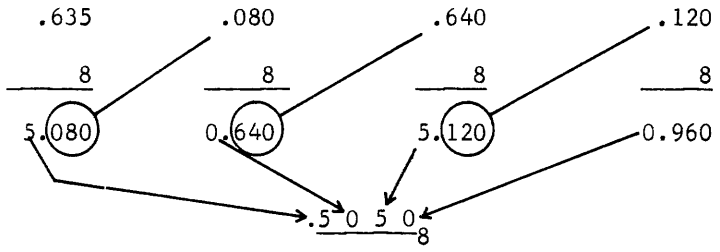
Therefore, $12345_{10} = 3039_{16}$

MULTIPLICATION (FRACTIONAL)

Example 1 Decimal to Octal

$$0.635_{10} = .\_\_\_\_\_{}_8$$

Step 1. Multiply the original number by the base desired, preserving the decimal point. Any digit to the left of the decimal point is part of the answer.



Step 2. Record the answers from the octal point to the right.

NOTE: Carry the answer out to one more place than in the original number (i.e. 4 places).

Example 2 Decimal to Binary (Fractional)

Convert 0.735_{10} to _____2

Step 1. Convert from decimal to octal as in the example above.

```
.735
  8
-----
5.880      .880
             8
           -----
           7.040      .040
                        8
                      -----
                      0.320      .320
                                   8
                                 -----
                                 2.560
```

0.5702_8

Step 2. Convert from octal to binary by inspection.

0.101 111 000 010

Therefore, $0.735_{10} = 0.101\ 111\ 000\ 010_2$

The foregoing conversion could have been made directly by multiplying by the new radix (2). However, this method becomes lengthy and it's difficult to determine where the significance of the new number ends.

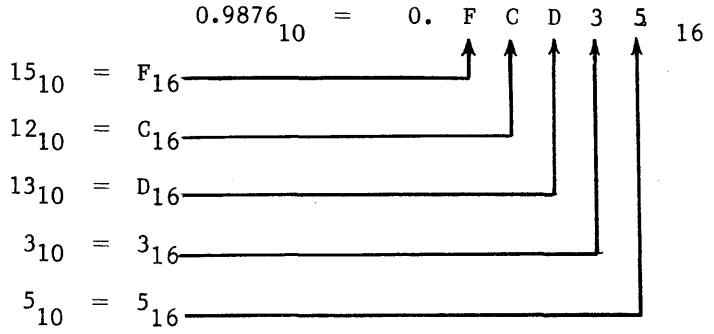
Example 3 Decimal to Hexadecimal (Fractional)

Convert 0.9876_{10} to _____16

Step 1. Multiply the origianl number by the base desired and mark off the decimal point. Digits to the left of the decimal point become part of the answer.

```
  .9876
    16
 59256
  9876
15.8016 ---->  .8016
                 16
               48096
                8016
              12.8256 ---->  .8256
                               16
                             49536
                              8256
                            13.2096 ---->  .2096
                                             16
                                           12576
                                            2096
                                          3.3536 ---->  .3536
                                                          16
                                                        21216
                                                         3536
                                                       5.6576
```

Step 2. Record the results of each operation and convert results to hexadecimal equivalent. The first character obtained becomes the most significant of final answer.



Well, you should now be able to convert any number from any given radix to any other given radix with little or no difficulty. Before going on to non-decimal arithmetic, complete the following practice problems using the conversion method suggested. Again, the answers are at the end of the chapter.

PRACTICE PROBLEMS

37. $7C2E_{16}$ = _____ 10

38. $0.CAB_{16}$ = _____ 10

39. 7654_8 = _____ 10

40. 0.015_8 = _____ 10

41. 1011100_2 = _____ 10

42. 0.01010101_2 = _____ 10

Polynomial expansion method

page 3-22

43. 999_{10} = _____ 2

44. 1001_{10} = _____ 8

45. 40960_{10} = _____ 16

Division method.

page 3-29

46. 0.8080_{10} = . _____ 2

47. 0.00009_{10} = . _____ 8

48. 0.00001_{10} = . _____ 16

Multiplication Method

page 3-33

49. 1110111_2 = _____ 8

50. 100110011_2 = _____ 16

51. 0.0000001_2 = . _____ 8

52. 0.10101010_2 = . _____ 16

53. 7654321_8 = _____ 2

54. 0.6655_8 = . _____ 2

55. $98B_{16}$ = _____ 2

56. $0.000F_{16}$ = . _____ 2

Direct inspection

page 3-27

57. 70707_8 = _____ 16

58. 0.003003_8 = . _____ 16

59. $FEE9_{16}$ = _____ 8

60. $0.1A2B3C_{16}$ = . _____ 8

Convert to binary, then to desired base by re-grouping.

page 3-27

Well, that should satisfy another of our objectives. You now know how to convert a number from one base to another. Once you have the numbers, or operands, expressed in the desired base, you should know how to perform arithmetic operations on those numbers.

ARITHMETIC OPERATIONS

BINARY ARITHMETIC

A digital computer operates internally with binary operands. Arithmetic operations may be performed with quantities in any number system. Binary numbers may be manipulated according to the following rules:

Addition

0	1	0	1
+1	+0	+0	+1
1	1	0	10

Carry the 1 to next higher bit position

Subtraction

0	1	1	10
-0	-0	-1	-1
0	1	0	1

borrow a 1 from next higher bit position (even though not present)

Addition

The following examples illustrate the principles of binary addition.

Example 1

Binary	Decimal	
1 1 1 ← carries		
0 1 0 1	5	Addend
0 0 1 1	+3	Augend
1 0 0 0	8	Sum

1st column: 1 plus 1 = 0, with a carry of 1
2nd column: 1 plus 0 = 1, plus the carry of 1 = 0 with carry of 1
3rd column: 1 plus 0 = 1, plus the carry of 1 = 0 with carry of 1
4th column: 0 plus 0 = 0, plus the carry of 1 = 1

Example 2

Binary	Decimal	
1 1←carries		
0 0 1 1	3	Addend
0 0 1 1	+3	Augend
0 1 1 0	6	Sum

1st column: 1 plus 1 = 0 with carry of 1
2nd column: 1 plus 1 = 0 with carry of 1 plus carry from 1st column = 1
3rd column: 0 plus 0 = 0 plus carry of 1 = 1
4th column: 0 plus 0 = 0

Subtraction

Binary subtraction is performed as illustrated in the following examples.

Example 1

Binary	Decimal	
0 1 1 1	7	Minuend
- 0 1 0 0	- +4	Subtrahend
0 0 1 1	+3	Difference

1st column: 1 minus 0 = 1
2nd column: 1 minus 0 = 1
3rd column: 1 minus 1 = 0
4th column: 0 minus 0 = 0

Example 2

Binary	Decimal	
0 1 0 0	+4	Minuend
- 0 0 1 1	- +3	Subtrahend
0 0 0 1	+1	Difference

1st column: 0 minus 1, must borrow. Borrowing the 1 from the 3rd place makes that bit a 0 and puts two 1's in the 2nd column; borrowing one of those leaves a 1 in the 2nd column and puts two 1's in the 1st column. The subtraction then becomes 1 from two 1's which leaves 1.
2nd column: 1 minus 1 = 0
3rd column: 0 minus 0 = 0
4th column: 0 minus 0 = 0

Multiplication

In any number system, multiplication is done using the same three basic steps:

Step 1. Form the partial product of the multiplicand and the least significant digit of the multiplier.
Step 2. Form the partial product of the multiplicand and next significant digit of multiplier, with this product shifted left one place. Repeat Step 2 for each digit of multiplier.
Step 3. Add the partial products to give the final product.

A decimal example is:

```
          259        Multiplicand
          139        multiplier
partial  2331        product of 1st multiplier digit
products  777        product of 2nd digit, shifted left
           259       product of 3rd digit, shifted left

         36001       sum is final product
```

Binary multiplication is identical, except no multiplication table need be learned. If the multiplier digit is a 1, add the multiplicand.

```
          1010   =  10(10)
          0101   =   5(10)

          1010              1st bit = 1, bring down multiplicand
partial   0000              2nd bit = 0, shift and add zeros
products  1010              3rd bit = 1, shift and add multiplicand
          0000              4th bit = 0, shift and add zeros

       0110010   =  50(10)  add partial products
```

Division

Binary division is carried out in the same manner as decimal division.

```
          101.
      ------------
10101 | 01110101.
        10101
        -------
        0100001
         10101
         -----
          1100
```

(1) 11101 is greater than 10101; enter a 1 in quotient and subtract. Bring down next bit.
(2) 10000 is less than 10101; enter a zero in quotient, bring down next bit of dividend.
(3) 100001 is greater than 10101; enter a 1 in quotient and subtract.
(4) 1100 is less than 10101; no more bits in dividend, 1100 is remainder unless the division is continued to the right of the binary point.

(5) Check, using decimal equivalents

$$21 \overline{)\,117}\;\;5$$
$$\underline{105}$$
$$12 \text{ remainder}$$

Check your proficiency with binary arithmetic by working out the following practice problems. The answers are at the end of the chapter.

PRACTICE PROBLEMS (binary)

	Addition			Subtraction	
61.	0101 0101	62.	0101 0001	63. 0101 0010	64. 1000 0101

	Multiplication			Division	
65.	0101 1001	66.	1010 1011	67. 1010/110010.	68. 110/1000000.

OCTAL ARITHMETIC *

It has previously been stated that all arithmetic operations within a computer are performed in binary. It has also been stated that octal and hexadecimal radices are used only to ease programming tasks. Fewer steps are involved and the operands become less cumbersome with a higher base system.

Perhaps you remember way back when you were required to learn the decimal multiplication tables. Multiplication and addition tables (matrices) have been inserted in this section for both radix 8 and radix 16 operations. However, you should know how the matrices were generated before you use them.

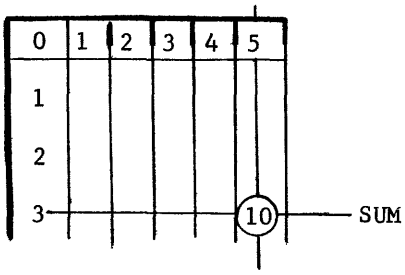
Addition

Octal addition may be performed by counting in units and remembering that there are no eights or nines. Example 7 + 5 = 7, 10, 11, 12, 13, 14. The sum of $7_8 + 5_8 = 14_8$. That method worked fine for those operands. Try another one. $247_8 + 263_8 =$ _____$_8$. The counting method still works if carries are considered.

Until you learn the octal addition table, you may use the matrix on page 3-41. For addition, locate one number on the left side, the other at the top, and read the sum from the coincident square.

* All values expressed in this section are radix 8 unless otherwise specified.

For Example: $3_8 + 5_8 =$ _____$_8$



Now let's again try to find the sum of 247_8 and 263_8.

```
 1<---- carry
 247
 263
----
  2
```

According to the matrix, 7 + 3 = 12. Record the 2 in the 8^0 column and the 1 at the top of the 8^1 column. Now find the sum of 1 + 4 + 6 (the 1 was the carry from the 8^0 column) by using the matrix. $1 + 4 + 6 = 13_8$. Record the 3 and carry the 1 to the top of the 8^2 column.

```
 1 1<---- carries
 2 4 7
 2 6 3
-------
   3 2
```

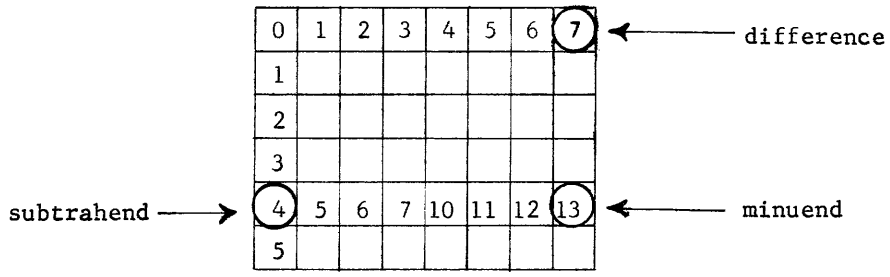
Now add 2 + 2 + 1. The matrix indicates that the sum would be 5 with no carry. Therefore, the sum of 247_8 and 263_8 would be 532_8 (not five hundred and thirty-two, but five-three-two octal).

Subtraction

Octal subtraction is performed by removing one unit at a time until you have removed the proper number of units. For example: $20_8 - 2_8 =$ _____? 20 minus 1 equals 17_8; minus 1 more equals 16_8. This method would be quite unsatisfactory using larger operands. Again, either you must learn how to subtract octally or use the matrix. Use the matrix for present needs, and you can learn how to subtract without the matrix as you become more familiar with the number system. Subtraction, using the matrix, is as follows: Locate the subtrahend digit on the left side of the matrix and follow that row across until you find the minuend digit within the matrix. The column number at the top of the matrix is the difference.

For example:

```
  13    minuend
- 4     subtrahend
  ?     difference
```



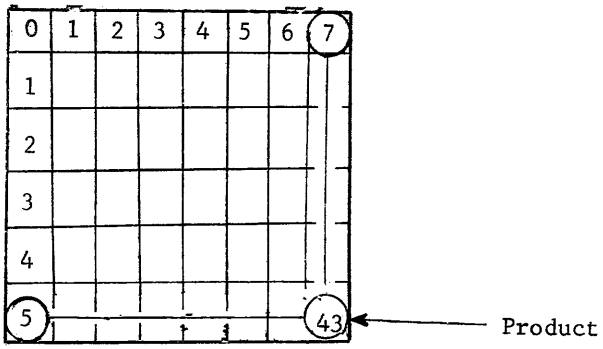
How would you use the table to subtract 6 from 53? Subtract 6 from 13 by borrowing one from the 5. Bring down the remaining 4.

Multiplication

Octal multiplication is the same as a series of additions. For instance, 6 x 7 = ? could be resolved by adding 6 + 6 + 6 + 6 + 6 + 6 + 6, using the addition matrix. Again, this process could become quite lengthy. A multiplication matrix (page 3-44) has been constructed for your use. To multiply, find one operand on the left side, the other at the top and read the product from the coincident square.

For example:

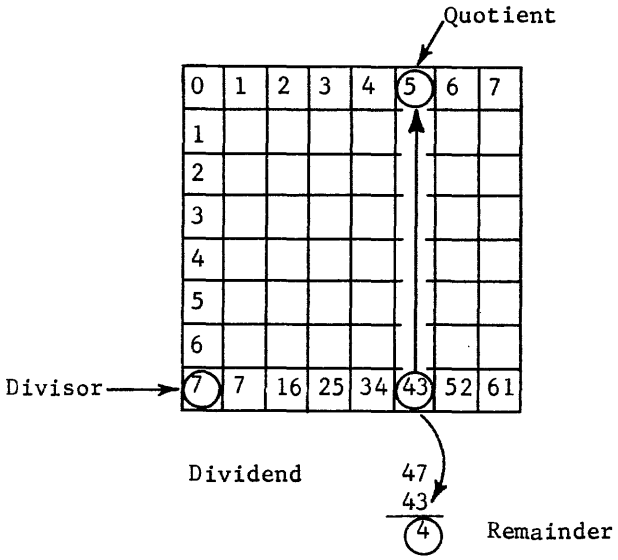
7 x 5 = ?



Division

Octal division could be accomplished by subtracting the divisor from the dividend and recording the number of times it was possible to subtract. Residue left over from the last operation would be the remainder. Again, the operation is quite simple if the matrix is used. To divide using the matrix, locate the divisor on the left side of the matrix and follow that row across to the operand closest to (without exceeding) the dividend. The quotient will be at the top of that column. The difference between the operand in the coincident square and the original dividend, is the remainder.

For Example: $7\overline{)47}$



Larger numbers could be divided using the same procedure in multiple steps similar to decimal division. For example:

$$
\begin{array}{r}
114 \\
7\ \overline{)\ 1024} \\
\underline{7} \\
12 \\
\underline{7} \\
34 \\
\underline{34} \\
0
\end{array}
$$

If the divisor contains multiple digits as in 3743 ÷ 247, the arithmetic becomes more difficult. Inspect the operands and record the first apparent digit of the quotient. Multiply that apparent digit by the divisor and subtract the product from the dividend.

```
      1
247 / 3743
      247
      125
```

Bring down the next digit of the dividend, inspect again, and record the second digit of the quotient.

```
      14
247 / 3743
      247
     1253
     1234
       17
```

When division is no longer possible, the residue of the last operation becomes the remainder. Therefore, 3743 ÷ 247 = 14, remainder 17.

Work the following practice problems to verify your ability to perform octal arithmetic. The answers are at the end of the chapter.

Addition

69. $\begin{array}{r} 5 \\ +\ 5 \\ \hline \end{array}$

70. $\begin{array}{r} 5 \\ +\ 1 \\ \hline \end{array}$

Subtraction

71. $\begin{array}{r} 5 \\ -\ 2 \\ \hline \end{array}$

72. $\begin{array}{r} 10 \\ -\ 5 \\ \hline \end{array}$

Multiplication

73. $\begin{array}{r} 5 \\ \times\ 11 \\ \hline \end{array}$

74. $\begin{array}{r} 12 \\ \times\ 13 \\ \hline \end{array}$

Division

75. $12\overline{)62}$

76. $6\overline{)100}$

You may have noticed that these operands are equalities to those used in the binary problems. The answers should then also be equalities.

Work a few more of the octal arithmetic problems that require more flexability from you. If they give you any trouble, a little more practice will be in order. If not, continue on to hexadecimal arithmetic.

Addition

77. 7654 + 233

78. 12345 + 71625

79. 351 + 153

Subtraction

80. 635 − 277

81. 500 − 250

82. 1000 − 400

Multiplication

83. 35 × 15

84. 67 × 45

85. 400 × 200

Division

86. 350 / 35

87. 10000 / 77

88. 100 / 40

OCTAL ARITHMETIC MATRICES

ADDITION

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

MULTIPLICATION

0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

HEXADECIMAL ARITHMETIC*

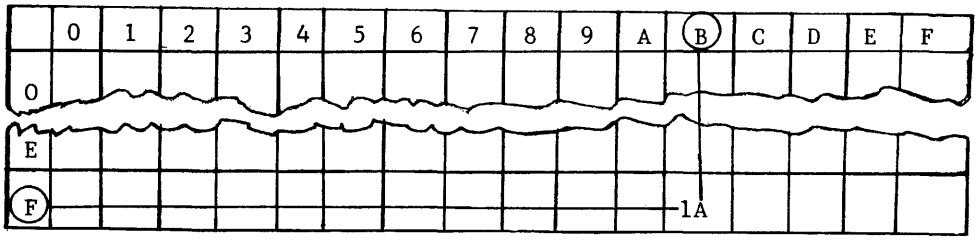
Hexadecimal arithmetic is understandably more difficult than either binary or octal because, with 16 discrete values, alpha characters are used as well as numbers. The letter A represents the value of 10 decimal and is used instead of 10 because it must be expressed with one character. This is necessary with positional number systems if arithmetic operations are to be performed.

Perhaps you recall the counting table back on page 3-17 which illustrated how to count hexadecimally. Addition could be performed by counting, subtraction by counting in reverse, multiplication by repeatedly counting, and division by repeatedly counting in reverse. For example: F + B = ?. F is equal to the decimal quantity 15, B is equal to 11 decimal, and the sum would be equal to 26 decimal. This could be expressed in hexadecimal as 1A. Remember, the combination 10 always represents the radix of its own system. Therefore, 10_{16} represents a value of 16_{10}, the hexadecimal radix. The value $1A_{16}$ represents the value 10_{16} plus A_{16} or, in decimal equivalents, $16_{10} + 10_{10} = 26_{10}$.

It is apparent that arithmetic operations in hexadecimal using large operands would become quite involved. Again, matrices have been prepared for your use and are on pages 3-47 and 3-48. Perform the four basic arithmetic operations with the aid of these matrices.

Addition

Add F_{16} and B_{16} using the hexadecimal addition matrix. Locate F on the left margin of the matrix, B at the top, and read the sum from the coincident square as illustrated.



Addition of larger numbers would require the same procedure for each column with the carry from the preceding column added. Adding a list of operands would be performed with successive adds. For example:

```
  1 1 <-- Carries
    A B
    1 C
  + D 4
  -----
  1 9 B
```

*All values expressed in this section are of radix 16 unless otherwise indicated.

B + C = 17 (7 with a carry of 1). 7 + 4 = B with no carry. The least significant digit of the answer is B. In the next column we now must add A = 1 + D + the carry (1). A + 1 = B; B + D = 18 (8 with carry of 1); and 8 plus the carry = 9. Therefore, the sum of the three listed operands is 19B.

Add the following list of operands

```
  AAA
  BBB
  CCC
+ DDD
```

Double check your answer by converting the operands to decimal, adding them, and reconverting your answer to hexadecimal. Both answers should agree if your procedures are correct. Incidentally, the answer should have been $310E_{16}$ or 12558_{10}.

Subtraction

Subtraction can be performed by using the hexadecimal matrix exactly the same as octal subtraction was performed using the octal addition matrix. Locate the subtrahend digit on the left side of the matrix and follow that row across to the minuend. The number at the top of the matrix in that column is the difference. In some instances, a borrow from the next position to the left may be required before subtraction can be performed. For example:

```
  A6
- 17
```

Seven cannot be subtracted from six without borrowing one (1×16^1) from the A. After the borrow, subtraction can be performed, but the value of A has been reduced by one to 9. The subtraction problem effectively becomes this:

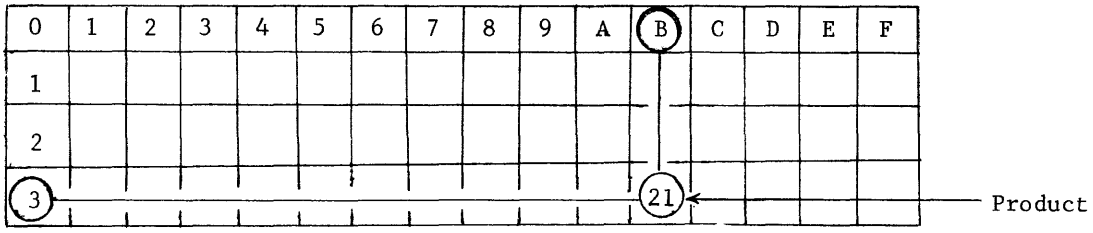
```
  9 ↷
  A  16
  1   7
-------
  8   F
```

The difference between A6 and 17 should be 8F. Add the difference and the subtrahend. The sum should be the same as the original minuend.

```
  1 ← Carry
  17
+ 8F
----
  A6
```

Multiplication

As you already realize, multiplication is a series of additions. A x 3 could be expressed as A + A + A and F x A could be expressed as F + F + F + F + F + F + F + F + F + F. Again, you can see that multiplication by this method could be cumbersome, especially with large operands. A hexadecimal multiplication matrix has been included on page 3-48 of this section for your use. Multiplication with two hexadecimal characters will produce a two-character product similar to decimal multiplication (1 x 2 = 02, 9 x 9 = 81, 7 x 3 = 21, etc.). You will notice that the product is within the matrix and that the multiplicand and multiplier are along the left side and top. Multiplication with the aid of the table is performed by locating the two operands and reading the product from the coincident square. For example: 3 x B = ?



Multiplication of larger numbers can be performed by listing partial products and then adding hexadecimally. For example:

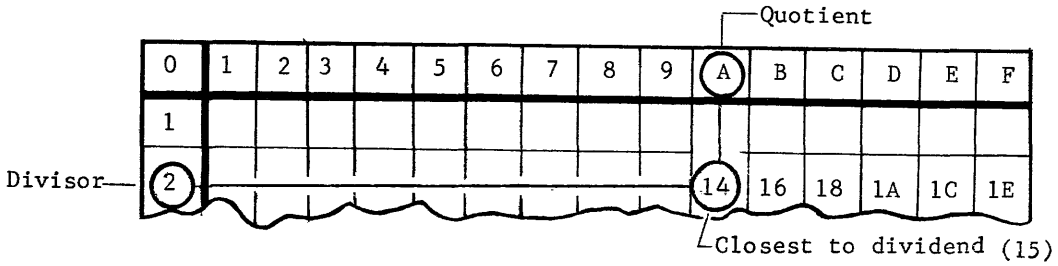
```
  ABC
  123
  ---
   24
   21      3 x ABC  =    2034
  1E
   18
   16     20 x ABC  =   15780
  14
   0C
   0B    100 x ABC  =   ABC00
   0A
  ---
  C33B4                 -----
                        C33B4
```

Division

Hexadecimal division can also be performed by using the multiplication matrix on page 3-41 exactly the same as the octal matrix was used to perform octal division. Locate the divisor on the left side of the matrix, follow that row across to the operand closest to (without exceeding) the dividend. The quotient will be at the top of that column. The difference between the original dividend and the value in the coincident square is the remainder.

For example

$$2 \,/\, \overline{15}$$



$$\begin{array}{r} \text{A} \\ 2 \,/\, \overline{15} \\ \underline{14} \\ 1 \end{array}$$

Therefore: $15 \div 2 = A + 1$ remainder

Doubtful answers can be easily double-checked by converting each hexadecimal character to binary, performing the operation in binary and re-converting the binary answer and remainder back to hexadecimal. Prove the foregoing division problem in binary.

$$2 \,/\, \overline{15} \text{ in radix 16} = 0/0 \,/\, \begin{array}{r} 1010. \\ \overline{00010101.} \\ \underline{010} \\ 0010 \\ \underline{010} \\ 0001 \end{array} \text{ in radix 2}$$

$$1010_2 = A_{16} \quad \text{and} \quad 01_2 = 1_{16}$$

Just as a review, for old time's sake, work that last hexadecimal multiplication problem in binary.

$$\begin{array}{r} \text{ABC} \\ \underline{\times \ 123} \end{array} \text{ radix 16} \quad + \quad \begin{array}{r} 1010\ 1011\ 1100 \\ \underline{0001\ 0010\ 0011} \\ 1010\ 1011\ 1100 \\ 1\ 0101\ 0111\ 100 \\ 1\ 0101\ 0111\ 100 \\ 1010\ 1011\ 1100 \\ \hline 1100\ 0011\ 0011\ 1011\ 0100_2 \\ \text{C} \quad 3 \quad 3 \quad \text{B} \quad 4_{16} \end{array} \text{ radix 2}$$

Try a few hexadecimal arithmetic problems to verify your ability. The answers are at the end of this chapter.

Addition

89. $\begin{array}{r} \text{CAB} \\ \underline{\text{BED}} \end{array}$

90. $\begin{array}{r} \text{BAD} \\ \underline{\text{DEED}} \end{array}$

91 $\begin{array}{r} \text{DOG} \\ \underline{\text{FOOD}} \end{array}$

Subtraction

92. AB999
    9FFFF

93. FEDCBA
    EDCBA9

94. ABCDE
    12345

Multiplication

95. 9876
    1234

96. 777
    AAA

97. 5252
    2525

Division

98. AB / AE

99. 999 / FFFF

100. F / 100000

Another way to double-check your arithmetic would be by converting each hexadecimal value to its decimal equivalent (by using the conversion table on page 3-49), performing the arithmetic operation in decimal, and then reconverting the answer back to its hexadecimal equivalent.

Well, another of the objectives of this chapter should now have been met. You should be able to perform arithmetic operations upon radix 2, 8, 10, and 16 operands. For operands of other bases, convert to the radix 10 equivalence, perform the arithmetic, and reconvert the answer back to the original base.

HEXADECIMAL ADDITION MATRIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

HEX	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

HEXADECIMAL MULTIPLICATION MATRIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	00	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	00	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	00	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	00	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	00	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	00	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	00	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	00	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	00	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	00	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

HEX	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

CONVERSION TABLE

HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.
1	1	10	16	100	256	1000	4096	10000	65536
2	2	20	32	200	512	2000	8192	20000	131072
3	3	30	48	300	768	3000	12288	30000	196608
4	4	40	64	400	1024	4000	16384	40000	262144
5	5	50	80	500	1280	5000	20480	50000	327680
6	6	60	96	600	1536	6000	24576	60000	393216
7	7	70	112	700	1792	7000	28672	70000	458752
8	8	80	128	800	2048	8000	32768	80000	524288
9	9	90	144	900	2304	9000	36864	90000	589824
A	10	A0	160	A00	2560	A000	40960	A0000	655360
B	11	B0	176	B00	2816	B000	45056	B0000	720896
C	12	C0	192	C00	3072	C000	49152	C0000	786432
D	13	D0	208	D00	3328	D000	53248	D0000	851968
E	14	E0	224	E00	3584	E000	57344	E0000	917504
F	15	F0	240	F00	3840	F000	61440	F0000	983040

HEX.	DEC.	HEX.	DEC.	HEX.	DEC.
100000	1048576	1000000	16777216	10000000	268435456
200000	2097152	2000000	33554432	20000000	536870912
300000	3145728	3000000	50331648	30000000	805306368
400000	4194304	4000000	67108864	40000000	1073741824
500000	5242880	5000000	83886080	50000000	1342177280
600000	6291456	6000000	100663296	60000000	1610612736
700000	7340032	7000000	117440512	70000000	1879048192
800000	8388608	8000000	134217728	80000000	2147483648
900000	9437184	9000000	150994944	90000000	2415919104
A00000	10485760	A000000	167772160	A0000000	2684354560
B00000	11534336	B000000	184549376	B0000000	2952790016
C00000	12582912	C000000	201326592	C0000000	3221225472
D00000	13631488	D000000	218103808	D0000000	3489660928
E00000	14680064	E000000	234881024	E0000000	3758096384
F00000	15728640	F000000	251658240	F0000000	4026531840

COMPLEMENT ARITHMETIC

The era of the computer has called for the introduction of a new line of reasoning to solve basic arithmetic problems. While in everyday life such signs as + (plus) and - (minus) are used to denote the difference between positive and negative numbers, these signs cannot be recognized or stored by a computer. Also, in everyday life, the combination of a group of numbers with mixed signs can be readily handled by adding the positive values, then adding the negative values, then determining the difference between the two, and affixing the sign of the larger number to the answer. The computer does not perform operations with signed numbers in this manner; instead, the computer uses complement arithmetic. Complement arithmetic is not a specially designed form of mathematics to be used only by computing devices. It could be used in everyday life in most of the situations that are normally encountered.

The dictionary defines complement as (a) something that fills up, completes, or makes perfect (b) the quantity or number required to make a thing complete (c) that which is required to supply a deficiency; one of two mutually complementing parts.

Perhaps you remember our discussion of the modulus of a device. We defined modulus as "the number of unique quantities that a particular device could express" and an odometer was used as an illustration.

A friend has a car that registers 67,232 miles on the odometer. First, what is the modulus of the device (disregard tenths) and next, what is the complement of that quantity?

You probably said that the modulus of the device was 100,000 and, if so, you were correct again. To find the complement, subtract from the modulus of the system (100,000 in this case). The answer is the "tens complement" of the original operand. If the device is radix 8 instead of radix 10, the foregoing procedure of subtracting, in octal, from the modulus would render the "eights" complement. A binary device and like procedure, with binary arithmetic, would result in the "twos" complement.

Complement, then, really means; of the total combinations in a given system, how many are left before repeating?

Consider these examples:

Decimal	Octal	Binary
1 0 1 0 1 0	1 0 1 0 1 0	1 0 1 0 1 0
Subtract, in decimal, from $1,000,000_{10}$ (modulus)	Subtract, in octal from $1,000,000_{8}$ (modulus)	Subtract, in binary from $1,000,000_{2}$ (modulus)
1,000,000 101 010	1,000,000 101 010	1,000,000 101 010
$898,990_{10}$	$676,770_{8}$	$010,110_{2}$
The "tens" complement of 101,010 decimal is 898,990.	The "eights" Complement of 101,010 octal is 676,770.	The "twos" complement of 101010 binary is 010,110.

Consider another situation. Instead of subtracting from the modulus, subtract from the "modulus minus one". For example:

Decimal	Octal	Binary
9 9 9 9 9 9	7 7 7 7 7 7	1 1 1 1 1 1
(modulus - 1)	(modulus - 1)	(modulus - 1)

Again, using the same values as before, subtract

999,999 101 010	777,777 101 010	111,111 101 010
$898,989_{10}$	$676,767_{8}$	$010,101_{2}$

1) In the decimal example, we subtracted from all 9's and the result was the "nines" complement.
2) In the octal example, we subtracted from all 7's and the result was the "sevens" complement.
3) In the binary example, we subtracted from all 1's and the result was the "ones" complement.

Compare the results of the two decimal, the two octal, and the two binary problems.

10's complement	898,990	Difference of 1
9's complement	898,989	
8's complement	676,770	Difference of 1
7's complement	676,767	
2's complement	010,110	Difference of 1
1's complement	010,101	

Now that you know what a complement is and how it is derived, let's see how it fits into the computer picture. A computer operates strictly in binary. The complements associated with binary arithmetic are "ones" and "twos" complement. Find the "ones" complement of the following numbers: 110011_2, 10001000_2, 111000_2.

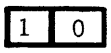
Examine the numbers and their "ones" complement carefully. Notice that this complement could have been formed simply by changing each 1 to a 0 and each 0 to a 1, an operation very easily performed by the computer.

The advantage of complements lies in the fact that addition and subtraction can be accomplished with the same device, obviating the necessity of separate additive and subtractive devices in the computer. This means that by using complements of numbers, both additive and subtractive operations can be performed with one device (which might be additive or subtractive in nature). Choice of the type of device lies completely with the design engineer.

At this point, it is important to realize that a computer or any arithmetic device is physically restricted to a definite modulus, similar to the odometer previously discussed. Remember that each bit position of a binary number will be represented by an electronic switch in the computer.

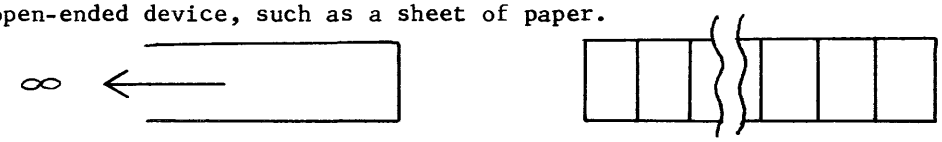
The quantity of these electronic devices dictates the number of bit positions which, in turn, determines the modulus of that particular machine.

Consider a machine with only two of these electronic devices.



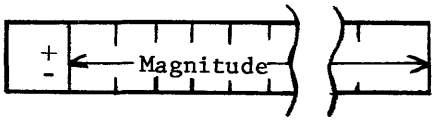
00
01
10
11
} 4 discrete values

What is the modulus of the machine? It could be expressed in binary as 100, in decimal as 4, or simply 2^2. A twelve-bit machine would have a modulus of 2^{12}, a sixteen-bit machine would have a modulus of 2^{16}, and a twenty-four bit machine would have a modulus of 2^{24}. Even the so-called "super computers" have a definite modulus that restricts the magnitude of operands that can be used at one time. The important thing to realize is that a computer is not an open-ended device, such as a sheet of paper.

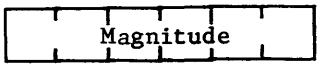


Open-ended device Device with a modulus

Perhaps you have noticed that throughout the chapter we have been dealing only with positive numbers. How would a negative number be expressed in the computer? There are only <u>two</u> possible signs by which we express a number and because a bit can express <u>two</u> values, we can reserve one bit position of our device to determine the sign of the operand. The sign of an operand usually precedes it, so we can use the left-most (high-order or most significant) bit to express the sign. This type of device will hereafter be referred to as a signed device.

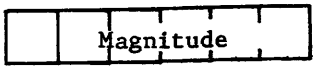


Does this now affect the modulus of the device? Let's see if it does. A six-bit machine could express values from 000 000_2 to 111 111_2 (00_8 to 77_8) if <u>no</u> bit position is used for the sign. A total of 100_8 positive values could be expressed.



6 bits of magnitude

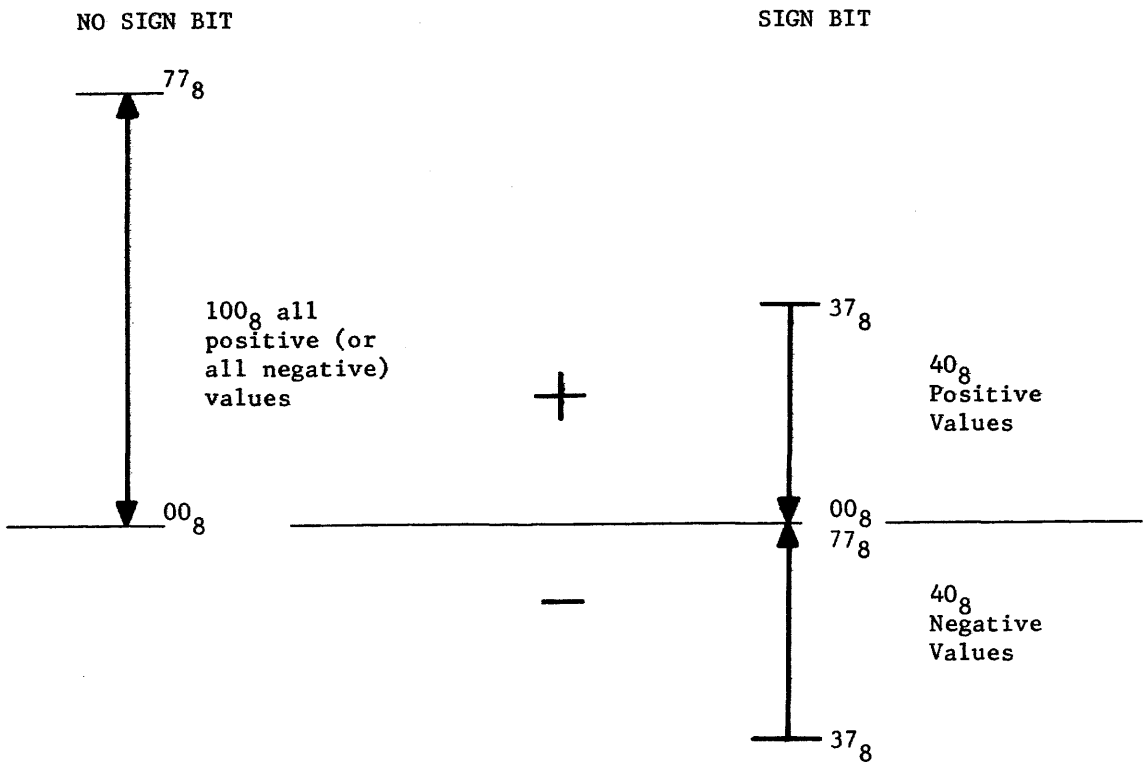
Now, if one bit position is reserved for the sign, the six-bit machine can only express values from 00 000_2 to 11 111_2 (00_8 to 37_8) or 40_8 values.



5 bits of magnitude

However, it is now possible to express 40_8 positive values <u>and</u> 40_8 negative values, a total of _____$_8$ values. (If your answer is 80, pile all of your books in the center of the floor and have a wiener roast).

Apparently, using one bit to express the sign <u>does not</u> affect the modulus of the device but <u>does</u> affect its range of numbers.



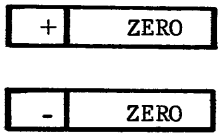
We can now express both positive and negative quantities with the same device without (appreciably) affecting the modulus. Maybe the parenthetical word "appreciably" deserves qualification.

Assume that a sign bit of "0" indicates that the quantity is positive and a sign bit of "1" indicates a negative quantity. It would then be possible to express a positive 6 and a negative 6, a positive 4 and a negative 4, a positive 2 and a negative 2, etc.

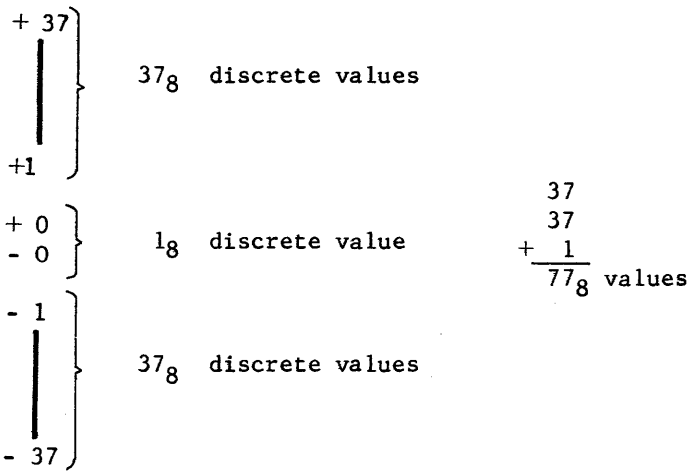
Wouldn't it also be possible to express a positive 0 and a negative 0? In arithmetic, zero is defined as the number between a set of all positive numbers and a set of all negative numbers. Positive 0 and negative 0 should define the same arithmetic quantity. By counting in reverse, we see the comparison between normal counting and counting with a signed device.

Normal counting	Signed device	
5	5	
4	4	
3	3	Positive
2	2	numbers
1	1	
0	0	
-1	0	
-2	1	
-3	2	Negative
-4	3	numbers
-5	4	
	5	

Two of the possible discrete combinations of the signed device are actually being used to express the same value.



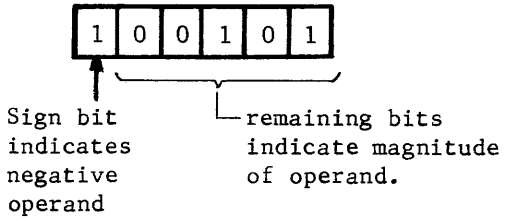
Therefore, the actual modulus of the device should be decreased by 1. Considering the 6-bit device used previously, the following discrete values could be expressed:



A 6-bit unsigned device could express 100_8 discrete values whereas a 6-bit signed device should express 1 less or 77_8 discrete values. Therefore, the modulus of a 6-bit unsigned device would be expressed as 2^6 but the modulus of a 6-bit signed device would be expressed as $2^6 - 1$ ($2^6 = 100_8$, $2^6 - 1 = 77_8$). A 24-bit signed device would have a modulus of $2^{24} - 1$ and a 24-bit unsigned device would have a modulus of 2^{24}.

There are two ways to express a given negative number in a signed device. The first is to express the desired magnitude of the operand and indicate that the operand is negative by the sign bit.

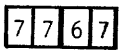
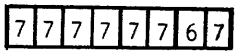
For example, $-00\ 101_2$ could be expressed in a signed device as:



This approach is used in some computer systems but requires special consideration before numbers of unlike signs can be added or subtracted.

The second approach is to express all negative numbers as the complement of their positive equivalents. The complement of a signed number depends upon the modulus of the device.

For example, express -10_8 in complemented form.

Modulus $8^2 - 1$	-10_8 would be expressed as	
Modulus $8^3 - 1$	-10_8 would be expressed as	
Modulus $8^4 - 1$	-10_8 would be expressed as	
Modulus $8^8 - 1$	-10_8 would be expressed as	

If each number was expressed in binary instead of octal, you would notice that the sign bit of each device is a one. That indicates that the number is actually the complemented form of its positive equivalence.

The following practice problems are to check your ability to recognize the sign and magnitude of an operand by inspection. If the number is negative (in complemented form), recomplement to find the true magnitude and attach a minus sign. Express your answer in octal.

101. $011\ 111_2$ = _____$_8$

102. $100\ 000_2$ = _____$_8$

103. $111\ 111_2$ = _____$_8$

104. $100\ 001_2$ = _____$_8$

105. $010\ 101_2$ = _____$_8$

106. $000\ 000_2$ = _____$_8$

107. $101\ 010_2$ = _____$_8$

108. $111\ 000_2$ = _____$_8$

109. $000\ 111_2$ = _____$_8$

110. $011\ 100_2$ = _____$_8$

Indicate how the following numbers would appear in a signed device of modulus $8^4 - 1$.

111. $+30_8$ = _____$_8$

112. -30_8 = _____$_8$

113. $+00_8$ = _____$_8$

114. -00_8 = _____$_8$

115. -37_8 = _____$_8$

116. $+37_8$ = _____$_8$

117. $+100_8$ = _____$_8$

118. -100_8 = _____$_8$

119. $+377_8$ = _____$_8$

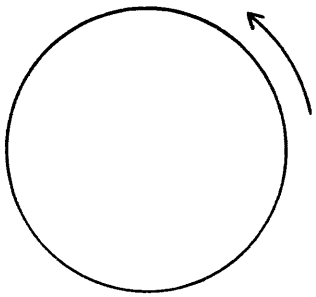
120. -377_8 = _____$_8$

A modulus of $8^4 - 1$ is equal to a binary device with a modulus of $2^{12} - 1$. The largest positive number that could be expressed in the device would be $011\ 111\ 111\ 111_2$ (3777_8) because of the sign bit.

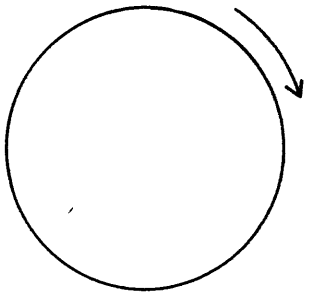
It was previously stated that both addition and subtraction can be performed with either an additive device or a subtractive device if complement arithmetic is used. Let's first use an additive device to see if both operations can be performed and then a subtractive device for the same two operations. The results should be the same with either device.

ADDITIVE DEVICE

An additive device is similar to a counter capable of counting only in a forward direction. A subtractive device is also a counter but capable of counting only in reverse.

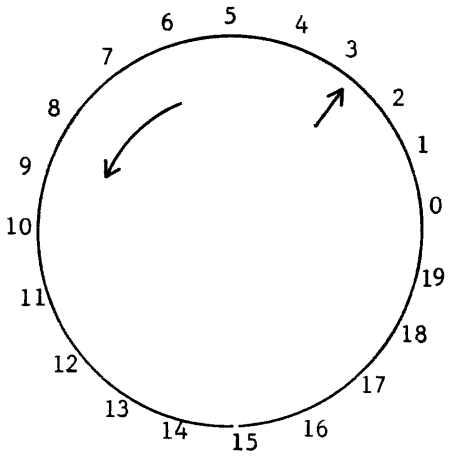


Additive Device
(Forward Counter)



Subtractive Device
(Reverse Counter)

Compare an additive counter to a roulette wheel that could only turn in one direction. Suppose that the pointer was on 3 and you wished to move it to 2.



The number 2 is only one unit from the present position but in a reverse direction. Remember that the additive device cannot move in that direction. Instead, it must move 19 units in the forward direction instead of 1 unit in the reverse direction (-1).

What is the modulus of the device?	20
How much did we desire to move the wheel?	-1
What is the complement of -1?	19
Move the wheel forward 19 units	
Now, where did the wheel stop?	2

Mathematically, the problem would have looked like this:

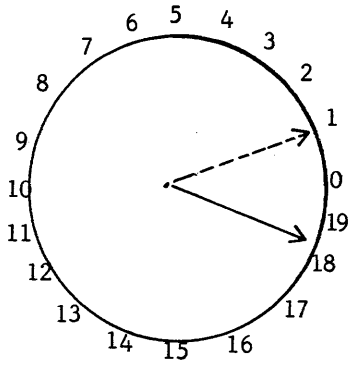
$$\begin{array}{r} +3 \\ +\ -1 \\ \hline +2 \end{array}$$

Although the numbers indicate that we are adding one, the sign of the number to be added indicates that we should have moved in the reverse direction. This was impossible with our one-way device. We had to move a complementary number of units in the forward direction (19 units).

Now let's use the same two operands but add +3 to -1. Mathematically, the problem would look like this:

$$\begin{array}{r} -1 \\ +3 \\ \hline \end{array}$$

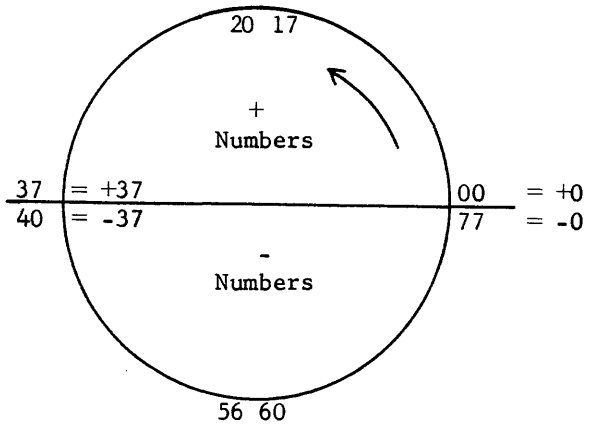
The sign of the number to be added indicates that we should move forward and the number indicates that the move should be 3 units. However, the starting point is now -1 which should be one place in the reverse direction from zero. -1 is therefore the position indicated by the number 19 on the roulette wheel.



$$\begin{array}{r} -1 \\ +3 \\ \hline \end{array}$$

When the device is moved forward 3 positions, as indicated by the number to be added, we find that the arrow now points to the 2 -- again the correct answer. In the first example, we effectively subtracted 1 by adding its complement (19); in the second example, we added 3 to -1 (19).

Now that we understand the principle of the forward counter, let's make a similar one that counts octally (modulus = 8^2 - 1).



The roulette wheel had twenty positions which represented twenty discrete values, and had only one position that indicated a value of zero. Our octal counter of signed numbers has two positions that indicate a value of zero (00_8 and 77_8). To eliminate the negative zero we must decrease the modulus of the device from 100_8 discrete values to 77_8 discrete values. This means that all complements should be formed by subtracting from the new modulus of 77_8 (the sevens complement). The same situation in binary resulted in the use of ones complement notation. Keep in mind that a computer is performing its addition in binary while we represent the same operation in octal.

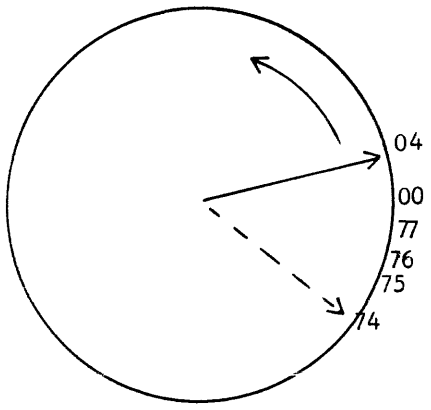
ADD +4
-7
-3_8

expressed in sevens complement notation

	0	4
∓	7	0
	7	4

$_8$

The problem indicates that, starting at a value of +4, we should add a quantity of 7 units, but in a reverse direction. Again, this is impossible with our onw-way device. Instead of moving 7 unit-positions in reverse, we move forward a complementary number of positions (70_8).



The pointer now indicates an answer of 74_8. Any time the upper octal digit is four or greater, the signed binary equivalent would indicate a negative operand. Recomplement to find the true value and attach the negative sign.

$74_8 = -3_8$ The correct answer.

Let's try another condition.

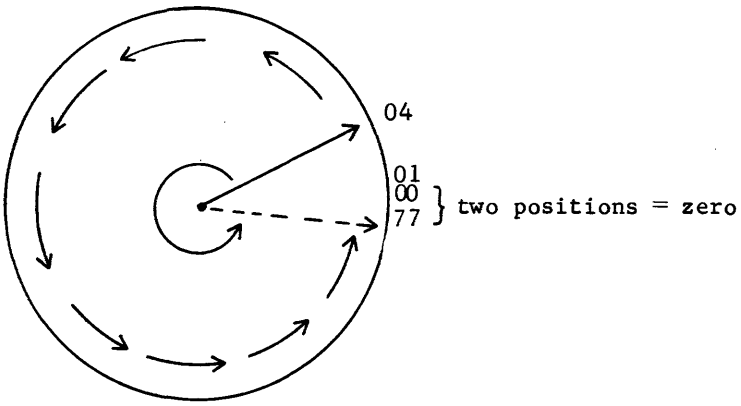
ADD +4
-2
+2 expressed in complement notation =

04		000 100
+75		111 101
101	1	000 001

carry

Our additive device had a modulus of $8^2 - 1$ and the results of the addition produced a carry <u>from</u> the sign bit position. What happens to the carry leaving the upper end of the device? Continue, but concentrate.

The illustration of the additive counter indicates that we moved 75 unit positions <u>and passed through two positions that both represent a value of zero</u>.



The extra position with a discrete value of zero caused us to count zero twice, which caused the answer to be one less than it should have been.

ADD (modulus $8^4 - 1$)

$+50_8$	=	0050_8
-27_8	+	$+ 7750_8$
$+21_8$		$1\ 0020_8$

$+ 3500_8$	=	3500_8
$- 3000_8$		4777_8
$+ 500_8$		$1\ 0477_8$

Both foregoing examples have a carry trying to leave the confines of the device when added in complementary form. Both answers are also incorrect; but, if the carry is added to the answer contained in the device, the answer would then be correct.

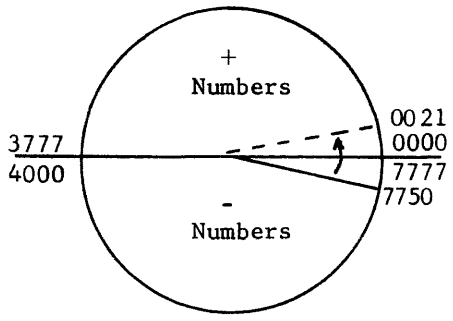
$$\begin{array}{l} 1\ 0020 \\ \hookrightarrow\ 1 \\ \hline 0021_8 \end{array} \qquad \begin{array}{l} 1\ 0477 \\ \hookrightarrow\ 1 \\ \hline 0500_8 \end{array}$$

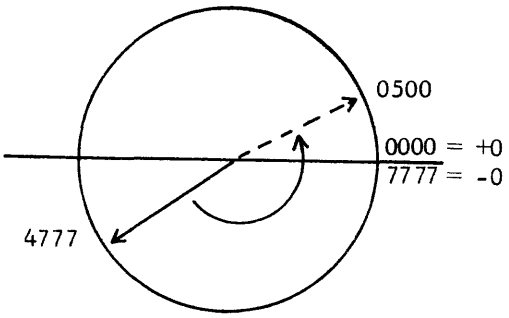
Let's call this procedure END AROUND CARRY (EAC) for an additive device. When EAC occurs, it always indicates that we have counted through zero and are compensating for that extra zero position. It is the end around carry which produces a modulus -1 by reducing the two zero positions effectively to one position.

Let's use the same operands again but rearrange them so that now $+50_8$ is being added to -27_8 and $+3500_8$ is being added to -3000_8.

ADD modulus $8^4 - 1$

$$\begin{array}{rcl} -27_8 & = & 7750 \\ +50 & = & +0050 \\ \hline +21_8 & & 1\ 0020 \\ & & +\ \hookrightarrow\ 1 \\ & & 0021_8 \end{array} \qquad \begin{array}{rcl} -3000_8 & = & 4777 \\ +3500_8 & = & +\ 3500 \\ & & 1\ 0477 \\ & & +\hookrightarrow\ 1 \\ & & \hline & & 0500_8 \end{array}$$





The foregoing examples expressed in binary also show the carry leaving the sign bit position of the answer.

$$\begin{array}{rcl} 7750_8 & = & \boxed{111}\boxed{111}\boxed{101}\boxed{000}_2 \\ 0050_8 & = & +\ \boxed{000}\boxed{000}\boxed{101}\boxed{000}_2 \\ \hline \text{carry} & & 1\ \boxed{000}\boxed{000}\boxed{010}\boxed{000}_2 \\ & & \qquad\qquad\qquad\quad 1 \\ \hline & & \boxed{000}\boxed{000}\boxed{010}\boxed{001}_2 = 21_8 \end{array} \qquad \begin{array}{rcl} 4777_8 & = & \boxed{100}\boxed{111}\boxed{111}\boxed{111} \\ 3500_8 & = & +\ \boxed{011}\boxed{101}\boxed{000}\boxed{000} \\ \hline \text{carry} & & 1\boxed{000}\boxed{100}\boxed{111}\boxed{111} \\ & & \qquad\qquad\qquad\quad 1 \\ \hline & & \boxed{000}\boxed{101}\boxed{000}\boxed{000}_2 = 500_8 \end{array}$$

OVERFLOW

At this time, it is important to realize that, in some instances, a carry may be generated by some of the magnitude bits and propagated upward to the sign bit position. This condition could cause the sign bit to change and cause the computer to produce a wrong answer. The computer checks each arithmetic operation and indicates that a fault (OVERFLOW) has occurred when the answer is incorrect. When overflow occurs, it means that two operands have been added or subtracted and the result has exceeded the modulus of the device.

For example:

ADD (modulus 2^6 -1)

		sign bit	magnitude bits
+37	=	0	11 111
+ 1		+ 0	00 001
40		1	00 000 $_2$ = 40_8 = -37_8

The result indicates the correct answer to be 40_8. However, 40_8 expressed in a 6-bit signed device indicates an actual value of -37_8 which is, of course, incorrect. The modulus of this device is 77_8 and the range of operands is from -37 to +37.

If the same operands were used in a modulus 2^9 -1 device, the sign bit would not be affected and the result would be correct.

ADD (modulus 2^9 -1)

$+37_8 = 037_8$

$+ 1_8 + 001_8$

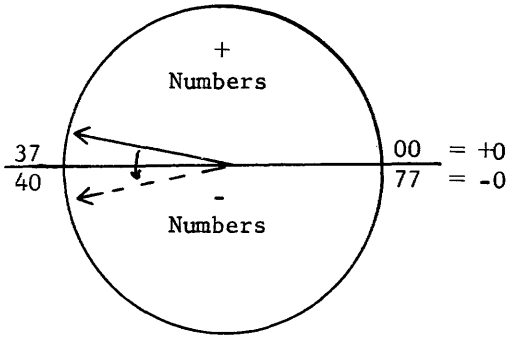
Sign bit

Magnitude bits

0 00 011 111

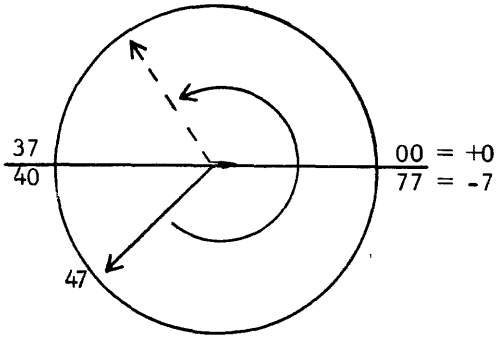
\+ 0 00 000 001

0 00 100 000 $_2$ = 040_8



ADD (modulus 2^6 -1)

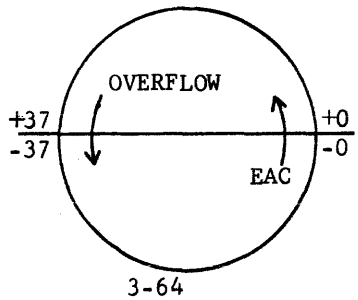
-30	=	100 111	$= 47_8$
-20	=	+ 101 111	+ $= 57_8$
-50		1 [010 110]	1 [26]
		1	1
		[010 111]$_2$ $= 27_8$	[27]



In the first example, we added two positive numbers and the result was negative. In the second, we added two negative numbers and the indicated result was positive. The indicated answer was incorrect because the sum of the numbers in both examples exceeded the modulus of the device and changed the sign bit.

1. If two numbers of <u>like</u> signs are <u>added</u> and the result has the opposite sign, the answer is <u>always</u> incorrect because the result has OVERFLOWED the modulus of the device.
2. If two operands of <u>unlike</u> signs are <u>subtracted</u> and the result has the same sign as the subtrahend, the answer is <u>always</u> incorrect and OVERFLOW has occurred. This will be illustrated in the following section covering subtractive devices.

NOTE: It is important to realize that overflow and <u>end around carry</u> are <u>completely unrelated</u>. End around carry indicates that we counted through zero; overflow indicates that a result has exceeded the modulus of the device. In the last example, where we added -30 and -20, the result was incorrect because of overflow even though end around carry was also accomplished.



If subtraction is required and the device is additive, the subtrahend is always complemented and addition is performed.

For example:

SUBTRACT - BY COMPLEMENTING AND ADDING (modulus $8^2 - 1$ or $2^6 - 1$)

$+15_8$		$= 15$	$= 001\ 101_2$
$-\ +\ 5_8$	complemented	$= 72$	$= 111\ 010_2$
10_8		$1\ \boxed{07}$ $\hookrightarrow 1$	$1\ \boxed{000\ 111}$ $\hookrightarrow 1$
		$\boxed{10}$	$\boxed{001\ 000}_2$

SUBTRACT - BY ADDING (modulus $2^6 - 1$ or $8^2 - 1$)

$+15$	$=$	15		15	$=\ 001\ 101$
$-\ -\ 7$	$=$	70	complemented	$=\ +\ 07$	$+\ 000\ 111$
24_8				24_8	$\boxed{010\ 100}_2$

The following problems are to be solved by addition. If subtraction is indicated, complement the subtrahend and add. The modulus of each device is $8^4 - 1$. Write yes or no beside your answer to indicate whether or not overflow has occurred. If yes, record the computers answer and the correct answer. If you wish, convert each problem to binary and use a modulus of $2^{12} - 1$.

Practice Problems (Solve by addition)

121. 235 ⊕ 173

122. -400 ⊕ -200

123. 3400 ⊖ 500

124. -3700 ⊖ -0300

125. 3777 ⊕ -0001

126. 3777 ⊖ 0001

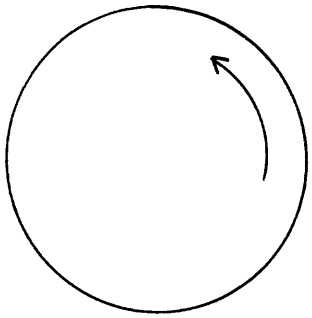
127. 2000 ⊕ 2000

128. 3700 ⊖ -200

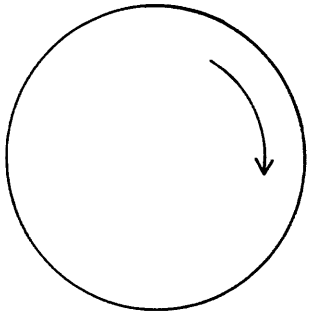
additive and subtractive operations have both been performed by using only an additive device and complement notation. Likewise, a subtractive device should also be capable of both operations.

SUBTRACTIVE DEVICE

The additive device previously discussed could only count in one direction -- forward. The subtractive device can only count in the reverse direction.



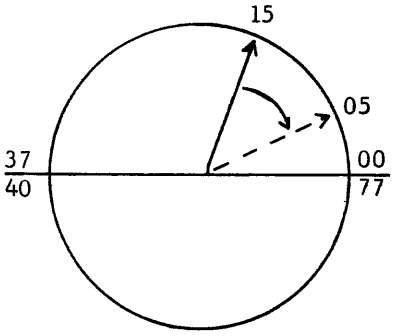
ADDITIVE DEVICE



SUBTRACTIVE DEVICE

Let's examine the operation of a subtractive counter.

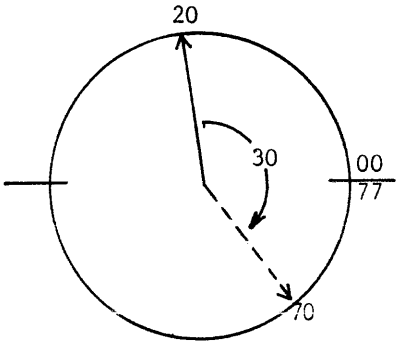
SUBTRACT 15
⊖ 10
05



When we passed through zero with an additive device, we always had an EAC; therefore, it seems probable that we should witness some strange phenomena when passing through zero in the reverse direction.

What would happen if a larger number is subtracted from a smaller one, such as:

20
⊖ 30
-10



The answer should be -10_8 (67 in complement notation), but counting through zero would again cause the answer to be incorrect by a value of one.

Let's examine what happens when we subtract the same numbers octally in a modulus $8^2 - 1$ device.

$$20_8 = \boxed{20}$$
$$30_8 = \ominus \boxed{30}$$
$$\overline{\text{(1) } \boxed{70}_8}$$

We find that we are required to borrow before subtraction can be performed. It's impossible to borrow from outside the device so, go end around and borrow one from within the device. The procedure will hereafter be referred to as "End Around Borrow" or EAB. The end-around operation effectively reduces the modulus by one by making the two zero positions occupy the same position.

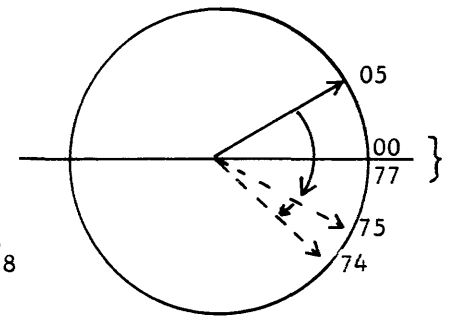
```
          20
          30
        ----
Borrow 1  70
     ↪    1
        ----
          67
```

With the additive device, we added one to the lower digit if the upper digit generated a carry and we called this procedure END AROUND CARRY. Likewise, with a subtractive device, we performed an END AROUND BORROW if the upper digit generated a borrow.

It can be seen from the reverse-counter illustration that subtracting a large absolute (positive) value from a smaller one will always cause the counter to pass through zero. Subtracting mathematically, the same procedure would always require a borrow. Whenever end around borrow is required of a subtractive counter, it is indicative that the counter has passed through zero and 1 more must be subtracted from the answer to compensate for that extra zero position.

For example:

```
   5        05
⊖ 10   =   10
-3₈      1 75
           ---
            1
           ---
           74 = -3₈
```

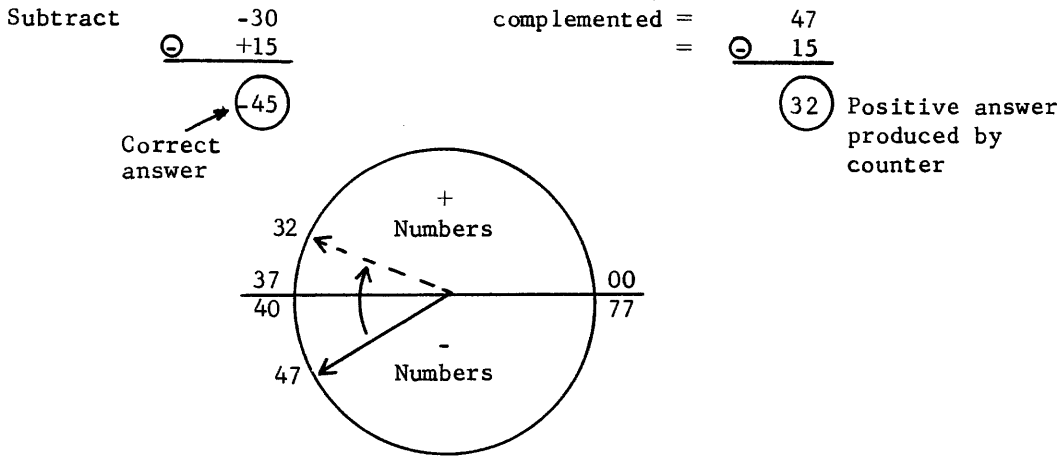


both equal to zero

EAB causes counter to move one more position to compensate for the extra zero

SUBTRACTIVE OVERFLOW

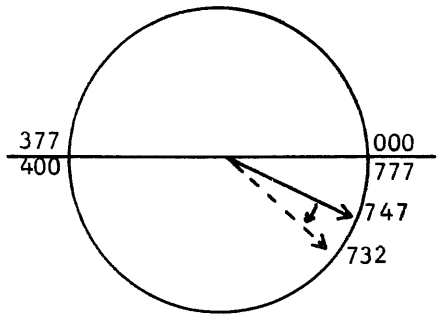
Perhaps you remember the conditions that would produce an incorrect answer with an additive device. We called this condition overflow because the sum of two operands was greater than the modulus of the device. Let's examine the subtractive counter to prove the rule about subtractive overflow (#2, page 3-64).



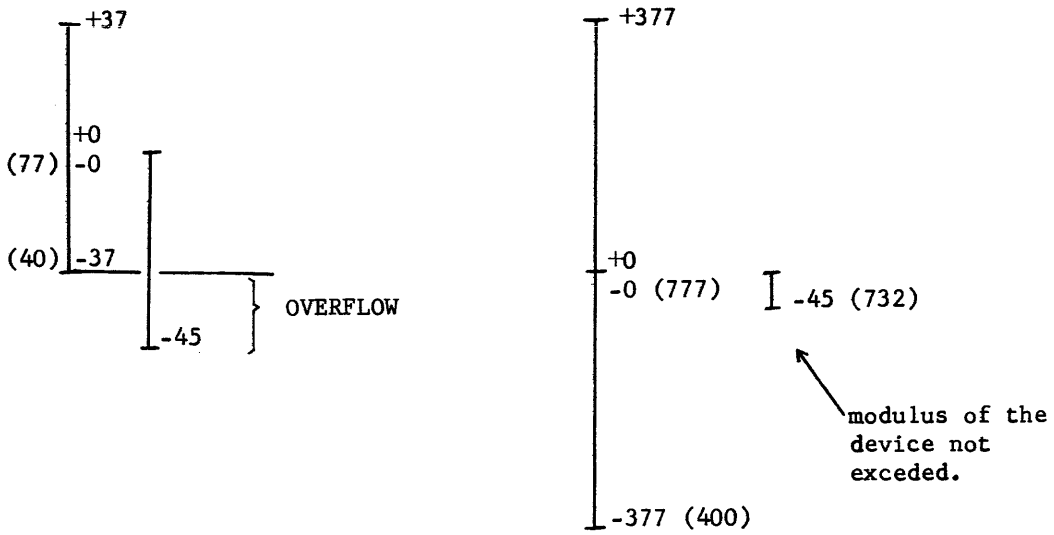
The counter has produced an incorrect result because, again, the modulus of the device has been exceeded. If the same two operands are used but the modulus of the device is expanded to 8^3 -1, the answer will now be correct.

For example:

$$\begin{array}{rcl} -30 & & 747 \\ & = & \\ \ominus\ +15 & & 015 \\ \hline -45_8 & = & 732_8 \end{array} = -45_8$$



Comparison of the two foregoing examples clearly illustrates what causes overflow. In the first example, the difference between the two operands was -45. The modulus of the device only permits expression of operands between -37 and +37. The second example used the same operands but the modulus of the device permits expression of values between -377 and +377.



Remember the rule for overflow? If two numbers of unlike signs are subtracted and the result has the same sign as the subtrahend, the modulus of the device has been exceeded and OVERFLOW has occurred.

Well, all of the desired objectives of this chapter have been satisfied. Turn back to the beginning of the chapter and review them once more. If you are confident that they have been achieved, complete the remainder of the practice problems and continue. If not, review the chapter again. A thorough understanding of number systems and complement notation is a necessity before you attempt to learn how to program a computer.

Conversions:

129. $111\ 000\ 111_2$ = _____$_{16}$

130. $111\ 000\ 111_2$ = _____$_{10}$

131. $111\ 000\ 111_2$ = _____$_{8}$

132. 735_8 = _____$_{16}$

133. $FADE_{16}$ = _____$_{10}$

134. 777_8 = _____$_{10}$

135. $FEED_{16}$ = _____$_{2}$

136. 3456_8 = _____$_{2}$

137. $DEED_{16}$ = _____$_{8}$

138. 100_{10} = _____$_{8}$

139. 200_{10} = _____$_{2}$

140. 300_{10} = _____$_{16}$

Arithmetic:

ADD (modulus $2^{12}-1$)

141.
```
  101 111 111 111
  000 000 000 001
```

142.
```
  011 111 111 111
  100 000 000 000
```

143.
```
  000 111 000 111
  000 111 000 111
```

144.
```
  000 000 000 000
  111 111 111 111
```

SUBTRACT (modulus 2^{12} -1)

145. 111 111 111 111
<u>000 000 000 000</u>

146. 111 000 111 000
<u>000 111 000 111</u>

147. 001 010 011 100
<u>100 011 010 001</u>

148. 111 110 101 100
<u>110 101 100 011</u>

149. Which of the preceding 8 practice problems caused an overflow condition?

150. Which of the preceding 8 practice problems had operands whose signs were such that overflow could have occurred?

MULTIPLY (modulus 2^{6} -1)

151. 011 111
<u>011 111</u>

152. 100 000
<u>011 111</u>

153. 111 111
<u>000 010</u>

154. 011 100
<u>010 001</u>

155. Which of the four multiplication problems have incorrect answers? Why?

156. How do you think the computer performs multiplication and reaches the correct answer?

157. How do you think a computer that can only add or only subtract performs multiplication?

158. What would the computer do if two negative operands were multiplied together?

DIVIDE (modulus $2^6 - 1$)

159. 010 001 $\overline{)\ 011\ 101}$

160. 111 101 $\overline{)\ 001\ 110}$

161. 000 100 $\overline{)\ 001\ 000_2}$

162. 000 111 $\overline{)\ 110\ 001_2}$

163. Which of the 4 divide problems gave the wrong answer? Why?

164. How do you think the computer performs division and produces the correct answer?

165. How can a computer add and subtract if the device is only additive?

166. Why is only an additive device or a subtractive device used in a computer?

167. What is significant about end around borrow and end around carry?

168. What does end around borrow accomplish?

169. What is the difference between overflow and end around carry? Are they related?

170. What causes overflow?

SUMMARY

This chapter has been concerned with computer mathematics including number systems, conversions, mathematical operations, complements, and overflow. You have learned that digital computers function by using binary arithmetic. The reason for binary arithmetic (radix 2) is that transistorized switches are used to represent digits of the operands and a switch has only two discrete prositions. We also discussed octal and hexadecimal number systems because each represents a multiple number of binary digits. Computers display results in either of the two bases because that result is more easily recognized than the same result in binary. For that reason, you learned how to perform conversions between these bases and our familiar decimal base. You also learned how to perform arithmetic operations with each of these number systems.

Next you discovered what is meant by a complement (it doesn't mean you are saying something nice to someone when spelled with an e) and why negative numbers can be expressed in complemented form. Additive and subtractive operations could be performed directly when using complements and the operands did not have to be examined to determine their sign. Ones complement arithmetic is always used with a signed binary device to compensate for two values of zero that are inherently present. An unsigned device, such as a counter, would use twos complement arithmetic because the combinations $00--0_2$ and $11--1_2$ each represent a different discrete value and the modulus would be <u>one</u> more than a similar signed device.

You learned that a computer could perform all four basic mathematical operations through the use of only one device -- either additive or subtractive. The advantage of one over the other is that the subtractive device will rarely produce an answer of negative zero. An answer of zero magnitude will always be indicated in the positive format of 0--0. A computer performs multiplication and division by complementing negative operands to their magnitude format and performing the operation by a series of multiple adds or multiple subtracts. The results of these operations are complemented again if the final answer and/or remainder should be negative.

You have seen that an additive device is nothing more than a one-way counter, and a subtractive device is also a counter but the reverse of the additive type. It should be mentioned that although a modern computer employs the use of either an additive or a subtractive counter, the device operates in parallel mode by simultaneously determining partial adds and carries or partial differences and borrows. This method, when employed with fast-switching transistors, enables the arithmetic operation to be performed in millionths of seconds (microseconds). This provides extremely fast computational speed. Remember the advantages of a computer? Speed, accuracy, and cost per computation. Doubling the speed of a device effectively decreases the cost per computation by one half, assuming no increase in cost for the faster machine.

You also learned that a signed summing device (adder), such as a computer would employ, sometimes produces the incorrect answer although the computer is still functioning perfectly. This undesirable situation occurs when the result of a given mathematical operation exceeds the modulus of the device. A device with a modulus of 8^3 -1 could only express operands with a magnitude up to and including $\pm 377_8$. Trying to express $\pm 400_8$ in that device would exceed the modulus and result in an incorrect answer and a condition called OVERFLOW. This condition would be detected by the computer and a fault would be indicated.

Collectively, this chapter should have prepared you with the necessary background in number systems and computer mathematics and should have stimulated your interest so that you wish to explore further the mysteries surrounding a digital computer. If this is the case, congratulations! Let's continue on to explore and conquer these mysteries.

EXPONENTIAL POWERS OF COMMON NUMBER SYSTEMS

Binary	Octal	Decimal	Hexadecimal
$2^0 = 1$	$8^\circ = 1$	$10^\circ = 1$	$16^\circ = 1$
$2^1 = 2$	$8^1 = 8$	$10^1 = 10$	$16^1 = 16$
$2^2 = 4$	$8^2 = 64$	$10^2 = 100$	$16^2 = 256$
$2^3 = 8$	$8^3 = 512$	$10^3 = 1,000$	$16^3 = 4,096$
$2^4 = 16$	$8^4 = 4,096$	$10^4 = 10,000$	$16^4 = 65,536$
$2^5 = 32$	$8^5 = 32,768$	$10^5 = 100,000$	$16^5 = 1,048,576$
$2^6 = 64$	$8^6 = 262,144$	$10^6 = 1,000,000$	$16^6 = 16,777,216$
$2^7 = 128$	$8^7 = 2,097,152$	$10^7 = 10,000,000$	$16^7 = 268,435,456$
$2^8 = 256$	$8^8 = 16,777,216$	$10^8 = 100,000,000$	$16^8 = 4,294,967,296$
$2^9 = 512$	$8^9 = 134,217,728$	$10^9 = 1,000,000,000$	$16^9 = 68,719,476,736$
$2^{10} = 1024$	$8^{10} = 1,073,741,824$	$10^{10} = 10,000,000,000$	$16^{10} = 1,099,511,627,776$

ANSWERS TO PRACTICE PROBLEMS

1. 170_{10}
2. 3612_{10}
3. $703,674_{10}$
4. 455_{10}
5. If you solved this one, read Page 8 again.
6. $1,000,000_{10}$
7. $.797_{10}$
8. $.16308_{10}$
9. $.67058_{10}$
10. $.03125_{10}$
11. $.5_{10}$
12. $.99580_{10}$
13. 25_{8}
14. 15_{16}
15. 70_{8}
16. 38_{16}
17. $111\ 000\ 011\ 110_{2}$
18. EIE_{16}
19. $101\ 010\ 101\ 010_{2}$
20. AAA_{16}
21. $1\ 011\ 111\ 010\ 101\ 101_{2}$
22. 137255_{8}
23. $1\ 101\ 111\ 011\ 101\ 101_{2}$
24. 157355
25. $.5_{8}$
26. $.A_{16}$
27. $.0004_{8}$
28. $.004_{16}$
29. $.000\ 000\ 001\ 010\ 011_{2}$
30. $.00A6_{16}$
31. $.111\ 110\ 101\ 100_{2}$
32. $.FAC_{16}$
33. $.000\ 100\ 100\ 011\ 101\ 010\ 111\ 100_{2}$
34. $.04435274_{8}$
35. $.111\ 111\ 111\ 111_{2}$
36. $.7777_{8}$
37. 31790_{10}
38. $.7917_{10}$
39. 4012_{10}
40. $.02539_{10}$
41. 92_{10}
42. $.33203_{10}$
43. $001\ 111\ 100\ 111_{2}$
44. 1751_{8}
45. $A000_{16}$
46. $110\ 011\ 101\ 101\ 100_{2}$
47. $.00027_{8}$
48. $.0000A_{16}$
49. 167_{8}
50. 133_{16}

Answers to Practice Problems (Contd.)

51. $.004_8$

52. $.AA_{16}$

53. $111\ 110\ 101\ 100\ 011\ 010\ 001_2$

54. $.110\ 110\ 101\ 101_2$

55. $100\ 110\ 001\ 011_2$

56. $.000\ 000\ 000\ 000\ 111\ 1_2$

57. $71C7_{16}$

58. $.0180C_{16}$

59. 177351_8

60. $.06425474_8$

61. 1010_2

62. 0110_2

63. 0011_2

64. 0011_2

65. $101\ 101_2$

66. $1\ 101\ 110_2$

67. 101_2, no remainder

68. 1010_2, 100 remainder

69. 12_8

70. 6_8

71. 3_8

72. 3_8

73. 55_8

74. 156_8

75. 5_8

76. 12_8, 4 remainder

77. 10107_8

78. 104172_8

79. 524_8

80. 336_8

81. 230_8

82. 400_8

83. 571_8

84. 3763_8

85. $100{,}000_8$

86. 10_8, no remainder

87. 101_8, 1 remainder

88. 2_8, no remainder

89. 1898_{16}

90. $EA9A_{16}$

91. If you knew it was impossible, why are you checking the answer?

92. $B99A_{16}$

93. $111\ 111_{16}$

94. $99\ 999_{16}$

95. $AD743F8_{16}$

96. $4F9B06_{16}$

97. $BF1BFDA_{16}$

98. 1_{16}, 3 remainder

99. $1A_{16}$, 675 remainder

100. $11\ 111_{16}$, 1 remainder

Answers to Practice Problems (Contd.)

101. $+37_8$

102. -37_8

103. -0_8

104. -36_8

105. $+25_8$

106. $+0_8$

107. -25_8

108. -7_8

109. $+7_8$

110. $+34_8$

111. 0030_8

112. 7747_8

113. 0000_8

114. 7777_8

115. 7740_8

116. 0037_8

117. 0100_8

118. 7677_8

119. 0377_8

120. 7400_8

121. 0430, no

122. 7177 = -600, no

123. 2700, no

124. 4377 = -3400, no

125. 3776, no

126. 3776, no

127. 4000 = -3777, yes

128. 4100 = -3677, yes

129. $1C7_{16}$

130. 455_{10}

131. 707_8

132. $1DD_{16}$

133. $64,222_{10}$

134. 511_{10}

135. $1\ 111\ 111\ 011\ 101\ 101_2$

136. $011\ 100\ 101\ 110_2$

137. 157355_8

138. 144_8

139. $011\ 001\ 000_2$

140. $12C_{16}$

141. $110\ 000\ 000\ 000_2$

142. $111\ 111\ 111\ 111_2$

143. 001 110 001 110

144. $111\ 111\ 111\ 111_2$

Answers to Practice Problems (Cont'd.)

145. $111\ 111\ 111\ 111_2$

146. $110\ 001\ 110\ 001_2$

147. $100\ 111\ 001\ 010_2$

148. $001\ 001\ 001\ 001_2$

149. Only problem number 147.

150. Problems 143, 145, 146, 147.

151. $001\ 111\ 000\ 001_2$

152. $001\ 111\ 100\ 000_2$

153. $000\ 001\ 111\ 110_2$

154. $000\ 111\ 011\ 100_2$

155. Problems 152, 153. It's impossible to multiply numbers in their negative (complemented) form. Multiplication is the product of absolute values.

156. Operands are examined for complements. Negative operands are complemented to magnitude representation and multiplication is then performed. If original signs were unlike, answer is complemented (multiplication with unlike signs result in a negative answer).

157. Multiplication is nothing more than multiple additive steps with several partial products similar to pencil and paper multiplication. The computer adds the multiplier to the partial product each time a 1 appears in the multiplicand. After a multiplicand bit has been examined, it is discarded and the next bit is examined until the process is complete.

158. The computer would complement both operands to their magnitude format and then multiply. The answer would be left positive because the product of two numbers with like signs is positive.

159. $000\ 001_2$, $001\ 100_2$ Remainder

160. Impossible to perform.

161. $000\ 010_2$, 000 000 remainder.

162. $000\ 111_2$, $000\ 000_2$ remainder.

163. Problems 160 and 162. Divisions cannot be performed upon numbers in complemented (negative) form.

164. The computer always divides a positive into a positive by complementing where necessary. The original operands are examined for negative quantities and the final answer and/or remainder is complemented where necessary. For example, the remainder sign is alwyas the same as the original sign of the dividend. If original dividend and divisor signs were alike, the answer is positive; if unlike, the answer is negative. The computer performs division in essentially the reverse process of multiplication; this is, by a series of subtract operations.

Answers to Practice Problems (Cont'd.)

165. Subtraction can be performed in an additive device by complementing the subtrahend and adding.

166. All four basic arithmetic operations can be performed with the use of only one type of device. If the device is subtractive,

1. Subtract two operands directly.
2. Add by complementing the addend and subtracting.
3. Multiply by a series of multiple adds. Addition is performed by complementing and subtracting.
4. Divide by a series of subtract operations.

167. End around carry occurs in an additive device if the counting operation passes through zero. End around borrow occurs in a subtractive device if the counting operation passes through zero.

168. It compensates for two positions of the counter that both represent the same arithmetic value - zero.

169. Overflow and end around carry are completely unrelated. Compare answers to problems 168 and 170 for difference.

170. Any time the modulus of a device is exceeded, (whether additive or subtractive), overflow occurs.

chapter IV
Programming

CHAPTER IV

PROGRAMMING

INTRODUCTION

It would be difficult for you to communicate with a person from another country unless you both understood a common language. Likewise, communications between you and a digital computer would be difficult without a common language.

Computers have not yet been designed that converse in English. A person associated with them must, therefore, converse in the computer's language--the language of numbers. You learned the binary number system in the preceding chapter and now should be able to communicate with a digital computer in its language.

The objective of this chapter is to familiarize you with the methods used to instruct, or program, a digital computer. Any discrete defined problem can be solved by a digital computer. You will be taught how to define the problem, diagram a solution, and then record that problem in a language meaningful to the computer.

A comparison can be made between a man sitting at a desk and a digital computer. Any computer consists of four main sections (input and output are both considered as part of the I/O section).

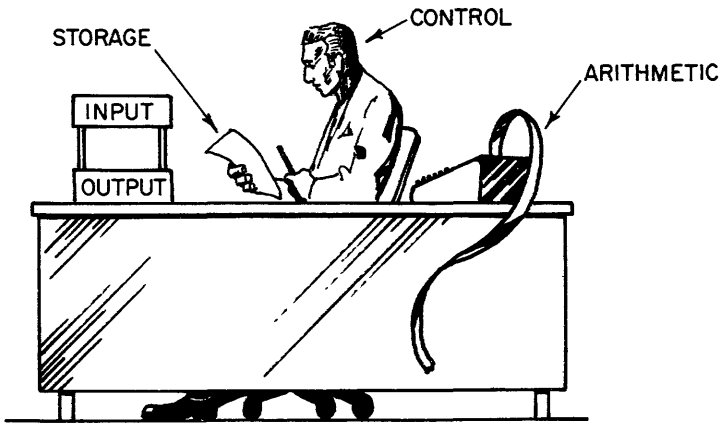
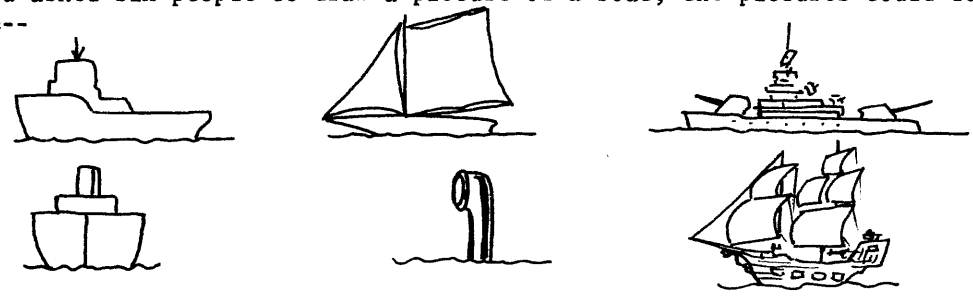


Figure 4-1. A "man-sized" computer

The Input/Output section is comparable to the in/out basket; the storage, or memory section, to the pencil and paper; the arithmetic section to the adding machine; and the control section to the man at the desk.

Man can accomplish tasks and make decisions because he can reason. The digital computer does not possess the ability to reason and thus must be "instructed" by someone who can.

If you asked six people to draw a picture of a boat, the pictures could look like these--



but, if you told the same six people to draw a picture of a rowboat, the pictures would more closely resemble each other. By defining what you specifically wanted, acceptable results were obtained. Now assume that those same six people have never seen a boat of any kind and that they will draw only as you "instruct" them. Would the pictures all appear the same? Would they all resemble a rowboat?

The pictures would be identical <u>if</u> each person responded exactly as the next and precisely followed your instructions. The pictures would all resemble a rowboat <u>if</u> your list of instructions, or program, was correct and complete.

A computer executes a list of instructions, or program, exactly as instructed. Whether right or wrong, it continues until commanded to stop. Instructing the computer how to solve your problems is an exact science known as programming.

WRITING A PROGRAM

The digital computer can only perform certain simple operations. The programmer's job is to simplify complicated mathematical equations and reduce them to basic step-by-step operations that a high-speed machine (the computer) can perform. The programmer can use the computer to solve almost any problem if he prepares his instructions properly and if they are entered into the computer in correct order. In writing a program, the programmer predetermines all operations the computer must perform to carry out or run the program and solve the problem. The complete programming operation may be divided into five steps.

1) Determine and precisely state the complete problem.

2) Analyze the problem, determining exactly what must be done.

3) Plan the problem by breaking it down into a sequence of logical steps or operations. The first breakdown should be very general, then it should be expanded by adding detailed steps. In the final breakdown,

care must be taken to keep the logical steps compatible with the machine.

4) Code the logical steps into an acceptable language.

5) Run and debug the program, making any necessary corrections.

If you were to drive across the United States, you would probably use a graphic diagram called a road map. The map indicates the intermediate points between the starting point and your destination. A computer program can also be graphically expressed using a diagram called a FLOW CHART.

FLOW CHART

A flow chart provides a means of visualizing the over-all problem and the manner in which it is to be processed. Symbols are used to denote general operations or functions. Refer to Figure 4-2 for a definition of these symbols. It is customary to first make a general, simple flow chart to get a broad view of the problem. Afterward, a more detailed chart is drawn. Then, using the detailed chart, the programmer takes several examples of the problem and applies them to see if anything should be deleted or added. When writing the program, it is wise to keep the flow chart handy for later use in debugging the program, if necessary.

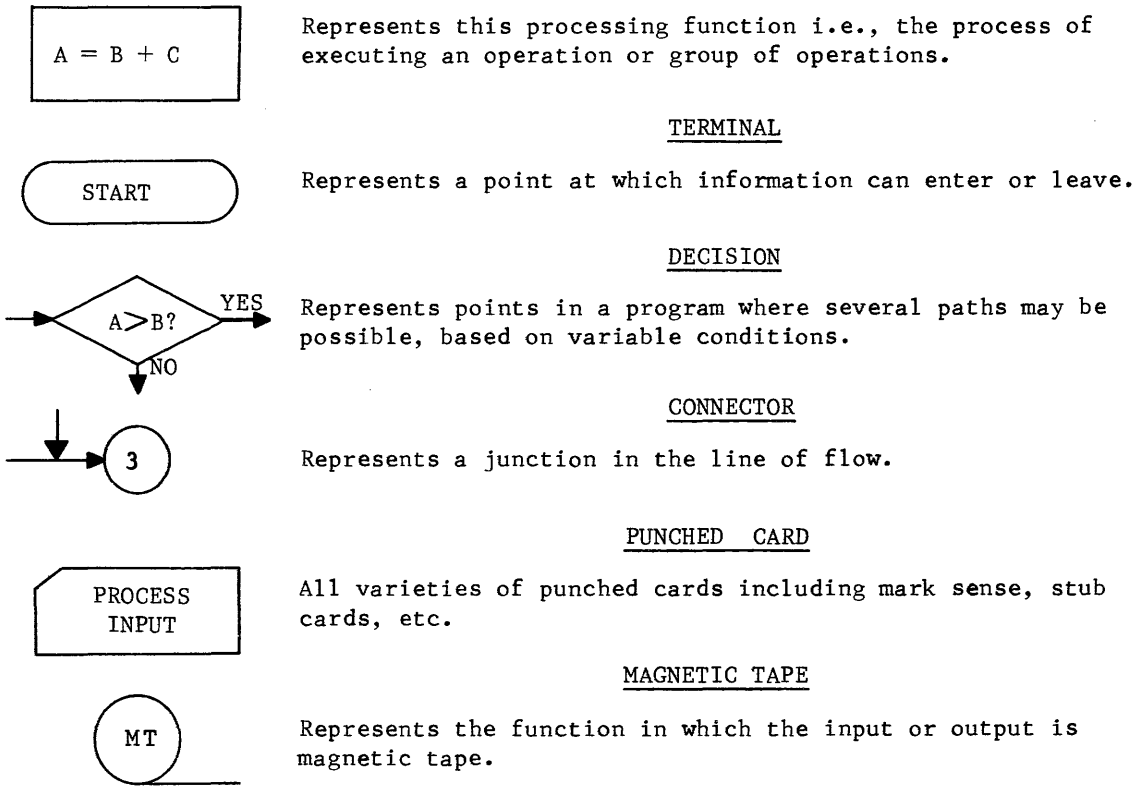
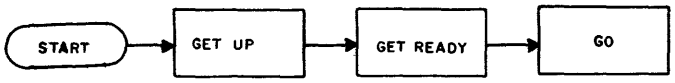
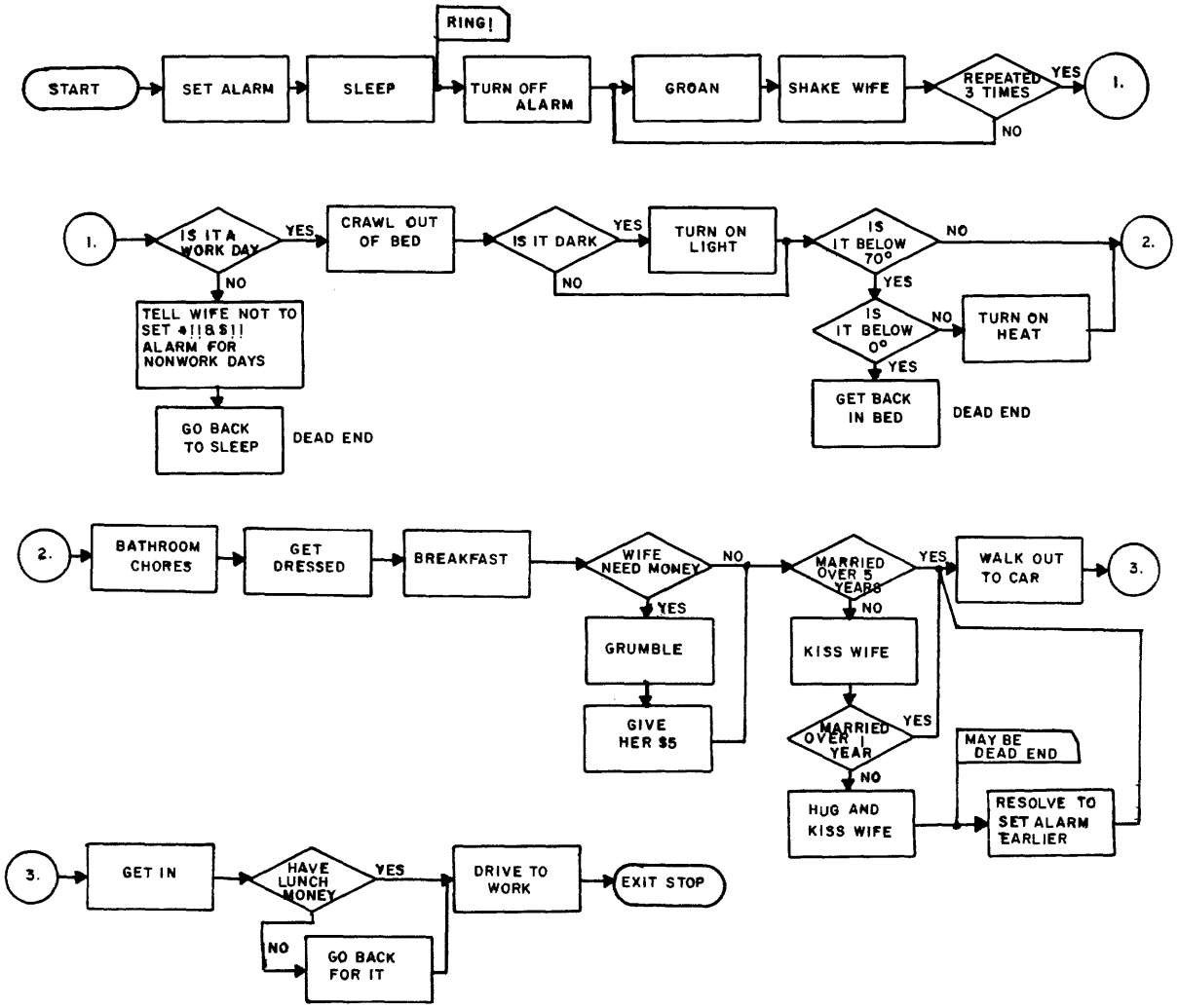


Figure 4-2. Flow Chart Symbols

The following example (figure 4-3) may help clarify the use and construction of a flow chart.



Simple Flow Chart



More Detailed Flow Chart

Figure 4-3. Flow Chart Examples

A flow chart is a diagram of the step-by-step solution of a problem. The flow chart must describe all of the operations and decisions that are required to find the solution, including the possibility of an error.

The following problem and suggested flow chart serve as an example.

In a salesman's commission calculation, there are five different commission formulas, depending on the product classification code. They are as follows:

Product Code	Commission Formula
1	15% x sale price
2	40% x (sale price - base price)
3	10% x base price
4	$10.00 + (5% x base price)
5	$15.00

Flow chart the total commission calculation for ten sales. See figure 4-4 for this illustration.

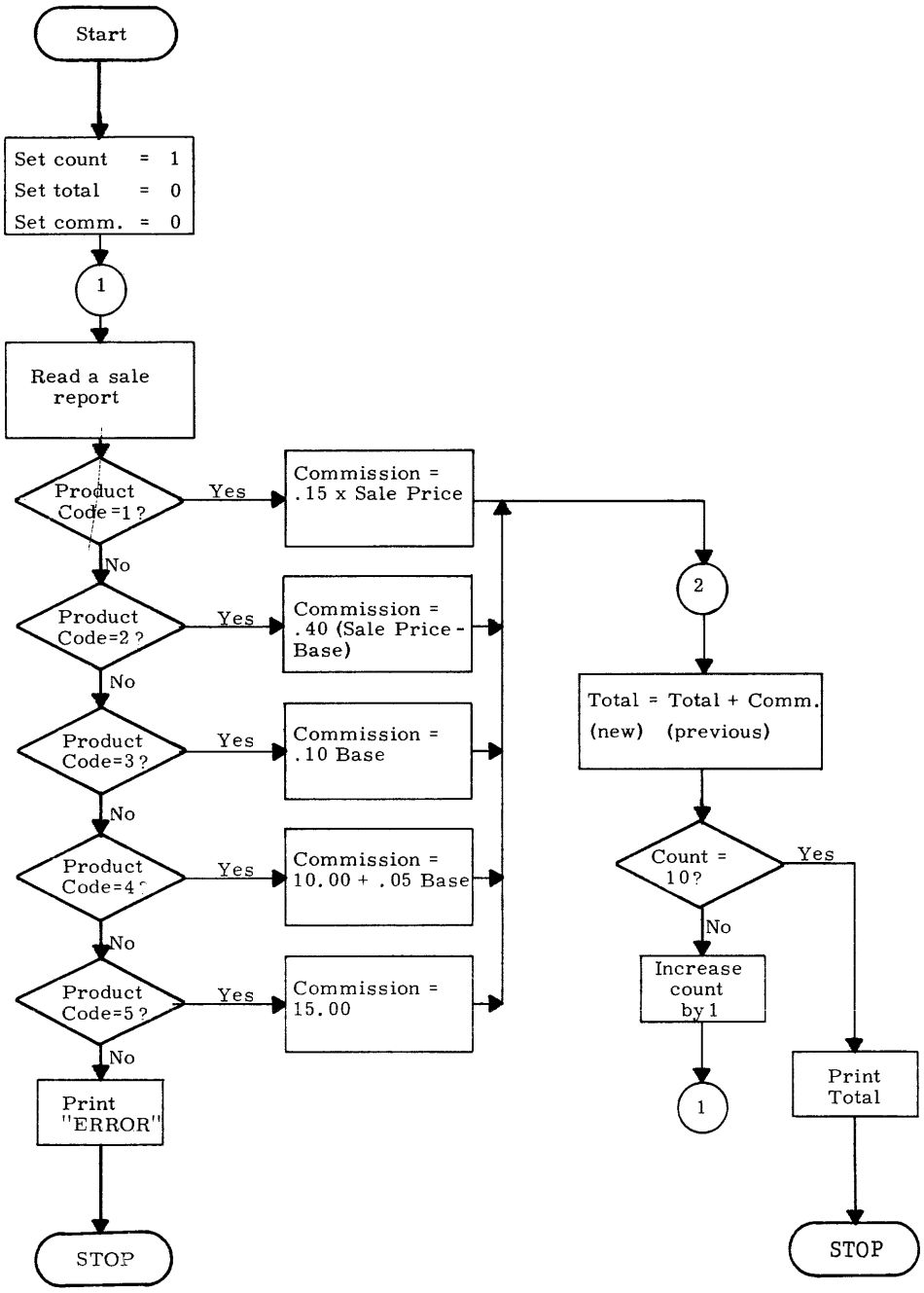


Figure 4-4. Commission Calculation Flow Chart

Draw your version of a flow chart that would solve this problem:

1) A snail climbing out of a well advances three feet each day but slips back two feet each night. How many days does it take the snail to climb out if it starts 30 feet below the top of the well?

Wasn't that fun? Try another one.

2) Using the following information, compute the tax for 100 employees and print the amount that each is required to pay. Net earnings are total income less $600 for each dependent.

Annual Net Earnings	Tax
Less than $2000	No tax
$2000--$4999.99	2% of amount over $2000
$5000 or over	$60 plus 5% of amount over $5000

And one more.

3) An item being sold has 3 quoted prices. The unit price is $1.50 for less than 20 items, $1.35 for 20-to-99 items, and $1.25 for quantities of 100 or more. Draw a flow chart to read 100 different sales reports and print the total cost of each sale.

NOTE:

Remember that a computer can make only one yes or no decision at any given time.

One solution to each flow chart problem is located at the end of the chapter. They may be quite different from your versions because a given problem may be solved by several varied methods.

You learned earlier that the complete programming operation was a sequence of five steps, and

1) the problems were determined and precisely stated.

2) each problem was analyzed.

3) each problem was divided into a series of logical steps with the aid of the flow chart.

4) the problems were coded--oh, oh, we haven't done that yet!

Well, you had better take a short break because this one is going to take a while.

A computer program may be coded by three different methods:

1) by writing the instructions in the computer's language (machine language),

2) by utilizing a programming aid known as an assembler and writing the program in a mnemonic language (assembler language), or

3) by using a sophisticated programming aid known as a compiler (compiler language).

Each of the three different methods will be discussed and comparisons made to determine the advantages of each.

PART I

MACHINE LANGUAGE PROGRAMMING

Machine language, also called absolute coding, is a series of numerical instruction codes that "tell" the computer what to do. Although the codes are actually in binary, they are normally represented and the programs are written in octal notation. A code of 30 instructs the computer to add; a 50 would designate multiplication; and division is indicated by a code of 51.

The add instruction (30) forms the sum of only two operands. The following problem would require the computer to add five different times.

$$X = A + B + C + D + E + F$$

$$\text{1st add}\ \begin{array}{r} A \\ B \\ \hline A+B \end{array} \qquad \text{2nd add}\ \begin{array}{r} A+B \\ C \\ \hline A+B+C \end{array} \qquad \text{3rd add}\ \begin{array}{r} A+B+C \\ D \\ \hline A+B+C+D \end{array}$$

$$\text{4th add}\ \begin{array}{r} A+B+C+D \\ E \\ \hline A+B+C+D+E \end{array} \qquad \text{5th add}\ \begin{array}{r} A+B+C+D+E \\ F \\ \hline A+B+C+D+E+F \end{array}$$

The computer executes these instructions very rapidly. It can do all four basic arithmetic operations (+,-,x,÷), search, locate, shift, store, interchange, and a variety of others, all in a few millionths of a second.

The instruction set, or repertoire, for a given computer may consist of only a few general instructions; whereas another computer may have an instruction repertoire of more than one hundred.

The numerical codes are fed into the computer via a variety of input media. Examples are magnetic tape, paper tape and punched cards. Each numerical code is followed by additional numbers specifying the operands, or the location of the operands, which are to be used in the operation.

INTERPRETATION OF NUMBERS

Each type of computer has its own instruction set which could vary greatly from that of another machine. Although an add instruction has been defined as a 30, another type of computer could have a different number to designate an add. We will adopt a portion of a typical instruction repertoire for our study of machine language.

The numbers in a computer are called "words" of its language. A computer word may be either an instruction or an operand, depending upon its use. The computer reads instructions which determines how the operands are to be affected.

Some basic terms must be defined before a study of the instructions can be undertaken. Some of the terms are: registers, switches, displays, and indicators.

TERMINOLOGY

Registers

A register is a device or circuit which temporarily stores numbers, consisting of bits or digits (usually a series of flip-flops). This definition is accurate and adequate if a person knows what circuits and flip-flops are. Consider the following, even though circuits and flip-flops are not employed.

The checkout girl at the grocery store temporarily stores the cost of each commodity in the keys of the cash register before she presses the add key to commit the new value to the memory of the cash register. The keyboard actually satisfies the definition of a register because it temporarily holds information.

In computers, much the same thing happens except keys are not used to hold the information; instead, an electrical circuit performs this function. The circuit is usually called a flip-flop circuit. Basically, registers hold information for short durations and the speed of entering the information or removing it depends upon the electrical circuit. All registers considered in this book are much faster than memory devices; the checkout clerk is, perhaps, fast at setting the keys but the motor-driven cash register is faster at putting information into memory.

Consider another comparison to further describe a register. The man at the desk stores information in his mind (temporary storage) before he puts the numbers on paper (permanent storage).

There are many registers in computers, each having a different function. They are considerably faster than the computer's memory and many times easier to change or correct.

Switches

Radios and lights in a house are turned on by using switches. There are many kinds of switches; e.g., flip-switches and pushbutton, to name but two. Switches in a computer allow an operator to turn it on, shut it off, put the numbers he desires into the registers and start the machine.

Displays

A register may display the numbers it contains by a series of lights; the computer displays its versatility by doing several things at once. Computer displays are of the visible type and use lights or a device similar to a television screen.

Indicators

If the computer detects some kind of error or malfunction, it may indicate that condition by lighting a red light. Overflow always produces a wrong answer and the computer would indicate the abnormal condition by lighting a red light. The computer may indicate that it is trying to communicate with some I/O equipment by lighting a green or yellow light.

Computer Console

All main computer controls and indicators are on the console (figure 4-5). Indicators are lamp modules, each of which displays an octal digit. The lamps, in response to signals from the computer, display the contents of the operational registers in octal form only when the computer is stopped; the display is blank when the computer is running.

Figure 4-5. Typical Control Data Console

The registers in the computer are identified by letters. The operational registers usually hold the end result of an operation or an instruction, an operand, or an address. Their contents are displayed on the console and may be manually changed by the operator. The transient registers used in formulating the result are secondary registers. They are usually not displayed and normally cannot be manually changed. See tables 4-1 and 4-2 for a description of the registers.

TABLE 4-1. OPERATIONAL REGISTERS OF THE COMPUTER

Register	Function
* A	Arithmetic
* Q	Auxiliary Arithmetic
B^1-B^3	Index Registers
P	Program Address
F	Instruction Register

* The arrow displayed between the two arithmetic registers indicates that the A register is on the left and the Q register is on the right (Figure 4-5).

TABLE 4-2. ARITHMETIC PROPERTIES OF REGISTERS

Register	Number of bits or stages	Modulus	Complement Notation	Arithmetic
A1 and A2	24	2^{24}-1	ones	signed
Q1 and Q2	24	2^{24}-1	ones	signed
B1-3	15	2^{15}	twos	unsigned
P	15	2^{15}	twos*	unsigned
F	24	2^{24}	twos	unsigned
S	15	2^{15}	Secondary registers with no arithmetic properties.	
Z	24	2^{24}		
X	24	2^{24}		

*Although the P register is a twos complement device with a 2^{15} modulus, a special circuit forces it to skip 77777 (modulus 2^{15}-1) when being incremented by one. It can be forced to 77777, however, which explains the modulus (2^{15}).

KEYBOARD

The entry keyboard on the console is the only means provided to manually enter data into the computer. Figure 4-6 shows the keyboard portion of the console. Refer to figure 4-5 for physical location of the keyboard on the console.

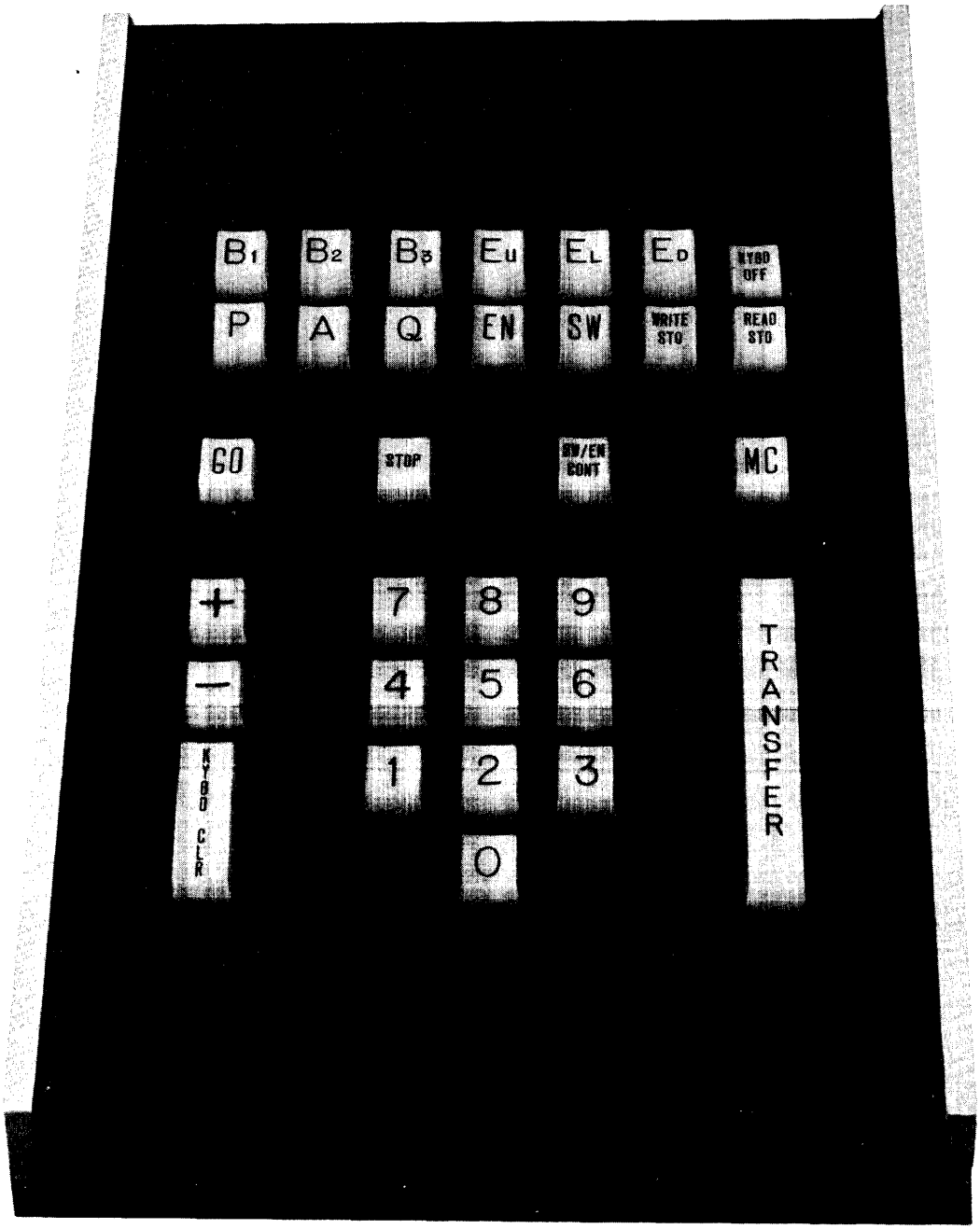


Figure 4-6. Keyboard

Table 4-3 lists the keyboard switches and gives their description.

TABLE 4-3

Switch Name	Illum.	Description
Keyboard Off	Yes	Deactivates all keyboard controls
Keyboard Clear (momentary)	Yes	Clears the communication register (Instruction Register) and Keyboard
GO (momentary)	Yes	Starts the computer at the address to which P has been set
Stop (momentary)	Yes	Brings the computer to a halt at the end of the current instruction
Transfer (momentary)	No	Enables the transfer of data between the communication register and a selected register or storage location
MC (momentary)	No	Performs both an internal and external master clear. Disabled when computer is in Go mode
B1 - B3	Yes	Enables the manual entry of data from the keyboard into index registers B1 - B3
P	Yes	Enables the manual entry of an address from the keyboard into the P register
A	Yes	Causes both A and Q to be displayed, but enables entry only into A
Q	Yes	Causes both A and Q to be displayed, but enables entry only into Q
Enter (EN)	Yes	Enables the manual entry of information into storage while computer is stopped
Sweep (SW)	Yes	Provides capability of examining contents of memory.
0-7	No	These switches allow entry of octal numbers directly into a register or memory (determined by top two rows of switches, figure 4-6)

NOTE:

The keyboard switches shown in figure 4-6 but not listed in table 4-3 permit special operations not discussed at the introductory level.

LOGICAL DESCRIPTION

The computer performs calculations and processes data in a parallel, binary mode through the step-by-step execution of individual instructions that are stored internally along with the data.

Functionally, computer operations may be divided into four major sections. Input/output provides communications between the computer and the external equipment; arithmetic performs the arithmetic and logical operations required for executing instructions; control coordinates and sequences all operations for executing an instruction by obtaining the instruction from storage and translating it into commands for the other sections. The basic computer is illustrated in figure 4-7. As with any computer, it includes the four previously-mentioned sections.

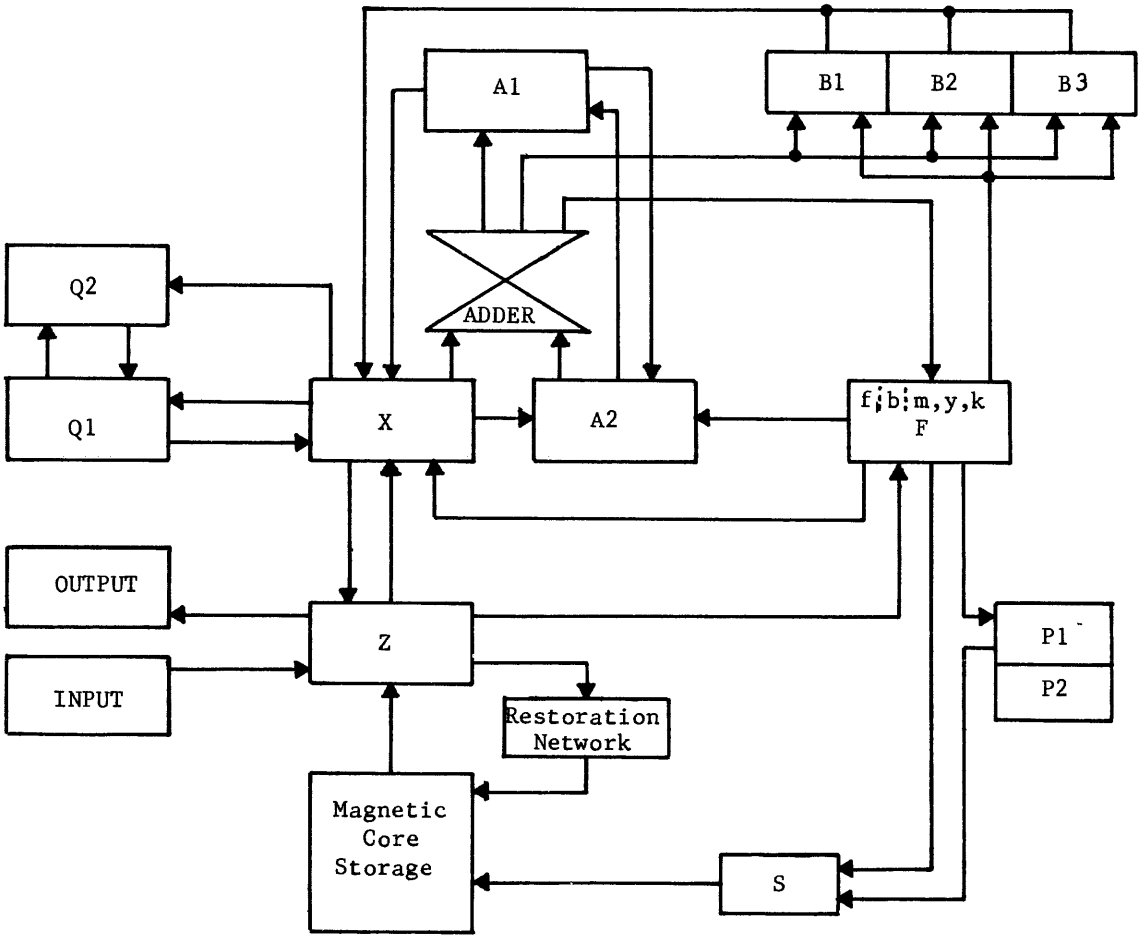


Figure 4-7. Basic Computer Block Diagram

B1 - 3, P1, P2 and S are 15-bit registers used for addresses or address modification.

A1, A2, F, Q1, Q2, X, and Z, are 24-bit registers used for full word operands or instructions.

INSTRUCTION FORMAT

The computer program consists of a series of instructions read from storage and executed one at a time. Each instruction is a 24-bit computer "word" which is routed from storage to the F register. The F register translates the instruction to determine what action should be taken. When the computer has executed that instruction, another is read from storage and the process is repeated.

The path the instruction follows as it is read from memory and placed in the F register is illustrated by figure 4-8.

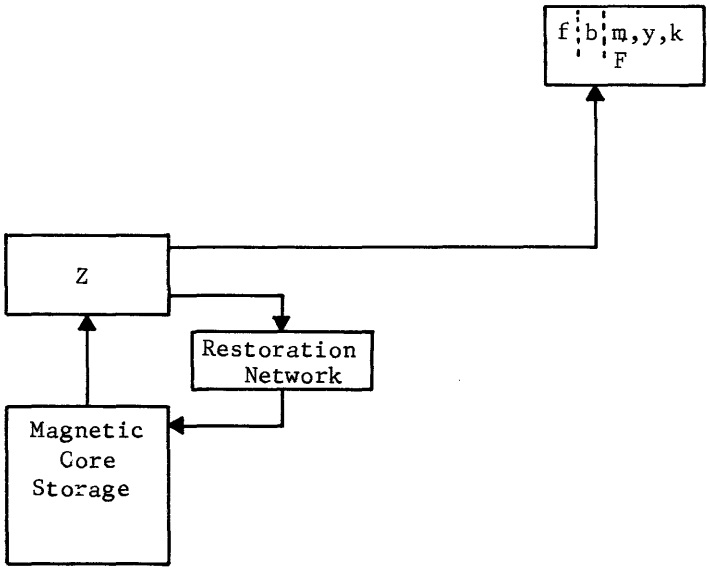


Figure 4-8. Instruction Flow Path

Figure 4-8 is a portion of the basic block diagram (figure 4-7). The instruction is read from storage into the Z register and then into the F (function) register. The Z register also allows the instruction to be restored back into storage after being destroyed by the reading process. The diagram indicates that the F register has several parts, each to accommodate a different portion of the instruction.

A variety of computers would have a variety of instructional formats, determined by the design and function of each machine. Figure 4-9 illustrates several different instructional formats, with the format for the basic computer outlined in the center.

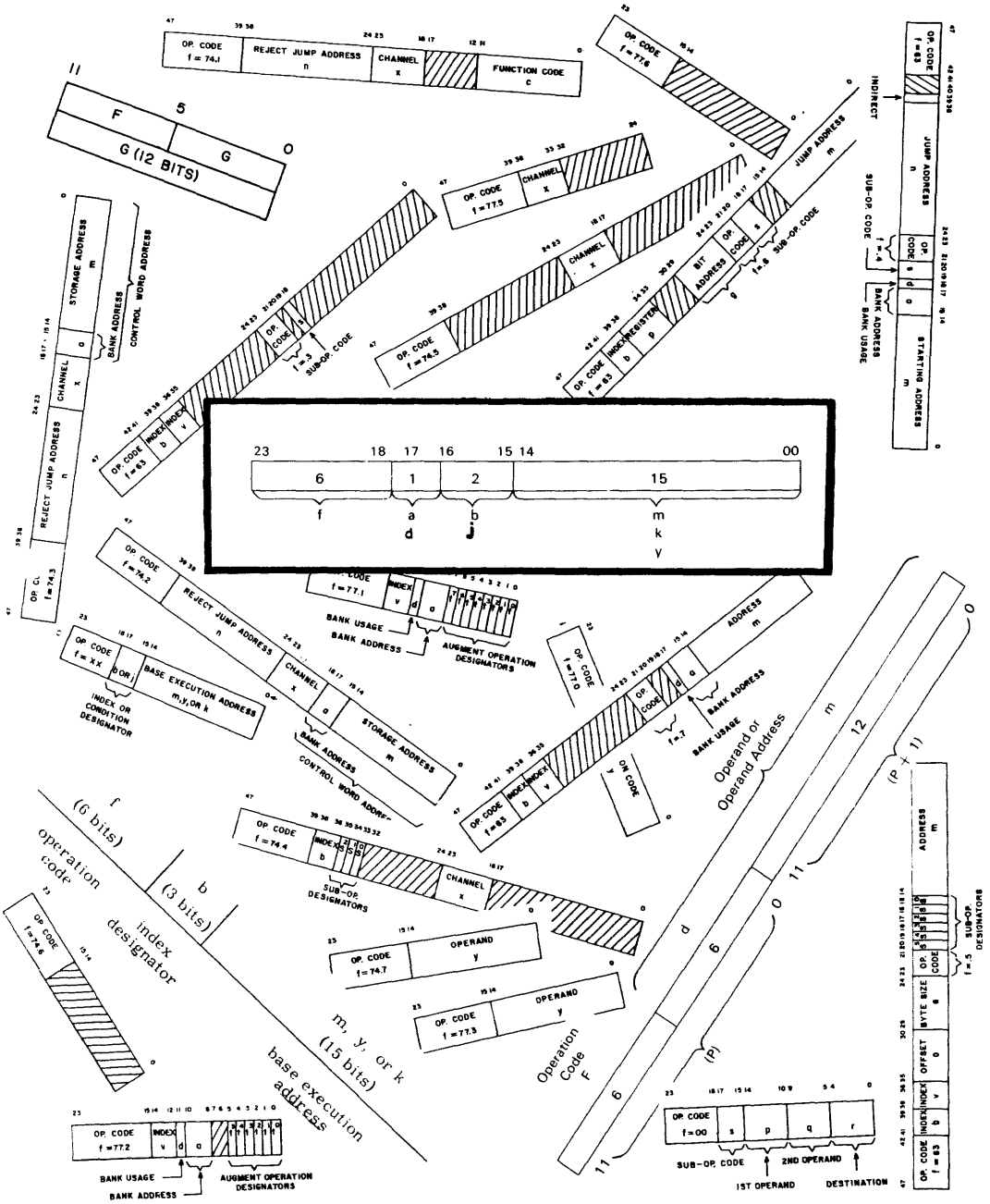


Figure 4-9. Instructional Formats

The F portion of the instruction determines its function and consists of six bits (two octal digits). A function code of 30 indicates that an add operation is to be performed. The two octal digits could be any combination from 00 through 77, equivalent to 64 different decimal combinations. Each combination designates a different instruction, but the basic computer will use only 25 combinations (25 instructions).

The next portion of the instruction format is a single bit referred to as the "a" or "d" designator (figure 4-10). The function of the special designator bit is determined by each instruction and will be discussed as individual instructions require it.

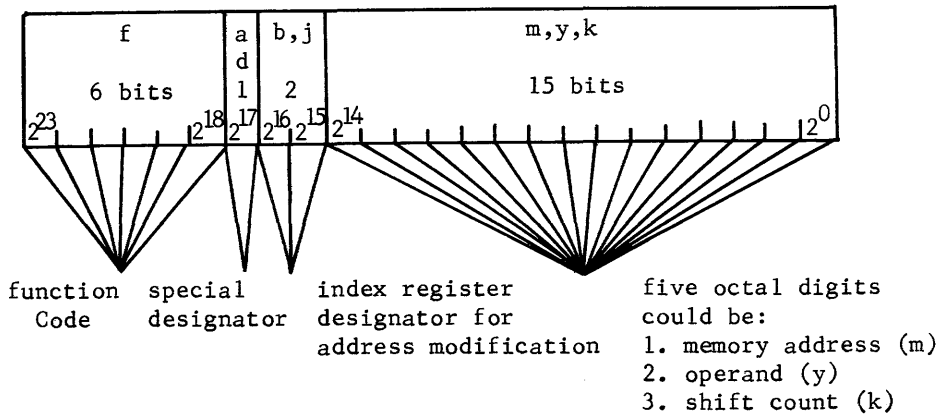


Figure 4-10. Basic Instruction Format

The index register designator consists of two bits and usually indicates that the contents of one of the three index registers (B1, B2, B3) will be used to modify the remaining 15 bits of the instruction. The index register designator can also be used in conjunction with the "a" designator for some instructions.

The remaining 15 bits of the instruction may be used as (1) the address of a storage location, (2) an operand, or (3) a shift count. It may be unmodified or, if desired by the programmer, modified by the contents of the indicated index register. Lower case letters (m, y, k) indicate that the address field is unmodified; capitalized letters indicate a modified address field.

The following examples indicate the relationship between the modified and unmodified address field:

1) The modified operand address M is represented by:

$M = m + (B^b)$ where: M = modified address of operand

m = unmodified address of operand (execution address)

(B^b) = contents of Index register b

If the Index designator = 0, then M = m.

2) The modified operand Y is represented by:

$Y = y + (B^b)$ where: Y = modified operand

y = unmodified operand (execution address)

(B^b) = contents of Index register b

If the Index designator = 0, then Y = y.

3) The modified shift count K is represented by:

$K = k + (B^b)$ where: K = modified shift count

k = unmodified shift count (execution address)

(B^b) = contents of Index register b

If Index designator = 0, then K = k.

Notice that 1) is the only case in which the address field actually contains an address. The interpretation of the address field is determined by the type of instruction being executed.

SYMBOL DEFINITIONS

The following designators will be used throughout the list of instructions.

a = addressing mode designator (a=0, direct addressing; a=1, indirect addressing)

b = index designator (unless otherwise stated)

d = shift designator - bit 2^{17} of instruction. If a 0, shift (A) register; if a 1, shift (Q) register.

f = function code (6 bits, octal 00 through 77).

j = jump, stop, or skip condition designator (see individual instructions)

k = shift count (unmodified)

m = word execution address (unmodified)

y = 15-bit operand

() = "the contents of". For example, (A1) indicates "the contents of" the A1 register.

ADDRESSING MODES

Three modes of addressing are used in the computer: No Address, Direct Address, and Indirect Address.

No Address

No address means that the address field of the instruction is not the address of a storage location. The address field would contain either an operand (y) or a shift count (k).

Direct Address

Direct address means that the address field does contain the base address of a storage location. If the two index designator bits are equal to zero, the address is not modified. The storage reference is made at the address indicated in the address field. If the two index designator bits are 01, 10, or 11, the address must be modified by adding the contents of the designated index register to the base address. The storage reference is then made at the address formed by $m + (B^b)$.

Indirect Address

If the address field of an instruction contains an address (m), indirect addressing is indicated by making the special designator bit (a) equal to a 1. With this mode, a memory reference is made at the storage location indicated by the address field (m). From that storage location, the lower 18 bits are read and replace the lower 18 bits of the instruction. The function code of the instruction remains the same. If the new "a" designator is a one, indirect addressing is again accomplished and the lower 18 bits of the instruction are again replaced by these from storage. The use of indirect addressing will become obvious as you gain proficiency as a programmer. The flow chart (figure 4-11) illustrates how the computer obtains the execution address (M).

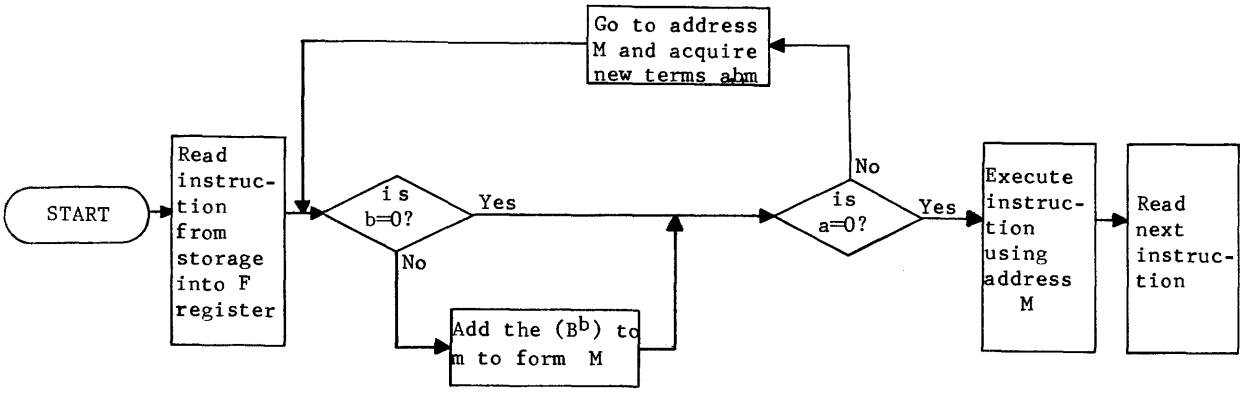
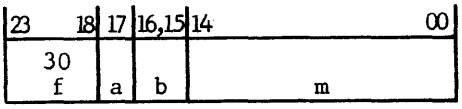


Figure 4-11. Formation of execution address (M)

The following examples illustrate how to derive the execution address (M) for the "Add to (A) register" instruction (function code 30). The same procedure would apply to any instruction that has "a" and "b" designators.

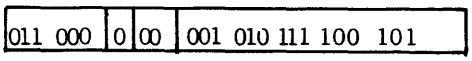


a = addressing mode designator (direct or indirect)

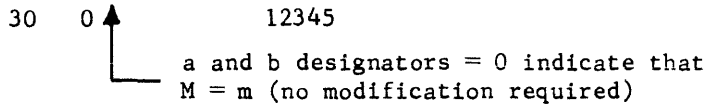
b = index register designator

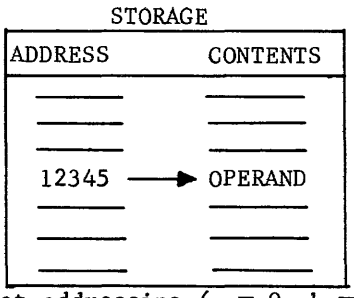
Instruction Description: Add the 24-bit operand from storage location M to the contents of the A2 register and form the sum in the A1 register

Example 1 Direct addressing (a and b = 0)

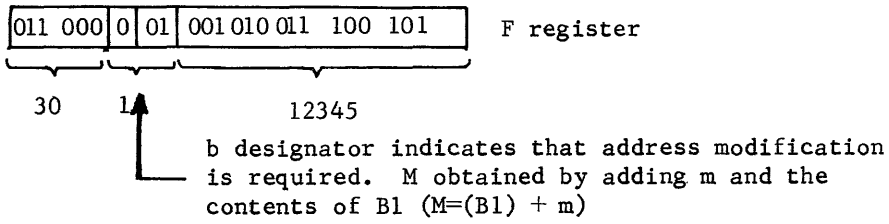


F register

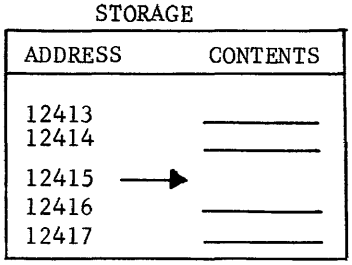




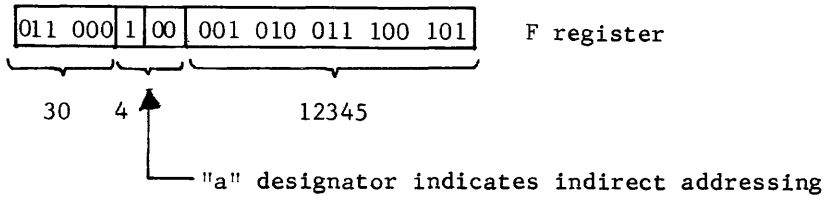
Example 2 Direct addressing (a = 0, b = 1)

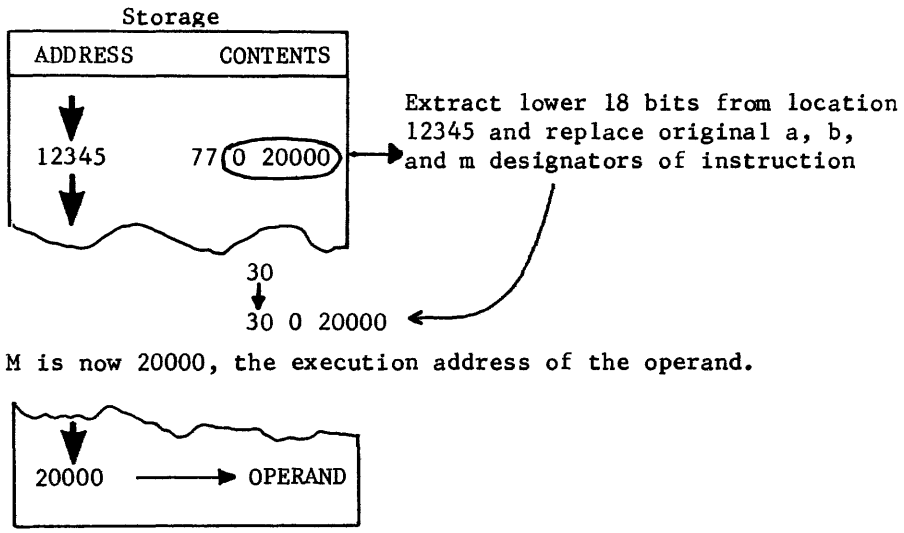


If B1 contains 00050, M = 12345 + 00050, which, in octal addition, is equal to storage address 12415.

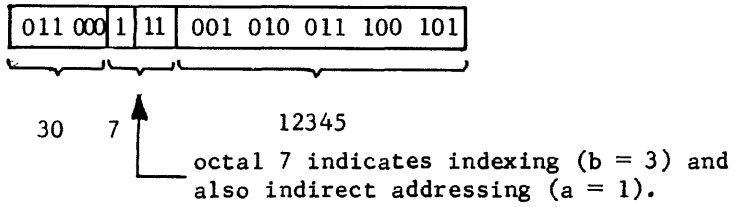


Example 3 Indirect addressing (a = 1, b = 0)



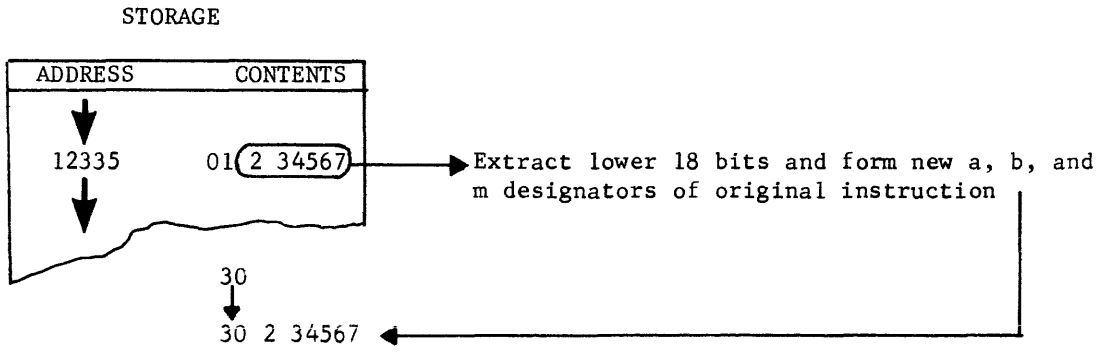


Example 4 Modified indirect addressing (a = 1, b = 3)

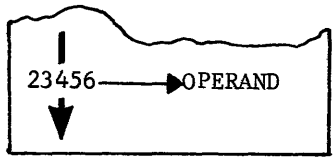


The indexing is performed first (figure 4-11). If (B3) is equal to 77767, m + (B3) = 12335: (12345 + 77767 = 12335).

The instruction now says to <u>indirect address</u> from storage address <u>12335</u>.



Instruction now indicates that indexing is again required. If (B2) is equal to 66666, M + (B3) = 23456: (34567 + 66666 = 23456). M is now 23456, the address of the operand.



At this point, it is important to realize that the program is being read from storage and each instruction also has a storage address. The program is normally located in a series of sequential storage locations followed by the required data (operands). Figure 4-12 illustrates this concept.

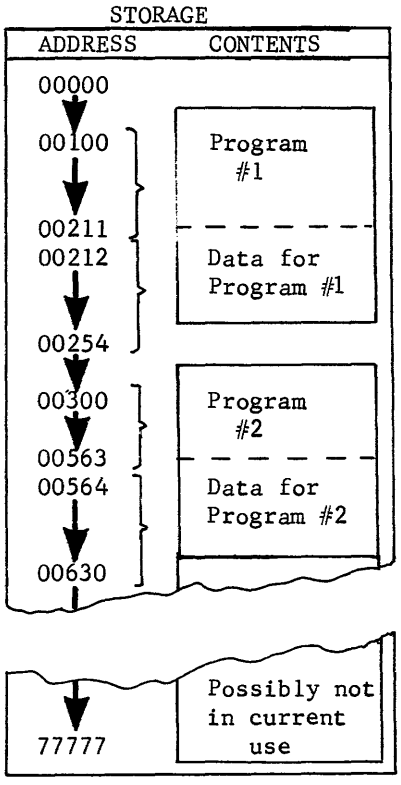


Figure 4-12. Programs and associated data in storage

Complete the following practice problems to ensure that you understand how to obtain the execution address if the instruction has "a" and "b" designators.

The answers are at the end of the chapter.

The Add instruction is at storage location 00010 for each problem and will be read from storage and placed in the F register. What would be the execution address (M) of the operand in each example?

4) 00010 30 0 22222

5) 00010 30 1 22222 (B1) = 12345

6) 00010 30 4 00077

↓

00077 50 0 00100

7) 00010 30 5 00123 (B1) = 12345

↓

00123 77 7 54321 (B2) = 33210

↓

12470 01 2 34567 (B3) = 43217

8) 00010 30 4 12345 (B1) = 00000

↓

12345 77 7 77770 (B2) = 00001

↓

77772 01 2 34567 (B3) = 00002

BASIC COMPUTER

Before analyzing the 25 instructions to determine their functions, it is necessary to analyze the basic computer block diagram. The basic computer can be divided into the four sections that are required of any computer system. Figure 4-13 illustrates the sections of the basic computer.

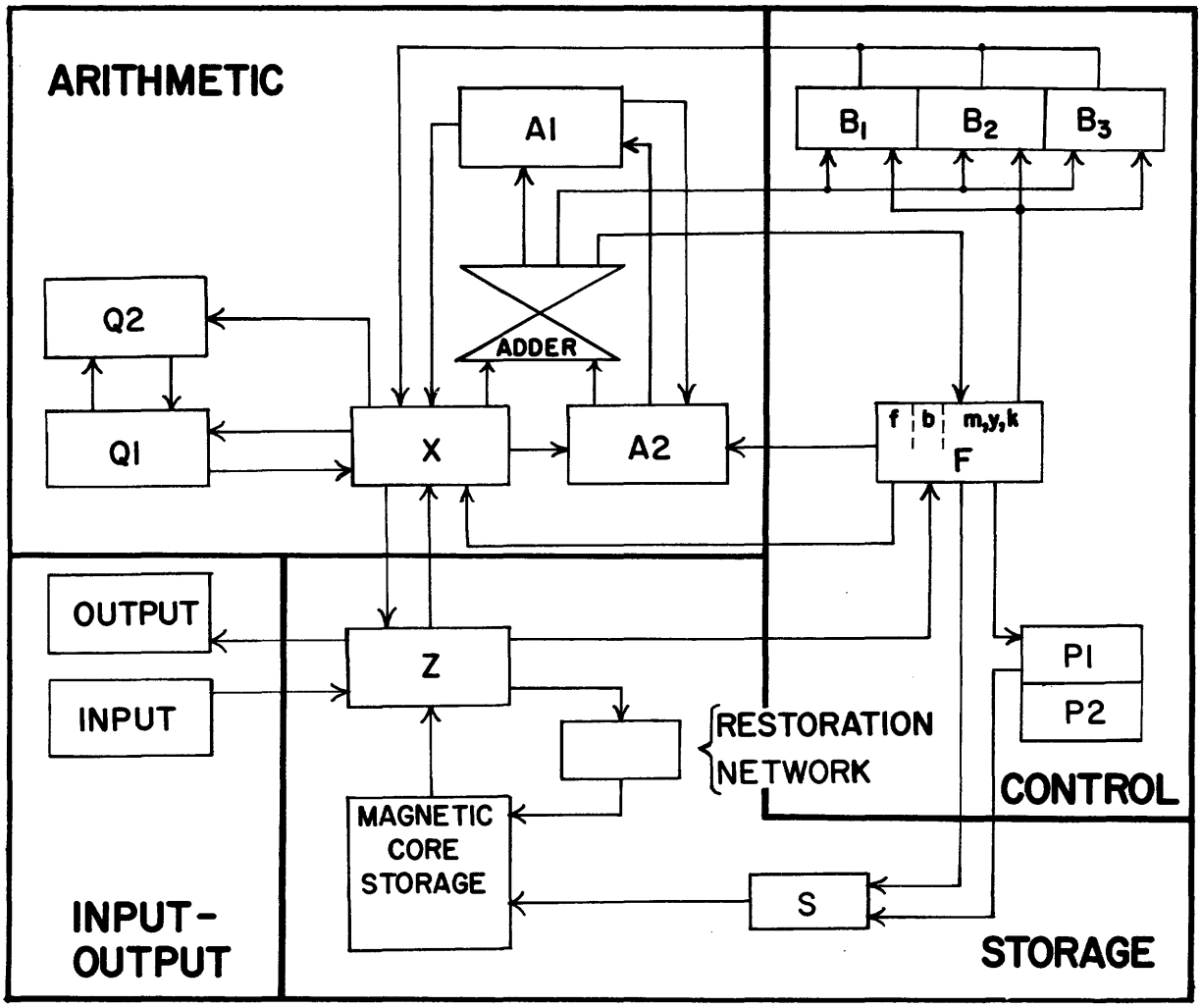


Figure 4-13. Basic Computer Sections

CONTROL SECTION

The control section directs the operations required to execute instructions and to exchange data with external equipment. It also establishes the timing relationships needed to perform the operations in the proper sequence.

The control section acquires an instruction from storage, interprets it, and sends the necessary commands to other sections to allow it to be executed. When that instruction has been executed, the control section acquires the next

instruction from storage and the execution process is repeated. Figure 4-14 illustrates the control section of the computer.

P Register

The P register is used to determine which instruction will be read from computer storage. Each storage location has a specific address which must be referenced before the contents of that memory location may be obtained. A programmer knows where his program is located in computer storage, usually a series of consecutive storage locations. The P1 and P2 registers together form a twos-complement counter that provides program continuity by generating, in sequence, the storage addresses which contain the individual instructions. Usually at the completion of each instruction, the count in P is advanced by one to specify the address of the next instruction.

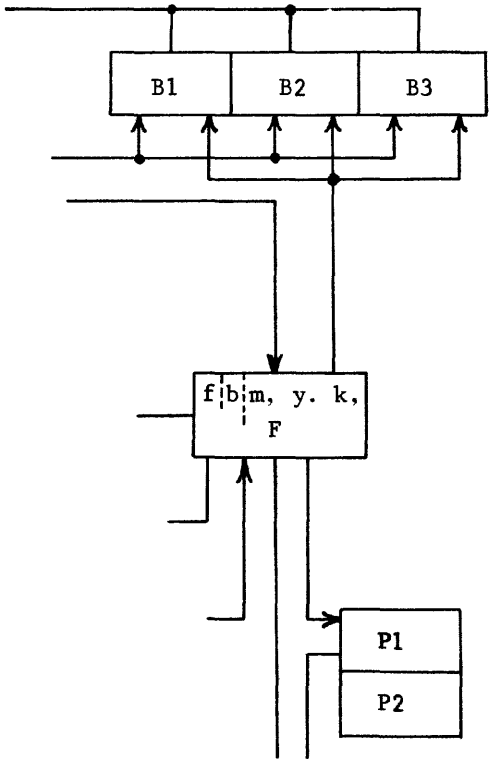


Figure 4-14. Control Section

F Register

The Function register (F), holds an instruction while it is executed. After executing an instruction, the computer performs an exit, jump exit, or skip exit. An exit advances the count in P by one and causes the computer to execute the instruction found at the storage location specified by the contents of P. A jump exit causes the computer to execute the instruction at the storage location specified by the execution address of the jump instruction. The execution address is, in this case, entered into P and specifies the starting location of a new sequence of instructions. A skip exit advances the count in P by two,

bypassing the next sequential instruction and executing the following one.

B Register

The index registers (B1-B3) are normally used to hold quantities to modify addresses. However, they may contain operands to be compared with other operands. All of the functions of the index registers will become evident as the instructions are discussed.

ARITHMETIC SECTION

The arithmetic section of the computer consists of two operational registers (A1 and Q1), and several secondary registers, as shown in figure 4-15.

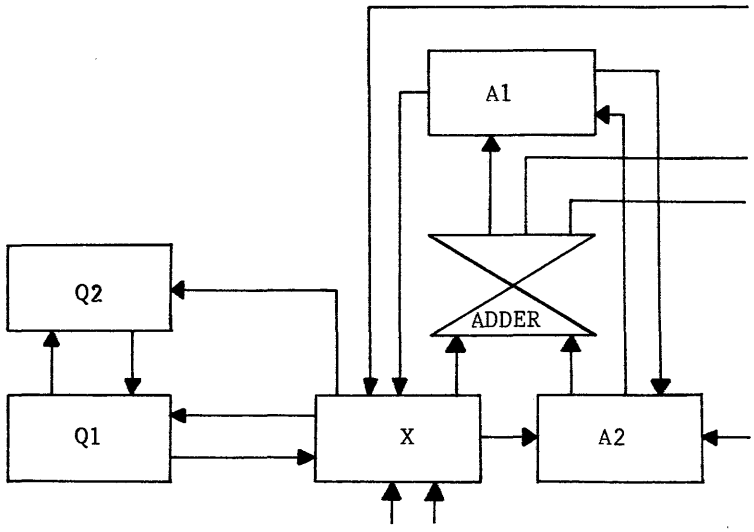


Figure 4-15. Arithmetic Section

A1 Register

The A1 register (accumulator) is the principal arithmetic register. Some of its more important functions are listed below:

1) All arithmetic operations use the A1 register in formulating a result. It is the only register with provisions for adding its contents and the contents of a storage location or another register.

2) The contents of the A registers may be shifted to the right or left. Right shifting is open-ended; the lowest bits are discarded and sign extended. Left shifting is circular; the highest-order bit appears in the lowest-order stage after each shift; all other bits move one place to the left. A1 and A2 are both required for shifting operations.

3) As a control for conditional instructions, "A" could hold the word which determines whether or not jump conditions are satisfied.

Q Register

The Q registers are auxiliary arithmetic registers and are generally used in conjunction with the A registers. The principal function of Q is to extend the A register during a multiply and a divide, thus providing a 48-bit register. Q1 contains the remainder after a divide and holds the most significant product digits after a multiply.

X Register

The X register is an exchange register used for a variety of functions, depending on the instruction being executed. For example, it contains the addend during an add operation, the subtrahend during a subtract, the multiplicand during a multiply, and the divisor during a divide. It also becomes part of the data flow path for several instructions.

Adder

The adder is used to form the sum, difference, quotient, and product during the execution of the arithmetic instructions. Address modification is also performed in the adder.

STORAGE SECTION

The magnetic core storage section of the computer (Figure 4-16) provides high-speed, random-access storage for 8192 words. It consists of two independent storage units, each with a capacity of 4096 words. They are called field 0 and field 1. These units operate together during the execution of a stored program and are considered one 8192 word storage system. The system can be expanded to a total of 32.768 (decimal) storage locations. The 15-bit address of an instruction would then indicate one of the 32,768 storage locations (modulus $2^{15} = 32{,}768_{10}$). Figure 4-16 shows the Contents of the Storage Section.

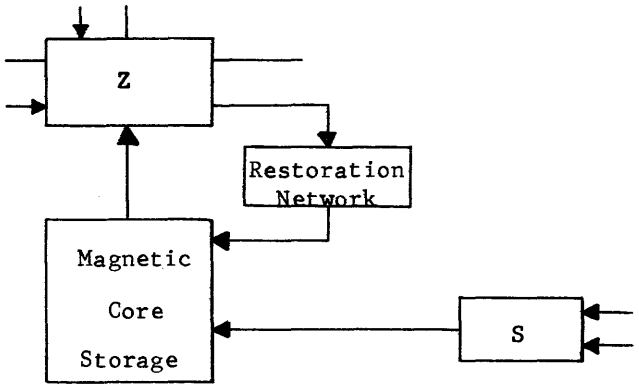


Figure 4-16. Storage Section

A word is 24 bits in length and is used as a 24-bit instruction or a 24-bit operand (data word). The location of each word in storage is identified by an assigned number or address. When a word is taken from (read) or entered into (written) storage, a reference is made to the storage address which holds the word.

The cycle time, or time for a complete storage reference, is 1.25 microseconds (usec).

S Register

The S register contains the address of the storage location currently being referenced. It may have received its address from the P register (instruction address) or from the address field of the F register (operand address).

Z Register

The proper storage location is then referenced and the 24-bit word is transferred to the Z register. Reading from memory destroys the contents of that storage location. However, the word is immediately rewritten by the restoration network into its original storage location. The Z register also sends the word to:

1) F if an instruction is being read.

2) X if an operand is being read.

3) The output circuits if an I/O operation is being performed.

INPUT-OUTPUT SECTION

The input-output section of the computer handles the flow of information to and from the computer. Prior to program execution, the data and instructions which comprise the program (input) are loaded into computer storage. After computation is complete the results (output) are transmitted from storage to an external

equipment.

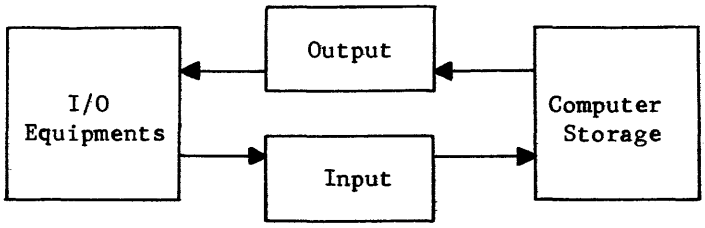


Figure 4-17. Input-Output Section

The computer communicates with external equipments through eight independent buffer channels which provide the normal exchange of data.

The input and output buffer channels are paired: channels 0 and 1, channels 2 and 3, channels 4 and 5, and channels 6 and 7. Each external equipment is connected to one of these pairs. It is possible to connect as many as eight different equipments to any given channel. All eight buffer channels may concurrently transmit information. Only one equipment, however, can use any one buffer channel at any given instant.

In the computer, input-output operations are independent of the main computer program. When data is transmitted, the main computer program initiates an automatic cycle which buffers data to and from computer storage. The main computer program then continues while the actual buffering of data is carried out independently and automatically.

The process of asynchronous input-output operations is termed a buffer. Buffer transmissions employ independent access to computer storage so that computation continues while the external equipment is loading information into, or extracting information from, computer storage. The rate of exchange is, in most cases, dictated by the external equipment.

The function of each section of the basic computer and the components that form each section have been briefly explained. You have learned that instructions are read in sequence from storage into the F register where each is translated. You also learned the basic instruction format and the instruction designators. With that knowledge, you can now proceed and examine the function of each of the 24 instructions. Some new designators will be used to explain the function of each instruction. The following is a complete list of the designators, including those previously discussed.

a	Addressing mode designator (bit 2^{17} of indirect addressable instructions)
A1	A1 register (accumulator)
$A1^n$	The binary digit in position n of the A1 register

→	Transmit to
b	Index designator
B^b	Designated index register
d	Shift designator bit (2^{17} of shift instructions)
Exit	Proceed to next instruction
f	6-bit function code (bits 2^{18}-2^{23} of instruction)
j	Condition designator for jump and stop instructions
Jump Exit	For next instruction, jump to address M
k	Unmodified shift count (bits 2^{00}----2^{14} of shift instructions)
K	Modified shift count
m	Unmodified operand address (bits ^{00}----2^{14} of storage reference instructions)
M	Modified operand address (M = (B^b) + m)
()	Indicates "The contents of" a register or a storage location
Q1	Auxiliary arithmetic register
RNI	Read next instruction
Skip Exit	Skip one instruction and read the second one next
y	Unmodified operand (bits 2^{00}----2^{14}, if instruction contains operand)
Y	Modified operand
///////	Fill with octal zeros

INSTRUCTION ANALYSIS

The instructions to be analyzed are included in the following Table 4-4. The mnemonic codes will become useful during the study of assembler language programming in Section II of this chapter. Try to associate the mnemonic code, function code, and instruction name as you study each instruction. The table also contains the page number where the explanation of each instruction may be found.

Table 4-4. INSTRUCTION LIST

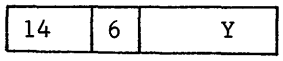
Mnemonic code	Function code	Instruction name	Page
ADA	30.	Add to A	4-42
AZJ	03. (0-3)	Compare (A) with zero	4-49
DVA	51.	Divide AQ by (M)	4-45
ENA	14.6	Enter A with y	4-35
ENA,S	14.4	Enter A with y (sign extended)	4-35
ENI	14.(1-3)	Enter index	4-37
ENQ	14.7	Enter Q with y	4-36
ENQ,S	14.5	Enter Q with y (sign extended)	4-36
HLT	00.0	Unconditional stop; RNI from m upon restart	4-57
INI	15.(1-3)	Increase index	4-54
ISD	10.(5-7)	Index skip (decremental)	4-56
ISI	10.(1-3)	Index skip (incremental)	4-54
LDA	20.	Load A	4-37
LDQ	21.	Load Q	4-39
MUA	50.	Multiply (M) by (A), product in QA	4-43
RTJ	00.7	Return Jump	4-51
SBA	31.	Subtract (M) from (A)	4-43
SHA	12.(0-3)	Shift (A) right or left	4-39
SHAQ	13.(0-3)	Shift (AQ) right or left	4-41
SHQ	12.(4-7)	Shift (Q) right or left	4-40
SJI-SJ6	00.j	Selective jump	4-50

Mnemonic code	Function code	Instruction name	Page
SLS	77.70	Selective stop; RNI at P + 1 upon restart	4-57
STA	40.	Store A	4-47
STQ	41.	Store Q	4-48
UJP	01.	Unconditional Jump	4-50

INSTRUCTION FUNCTIONAL ANALYSIS

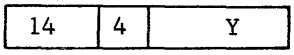
Interregister Transfer Instructions

ENA Enter A with y



Clear the A register and enter the 15-bit operand from the address field of the F register. The largest operand possible with this instruction is 00077777_8 (2^{15}). The A register cannot be entered with a negative operand with the ENA instruction.

ENA,S Enter A with y (sign extended)

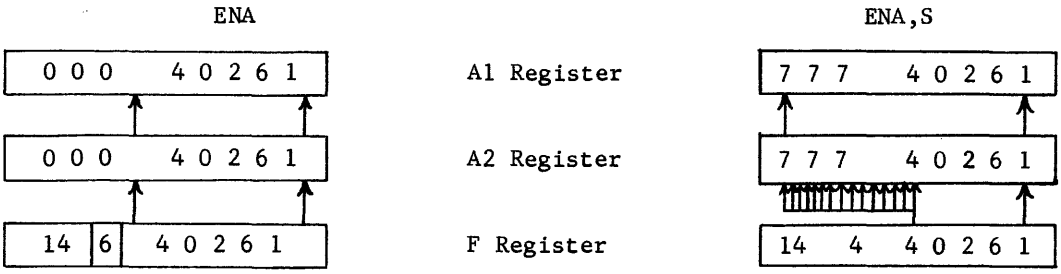


This instruction is similar to the ENA instruction except that the sign bit of the 15-bit operand is extended throughout the upper 9 bits of the A register. The largest operand (signed) that can be entered into the A register using this instruction is $\pm 37777_8$. However, both positive and negative operands may be used (modulus $2^{15}-1$).

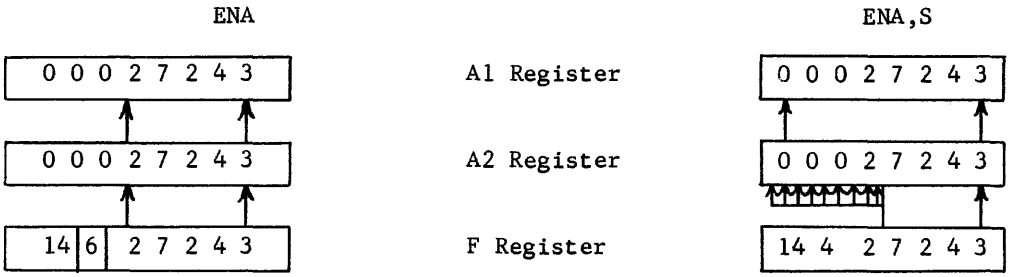
NOTE:

The A1 and A2 registers are normally equalized by frequent transfer pulses. Both registers are entered with the operand during the execution of the ENA and ENA,S instructions.

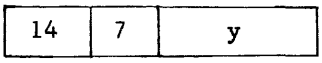
Example #1



Example #2

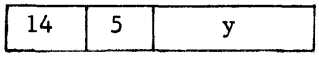


ENQ Enter Q with y

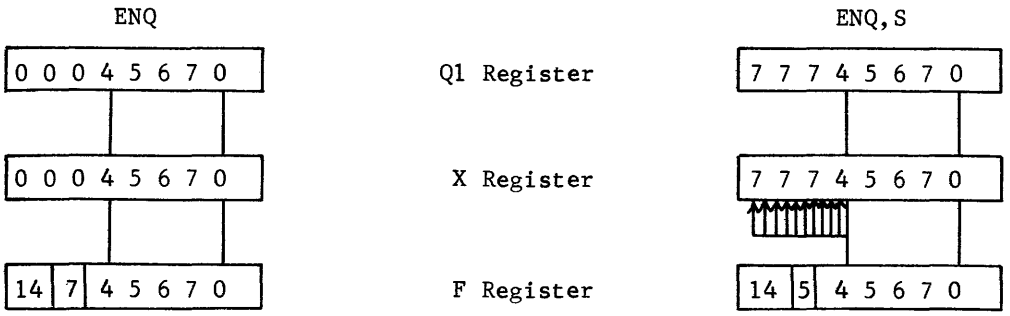


Clear the Q register and enter with the 15-bit operand y. Similar to ENA except that the Q register is involved.

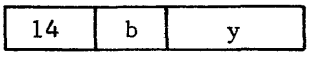
ENQ,S Enter Q with y (sign extended)



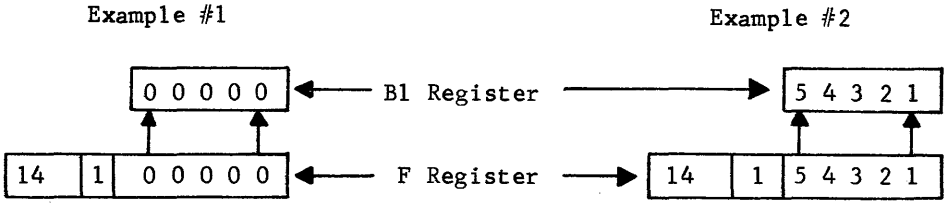
Similar to ENA,S except that the Q register is involved.



ENI Enter index with y

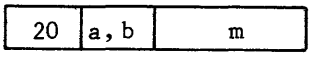


Replaces (B^b) with operand y (a = 0, b = 1-3). If b = 0, the instruction becomes a "do nothing".



Load Instructions

LDA Load A



a = addressing mode designator
b = index register designator
m = storage address; M = m + (B^b)

The contents of the A1 and A2 registers are replaced by the 24-bit operand from the storage location specified by M. The contents of the storage location remain unchanged.

Address modification to obtain M is accomplished in the adder. The contents of the specified index register are transferred to the X register and m to the A2 register; the sum is formed in the adder and transferred back to the F register as M.

M is then transferred from the F register to the S register and storage reference is made to obtain the operand. Figure 4-18 illustrates the flow paths in sequence for a 20 3 12345 instruction.

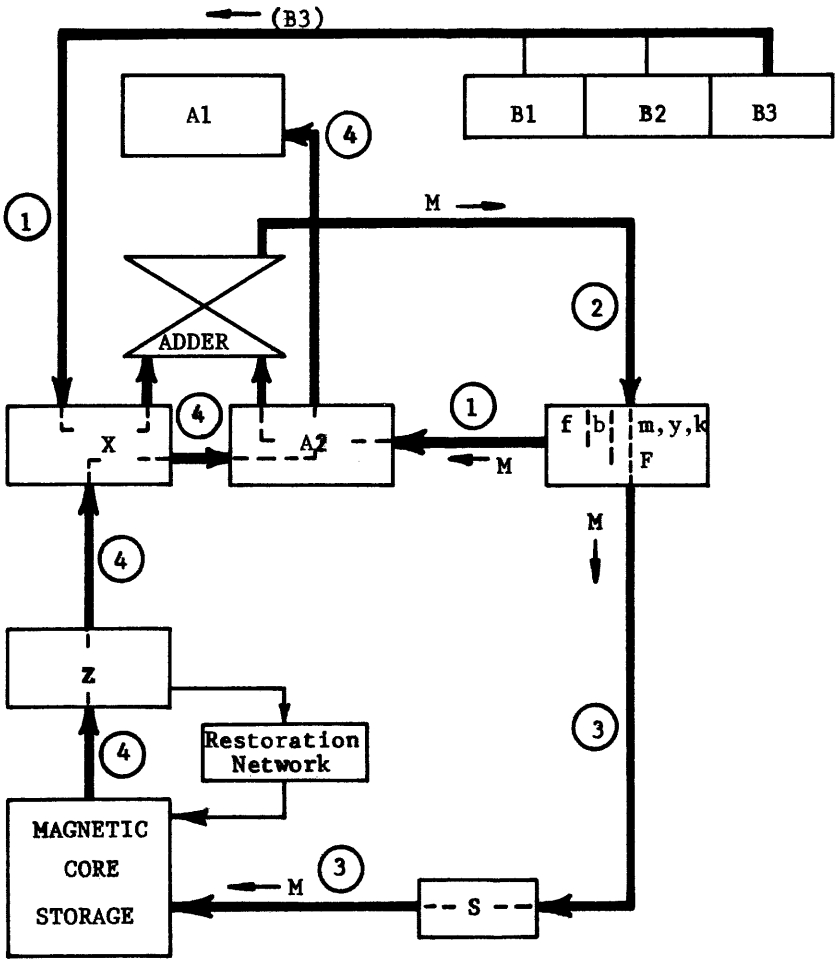


Figure 4-18. Load A Instruction

1) (B3)→X, m→A2

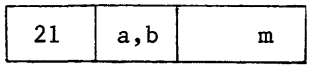
2) M→F address field (M=(B^b) + m)

3) M→S→Magnetic Core Storage

4) Operand→Z→X→A2→A1

If indirect addressing had been specified (a = 1), step #4 of Figure 4-18 would transfer 18 bits from storage → Z → F. The instruction would again be translated with its new a, b, and m designators to determine whether or not index modification and/or indirect addressing is required. When M is finally determined, storage would be referenced at location M and the operand from this location transferred to the A registers.

LDQ Load Q



The load Q instruction is similar to the Load A instruction except that the operand is transferred from X → Q1 instead of from X → A2 → A1. Q2 is not affected by the LDQ instruction (this register is used mainly by any instruction that requires shifting in Q--SHQ, SHAQ, MUA, DVA).

Shift Instructions

The format for the shift instructions is quite unique and different than those discussed previously. Figure 4-19 illustrates the format used by all shift instructions.

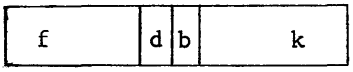


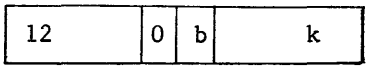
Figure 4-19. Shift Instruction Format

The "d" designator (bit 2^{17}) indicates which register is to be shifted. If d = 0, shift (A) register; if d = 1, shift (Q) register.

The "b" designator represents one of the index registers. The shift count, k, may be modified to yield K (K = (B^b) + k).

The "k" in the address field represents the shift count. A positive shift count indicates a left shift whereas a negative shift count indicates a right shift. All left shifts are end around (bit 2^{23} → bit 2^{00}) and all right shifts are end-off with sign bit extension.

SHA Shift A



d = 0

The Shift A instruction shifts the (A) register right or left, depending on the sign bit of the modified shift count, K. The d designator bit must be a "0" for the SHA instruction. Shifting is accomplished in parallel between the A1 and A2 registers. Figure 4-20 illustrates how the (A) register is left-shifted one place by a 12 0 00001 instruction.

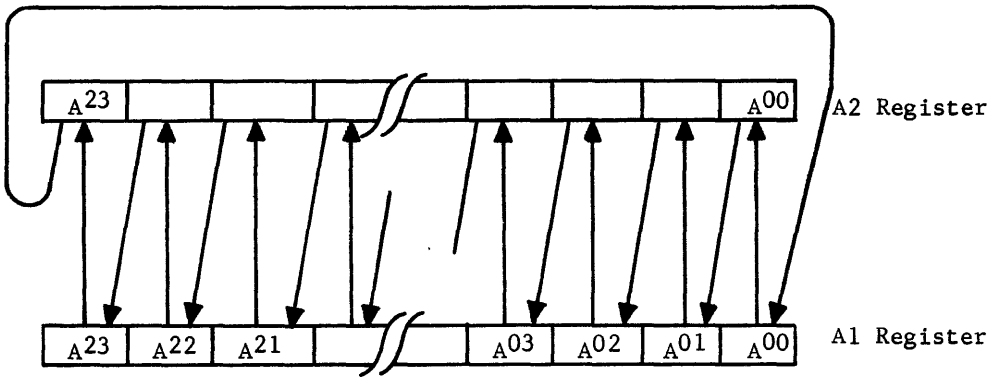


Figure 4-20. Shift A Left

If the instruction had a negative shift count, the shift would be to the right. Figure 4-21 illustrates the result of a 12 0 77776 instruction.

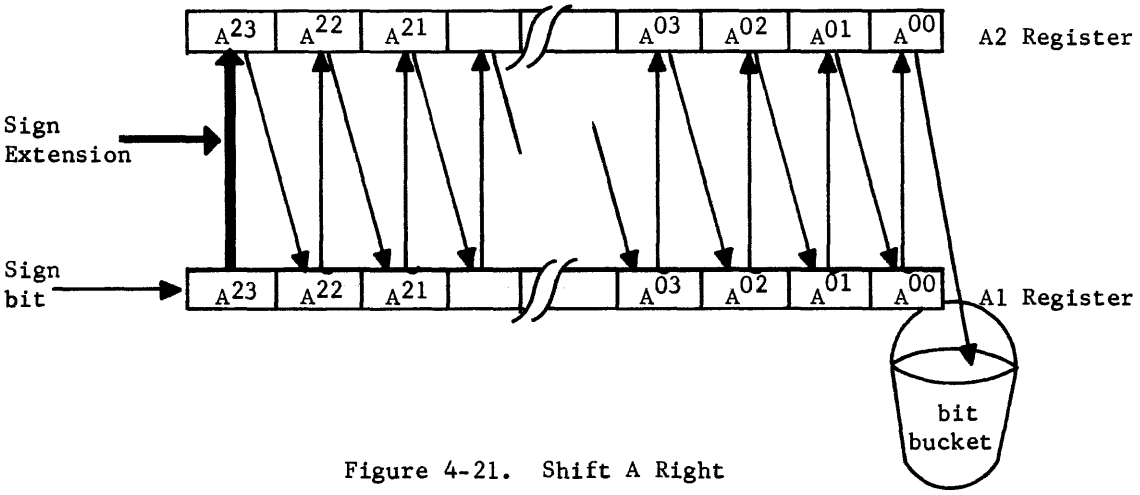


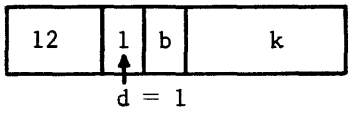
Figure 4-21. Shift A Right

1) Shift right A2 ⟶ A1--A1 contains (A2) right shifted one position.

2) Equalize registers A1 ⟶ A2 (A2 = A1)--extends sign by $A1^{23}$ transfer.

3) If the shift count had been 77775 instead of 77776, steps 1 and 2 would be repeated.

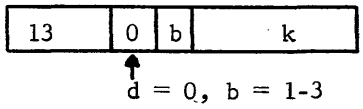
SHQ Shift Q



The Shift Q instruction is essentailly the same as the shift A except that the (Q) register contents are shifted instead of (A). The instruction format is the

same except that the d Designator is a "1" for the Shift Q. The largest practical shift count would be 00027_8 for left shifts, and 77751_8 for right shifts.

SHAQ	Shift AQ



The Shift AQ instruction shifts both the A and Q registers as one 48-bit register. A shift count (k) of 00030_8 would interchange the contents of the two registers (left shift). A shift count of 77747_8 would shift (A) into Q. The A register would contain a 24-bit extension of the sign bit; the original (Q) would be lost. Figure 4-22 illustrates the AQ left shift and the AQ right shift procedures.

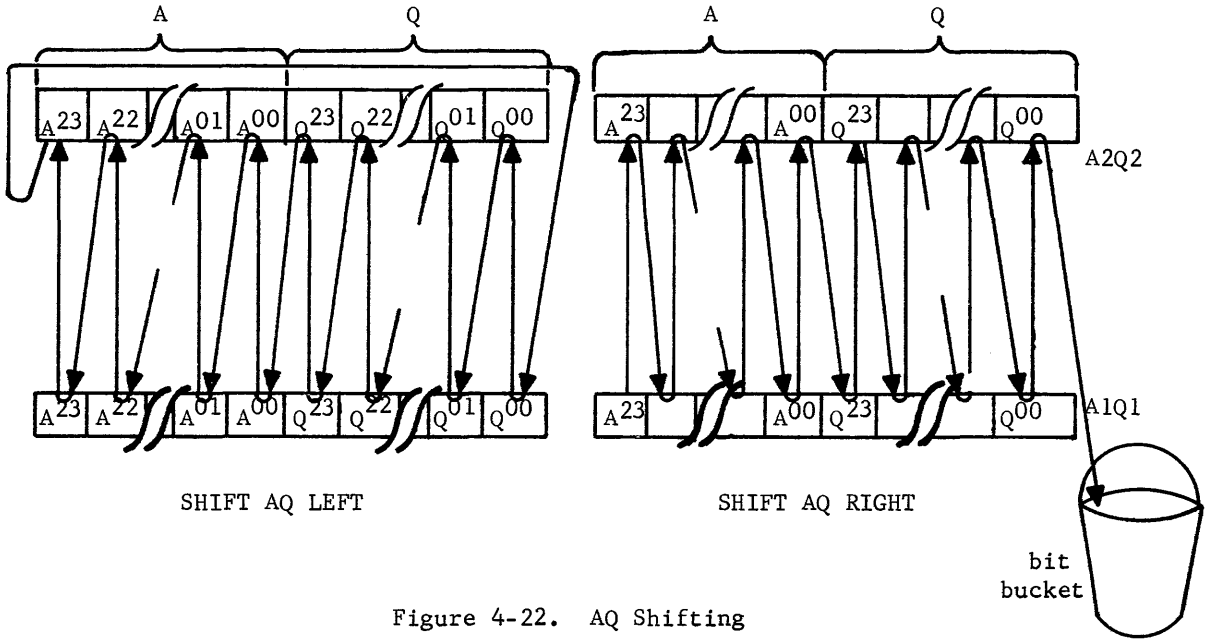


Figure 4-22. AQ Shifting

Shift AQ left

1) bit 2^{23} of Q2 is shifted into bit 2^{00} of A1

2) bit 2^{23} of A2 is shifted end around into bit 2^{00} of Q1

Shift AQ right

1) bit 2^{00} of A2 is shifted into bit 2^{23} of Q1

2) bit 2^{00} of Q2 is shifted off and lost

3) sign bit is extended in A

What would each of the following instructions accomplish? For all examples, assume that:

(B1) = 00007 (A) = 00 00 5000

(B2) = 00014 (Q) = 40 00 0000

(B3) = 77770

9) 12 0 00015 K = ____? (A) final = 2000 0001 ?

10) 12 3 00004 K = ____? (A) final = 0000 0500 ?

11) 12 6 77766 K = ____? (A) final = 0000 5000 ? (Q) final = 0000 0004 ?

12) 12 7 00001 K = ____? (A) final = 0000 5000 ? (Q) final = 7740 0000 ?

13) 13 0 00014 K = ____? (A) final = 5000 4000 ? (Q) final = 0000 0000 ?

14) 13 3 00003 K = ____? (A) final = 0000 0240 ? (Q) final = 0200 0000 ?

Check your solutions with those at the end of the chapter.

Arithmetic Instructions

1. All four arithmetic instructions use full-word (24-bit) operands.

2. All modes of address modification (a and/or b) apply to these instructions.

3. One storage reference is made for each instruction unless indirect addressing is designated (a = "1")-- if designated, at least two storage references are required (one to obtain new a, b, m designators, another to obtain operand). Contents of storage location (M) are unchanged.

4. If the modulus (2^{24}-1) of the A register is exceeded during the execution of the ADA of the SBA instructions, an arithmetic overflow fault is produced (reference overflow section of Chapter III). If the modulus of A is exceeded during the execution of the DVA instruction, a divide fault is produced.

ADA Add to (A)

30	a,b	m

The ADA instruction obtains an operand from storage location M and adds it to the (A) register. The sum is formed by the adder and transferred to both A1 and A2. Direct and/or indirect addressing may be used with the ADA instruction. The original operand in the A registers is replaced by the sum of A and (M); the contents of storage location M are not changed.

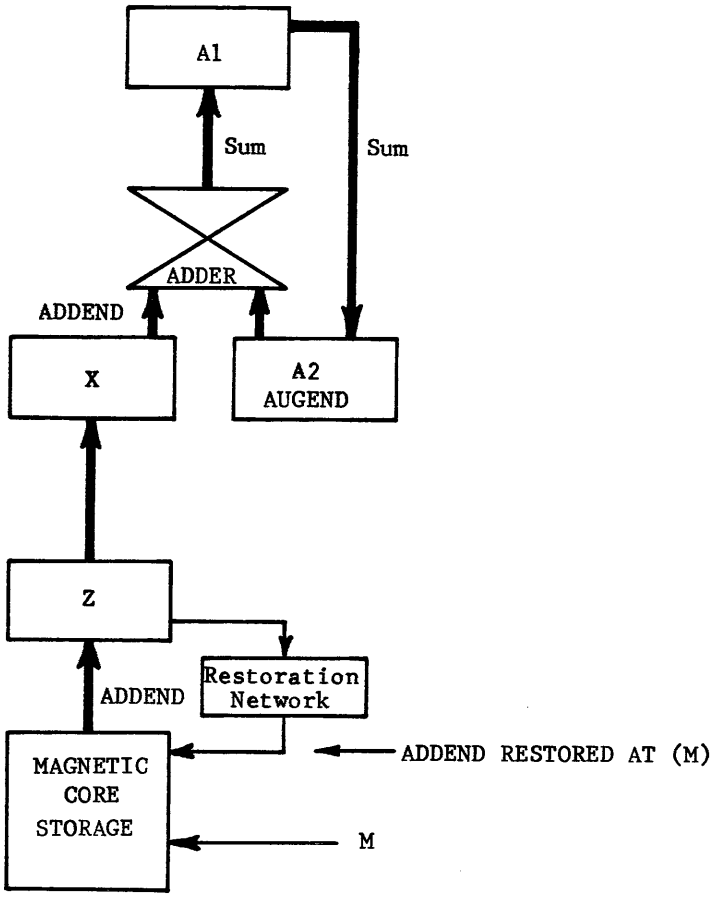


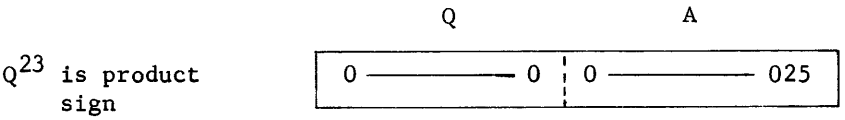
Figure 4-23. ADA Instruction data flow paths

SBA	Subtract from A	31	a,b	m

This instruction obtains the subtrahend from storage at address M and subtracts from the minuend contained in the A registers. The difference is formed in the adder and transferred into the A1 and A2 registers. The flow paths are exactly the same as for the ADA instruction; however, the ones complement of the subtrahend is sent to the adder by the X register and addition is performed (subtraction can be accomplished by complementing the subtrahend and adding).

MUA	Multiply A	50	a,b	m

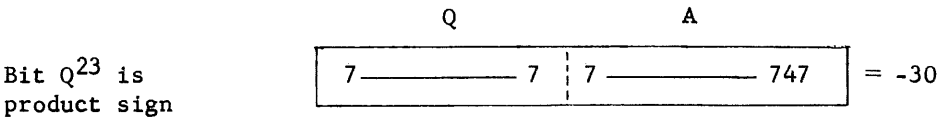
The MUA instruction forms the 48-bit product of the multiplicand from storage location M and the multiplier contained in the A register (both 24-bit operands). The 48-bit product is contained in the QA registers with the most significant bit in Q^{23} and the least significant bit in A^{00}. The actual multiplication is performed via a series of add and shift operations initiated by the MUA instruction. For example, assume (A) = 000 00003 and storage location 00100 contains 00 0 00007. The instruction 50 0 00100 would obtain the multiplicand (000 00007) from storage location 00100 (M) and multiply by (A). The final product would be equal to 25_8 and would appear in the QA registers as:



If the two operands (multiplicand and multiplier) have unlike signs, the product will be negative and will appear in complemented format. For example, multiply 6 by -4 with the instruction 50 0 00100

(00100) = 000 00006 = 6
(A) = 777 77773 = -4
-30_8

the final product would appear in QA as:



Assume that the multiplicand and multiplier are in storage at addresses 00500 and 00501. A simple program to form the product could be written as follows.

00100	20 0 00500	Load A with multiplier from location 00500
00101	50 0 00501	Multiply (00501) by (A)
00102	HALT	
00500	Multiplier	
00501	Multiplicand	

The program and data are in storage. Master Clear the computer (M/C), set the P register = 00100 (address of first instruction) and press the GO button on the console. The first instruction loads the multiplier into the A register, the second instruction performs the multiplication, and the third instruction stops the computer.

DVA Divide A

51	a,b	m

This instruction divides the 48-bit dividend contained in AQ by the 24-bit divisor from storage location M. The quotient is found in A and the remainder is in Q. The uppermost bit of A (A^{23}) determines the sign of the entire dividend. If the dividend and divisor signs are unlike, the quotient in A will be negative. The remainder always has the same sign as the dividend. For example:

A divide fault will occur if the quotient exceeds the modulus of the A register. For example, the division of a positive dividend by a positive divisor should yield a positive answer. If the 48-bit dividend 0000 0002 0000 0000_8 is divided by the 24-bit divisor 0000 0004_8, the quotient in the A register would be 4000 0000_8. In complement notation, that quotient would represent the negative quantity 3777 7777_8. The divide fault was a result of using a dividend too large with respect to the divisor. If the dividend is reduced by one to 0000 0001 7777 7777_8 and divided by the same divisor (0000 0004_8), the answer would be correctly expressed as 3777 7777_8, the largest positive operand that could appear in the 24-bit A register.

NOTE:

If a divide instruction follows a multiply, the contents of the A and Q registers must be interchanged. A multiply forms the product in QA whereas a divide instruction assumes the dividend to be in AQ. Therefore, (A) and (Q) must be interchanged. This can be accomplished by the Shift AQ instruction (SHAQ).

The following program employs some of the instructions explained to this point. Work the program and list the effect of each instruction. Remember, the computer reads the instructions, one at a time in sequence, unless "instructed" to do otherwise.

15) Master Clear, Set P = 00100, and press Go. Beside each instruction, list its function and its effect in the program.

```
00100    14 4 00050

00101    14 7 77777

00102    14 3 12345

00103    14 2 11225
```

```
00104       14 1 34567

00105       30 2 66666

00106       50 4 00120

00107       13 1 43240

00110       51 0 00116

00111       12 4 00025

00112       31 4 00121

00113       HALT (to be explained)

00114       00 0 00045

00115       77 7 77774

00116       77 7 77771

00117       77 7 77677

00120       00 0 00115

00121       00 0 00117
```

The computer is stopped by the HALT instruction at address 00113. The register contents are displayed on the console. What should be displayed by each of the following registers when the computer stops?

P =

F =

B1 =

B2 =

B3 =

A =

Q =

If you do not agree with the answers at the end of the chapter, review addressing modes and any troublesome instructions.

Store Instructions

A store instruction is used to write (record) an operand in storage at address M. Indirect addressing and address modification may be used.

STA	Store A

40	a,b	m

This instruction copies the (A) register into storage location M. Original contents of storage location M are destroyed. The contents of A are now held in storage and also retained in A. Figure 4-24 illustrates the data flow paths for the Store A instruction.

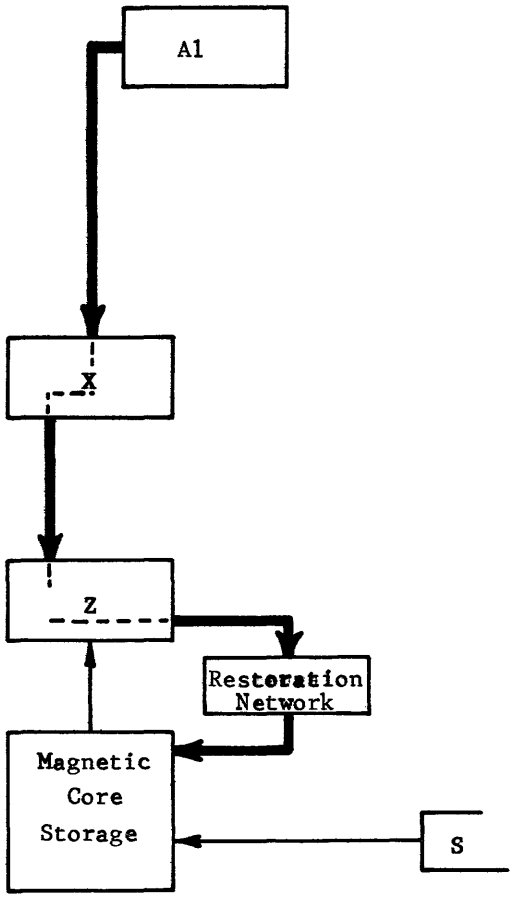


Figure 4-24. Store A

STQ **Store Q**

41	a,b	m

The Store Q instruction copies the contents of the Q register into storage at location M. A copy of the operand is now contained in storage and still retained in the Q register.

What would the following program accomplish?

16) M/C, set P = 00100, Go

```
00100    20 0 00112
00101    30 0 00113
00102    30 0 00114
00103    30 0 00115
00104    14 5 00000
00105    13 0 00030
00106    51 0 00116
00107    40 0 00117
00110    41 0 00120
00111    HALT
00112    0000 0020
00113    7777 7677
00114    0000 0037
00115    0000 0031
00116    0000 0004
```

17) If storage location 00113 contains 77777577, the preceding program would not produce the correct result. Why? Work the program to see if you can decide why the answer and remainder would be wrong before you check the solution.

18) How could you modify the program to always achieve the correct results, regardless of which operands are used?

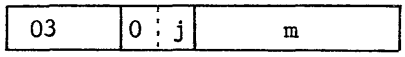
19) What would be required to prepare the Q register for division if the Divide instruction follows a multiply?

Jump Instructions

A jump instruction allows the current program sequences to terminate and initiates a new sequence at the jump address. The programmer may transfer control to any point in his program by jumping directly to the desired instruction.

The jump may be automatic or may be conditioned by the contents of a register or switches on the console. If the jump conditions are not satisfied, the next sequential instruction is executed.

AZJ "A" zero jump



d = 0

j designates a condition

m = M = jump address

The AZJ Instruction compares the contents of the A register with zero for greater than, equal to, or less than conditions. The j designator determines which comparison is being made. If the tested condition is met, a jump is made to M.

Condition Mnemonic	Jump designator j	Test Condition
AZJ,EQ	0	(A) = 0 (EQual to zero)
AZJ,NE	1	(A) ≠ 0 (Not Equal to zero)
AZJ,GE	2	(A) 0 (Greater than or Equal to zero)
AZJ,LT	3	(A) 0 (Less Than zero)

Positive zero (00000000) and negative zero (77777777) are considered equal to zero for the AZJ,EQ and the AZJ,NE instructions. The AZJ,GE and AZJ,LT instructions examine only the sign of the A register; therefore, negative zero is considered to be less than positive zero.

Indirect addressing and address modification are not possible with the AZJ instruction.

Determine if a jump will be made under each of the following conditions.

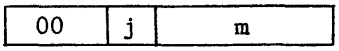
20) sign of A negative, AZJ,EQ instruction

21) (A) = 77777776, AZJ,NE instruction

22) (A) = 40000000, AZJ, GE instruction

23) (A) = 77777777, AZJ,LT instruction

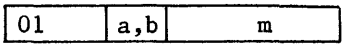
SJ1-6 Selective Jump



j = 1-6

The SJ1-6 Instruction causes a jump to m if the console jump key (specified by j) is set. For example, the instruction 00 5 12345 would cause a jump to the instruction at storage address 12345 only if jump key 5 is set. The condition of the other jump keys is insignificant for the SJ5 instruction.

UJP Unconditional Jump



a = addressing mode designator

b = index register designator

m = unmodified jump address ($M=(B^b)+m$)

When the UJP Instruction is executed, a jump is always made to M. Indirect addressing and address modification may both be used.

In the following program, what would be the jump address (M) of the instruction at storage location 00102? M/C, set P = 00100, Go

24) 00100 14 3 33000

00101 14 2 77776

00102 01 6 00105

00103 12 3 45670

00104 56 7 45102

00105 77 7 45670

The normal program sequences permit execution of a consecutive list of instructions previously stored in computer storage. The P register always contains the address of the current instruction and is merely updated by one to read the next consecutive instruction. If a jump instruction causes a program jump, the jump address is forced into the P register and then to the S register. Figure 4-25

illustrates how the jump address conditions the P and S registers to obtain the instruction from the jump address.

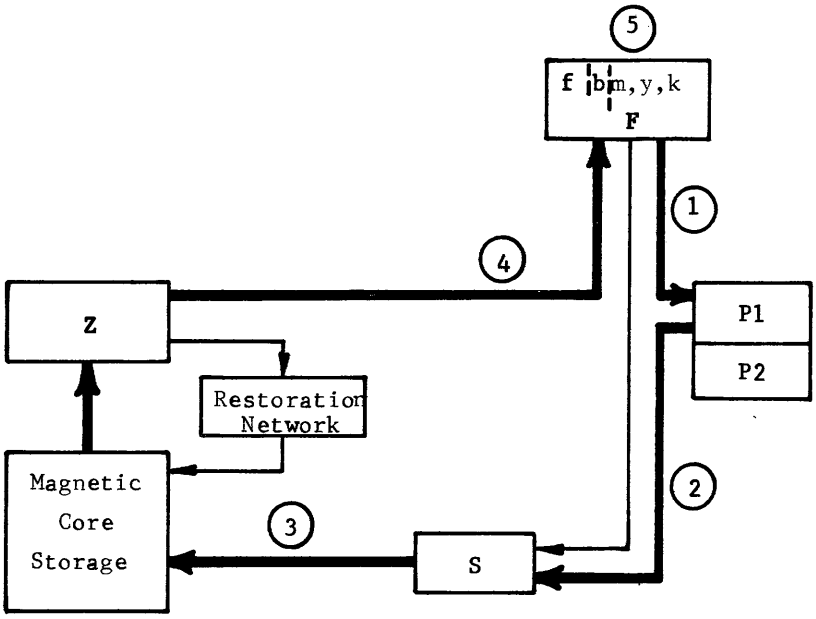


Figure 4-25. Jumping to a New Instruction

1) Determine jump address and force P register to new address

2) Transfer jump address to S register

3) Initiate storage at jump address

4) Read instruction from jump address and Transfer to F register

5) Execute new instruction

NOTE:

The direct line from F to S is to reference storage for operands. Program continuity would be lost if P was forced to the operand address.

RTJ Return Jump

00	7	m

The Return Jump Instruction provides the means of jumping out of the main program to a subprogram and returning back to the next sequential instruction in the

main program. Frequently-used routines, such as trigonometric functions, are not rewritten for every new program. Instead, the standard routines are retained and, if a program requires them, entered into computer storage as a subprogram.

Assume that a program in storage calls for a cosine function. Instead of using the cosine function as a part of the main program, the program return jumps to the cosine subprogram and records the return address of the main program (P)+1. The subprogram derives the desired cosine function and then jumps back to the first instruction of the subprogram where the return address to the main program is recorded. That instruction provides the address of the next main program instruction.

Figure 4-26 illustrates the function of the Return Jump instruction.

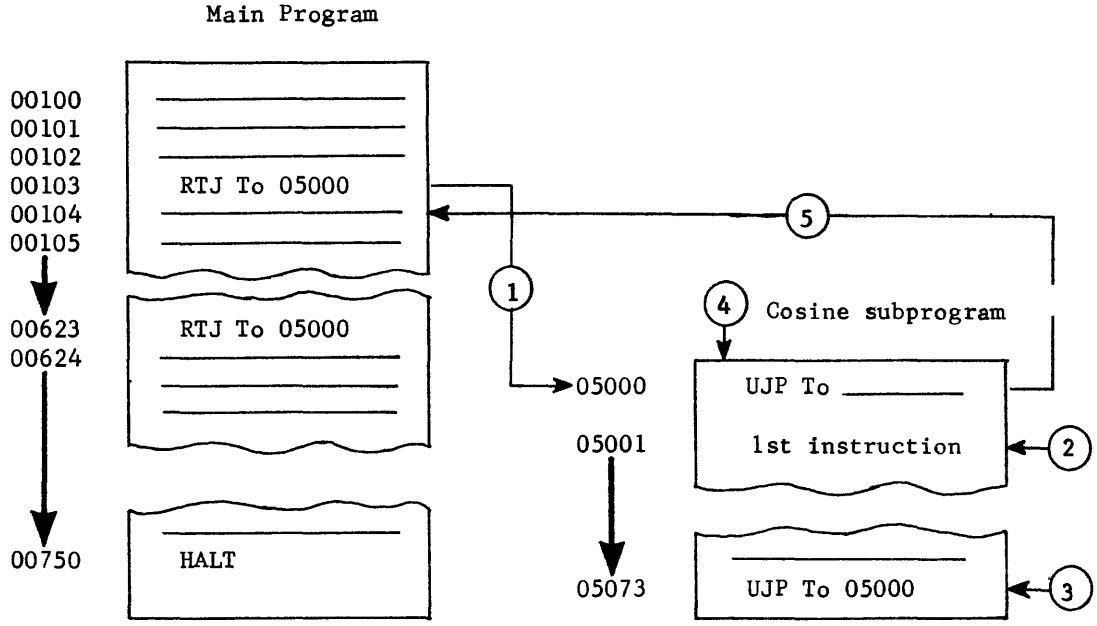


Figure 4-26. Return Jump

1) Jump to address 05000 (subprogram) and write (P) + 1 (00104) in address field.

2) First instruction to be executed is at address 05001.

3) Last instruction of subprogram (address 05073) is a jump back to start of subprogram.

4) First instruction of subprogram (address 05000) provides for automatic return to proper step of main program.

5) Main program continues at address 00104.

Storage location 00623 also contains a Return Jump to the cosine subprogram. What address would be written in the lower 15 bits of location 05000 this time?

The first and last instructions of the subprogram must contain unconditional jumps. The address field of the first instruction is automatically conditioned by the return jump instruction (records (P) + 1) but the programmer must place the proper jump address in the last instruction.

The following program is an exercise using the four jump instructions. Work the program and answer the questions that follow. M/C, set P = 01000, set Jump Key 1, Go.

```
01000        14 6 77777

01001        03 3 01010

01002        00 1 01004

01003        01 0 01010

01004        30 0 01011

01005        00 7 01012

01006        03 2 01005

01007        01 0 01003

01010        HALT

01011        37 7 00001

01012        01 0 77777

01013        01 0 01012
```

25) Draw lines on the preceding program to indicate the sequence in which the instructions were executed.

26) Did you find a subprogram? How many instructions did it contain?

27) How many instructions were executed by the program? Count each instruction each time it is used. Were any instructions executed more than once?

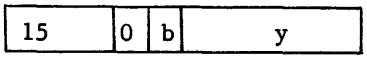
28) Which storage locations contained operands?

If you correctly answered the questions about the jump program, you can JUMP with joy to the next group. If not, JUMP back and review the jump instructions again.

Increment, Decrement, Increase Index Registers

The index registers normally contain address modifiers or operands to establish parametric limits. This group of instructions permits arithmetic operations on the modifier and operands.

INI Increase Index



d = 0

b = 1-3

If b = 0, the INI instruction becomes a "pass" or "do-nothing" instruction. The signs of the 15-bit operands (y and (B^b)) are extended and the (B^b) is increased by y in the adder. The lower 15 bits of the 24-bit sum is transferred to the designated index register. Figure 4-27 illustrates the function of the Increase Index instruction.

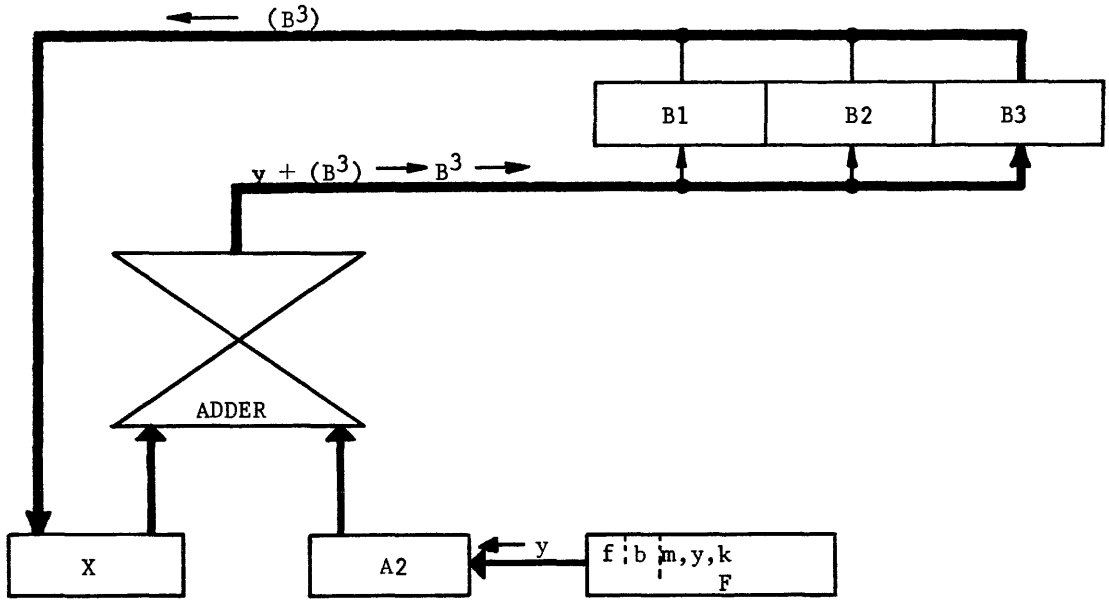
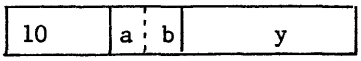


Figure 4-27. Increase Index

ISI Index Skip, Incremental



d = 0

b = index register under comparison

y = comparison operand

The ISI instruction compares the (B^b) with the operand y. If equal, B^b is cleared (all zeros) and program control skips one instruction to P + 2. If unequal, (B^b) is incremented by one in the adder and the instruction at P + 1 is executed.

The ISI instruction is frequently used as a counter to control the number of passes made through a program. The flow chart of the commission problem (Figure 4-4) contained several decisions, one of which asked: Is count = 10? That decision would undoubtedly be made by the ISI instruction.

The following program illustrates how a similar problem could appear in machine language. The program is designed to find the sum of six operands contained in storage at locations 00200 through 00205.

M/C, set P = 00100, Go

```
00100      20 0 00200
00101      30 1 00201
00102      10 1 00004  <------ Pass counter
00103      01 0 00101
00104      HALT
```

1st pass

1) Load A with first operand (00200) and add second operand from 00201 to it (M = 00201 because B1 was cleared by M/C).

2) Compare (B1) with y. Not equal so increment (B1) to 00001 and exit to P + 1.

3) Jump back to Add instruction.

2nd pass

1) Add (00202) to (A). The A register now contains the sum of the first three operands.

2) Compare, increment (B1) to 00002, and exit to P + 1 (jump back to add).

3rd pass

1) Add (00203) to (A). Sum of first four operands now in A.

2) Compare B1 (now 00002) with y (00004). Not equalities so increment (B1) to 00003 and exit to P + 1 (jumps back to add).

4th pass

1) Add (00204) to (A). Sum of first five operands now in A.

2) Compare B1 (now 00003) with y (00004). Still not equalities so increment (B1) to 00004 and exit to P + 1 (jumps back to add).

5th pass

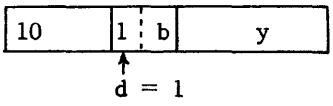
1) Add last operand (00205) to subtotal in A register.

2) Compare B1 (now 00004) with y (00004). They are equal. Clear B1 to all zeros and skip exit to P + 2, passing over the jump instruction that caused the loop.

3) Instruction at P + 2 (00104) halts the program and leaves sum of operands in A register.

The preceding program could be used to add any number of operands. How would you modify the program to add a list of 100_8 operands?

ISD Index Skip Decremental



b = index register under comparison

y = comparison operand

The ISD instruction is a variation of the ISI instruction and would have similar applications in a program. his instruction decrements the contents of the index register designated by b of the instruction. The reverse counter could be used to add the same six operands discussed with the ISI instruction.

For example:

```
00100    14 1 00004
00101    20 0 00205
00102    30 1 00200
00103    10 5 00000
00104    01 0 00102
00105    HALT
```

This program starts with the last operand in the list and adds the list in reverse. One more instruction is required than in the previous program because the index register must be initially conditioned.

Problem #29

Rewrite the preceding problem and eliminate the ENI instruction. Modify the remaining instructions as necessary to provide the correct number of passes to add six operands using theISD instruction. Check your solution.

Program Termination Instructions

A digital computer executes instructions in sequence and accomplishes only what that instruction specifies. The computer does not tire, does not require coffee breaks, and doesn't even stop for lunch--unless instructed to do so.

Indirect addressing and address modification may not be used with these instructions.

SLS Selective Stop

77	7	0/////////

lower 12 address bits are not used but are normally all zeros.

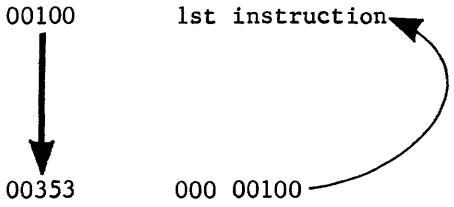
The selective Stop instruction is conditioned upon the setting of the console Stop switch (Figure 4-6). Program executions terminate when the SLS instruction is executed if the stop switch is set. If not set, the instruction becomes a pass or do-nothing.

If the program is terminated by the SLS instruction, pressing the GO switch on the console re-initiates the program at P + 1.

HLT Halt

00	0	m

This instruction unconditionally halts the program. The address m designates the next instruction to be executed if the computer is restarted. For example:



Restart causes jump back to 1st instruction and program is re-executed.

Compare the SLS and HLT instructions

SLS	HLT
stops only if switch is set, reads next instruction (RNI) from P + 1	stops unconditionally RNI at m

The HLT instruction is normally the last instruction of a program and is also the one that terminates this section. With the machine language instructions discussed, you should be able to write some useful programs.

Answer the following review questions before continuing on to Assembler Language Programming. The answers are at the end of the chapter.

30) What is an instruction?

31) How many instructions are in a 24-bit word?

32) The operation code (f) is held in the upper 6 bits of what register?

33) What is the chief difference between an ENA (14.6) and a LDA (20)?

34) What happens to the original contents of the register used on a Load instruction

35) What is the chief difference between a Load instruction and a Store instruction?

36) $A_{(i)} = 00002170$, $(B^1) = 00007$. After executing a 12 1 77760 instruction, what will (A_f) equal?

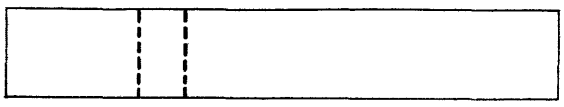
37) (A) = 77777113. What will (A) equal after a 14 6 00003 instruction? A 14 4 00003 instruction?

38) Computer words are 24 bits in length. These words can be either of two things:

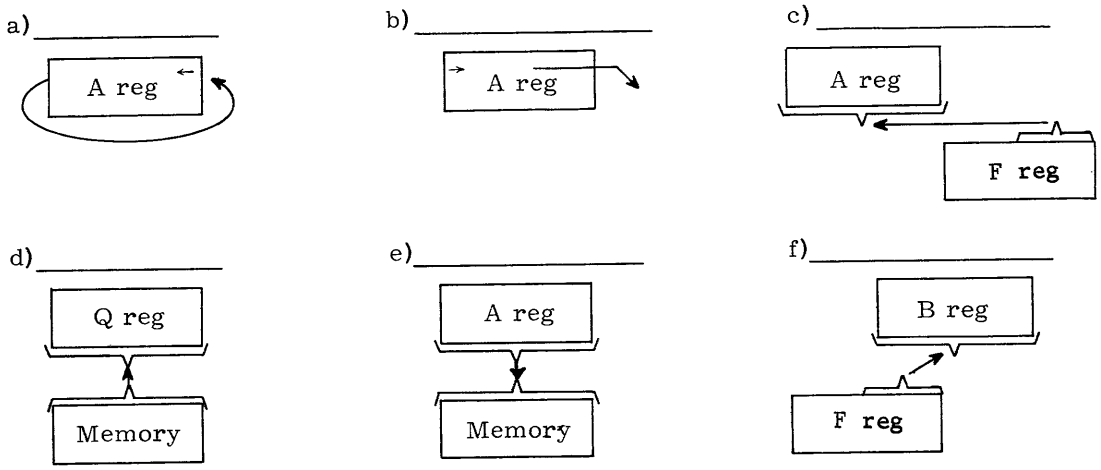
1) _____ = telling the computer what to do.

2) _____ = what the computer uses to solve problems.

39) In the rectangle provided below, fill in the different portions of the computer instruction and the number of bits in each area.



40) Which instructions are graphically represented by the following diagrams?



41) Draw the graphical representation of the following instructions.

a) 01

b) 51

c) 30

d) 31

42) M/C, set P = 02000, GO. What is (A) final?

```
02000    14 4 00360
02001    00 0 02000
```

43) M/C, set P = 03010, GO. What is (A) final?

```
03010    14 6 77760
03011    00 0 03010
```

44) M/C, Set P = 02500, GO. What is (A) final?

```
02500     14 4 02502
02501     00 0 02500
02502     01 2 34567
```

45) M/C, set P = 13000, GO. What is (A) final and (Q) final?

```
13000     14 4 13003
13001     21 0 13003
13002     00 0 13000
13003     00 0 00002
```

46) M/C, set P = 66000, GO. What is (A) final and (Q) final?

```
66000     21 0 66003
66001     14 4 66004
66002     00 0 66000
66003     76 5 43210
66004     00 1 23456
```

47) M/C, set P = 14000, GO. What is (A) final and (Q) final?

```
14000     14 4 00002
14001     21 0 14004
14002     30 0 14004
14003     00 0 14000
14004     00 0 00004
```

48) M/C, set P = 01000, GO. What is (A) final?

```
01000     14 4 06030
01001     12 0 77766
01002     00 0 01000
```

49) M/C, set P = 03000, GO. What is the final contents of the A, Q, and P Registers?

```
03000     20 0 03006
03001     21 0 03007
03002     12 0 00003
03003     13 0 77767
03004     40 0 03005
03005     10 0 00000
03006     30 0 37001
03007     00 0 77776
```

50) M/C, set P = 15000, GO. What is (A) final?

```
15000     14 4 00007
15001     30 0 15004
15002     31 0 15005
15003     00 0 15000
15004     00 0 00011
15005     00 0 00014
```

51) M/C, set P = 11110, GO. What is (A) and (Q) final?

```
11110     14 4 77774
11111     50 0 11120
11112     13 0 00030
11113     51 0 11117
11114     00 0 11110
11115
11116
11117     77 7 77775
11120     77 7 77664
```

a) When multiplying, use the standard arithmetic rules. Fill in the blanks.

1) A + times a + give a _____answer

2) A + times a - gives a _____answer

3) A - times a + gives a _____answer

4) A - times a - gives a _____answer

b) Similar rules exist for division. Complete the samples.

1) A + into a + gives a _____answer and a _____remainder

2) A + into a - gives a _____answer and a _____remainder

3) A - into a - gives a _____answer and a _____remainder

4) A - into a + gives a _____answer and a _____remainder

c) Do your arithmetic in OCTAL.

52) M/C, set P = 11000, GO. What is (A) final and (P) final?

```
11000    14 4 77777
11001    03 0 11003
11002    00 0 11000
11003    30 0 11012
11004    03 1 11006
11005    00 0 11000
11006    31 0 11013
11007    03 2 11011
11010    00 0 11000
11011    00 0 11000
11012    00 0 00001
11013    00 0 00002
```

53) M/C, set P = 06000, GO. What are the final contents of the A, P, and B1 registers?

```
06000    14 4 01010
06001    30 1 07000
```

```
06002     10 1 00003
06003     01 0 06001
06004     00 0 06000
07000     00 0 00670
07001     77 7 77517
07002     00 0 06673
07003     00 0 07213
07004     77 7 77657
```

54) M/C, set P = 07000, GO. List the final contents of

A
Q
F
P
B1

```
07000     10 1 00006
07001     20 1 07012
07002     30 0 07013
07003     40 1 07012
07004     21 1 07015
07005     13 0 00030
07006     51 0 07015
07007     03 3 07012
07010     01 0 07000
07011     00 0 07000
07012     00 0 07000
07013     00 0 00004
07014     76 0 10103
07015     77 7 77774
07016     00 0 00000
```

55) M/C, set P = 08006, GO. What is (A) final and (P) final?

```
08006     14 4 40000
```

```
08007     03 2 08009

08008     00 0 08006

08009     00 0 08006
```

56) M/C, set P = 10000, GO. List the contents of all operational registers when the computer stops.

```
10000     14 6 00201

10001     12 0 00017

10002     12 0 77767

10003     40 0 10013

10004     14 6 00000

10005     30 1 11000

10006     10 1 00003

10007     03 2 10014

10010     50 0 11003

10011     13 0 00030

10012     51 0 11004

10013     00 0 00000

10014     01 0 10005

11000     00 0 00567

11001     00 0 01346

11002     77 7 76541

11003     00 0 00012

11004     77 7 77774
```

57) M/C, set P = 12000, GO. List the final contents of the A, Q, P, and F registers.

```
12000     14 4 77777

12001     03 3 12007

12002     00 0 12000
```

12003 01 0 12000

12004 20 0 12003

12005 21 0 12001

12006 01 0 12002

12007 00 7 12003

58) M/C, set P = 12000, GO. What is (A) final?

12000 20 4 12002

12001 00 0 12000

12002 00 0 60732

59) M/C, set P = 07000, GO. What is (A) final?

07000 20 4 07000

07001 00 0 07000

60) M/C, set P = 13000, GO. Where does the computer stop?

13000 00 7 13013

13001 01 0 13017

13002 00 0 21000

13003 01 0 13013

13004 01 0 13003

13005 00 0 14000

13006 00 0 15000

13007 01 0 13003

13010 14 0 13017

13011 00 7 13003

13012 00 7 13004

13013 01 0 13004

13014 01 0 13007

13015 14 9 13002

13016 01 0 13011

13017 01 0 13015

13020 00 0 13000

What are the final contents of storage locations 13004, 13011, and 13013?

Now that you have had the opportunity to work a few practice problems, try to write a machine language program to solve the following problem.

PROBLEM

A drove of turkeys and sheep have 99 heads and feet, there are twice as many turkeys as there are sheep. When the computer stops, the number of sheep should appear in the Q register (in octal, of course), and the number of turkeys should be stored at location 00200.

Remember the steps to consider when writing a program.

1. Determine and state the complete program.
2. Analyze the problem, determine what must be done.
3. Sequence the problem with a flow chart.
4. Code the steps into machine language.

Rule 1. Determine and state

The problem is to determine the number of each animal. The number of turkeys is a key figure because there are twice as many turkeys as sheep. The problem is stated using decimal operands. Convert the operands to octal to be compatable with the computer's language.

$$99_{10} \text{ heads and feet} = 143_8 \text{ heads and feet.}$$

Rule 2 Analyze

Derive a formula that could be used to algebraically solve the problem.

a. Twice as many turkeys as sheep

b. Two turkeys would have two heads and four feet (6X)

c. Each sheep has one head and four feet (5X)

d. Each possibility would have 11 heads and feet

$$\text{turkeys} = 2 \times \text{sheep} \quad (T = 2S)$$

$$\therefore T \times 3 + S \times 5 = 143_8$$

Rule 3 Plan the problem

A man would work the problem algebraically; the computer can rapidly furnish the random number of turkeys by using sequential numbers beginning with the number 1. Let's hope none of the sheep have only three legs and that there are no two-headed turkeys. Construct a flow chart before attempting to code the problem.

Rule 4 Code the problem in an acceptable language (machine language).

Write your program in machine language on the following form. It is referred to as an "Absolute Coding Form" and is typical of the many forms suitable for machine language programming. The form was specifically chosen to be compatable with the adopted instruction format.

STORAGE LOCATION		INSTRUCTION f a,b m, y, k	COMMENTS
	0		
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	0		
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	0		
	1		
	2		
	3		
	4		
	5		
	6		
	7		
	0		
	1		
	2		
	3		
	4		

The flow chart and the coding of the problem have been left as a challenge to you. The program, including data, can be written in less than 30 program steps. When you have completed your flow chart and program, you may check your solutions with those at the end of the chapter (#61,62). The program solution has been included only as a guide and is not necessarily the most efficient way to achieve the desired result. Make sure that your program allows for the possibility of not finding a solution and will stop after a given number of passes.

The program you have just written must now be entered into the computer. This must be accomplished manually from the console (enter mode) for a program written in machine language.

PART II

ASSEMBLER LANGUAGE PROGRAMMING

Now that you have learned machine language programming, you can appreciate the difficulties presented with that type.

There must be an easier way and, by ADA, there is (ADA could be a girl's name or, it could be the mnemonic code for an "Add to A" instruction).

To begin with, an Assembler is not a black box; it is a program consisting of a series of translatory routines. The purpose of the routines is to make computer programming easier by using symbols for operations and names of locations. Numbers are used if necessary for constants but no reference is given to numbered memory locations.

This method of programming is so named because the computer is expected to do some "assembline" (the changing of the written language into machine code before the program can be executed). The following is a comparison of machine and assembly languages.

Machine Code	Assembly Code
20 0 XXXXX	LDA COSINE
20 is the code for loading the A register with the contents of the memory location specified by the address xxxxx.	LDA is the assembly language (mnemonic) code which means load the A register with the contents of the memory location identified by the name Cosine.
	NOTE: Attach a location name that is meaningful and one that defines the operation.

The assembly language has several advantages over machine coding. The mnemonic letters and tags indicate which action is to take place, whereas the machine code is only a list of octal digits with little or no meaning outside the mind

of the programmer. There is also another advantage of the assembly code; the programmer need not be concerned with addresses of the instructions themselves. The assembler (the program in the computer which changes the assembly code to machine code) has the responsibility of assigning addresses.

The following diagram shows the order of operations of a program written in assembly language.

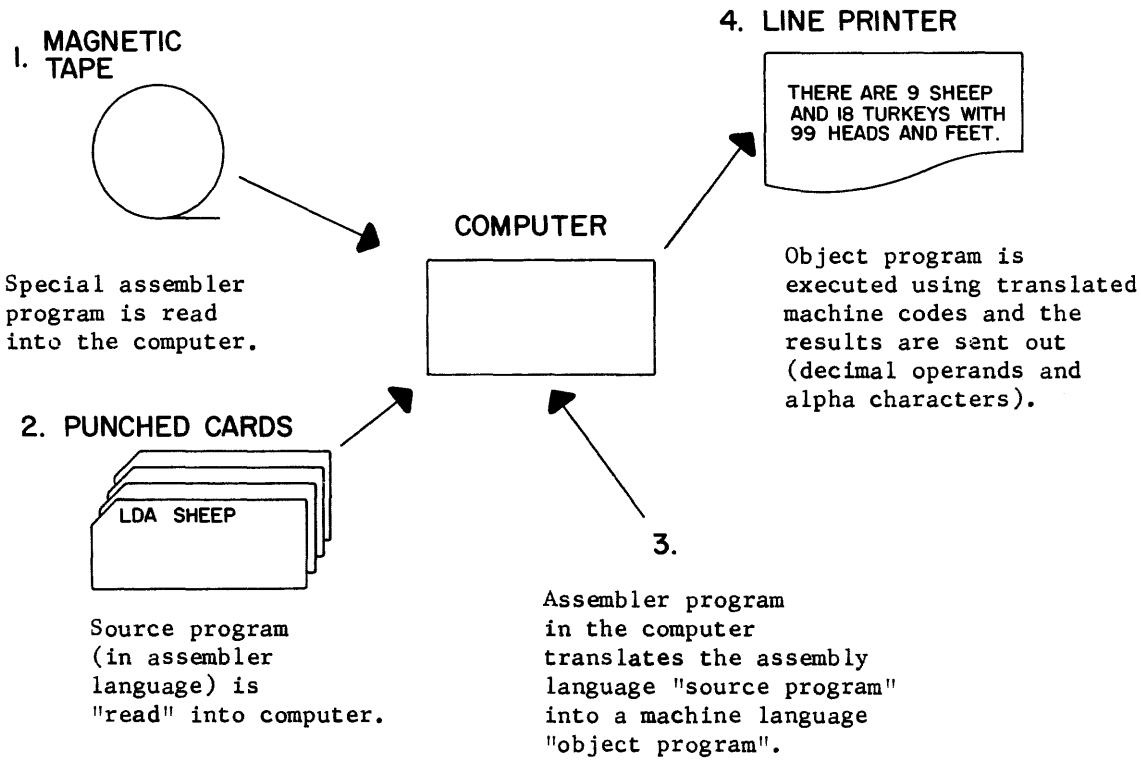


Figure 4-28. Order of Operations

The program is generated by following the same steps used for machine language programming.

1) Determine and precisely state the problem.

2) Analyze the problem.

3) List the solution to the problem in a series of logical steps (flow chart).

4) Code the logical steps into an acceptable language.

One type of computer language highly acceptable and in extensive use is the assembler language. Each type of computer system has its own instruction format

and repertoire. Therefore, each type of system must have its own assembler to translate the mnemonic codes into its language. The assembler may be identified as

CODAP	(COntrol Data Assembly Program),
CAP	(Computer Assembly Program),
COMPASS	(COMputer ASsembly System),
ASCENT	(Assembly System for CENTral processor),
ASPER	(Assembly System for PERipheral processor) or by some other appropriate name.

The COMPASS assembler is compatable with the instruction format and repertoire discussed in the machine language section of this chapter. Therefore, the following discussion will be directed toward the COMPASS assembler and programs will be known as COMPASS programs. Although coding methods for other assemblers may vary slightly, the procedures are essentially the same.

The assembly program is written using the mnemonic codes to identify the desired operation. The following table is an extract from Table 4-4 and designates the mnemonic code for each instruction.

TABLE 4-5. MNEMONIC INSTRUCTION CODES

Mnemonic code	Machine code	Instruction Name	Mnemonic code	Machine code	Instruction Name
ADA	30.	Add	LDQ	21.	Load Q
AZJ	03.(0-3)	A zero jump	MUA	50.	Multiply
DVA	51.	Divide	RTJ	00.7	Return Jump
ENA	14.6	Enter A	SBA	31.	Subtract
ENA,A	14.4	Enter A	SHA	12.(0-3)	Shift A
ENI	14.3	Enter index	SHAQ	13.(0-3)	Shift AQ
ENQ	14.7	Enter Q	SHQ	12.(4-7)	Shift Q
ENQ,S	14.5	Enter Q (extended)	SJI-6	00.j	Selective Jump
HLT	00.0	Halt	SLA	77.7	Selective Stop
INI	15.(1-3)	Increase index	STA	40.	Store A
ISD	10.(5-7)	Index skip (decremental)	STQ	41.	Store Q
ISI	10.(1-3)	Index skip (incremental)	UJP	01.	Unconditional Jump
LDA	20	Load A			

CODING FORM

The appropriate program steps are recorded sequentially on a coding form. Figure 4-29 illustrates the coding form used to record COMPASS assembler programs.

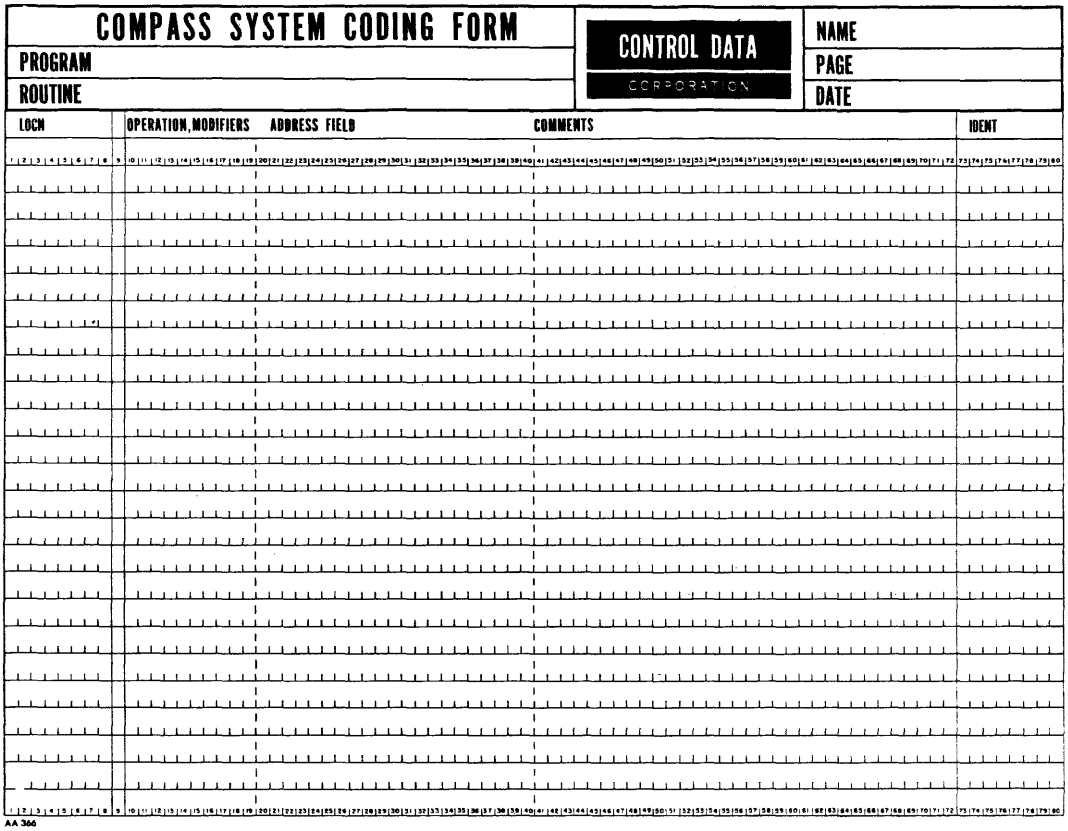
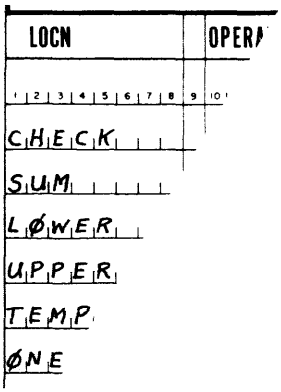


Figure 4-29. COMPASS Coding Form

Location Field--Columns 1-8



The location field is used to assign an alphanumeric location symbol to an instruction or constant (operand), permitting execution addresses to be represented symbolically. The first character must be alphabetic but subsequent characters may be alphabetic, numeric, or a period. Other punctuation marks and special symbols may not be used in the location field.

The location symbol must not contain more than eight characters (length of the field). Examples of acceptable characters are SIDEA, SIDEB, XSQUARE, DIVISOR, TURKEYS, SHEEP. The symbols 5TURKEYS and ?SHEEP would be illegal and would constitute errors. An asterisk in column 1 would identify the entire line (columns 2 through 72) as a comment and the information on that line would not be assembled.

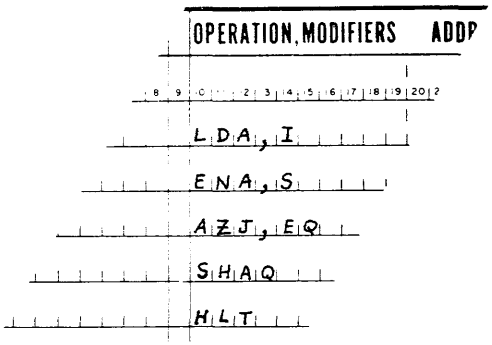
NOTE:

> The LOCN column may be left blank unless that address is referenced out of sequence (to obtain an operand or as a jump address).

Column #9

Column 9 is a demarcation line between the location field and the operation field and is always left blank.

Operation Field--columns 10-19

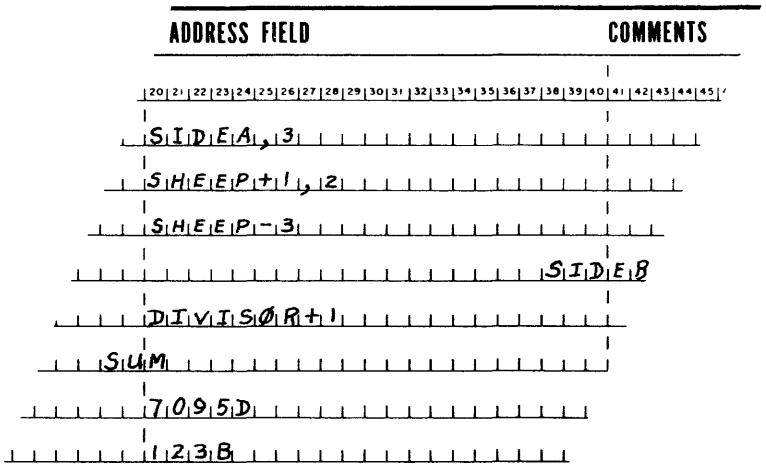


The operation field contains a mnemonic instruction code with related modifiers, a pseudo-mnemonic instruction (to be explained), or the octal values 00-77 to indicate the machine code for an instruction. The field begins in column 10 and is terminated by the first blank. If the address field is blank, the assembler translates it as a 00 machine code.

Related modifiers may be used in conjunction with the mnemonic code. If used, they must follow the mnemonic code, separated by a comma. The AZJ mnemonic code must be followed by a modifier to designate which type of comparison is to be made of the A register. The modifiers are:

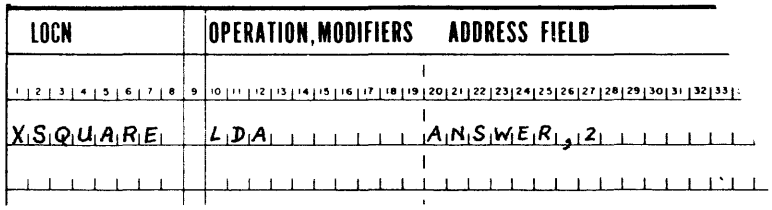
I	denotes indirect addressing	(LDA,I)
S	denotes sign extension	(ENA,S)
EQ	equal	(AZJ,EQ)
NE	not equal	(AZJ,NE)
GE	greater than or equal	(AZJ,GE)
LT	less than	(AZJ,LT)

Address Field--Columns 20-40 (with exceptions)



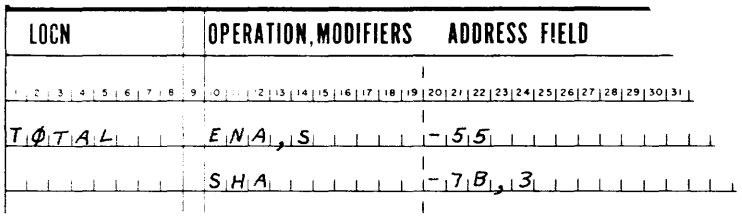
The address field may contain a mnemonic code, numerics, or an alphanumeric combination to represent a storage location (m), an operand (y), a shift count (k), or data.

The mnemonic code may designate a storage location or a reference point to designate some other storage location. For example, if the code SHEEP represents storage location 12345, SHEEP + 3 and SHEEP -4 would represent storage locations 12350 and 12341 respectively. Index register designators follow the mnemonic code, separated by commas. For example, the mnemonic code:



would be assembled as 20 2 XXXXX in machine language (XXXXX is the address equated with ANSWER).

If the address field contains an operand or shift count, it may be expressed in either octal or decimal. The number 7095 is decimal (radix 10) whereas the number 123B is octal (the B specifies octal). The assembler converts decimal operands to their octal equivalents (actually binary in the computer), obviating the necessity of conversion by the programmer. A minus sign (-) may be used to indicate negative numbers instead of the machine language type complement notation. For example, the assembler language instructions

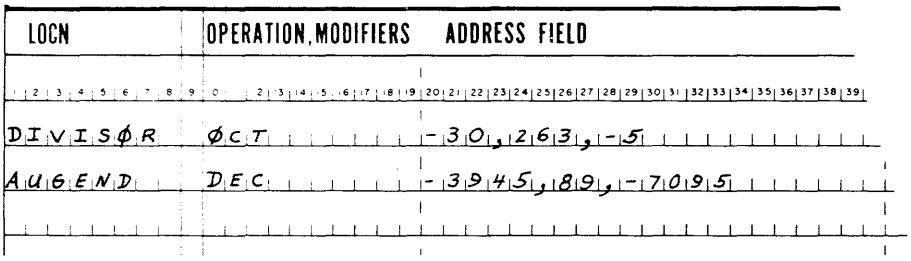


would be assembled in machine language as:

```
14 4 77710          (55 decimal = 67 octal)

12 3 77770          (shift count K = (B3) + k)
```

Data to be used by the program is normally stored immediately following the program in memory. Several operands may be expressed on the same line if separated by commas (each comma designates a new subfield). Although several subfields appear on the same line, the assembler would place each one in a separate storage location. Operands expressed on the same line must be all octal or all decimal, indicated by OCT or DEC in the operation field. For example, the assembly language expressions:



could appear in storage as:

```
12346    77777747    12353    77762110
12347    00000263
12350    77777772
12351    77770226
12352    00000131
```

The octal operands could be obtained from storage by making reference to DIVISOR, DIVISOR + 1, and DIVISOR + 2. Likewise, the decimal operand -7095 could be obtained by referencing storage location AUGEND + 2.

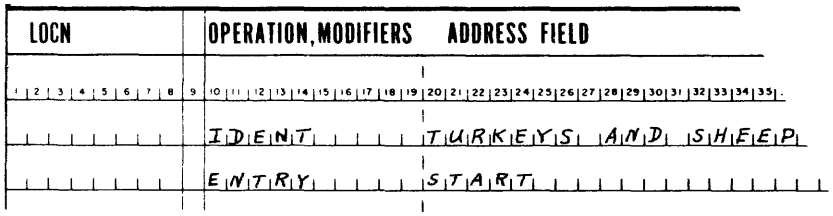
The address field begins with the first non-blank characters following the operation field and is terminated by a blank column. The field must begin before column 41 and is always terminated by column 73. All location symbols expressed in the address field must be defined by a similar symbol in the location field.

Comments

The comments area is provided for program remarks. This area is solely for the benefit of the programmer and has no effect on the program.

PSEUDO INSTRUCTIONS

One other area that must be discussed concerns pseudo instructions. A pseudo instruction provides information to the assembler that accomplishes some operation that the computer cannot accomplish with its own instructions. The OCT and DEC terms used to define data are examples of pseudo instructions. Another example of a pseudo instruction is the program identification symbol. Like other pseudo instructions, it is contained in the operation field.



The operation term is IDENT and identifies the program name as TURKEYS and SHEEP. The program name is located in the address field of the format. Only the first eight columns of the address field are actually used by COMPASS as the program identifier (TURKEYS).

Another pseudo instruction is the ENTRY statement which defines the entry point into the program. The entry point must be defined to allow proper assembly of the program.

With these few basic facts concerning the COMPASS assembler, you should be able to write a program coded in assembly language. The turkeys and sheep program discussed earlier is illustrated on the following page. Is should serve as a comparison between machine and assembler language programming. By comparing the two programs, it is evident that--although assembler language programming has numerous advantages over machine language--one assembly language statement is still required for each comparable machine language instruction.

After the assembly language program is recorded on a coding form, the form provides the necessary information to punch the source program (Figure 4-28). Each line on the coding form represents one card in the source deck. The deck would be loaded in the card reader and, in turn, into the computer for assembly. Figure 4-31 illustrates how the source deck for the turkeys and sheep problem would appear.

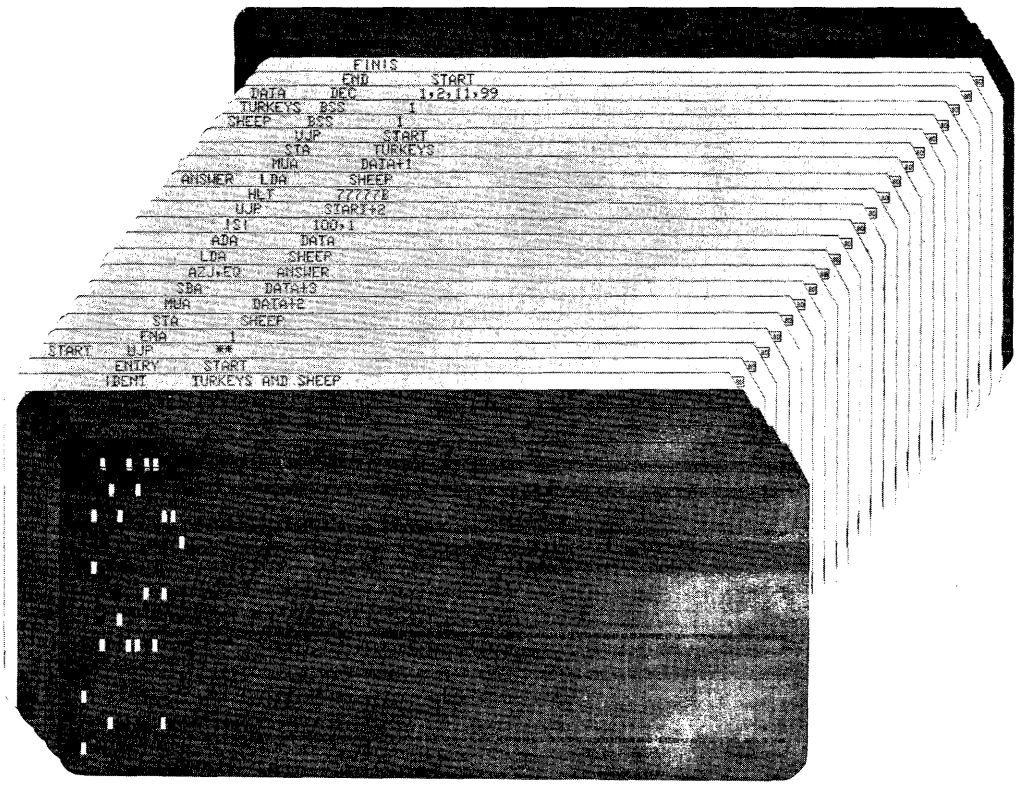


Figure 4-31. Source Deck

The turkeys and sheep problem was executed by a CDC 3200 computer in COMPASS. After execution, storage location TURKEYS contained the binary equivalent of 00000022_8 (18 decimal) and storage location SHEEP contained the binary equivalent of the number $0000\ 0011_8$ (9 decimal).

MONITOR SYSTEM

The following listing is a printout of the object program for the turkeys and sheep problem. The assembler is under control of another software program called a monitor or operating system. The monitor for the 3200 computer is SCOPE (Supervisory Control Of Program Execution). All programs, whether written in a compiler or assembler language, are compiled (or assembled) and executed under control of SCOPE. Although a program is assembled by COMPASS, the assembler is actually a subprogram of SCOPE. After assembly, control is returned to SCOPE

and the program may be executed.

A program is assembled starting at relative address 00000 and written on magnetic tape. At the end of the assembly run, the machine language program is on magnetic tape.

Program execution is accomplished under control of SCOPE by reading two cards, Load and Run (see Figures 4-31 and 4-32). The program is then read into computer storage starting at an address determined by SCOPE and printed on the listing following the Run statement (address 17712). All addresses and instruction m fields must be modified as the program is loaded into computer storage. The first instruction will be placed in magnetic core storage at address 17712. If the instruction contains an address in its m field, indicated with a P in the listing, the assembled address will also have 17712 added to it.

For example, the AZJ,EQ instruction would be in storage location 17717 (17712 + 00005) and would appear as 03 0 17725.

Locations SHEEP and TURKEYS would be assigned addresses of 17731 and 17732 respectively.

After SCOPE has relocated the program in magnetic core storage, a return jump is made to the entry point of the program (START) and a return address to SCOPE is written where the ** appears on the listing. The first instructions to be executed is at P + 1 (review the RTJ instruction). After the entire program has been executed, including all desired I/O operations such as Print-outs, a jump is made back to Start which returns control back to SCOPE. SCOPE will then immediately assemble, compile, or execute the next job, as indicated by the control cards it reads.

The function of the SCOPE control cards shown in the source deck (Figure 4-31) is listed below. Each control card is identified by SCOPE by a $\frac{7}{9}$ multipunch in column 1.

$\frac{7}{9}$ SEQUENCE

This indicates the start of a new job to SCOPE and provides the computer operator with a job sequence number between 1 and 999.

$\frac{7}{9}$ JOB

This provides information to SCOPE for automatic accounting. The first field following job is reserved for an 0-8 character charge number, the second for an 0-4 character programmer identification, and the third for a time limit. If the job is not completed within the prescribed time, control is returned to SCOPE and the next job is processed. Even if left blank, the fields must be delineated with commas.

The last field may contain one of several information messages to SCOPE. In this case, the ND indicates that no memory dump is desired if the program

terminates abnormally.

7/9 EQUIP

This card allows the programmer to assign certain operations to a definite piece of I/O equipment. For example, logical unit 56 is assigned as a Load and Go unit but no specific type of equipment is permanently assigned. The "56 = MT" indicates that the assembled program will be written on magnetic tape. Some units are permanently assigned for a given system and need not be equipped. By using the EQUIP statement, the programmer is allowed more latitude in the choice of I/O equipments.

7/9 COMPASS

This card informs SCOPE that the tab to follow is written in assembler language and that COMPASS is required to assemble it into machine language. Upon reading COMPASS, the assembler is read into memory from magnetic tape. The parameters on the COMPASS card are then read by COMPASS.

L indicates that a listing of the program is desired on the line printer.

X indicates that the program is to be executed and places the assembled program on logical unit 56 (previously equipped, as magnetic tape).

P would indicate that the assembled program will also be punched on cards.

Following assembly, the Finis card returns control to SCOPE, which then reads the Load card.

7/9 LOAD

The load card causes the monitor to load the program from logical unit 56. At load time, the program is relocated in memory and all addresses are modified to reflect the relocation. After the program is loaded in memory, the next card is read.

7/9 RUN

The run card informs SCOPE that the loaded program should now be executed. SCOPE does a RTJ to the Entry point and stores a return address. Upon completion of execution, control is returned to SCOPE which then reads the next card.

77/88

The double 7/8 indicates an End of File to SCOPE and that the job is completed. At this time another job may be processed under control of SCOPE. Any programs

residing in magnetic core storage--except SCOPE itself--may be destroyed as the new program is assembled and executed.

The listing for the TURKEYS and SHEEP program is illustrated in Figure 4-32. Notice that the program was not assigned a relocation address until after the Run card had been read. The same program may be assigned a different relocation address if loaded again. The relocation address is assigned by SCOPE at LOAD time.

```
COMPASS-32    (2.1)                TURKEYS


  ENTRY-POINT SYMBOLS
     START          00000



  LENGTH OF SUBPROGRAM        00025
  LENGTH OF COMMON            00000
  LENGTH OF DATA              00000


COMPASS-32    (2.1)                TURKEYS
                                                    ENTRY     START
      00000  01077777 01 0  77777 0  START    UJP       **
      00001  14600001 14 1  00001 2            ENA       1
      00002  40000017 40 0 P00017 0            STA       SHEEP
      00003  50000023 50 0 P00023 0            MUA       DATA+2
      00004  31000024 31 0 P00024 0            SBA       DATA+3
      00005  03000013 03 0 P00013 0            AZJ,EQ    ANSWER
      00006  20000017 20 0 P00017 0            LDA       SHEEP
      00007  30000021 30 0 P00021 0            ADA       DATA
      00010  10100144 10 0  00144 1            ISI       100,1
      00011  01000002 01 0 P00002 0            UJP       START+2
      00012  00077777 00 0  77777 0            HLT       77777B
      00013  20000017 20 0 P00017 0  ANSWER    LDA       SHEEP
      00014  50000022 50 0 P00022 0            MUA       DATA+1
      00015  40000020 40 0 P00020 0            STA       TURKEYS
      00016  01000000 01 0 P00000 0            UJP       START
      00017                          SHEEP     BSS       1
      00020                          TURKEYS   BSS       1
      00021          00000001        DATA      DEC       1,2,11,99
      00022          00000002
      00023          00000013
      00024          00000143
                                                END       START

                NUMBER OF LINES WITH DIAGNOSTICS          0

LOAD,56
RUN


SUBP
 17712  TURKEYS

ENTR
 17712  START
 02201  AET
 02453  START2
```

Figure 4-32.

Other programs of a more practical nature than the Turkeys and Sheep problem can also be written in COMPASS. For example, suppose you wish to solve the equation $X^4 - 46X^3 + 164X^2 + 6814X - 20757$ and derive the four roots of X. It could be solved algebraically,but consider how it could be solved by a computer.

Remember the steps to consider when writing a program?

1. Define the problem---$X^4 - 46X^3 + 164X^2 + 6814X - 20757$.
2. Draw a flow chart---Figure 4-33.
3. Code the problem into a computer language---Figures 4-34 and 4-35.
4. Delay the program---Correct any diagnostics indicated on the assembly listing before execution.

In order to solve the equation, some arbitrary valve would be assigned to X and inserted into the equation. If the result of the equation equals zero, a root has been found. Increment the value of X by one and check that value. Store the roots as they are found before looking for the next root. Limits must be established to allow the program to stop if four roots are not found. Provisions should also be made to stop immediately after the fourth root has been found even though the upper limit has not been reached. Examine the flow chart (Figure 4-33).

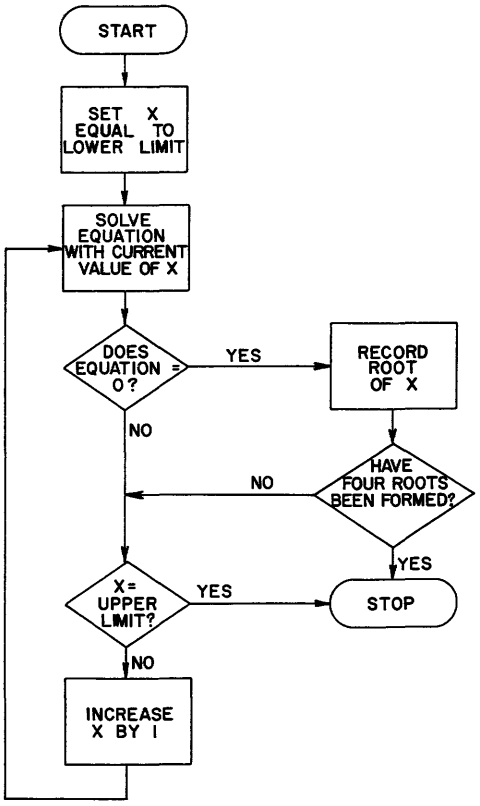


Figure 4-33. Polynomial Expression Flow Chart

After the flow chart is completed, code the probelm in assembly language. One solution to the problem is illustrated by Figures 4-34 and 4-35. Notice that the tags assigned as addresses allude to the meaning of the operation being performed.

Decimal values between -50 and +50 will be checked for possible roots of X, starting at -50.

The first page of the program solves the equation with -50 as the value for X. If -50 was a root, the results of the subtract (last line, Figure 4-34) would be zero. The next instruction jumps to a store the root routine if the root was found. If not, the limits of the program are checked and another decision is made. If the upper limit has been reached, return to SCOPE. If still within limits, increase value of X by one and go through program again.

If the four roots of X are found before the upper limit is reached, the ISI instruction in the store the root routine causes a skip to address STØP and the UJP instruction causes a jump back to Start, which returns control to SCOPE.

Program results are usually printed out before control is returned to SCOPE. Otherwise, another job could be immediately processed and the contents of RØØTS through RØØTS + 3 could be destroyed. However, the routine to convert the octal answers to BCD (binary coded decimal) and then accomplish the printing becomes quite involved and lies beyond the SCOPE of this course.

COMPASS SYSTEM CODING FORM

CONTROL DATA CORPORATION

PROGRAM FIND THE ROOTS OF: $X^4-46X^3+164X^2+6814X-20757$

ROUTINE

NAME JOHN DOE

PAGE 1

DATE NOV. 1

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS	IDENT
	IDENT	XRØØTS	PROGRAM NAME	
	ENTRY	START	DEFINES ENTRY POINT FOR SCOPE	
START	UJP	*,*	SCOPE RELOCATABLE RETURN ADDRESS	
	LDA	LOWER	MAKE X EQUAL TO LOWER LIMIT	
	40	X	STORE CURRENT VALUE OF X	
	MUA	X	FORM X^2	
	STA	X+1	STORE CURRENT VALUE OF X^2	
	MUA	X	FORM X^3	
	STA	X+2	STORE CURRENT VALUE OF X^3	
	MUA	X	FORM X^4	
	STA	X+3	STORE CURRENT VALUE OF X^4	
	LDA	CØNSTANT	LOAD A WITH 46 (56 OCTAL)	
	MUA	X+2	FORM $46X^3$	
	STA	TEMP	TEMPORARILY STORE $46X^3$	
	LDA	CØNSTANT+1	LOAD A WITH 1644 (2444 OCTAL)	
	MUA	X+1	FORM $1644X^2$	
	STA	TEMP+1	TEMPORARILY STORE $1644X^2$	
	LDA	CØNSTANT+2	LOAD A WITH 68144 (105236 OCTAL)	
	MUA	X	FORM $68144X$	
	SBA	CØNSTANT+3	SUBTRACT 20757 FROM $68144X$	
	ADA	TEMP+1	ADD $1644X^2$ TØ $(68144X-20757)$	
	ADA	X+3	ADD X^4 TØ $(1644X^2+68144X-20757)$	
	SBA	TEMP	SUBTRACT $46X^3$ FRØM $(X^4+1644X^2+$	
*			$68144X-20757)$	

AA 366

Figure 4-34.

COMPASS SYSTEM CODING FORM

CONTROL DATA CORPORATION

PROGRAM FIND THE ROOTS

ROUTINE

NAME JOHN DOE

PAGE 2

DATE NOV. 1

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS	IDENT
	AZJ,EQ	RECØRD	JUMP IF (A1)=ZERØ (RØØT FØUND)	
CHECK	LDA	LØWER	NØT A RØØT, LØAD A WITH X	
	SBA	UPPER	SUBTRACT UPPER LIMIT	
	AZJ,EQ	STØP	JUMP IF UPPER LIMIT IS REACHED	
	LDA	LØWER	UPPER LIMIT NØT REACHED - LØAD A	
	ADA	ØNE	WITH VALUE ØF X AND ADD 1	
	STA	LØWER	STØRE NEW VALUE ØF X	
	UJP	START+1	TRY AGAIN WITH NEW VALUE ØF X	
RECØRD	LDA	LØWER	LØAD A WITH THE RØØT ØF X	
	STA	RØØTS,2	STØRE RØØT	
	ISI	3,2	HAVE FØUR RØØTS BEEN FØUND? IF	
*			NØT, ADD 1 TØ (B2) AND GØ TØ P+1	
*			IF YES, CLEAR B2 AND SKIP TØ P+2	
	UJP	CHECK	RETURN TØ UPPER LIMIT CHECK	
STØP	UJP	START	RETURN TØ SCØPE UPØN COMPLETIØN	
LØWER	DEC	-50		
UPPER	DEC	50		
ØNE	DEC	1		
CØNSTANT	DEC	46, 64, 68, 14, 20757		
TEMP	BSS	2	STØRAGE FØR 46X³ AND 64X²	
RØØTS	BSS	4	STØRAGE FØR RØØTS ØF X	
X	BSS	4	STØRAGE FØR X, X², X³, AND X⁴	
	END	START	INDICATES END ØF PRØGRAM	

AA 366

Figure 4-35.

The listing for the X RØØTS program is illustrated in Figure 4-36.

The four roots were stored at addresses RØØTS, RØØTS + 1, RØØTS + 2, and RØØTS + 3 by the program. During loading,SCOPE relocated the entire program by a factor of 17654.

```
SUBP
 17654  XROOTS

ENTR
 17654  START
```

Location Start was relocated from relative address 00000 to address 17654 and the remainder of the program followed in sequential locations. Therefore, the four roots derived by the program were placed in storage locations 17727, 17730, 17731, and 17732 during execution (17654 + 00053 = 17727).

After execution, those addresses were examined and were found to contain 77777764, 00000003, 00000021, and 00000045 respectively. The decimal equivalents of the octal answers (-13, +3, +21, and +45) would be -11, +3, +17, and +37. If each of those values is substituted into the equation for X, the result is zero which indicates that each is actually a root.

```
                                          ENTRY     START              DEFINES ENTRY POINT FOR SCOPE
00000  01077777 01 0  77777 0  START     UJP       **                 SCOPE RECORDS RETURN ADDRESS
00001  20000042 20 0 P00042 0            LDA       LOWER              MAKE X EQUAL TO LOWER LIMIT
00002  40000057 40 0 P00057 0            40        X                  STORE CURRENT VALUE OF X
00003  50000057 50 0 P00057 0            MUA       X                  FORM X2
00004  40000060 40 0 P00060 0            STA       X+1                STORE CURRENT VALUE OF X2
00005  50000057 50 0 P00057 0            MUA       X                  FORM X3
00006  40000061 40 0 P00061 0            STA       X+2                STORE CURRENT VALUE OF X3
00007  50000057 50 0 P00057 0            MUA       X                  FORM X4
00010  40000062 40 0 P00062 0            STA       X+3                STORE CURRENT VALUE OF X4
00011  20000045 20 0 P00045 0            LDA       CONSTANT           LOAD A WITH 46 (56 OCTAL)
00012  50000061 50 0 P00061 0            MUA       X+2                FORM 46 X3
00013  40000051 40 0 P00051 0            STA       TEMP               TEMPORARILY STORE 46X3
00014  20000046 20 0 P00046 0            LDA       CONSTANT+1         LOAD A WITH 164 (244 OCTAL)
00015  50000060 50 0 P00060 0            MUA       X+1                FORM 164X2
00016  40000052 40 0 P00052 0            STA       TEMP+1             TEMPORARILY STORE 164X2
00017  20000047 20 0 P00047 0            LDA       CONSTANT+2         LOAD A WITH 6814 (15236 OCTAL)
00020  50000057 50 0 P00057 0            MUA       X                  FORM 6814X
00021  31000050 31 0 P00050 0            SBA       CONSTANT+3         SUBTRACT 20757 FROM 6814X
00022  30000052 30 0 P00052 0            ADA       TEMP+1             ADD 164X2 TO (6814X-20757)
00023  30000062 30 0 P00062 0            ADA       X+3                ADD X4 TO (164X2+6814X-20757)
00024  31000051 31 0 P00051 0            SBA       TEMP               SUBTRACT 46X3 FROM (X4164X2+
                                                                      6814X-20757)
00025  03000035 03 0 P00035 0            AZJ,EQ    RECORD             JUMP IF (A)=ZERO (ROOT FOUND)
00026  20000042 20 0 P00042 0  CHECK     LDA       LOWER              NOT A ROOT,LOAD A WITH X
00027  31000043 31 0 P00043 0            SBA       UPPER              SUBTRACT UPPER LIMIT
00030  03000041 03 0 P00041 0            AZJ,EQ    STOP               JUMP IF UPPER LIMIT IS REACHED
00031  20000042 20 0 P00042 0            LDA       LOWER              UPPER LIMIT NOT REACHED - LOAD A
00032  30000044 30 0 P00044 0            ADA       ONE                 WITH VALUE OF X AND ADD 1
00033  40000042 40 0 P00042 0            STA       LOWER              STORE NEW VALUE OF X
00034  01000001 01 0 P00001 0            UJP       START+1            TRY AGAIN WITH NEW VALUE OF X
00035  20000042 20 0 P00042 0  RECORD    LDA       LOWER              LOAD A WITH ROOT OF X
00036  40200053 40 0 P00053 2            STA       ROOTS,2            STORE ROOT
00037  10200003 10 0  00003 2            ISI       3,2                HAVE FOUR ROOTS BEEN FOUND  IF
                                                                       NOT,ADD 1 TO (B2) AND GO TO P+1
                                                                       IF YES,CLEAR B2 AND SKIP TO P+2
00040  01000026 01 0 P00026 0            UJP       CHECK              RETURN TO UPPER LUMIT CHECK
00041  01000000 01 0 P00000 0  STOP      UJP       START              RETURN TO SCOPE UPON COMPLETION
00042          77777715         LOWER     DEC       -50
00043          00000062         UPPER     DEC       50
00044          00000001         ONE       DEC       1
00045          00000056         CONSTANT  DEC       46,164,6814,20757
00046          00000244
00047          00015236
00050          00050425
00051                           TEMP      BSS       2                  STORAGE FOR 46X3 AND 164X2
00053                           ROOTS     BSS       4                  STORAGE FOR ROOTS OF X
00057                           X         BSS       4                  STORAGE FOR X,X2,X3,AND X4
                                          END       START              INDICATES END OF PROGRAM

                                NUMBER OF LINES WITH DIAGNOSTICS        0
```

Figure 4-36.

The same program was again assembled but three cards in the source deck were replaced with cards that contained errors. Figure 4-37 illustrates the resulting listing from the erroneous source program.

One indication that something is wrong with the program is the statement LOADING DELETED. Serious errors prevent proper execution and, therefore, the loading of the assembled program from tape into storage would only waste time.

Another indication of trouble is the statement NUMBER OF LINES WITH DIAGNOSTICS 3. Upon examination of the listing, three alpha characters are noted at the left margin of the page. The first is the letter 0 opposite relative address 00015. The 0 indicates that there is an error in the operation field. Examination of the field shows the erroneous mnemonic code MUL instead of MUA.

The next error is indicated by the letter U opposite relative address 00032. The U indicates an undefined address. The address field should contain ONE but contains only ON. Address ON does not appear in the location field and is therefore undefined.

The third error is indicated by the A opposite relative address 00037. The A indicates that an error exists in the address field. The interpretation of the address field of the ISI instruction would be "skip if B5 is equal to 3". The 3200 computer has only three index registers (B1, B2, B3) and, therefore, examination of B5 is an impossibility. SCOPE detected the error because it "knew" that the computer did not have a B5.

SUMMARY

You should now be able to write a program in assembly language that will run on a CDC 3200 computer. Procedures for output operations are explained in detail in the 3200 SCOPE/COMPASS reference manual, if printed results are desired. However, the important point is that you are now able to write a program. Actual program results are of secondary importance at the introductory level.

```
                                                    ENTRY     START          DEFINES ENTRY POINT FOR SCOPE
      00000  01077777 01 0   77777 0  START        UJP       **             SCOPE RECORDS RETURN ADDRESS
      00001  20000042 20 0 P00042 0                LDA       LOWER          MAKE X EQUAL TO LOWER LIMIT
      00002  40000057 40 0 P00057 0                40        X              STORE CURRENT VALUE OF X
      00003  50000057 50 0 P00057 0                MUA       X              FORM X2
      00004  40000060 40 0 P00060 0                STA       X+1            STORE CURRENT VALUE OF X2
      00005  50000057 50 0 P00057 0                MUA       X              FORM X3
      00006  40000061 40 0 P00061 0                STA       X+2            STORE CURRENT VALUE OF X3
      00007  50000057 50 0 P00057 0                MUA       X              FORM X4
      00010  40000062 40 0 P00062 0                STA       X+3            STORE CURRENT VALUE OF X4
      00011  20000045 20 0 P00045 0                LDA       CONSTANT       LOAD A WITH 46 (56 OCTAL)
      00012  50000061 50 0 P00061 0                MUA       X+2            FORM 46 X3
      00013  40000051 40 0 P00051 0                STA       TEMP           TEMPORARILY STORE 46X3
      00014  20000046 20 0 P00046 0                LDA       CONSTANT+1     LOAD A WITH 164 (244 OCTAL)
    0 00015  00000000 00 0   00000 0                MUL       X+1            FORM 164X2
      00016  40000052 40 0 P00052 0                STA       TEMP+1         TEMPORARILY STORE 164X2
      00017  20000047 20 0 P00047 0                LDA       CONSTANT+2     LOAD A WITH 6814 (15236 OCTAL)
      00020  50000057 50 0 P00057 0                MUA       X              FORM 6814X
      00021  31000050 31 0 P00050 0                SBA       CONSTANT+3     SUBTRACT 20757 FROM 6814X
      00022  30000052 30 0 P00052 0                ADA       TEMP+1         ADD 164X2 TO (6814X-20757)
      00023  30000062 30 0 P00062 0                ADA       X+3            ADD X4 TO (164X2+6814X-20757)
      00024  31000051 31 0 P00051 0                SBA       TEMP           SUBTRACT 46X3 FROM (X4164X2+
                                                                            6814X-20757)
      00025  03000035 03 0 P00035 0                AZJ,EQ    RECORD         JUMP IF (A)=ZERO (ROOT FOUND)
      00026  20000042 20 0 P00042 0  CHECK        LDA       LOWER          NOT A ROOT,LOAD A WITH X
      00027  31000043 31 0 P00043 0                SBA       UPPER          SUBTRACT UPPER LIMIT
      00030  03000041 03 0 P00041 0                AZJ,EQ    STOP           JUMP IF UPPER LIMIT IS REACHED
      00031  20000042 20 0 P00042 0                LDA       LOWER          UPPER LIMIT NOT REACHED - LOAD A
    U 00032  30000000 30 0   00000 0                ADA       ON              WITH VALUE OF X AND ADD 1
      00033  40000042 40 0 P00042 0                STA       LOWER          STORE NEW VALUE OF X
      00034  01000001 01 0 P00001 0                UJP       START+1        TRY AGAIN WITH NEW VALUE OF X
      00035  20000042 20 0 P00042 0  RECORD       LDA       LOWER          LOAD A WITH ROOT OF X
      00036  40200053 40 0 P00053 2                STA       ROOTS,2        STORE ROOT
    A 00037  10100003 10 0   00003 1                ISI       3,5            HAVE FOUR ROOTS BEEN FOUND  IF
                                                                             NOT,ADD 1 TO (B2) AND GO TO P+1
                                                                             IF YES,CLEAR B2 AND SKIP TO P+2
      00040  01000026 01 0 P00026 0                UJP       CHECK          RETURN TO UPPER LUMIT CHECK
      00041  01000000 01 0 P00000 0  STOP         UJP       START          RETURN TO SCOPE UPON COMPLETION
      00042          77777715        LOWER        DEC       -50
      00043          00000062        UPPER        DEC       50
      00044          00000001        ONE          DEC       1
      00045          00000056        CONSTANT     DEC       46,164,6814,20757
      00046          00000244
      00047          00015236
      00050          00050425
      00051                          TEMP         BSS       2              STORAGE FOR 46X3 AND 164X2
      00053                          ROOTS        BSS       4              STORAGE FOR ROOTS OF X
      00057                          X            BSS       4              STORAGE FOR X,X2,X3,AND X4
                                                  END       START          INDICATES END OF PROGRAM

                                        NUMBER OF LINES WITH DIAGNOSTICS      3

LOAD,56
LOADING DELETED
```

Figure 4-37. 4-88

PART III

COMPILER PROGRAMMING

More sophisticated programs, known as compilers, enable programmers to be more concerned with what has to be done, rather than how the computer does it. Assembly languages involve a mnemonic or relative technique of coding each computer instruction, whereas the compiler enables programmers to write one statement for many computer instructions. This ultimate degree of programming is referred to as Compiler Language programming.

When writing programs in a compiler language there is no need to be concerned with the octal numbers or the actual instructions of the machine.

It is important to realize that a compiler is not a black box or part of the computer hardware. It is a stored computer program that translates a programmer's written language (source language) into machine code (object language). The differences in the languages are apparent. Compare the three solutions to the following problem.

Problem: Add the contents of registers 40 and 50 and store the sum in register 51.

Machine Coding	Assembler Coding	Compiler Coding
20 0 00040 (LDA)	LDA X	Result = X + Y
30 0 00050 (ADA)	ADA Y	
40 0 00051 (STA)	STA Result	

DIFFERENT COMPILERS

There are many compiler languages written for different applications. ALGOL (an algebraic compiler), COBOL (used in business applications), and FORTRAN, (an algebraic compiler used in scientific work), are three of the most common. These compilers have to be written for the specific machine with which they are going to be used. However, there are only slight differences in the FORTRAN language for different computers. A general discussion of FORTRAN*, a contraction of FORmula TRANslation, illustrates one kind of a compiler language.

Figure 4-38 shows the order of compililing and executing a program written in a compiler language.

---

* Computer Programming Concepts

* FORTRAN Auto Tester

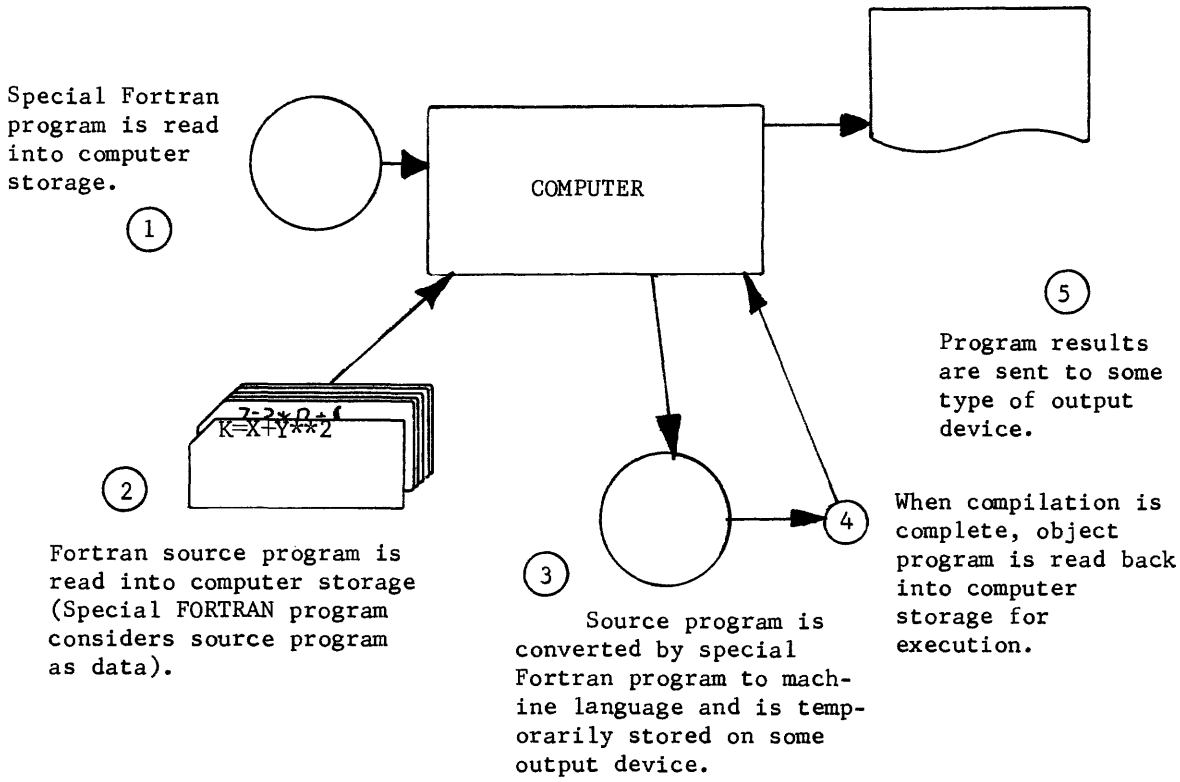


Figure 4-38. Compiler Language Programming

Any number that appears in a FORTRAN program as an operand is a constant, whereas any quantity expressed by a name is a variable. In the arithmetic statement X = 2Y + 3Z, 2 and 3 are constants but X, Y, and Z are variables.

CONSTANTS

Two kinds of constants exist in FORTRAN. Whole numbers, or integers, are expressed in fixed point. Extremely large numbers or fractions may be equated to a coefficient raised to some power and are expressed in floating point.

The desired method of expression is indicated by the programmer as the program is written. If floating point is indicated, decimal points are automatically "floated" into place as needed to perform the arithmetic operations. Hence, the term "floating point."

FORTRAN distinguishes between floating and fixed point constants by the presence or absence of decimal point, respectively. Thus 3 is a fixed point constant,

but 3.0 (3., 3.000, etc.) is a floating point constant.

If a constant is positive, the plus (+) sign may be omitted. If it is negative, it must be preceded by a minus sign.

Examples of acceptable fixed point constants are:

0

6

+400

-1234

32767

Examples of unacceptable fixed point constants are:

12.78 (decimal points not allowed)

1604A (letters not allowed)

Examples of acceptable floating point constants are:

0.0

6.0

-20000.

-.0002784

+15.016

VARIABLES AND THE NAMES OF VARIABLES

The term variable is used in FORTRAN to denote any quantity that is referred to by name rather than by explicit appearance; the variavle is able to assume any number of values.

Variables may be either fixed point or floating point quantities. A fixed point variable is one that takes on any of the values permitted of a fixed point constant. The name of a fixed point variable is given in a combination of one to eight alphanumeric characters. The first letter must be an I, J, K, L, M, or N. Examples of fixed point variables are:

I

KLM

MATRIX

Examples of unacceptable names of fixed point variables are:

J12345670	-	too many characters
ABC	-	does not begin with the correct letter
5M	-	does not begin with a letter
$J78	-	contains a character other than a digit or letter
J34.5	-	contains a character other than a digit or letter

A floating point variable is one represented inside the machine in the same form as a floating point constant. Data is usually set up as a floating point variable because of the convenience provided by the automatic handling of all decimal points. The name of a floating point variable is given in combinations of one to eight alphanumeric characters of which the first is a letter other than I, J, K, L, M, or N.

Examples of acceptable names of floating point variables are:

AVAR

FRONT

G

F00009

Examples of unacceptable names of floating point variables are:

A12345670	-	too many characters
8BOX	-	does not begin with a letter
KJLI	-	does not begin with the correct letter
*BCD	-	contains a character other than a digit or a letter
A + B	-	contains a character other than a digit or a letter
B9.35	-	contains a character other than a digit or a letter

The compiler places no significance in names; it merely inspects the first letter to determine whether the variable is fixed or floating point. A name such as B7 does not specifically mean B times 7, B to the seventh power, or B_7. The programmer should assign names to variables that allude to their meaning, but no meaning as such is attached to the symbols by the compiler.

Every combination of letters and digits constitutes a separate name. Thus the name ABC is not the same as the name BAC, and the names A, AB, and AB7 are all

different in FORTRAN.

OPERATIONS

FORTRAN provides for five basic arithmetic operations: addition, subtraction, multiplication, division, and exponentiation. Each of these operations is represented by a symbol:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Note that the combination ** is considered one symbol; there is no confusion between ** and * since it is never permissible to write two operation symbols side by side.

EXPRESSIONS OR STATEMENTS

A FORTRAN expression contains constants, variables, functions, or any combination of these, separated by operation symbols, commas, and parentheses. If forms a meaningful mathematical statement. Some examples of expressions follow in table 4-6.

TABLE 4-6. EXAMPLES OF FORTRAN EXPRESSION

EXPRESSION	MEANING
K	The value of the fixed point variable K.
3.14159	The value of the floating point constant 3.14159
A+2.1828	The sum of the value of A and 2.1828.
ZETA-SIGMA	The difference between the values of ZETA and SIGMA
X*Y	The product of the values of X and Y.
OMEGA/16.2832	The quotient formed when the value of OMEGA is divided by 16.2832.
C**2	The value of C raised to the second power

EXPRESSION	MEANING
(A+F)/(X+2.)	The sum of the values of A and F divided by the sum of the value of X and 2.
1./(X**2+Y**3)	The reciprocal of (X^2+Y^3).

In writing expressions the programmer must observe certain rules in order to correctly convey his intentions.

1) Two operation symbols must not appear next to each other. Thus A*-B is not a valid expression, but A*(-B) is.

2) Parentheses must be used to indicate grouping just as in ordinary mathematical notation. Thus $(X+Y)^3$ must be written (X+Y)**3 to convey the correct meaning.

3) When the hierarchy of operations in an expression is not completely specified by the use of parentheses, the standard FORTRAN sequence is as follows: All exponentiations are done first, then all multiplications and divisions, and finally all additions and subtractions.

4) Within a sequence of consecutive multiplications and/or divisions, or additions and/or subtractions, in which the order of the operations to be performed is not completely specified by the use of parentheses, the operations are performed from left to right.

5) An exponent may itself be an expression. Thus the expression X**(I+2) is perfectly acceptable.

6) Parentheses indicate grouping. Specifically, they never imply multiplication. Thus, the expression(A+B) (C+D) is incorrect; it should be written (A+B)*(C+D).

ARITHMETIC STATEMENTS

The most common statement is the arithmetic statement, which is an order to FORTRAN to perform a computation. Its general format is A = B, in which A is a variable name, written without a sign, and B is any expression defined above. The = sign in an arithmetic statement is not used in the same way as it is in ordinary mathematical notation. In FORTRAN, the = sign means <u>is replaced by</u>. Statements such as Z-RHO = ALPHA + BETA, in which Z is unknown and the others are known, are not permitted. The only legitimate form of arithmetic statement is one in which the left side of the statement is the name of a single variable. The meaning of the = sign in this case is <u>to replace</u> the value of the variable named on the left with the value of the expression on the right. Thus the statement A = B + C is an order to form the sum of the values of the variables B and C and to replace the value of the variable A with that sum.

Another example of arithmetic statement brings out the special meaning of the = sign. A statement such as N = N + 1 means: replace the value of the variable N with its old value plus 1. This kind of statement, which is clearly not an equation, finds frequent use.

The following examples (Table 4-7) show acceptable arithmetic statements with their equivalent normal mathematical forms when such equivalents exist. Variable names have been chosen arbitrarily. It is assumed that previous statements have established values of the variables on the right side.

TABLE 4-7. EQUIVALENT FORTRAN/MATHEMATICS STATEMENT

Arithmetic Statement	Original formula
A = (S+X)/(R-T)	$A = \frac{S+X}{R-T}$
GAMMA = -1./(2.*X)+A**2/(4.*X**2)	$GAMMA = \frac{-1}{2X} + \frac{A^2}{4X^2}$
H = 1.112*D*(R1*R2/(R1-R2))	$H = 1.112D \frac{R1\ R2}{R1-R2}$
PI = 3.1415927	$\pi = 3.1415927$
Z = X*(X**2-Y**2)/(X**2+Y**2)	$Z = \frac{X^2 - Y^2}{X^2 + Y^2}$

WRITING FORTRAN PROGRAMS

FORTRAN programs are also written on a coding form. Because assemblers and compilers represent different degrees of programming, the coding format for FORTRAN programs is quite different from the assembler format discussed earlier. Figure 4-39 illustrates the FORTRAN coding form.

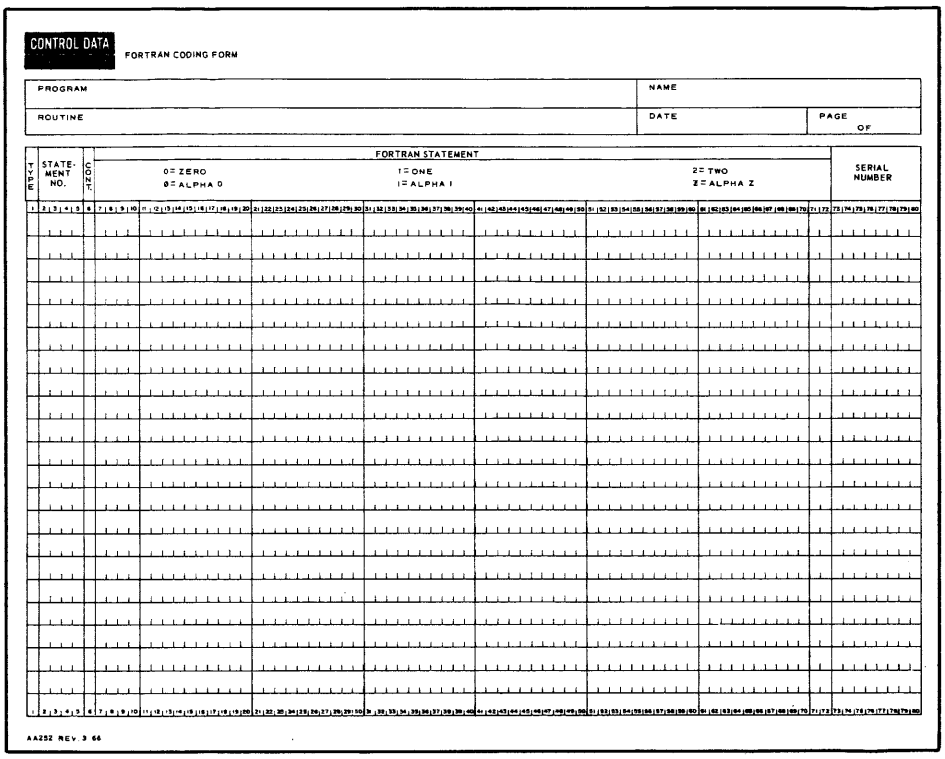
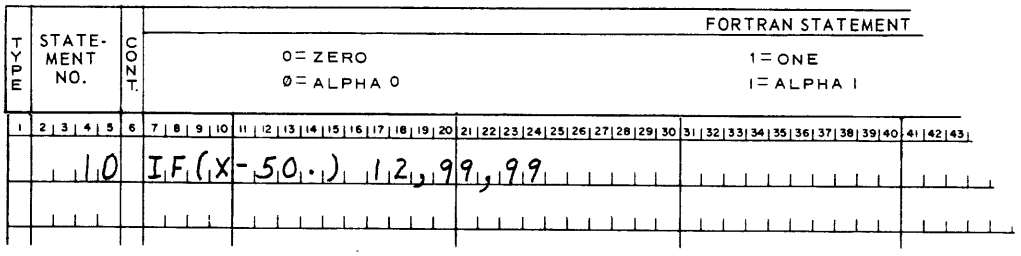


Figure 4-39. FORTRAN Coding Form



Column 1, Type

The letter C in column 1 identifies the entire statement as a comment. The comment is not processed by FORTRAN but does appear on the printed listing of the program.

Columns 2 to 5, Statement Number

The statement number is normally right justified in column 2-5. However, if the programmer desires, column 1 may also be used, allowing expression of statement numbers up to 32767_{10}.

Column 6, Continuation

Any FORTRAN character with the exception of a blank or a zero in column 6 indicates that the statement is a continuation of the preceding line. Up to 10 lines can be used for one FORTRAN statement (FORTRAN statement located in columns 7-72).

Columns 7 to 72, FORTRAN Statement

All statements must start in column 7 or a following column. Each column may contain only 1 character. In the foregoing example, the statement contains 18 characters. Any statement containing more than 66 characters must be continued on the next line.

Columns 73 to 80, Serialization

These columns are used to serialize the lower deck that will be punched from the information on the coding form. Serial numbers are not processed by FORTRAN but do appear on the program listing.

DATA LIST STATEMENTS

If a problem is to be executed only once, the data can be entered with the program in the form of constants in statements. Examples of this are:

```
X = 42.5

ALPHA = 237.062
```

Programs are usually set up to read in data from punched cards at the time the program is executed. The data cards normally follow this program and must be in proper sequence. The same FORTRAN statements can then be used to process as many sets of data as desired, by changing only the data cards.

Read Statement

Data is entered into the computer from cards by the use of a Read statement. The Read statement contains a list of the variable names to be read from a card. The variables are read off the card from left to right. One Read statement reads at least one card. Several variables may appear in one card but they cannot be read with separate Read statements. The Read statement is called a data list statement because it contains a list of the variables to be read into the computer from the card.

Punch Statement

Output data may be punched into a card from the computer. A Punch statement is used for this purpose. The values appear on the card in the same order as the list of variables appear in the Punch statement. Examples of READ and PUNCH statements are:

```
READ 10, X, Y, Z

PUNCH 25, ALPHA, K, JET
```

Note the numbers which separate the name of the statement and the list of variables. Each number specifies a Format statement (discussed next). The Punch statement is also called a data list statement. Other examples of data list statements are Print, Write Output Tape, and Read Input Tape. They are used to print the value of variables on the printer and write on or read from magnetic tape.

FORMAT STATEMENTS

The data list statement tells the computer which variables are to be read into or out of the computer. It contains no information about the variable itself, other than its floating or fixed point classification. It does not tell how many columns are allocated to each variable, or where the decimal point is in a floating point number.

This type of information is contained in a Format statement. Suppose one desires to read the values for three variables, ALPHA, XRAY, and IDA from a punched card. One must know how these values are punched in a card and prepare a Format statement to guide the Read statement in reading the information properly. The following example illustrates the Read and Format statements for the three variables; ALPHA, XRAY, and IDA (ALPHA and XRAY are floating point, IDA is fixed point).

TYPE	STATEMENT NO.	CONT.	FORTRAN STATEMENT (0=ZERO Ø=ALPHA O 1=ONE I=ALPHA I)
1	2 3 4 5	6	7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
			READ 10, ALPHA, XRAY, IDA
	10		FØRMAT (F5, F6, I5)

The Read statement specifies the reading of the first data card following the source program. Its interpretation is "Read, as to the format of statement 10, the values for ALPHA, XRAY, and IDA." The format specifies that data is located in three subfields (Figure 4-40). Information may be contained in all 80 columns but only the data in the subfields specified by the FORMAT statement will be used.

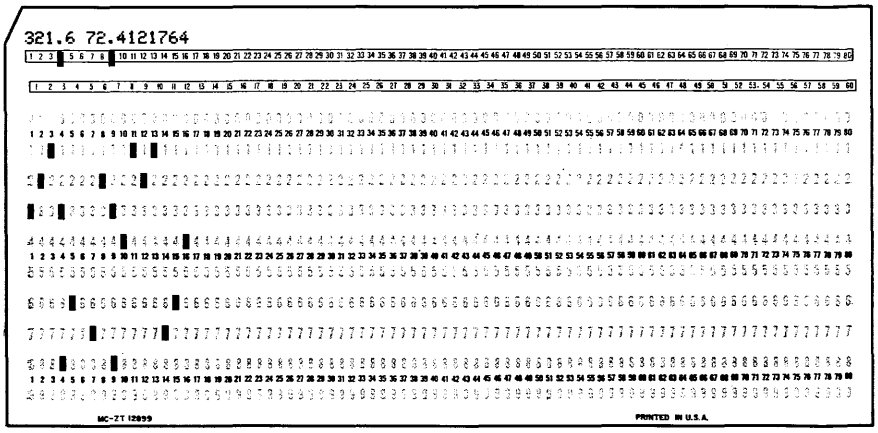


Figure 4-40. FORTRAN Data Card

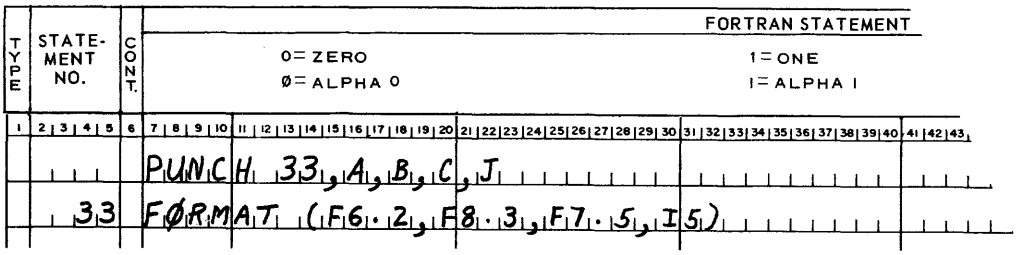
The first field specification (F5) tells how many columns are associated with the first variable ALPHA, and the relative position of the decimal point on the data card determines the magnitude of the operand. The field includes the first five columns on the card with one digit to the right of the decimal point. The decimal point occupies one column. The letter F is used to indicate floating point constants in the data list.

The field specification for the variable XRAY is F6. F is for floating point and the 6 for six columns that XRAY occupies. The next variable is IDA, which is a fixed point variable and occupies five columns. The letter I is one of those used to indicate fixed point variables. The field specification is, therefore, I.

Although they are part of the same deck containing the source program, the data cards are not read into the computer until after the source program has been compiled and the object program is being executed.

One data card is read for each READ statement in the program. If the program loops back on the same READ statement, a new data card will be read during each pass.

The PUNCH statement requires a similar Format statement to indicate the field length. The field F6.2 indicates that two places of significance will follow the decimal point and three significant digits will proceed the point.



Assume that a program has just solved a problem and derived the values for A, B, C, and J, as +31.68, -208.662, +00024, and -6011 (expressed in decimal). The preceding PUNCH and FORMAT statements would punch those values into a card, as illustrated by Figure 4-41.

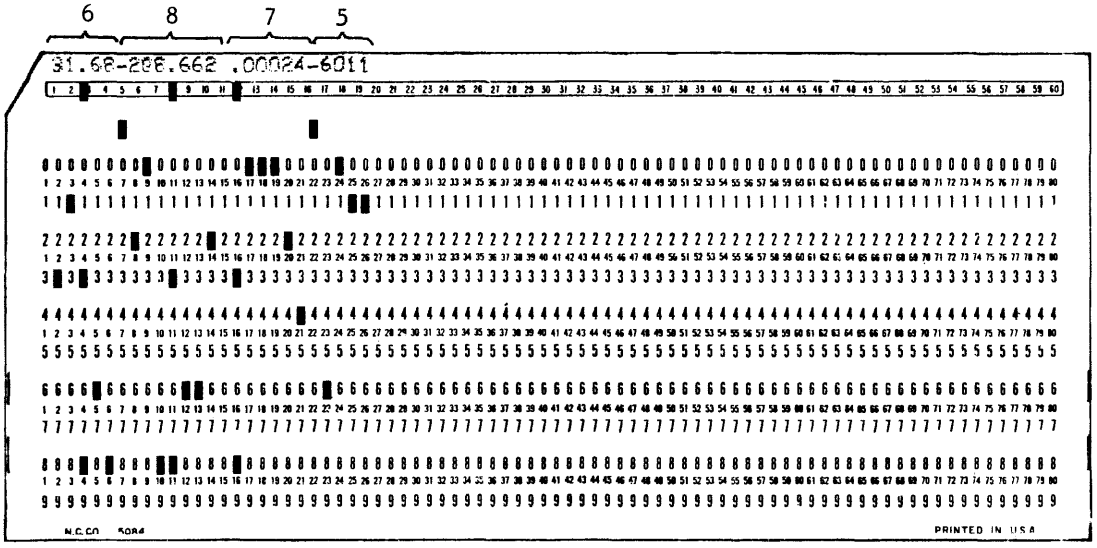
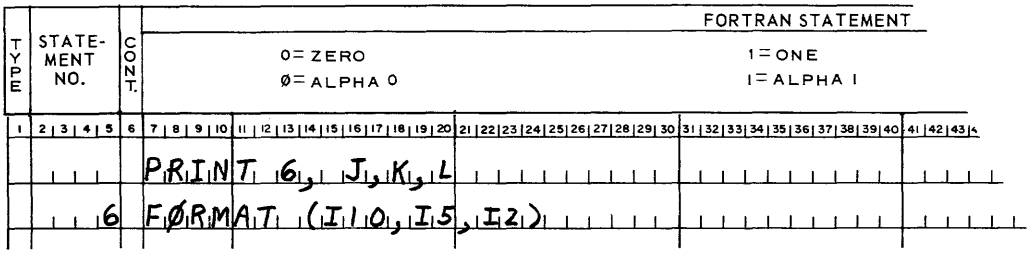


Figure 4-41. Results of Punch Statement

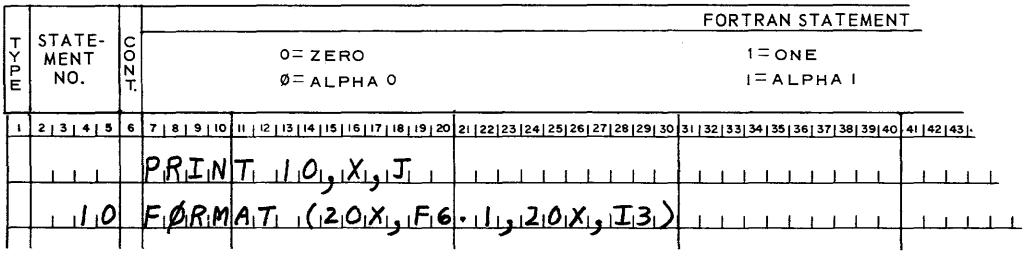
Consider an example using the Print statement. If J = -307767, K = +32, and L = +4, the statements



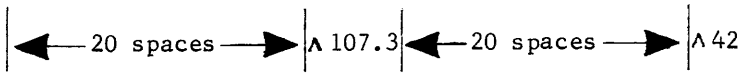
would cause the line printer to print:

|^^^-307767^^^32^4 (^ indicates space)

Two other field specifications give added flixibility in reading or printing data. The "X", or skip character specification, is used to skip characters during input, or to enter blanks during output. During input, nX indicates that the next "n" characters coming in by some input equipment are to be ignored. During output, nX indicates that "n" blanks are to be inserted in the output record before the next variable is moved from memory. If X = 107.3 and J = 42, the FORTRAN statements:

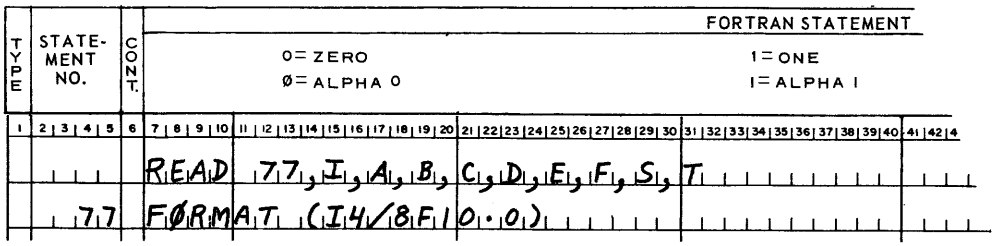


should result in the following printout.



The / or slash specification, is used to a skip line, a card or a magnetic tape records. During input, the slash indicates that the rest of the record or card is to be skipped and that input is to proceed with the next record or card. During output, the slash begins a new line, record, or card, depending upon the output equipment being used.

As an example, the following statements would read two cards: a 4-digit integer from the first card and 8 floating point numbers from the second card.



CONTROL STATEMENTS

The Go To Statement

The GO TO statement provides a means of transferring control to some statement other than the next one in sequence. The statement takes the form of GO TO N in which N is the number of another statement anywhere in the program. When this statement is encountered in a program, the next statement executed is the one specified by the statement number. In the following example,

```
      GO TO 76
   33 B = 3.**2/7.61
   76 A = 4.2*4.**6
```

statement 33 is skipped and statement number 76 is executed following the GO TO statement. The GO TO statement provides a way of altering the program unconditionally by being compiled as an Unconditional Jump in machine language.

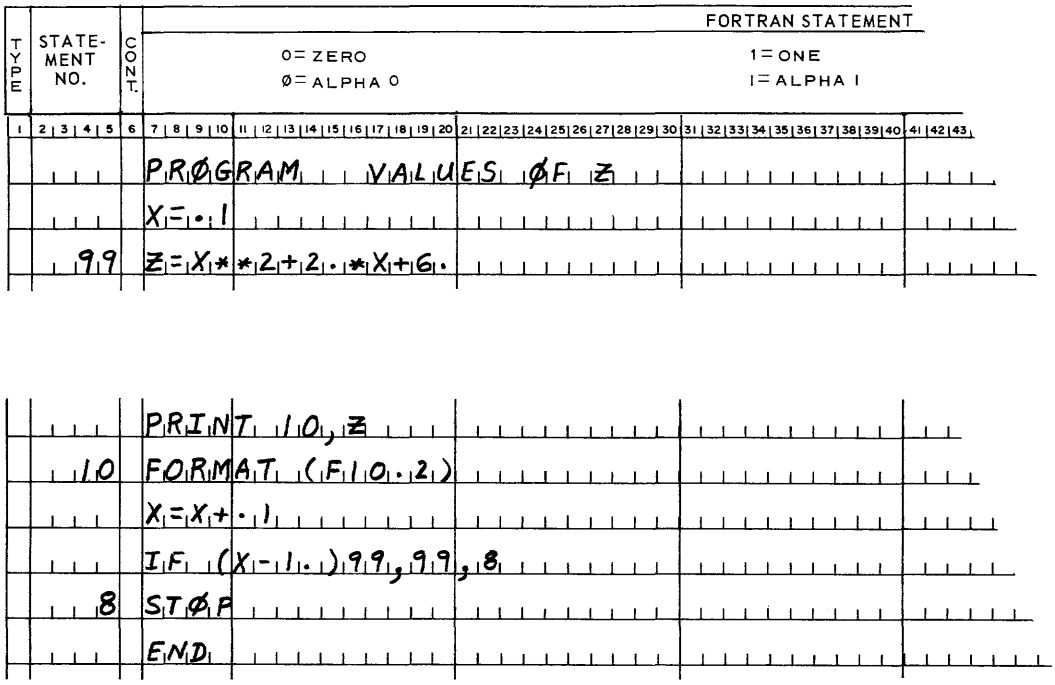
The IF Statement

The IF statement provides for a conditional transfer of control based on data or computed results. The general form of the IF statement is: IF (e) N1, N2, N3, where e represents an expression and N1, N2, N3, are statement numbers. If the value of e is negative, control is transferred to N1; if the value of e is zero, control is transferred to N2; and if the value of e is positive, control is transferred to statement N3. The IF statement is compiled as a machine language AZJ instruction. The following example illustrates the usefulness of the IF statement.

Assume an order to print 10 values of Z as X varies from .1 to 1. in the following equation.

$$Z = X^2+2X+6$$

The FORTRAN program could be written as follows:



The STOP Statement

The Stop statement causes a stop in the execution of the object program. This statement takes effect only when executing the object program, and does not cause the termination of compilation. After a Stop statement the computer cannot continue with the same program. This statement can be of the form STOP or STOP N where N is the octal number displayed in the accumulator after the statement is executed.

The PAUSE Statement

The Pause statement causes a temporary stop during program execution. This is useful, for it enables the operator to see how long it takes to get to this point in the program. In order to restart the program execution, it is necessary to press the Run switch and program execution continues with the statement following PAUSE. This statement can be of the form PAUSE or PAUSE N where N is an octal number displayed in the accumulator after execution of the Pause statement.

The END Statement

The End statement is used as a signal to the compiler that the end of the program has been reached. The last statement of every program must be an END, otherwise

the program is not compiled. The End statement marks the end of the source program and tells the compiler to complete the production of the object program.

The CONTINUE Statement

The Continue statement is a do-nothing instruction. Control passes to the next sequential program statement.

After the FORTRAN program is written on the FORTRAN coding form, it is keypunched and becomes the source deck. Figure 4-42 illustrates the same turkeys and sheep problem discussed in the assembly language section, coded on the FORTRAN coding form. Figure 4-43 illustrates the source deck for the same problem. Compilation is much faster in integer mode arithmetic than in floating point and, whenever possible, integer mode arithmetic should be indicated. Sheep and turkeys are normally expressed in multiples of whole units. The J preceding each variable specifies integer mode arithmetic.

CONTROL DATA FOR' 'N CODING FORM

PROGRAM TURKEYS AND SHEEP NAME

ROUTINE DATE PAGE OF

TYPE	STATEMENT NO.	CONT.	FORTRAN STATEMENT (0 ZERO 0 ALPHA 0 / 1 ONE 1 ALPHA 1 / 2 TWO 2 ALPHA 2)	SERIAL NUMBER
			PRØGRAM TURKEYS AND SHEEP	
			JSHEEP=1	
	1		JTURKEYS=JSHEEP*2	
			IF(JSHEEP*5+JTURKEYS*3-99)11,10,999	
	11		JSHEEP=JSHEEP+1	
			GØ TØ 1	
	10		PRINT 2,JTURKEYS,JSHEEP	
	2		FØRMAT(2I10)	
			STØP	
	999		STØP 77	
			END	

AA252 REV 3 66

Figure 4-42. Turkeys and Sheep Program

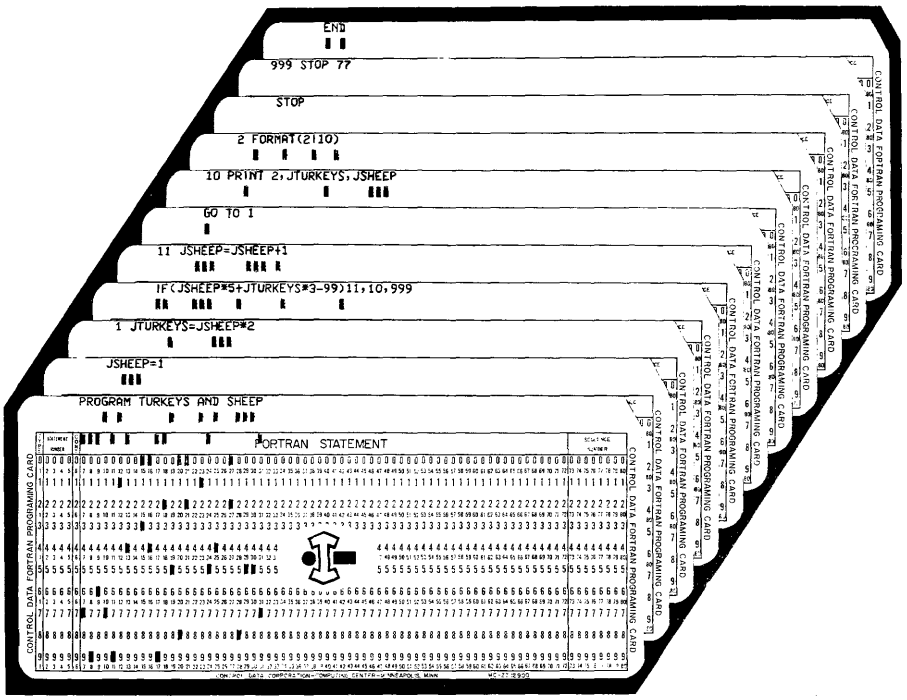
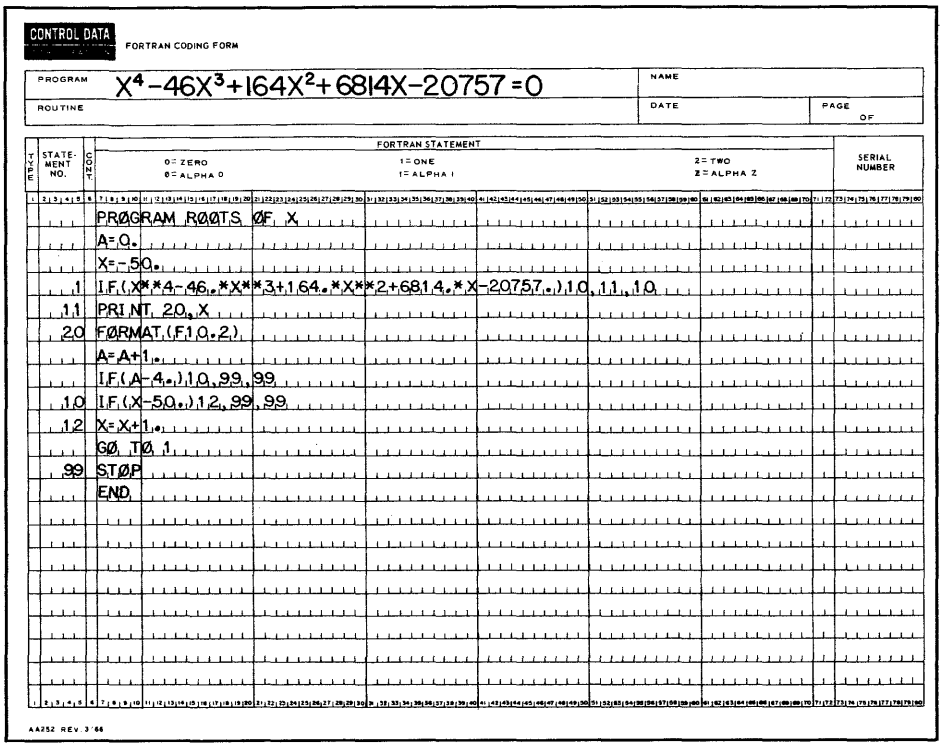


Figure 4-43. FORTRAN Source Deck

One program previously coded into assembly language was the one to find the four roots of X in the equation $X^4-46X^3+164X^2+6814X-20757 = 0$. Figure 4-44 illustrates how that problem could be expressed in FORTRAN. Although the problem could be more efficiently expressed, the following format more closely resembles the one used in coding the problem in assembly language.



CONTROL DATA FORTRAN CODING FORM

PROGRAM $X^4-46X^3+164X^2+6814X-20757=0$ NAME

ROUTINE DATE PAGE OF

TYPE	STATEMENT NO.	CONT	FORTRAN STATEMENT (0=ZERO O=ALPHA O; 1=ONE I=ALPHA I; 2=TWO Z=ALPHA Z)	SERIAL NUMBER
			PRØGRAM RØØTS ØF X	
			A=0.	
			X=-50.	
	1		I F (X**4-46.*X**3+164.*X**2+6814.*X-20757.) 10, 11, 10	
	11		PRINT 20, X	
	20		FØRMAT (F10.2)	
			A=A+1.	
			I F (A-4.) 10, 99, 99	
	10		I F (X-50.) 12, 99, 99	
	12		X=X+1.	
			GØ TØ 1	
	99		STØP	
			END	

AA252 REV. 3 '66

Figure 4-44. Roots of X Program

A very practical computer application is demonstrated by the following listing. The FORTRAN program calculates income tax deductions for varied salary levels and exemptions.

The first section of the listing is a reproduction of the source deck statements. The machine language program generated by the compiler is illustrated next and, finally, the actual printed tax table generated by the program is shown. Notice that the tax table has a format error--the exemptions are not centered over the columns. How would you modify the program to correct the error?

INCOME TAX PROGRAM

```
      PROGRAM RLI
      DIMENSION TAX(6)
      PRINT 92
   92 FORMAT(1H0)
      PRINT 103
  103 FORMAT(55X, 10HEXEMPTIONS)
      PRINT 79
   79 FORMAT(37X,1H1,10X,1H2,10X,1H3,10X,1H4,10X,1H5,10X,1H6,/)
      A=5000.
   76 I=1
      E=1.
   13 TAX(I)=.22*(A-600.*E)
      I=I+1
      IF(I-6)3,3,7
    3 E=E+1.
      GO TO 13
    7 PRINT 1, A, TAX
    1 FORMAT (21X,7F10.2)
      IF(A-10000) 33,17,17
   33 A=A+ 250
      GO TO 76
   17 STOP
      END
```

ACTUAL RESULTS OF PROGRAM

	EXEMPTIONS					
	1	2	3	4	5	6
5000.00	968.00	836.00	704.00	572.00	440.00	308.00
5250.00	1023.00	891.00	759.00	627.00	495.00	363.00
5500.00	1078.00	946.00	814.00	682.00	550.00	418.00
5750.00	1133.00	1001.00	869.00	737.00	605.00	473.00
6000.00	1188.00	1056.00	924.00	792.00	660.00	528.00
6250.00	1243.00	1111.00	979.00	847.00	715.00	583.00
6500.00	1298.00	1166.00	1034.00	902.00	770.00	638.00
6750.00	1353.00	1221.00	1089.00	957.00	825.00	693.00
7000.00	1408.00	1276.00	1144.00	1012.00	880.00	748.00
7250.00	1463.00	1331.00	1199.00	1067.00	935.00	803.00
7500.00	1518.00	1386.00	1254.00	1122.00	990.00	858.00
7750.00	1573.00	1441.00	1309.00	1177.00	1045.00	913.00
8000.00	1628.00	1496.00	1364.00	1232.00	1100.00	968.00
8250.00	1683.00	1551.00	1419.00	1287.00	1155.00	1023.00
8500.00	1738.00	1606.00	1474.00	1342.00	1210.00	1078.00
8750.00	1793.00	1661.00	1529.00	1397.00	1265.00	1133.00
9000.00	1848.00	1716.00	1584.00	1452.00	1320.00	1188.00
9250.00	1903.00	1771.00	1639.00	1507.00	1375.00	1243.00
9500.00	1958.00	1826.00	1694.00	1562.00	1430.00	1298.00
9750.00	2013.00	1881.00	1749.00	1617.00	1485.00	1353.00
10000.00	2068.00	1936.00	1804.00	1672.00	1540.00	1408.00

Remember the five steps required to write a computer program? You were warned that step number 4 would be a "giant step." The discussion, from that warning to this point, has been entirely devoted to the "coding" process.

You learned how to code in machine language, assembler language (COMPASS), and a compiler language (FORTRAN). The turkeys and sheep problem was illustrated in all three languages. From that one problem alone, the advantages of each programming language should be quite evident.

Machine language is fine for short programs that will be used only once and for special maintenance purposes. However, entering a long program in machine language would waste computer time and become quite laborious.

An assembler language has advantages over machine language programming. Instructions are associated with named locations and the mnemonic instruction code is used instead of the numerical code. However, one assembler language statement is still required to generate one machine language instruction.

The compiler offers the most effecient method of programming. With this language, the programmer is not concerned about storage locations. In fact, the programmer isn't even concerned about which machine language instructions are generated by the compiler. A compiler language saves valuable programming time by accepting problems written in standard scientific notation.

The final step is accomplished after the program has been written and run on the computer. Debugging is done to your program, not to the computer. Any computer will operate reliably for long periods of time--if it is properly instructed. If not, it becomes confused and bewildered, and supplies vast amounts of incorrect information. The following procedures are to provide an insight that would enable you to debug your programs (if required).

PART IV

DE-BUGGING TECHNIQUES

INTRODUCTION

Murphy's rule: If it is at all possible for something to go wrong, it will! Nowhere is this rule more applicable than in programming.

Several years ago it was common practice to de-bug programs, no matter how long or how complex, by simply presenting the previous deck of newly punched cards to the computer operator, crossing the fingers, closing the eyes, and glowing inwardly in hopeful anticipation of the great event which was about to rock the programming world. The perfect program! On rare occasions when a program actually would run perfectly on the first try, the lucky programmer was presented with the "award of the silver wire" for distinguished achievement. Although computers and de-bugging techniques have changed, the old habits of the silver wire era are unfortunately still with us.

The intention of this section is to outline a practical approach to program de-bugging, an approach based on improved techniques and experience rather than being entirely scientific or entirely artistic.

GENERAL THOUGHTS AND OBSERVATIONS

Program errors may be classified in two areas--those that are a result of improper coding formats and those that are caused by an improper sequence of commands.

Improper sequence errors are more difficult to detect and may occur in any level of programming. Errors caused by improper coding format occur only when a programming aid such as an assembler or a compiler is being utilized.

FORMAT ERRORS

Assemblers and compilers are designed to allow the programmer to concentrate more on the problem and less on computer-orientated operations. The software translates mnemonic coded instructions (assembler) or formulas (compilers) into machine language instructions. As a program is assembled or compiled, the software is looking for pre-established commands to determine what operations are to be accomplished. If the programmer does not follow the pre-established

formats, the software cannot interpret the commands and format errors result. These errors are detected by the software and printed out as the source program is processed. A few years ago, software systems were designed to stop when an error was detected. That method was abandoned because each stop tied up the computer and wasted expensive time. As programs became more sophisticated, de-bugging routines became more comprehensive. The present philosophy is to process the entire source program and detect all errors on the same run. All diagnostics are then printed on the assembly listing (Figure 4-37). The source program can then be de-bugged to correct all errors. The second run should then be format error free.

SEQUENCE ERRORS

Sequence errors occur because the programmer failed to maintain program continuity or because some of the idiosyncratic traits of a particular computer were overlooked. For example, the product is formed in the QA registers after a multiply whereas the dividend must be in the AQ registers before a divide. Some computers assume the operand to be in the QA registers for both the multiply and the divide instructions.

Assemblers and compilers are now written to include various de-bugging aids, conversion routines, and certain corrective features. As you become more familiar with software, many of these features may be employed to facilitate program de-bugging.

INFLUENCE OF PROGRAM STRUCTURE

A particular program can normally be written several different ways and still accomplish the same task. If the programmer does have such latitude, he should strive to write the program to facilitate the de-bugging that follows as inevitably as "death and taxes". If the program is of the straight-line type, it should be divided into distinct segments that can be de-bugged individually. An error in one segment may then be detected before it is propagated throughout the program. If the program contains iterative loops or sub-routines, pseudo instructions may be inserted to provide program continuity and the sub-routines extracted for individual de-bugging.

A program containing fault detection branches may be validated by purposely inserting "bad" data. Otherwise, the program may appear faultless even though some branches contain errors.

The popular programming trend is to attain a high degree of sophistication. However, such programs are more difficult to de-bug which often requires excessive amounts of time. A good rule is to sophisticate a program only to the extent that time and available memory dictate. Trick programming may be of academic interest but is a form of "teasing a barking dog".

All programs should be fully documented to explain the intent and function.

Each operation should be explained to provide a chronological picture of what is being attempted. Software systems are now so extensive that they are written in segments by a team. Frequently, members of the team are physically separated and not in close communication. By fully documenting the program, other team members can more easily determine what the program is supposed to accomplish. Many hours are wasted simply because a departed programmer did not document his "wild" program.

De-bugging is the final stop in the writing of a program. If the preceding steps have been effectively accomplished, the de-bugging process will be minimized. Although the perfect program is the programmer's dream, he should take the "easy to de-bug" approach "just in case" perfection is not attained.

SUMMARY

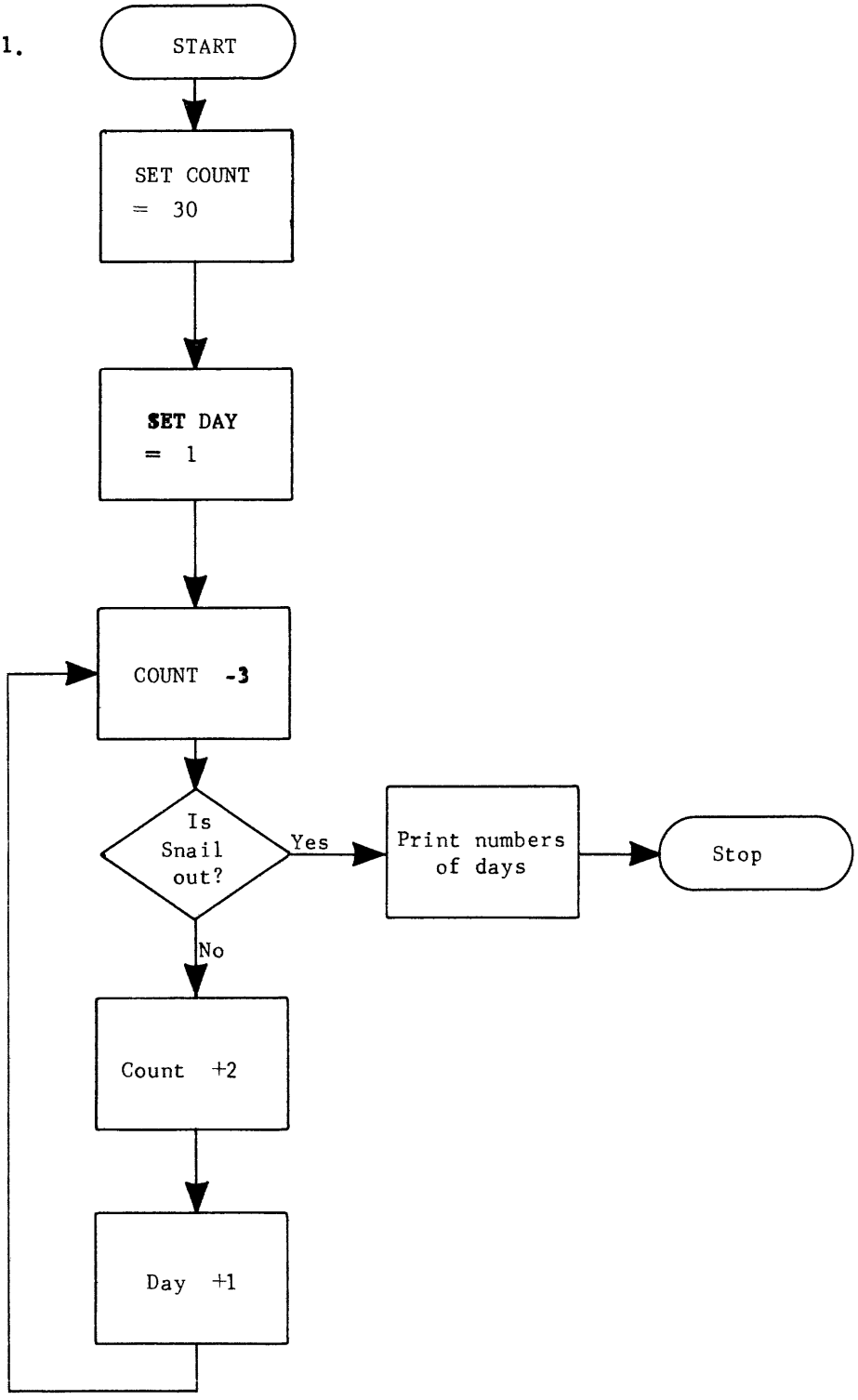
You should now know how to program a digital computer and what steps are required to define the problem, chart the solution, code the instructions, and run the program. If by some remote quirk of fate your program does not run, you should also be familiar with some of the avenues of escape. Familiarity with proper de-bugging techniques will prevent those avenues from becoming dead-end streets.

Volume II explains the function of computer hardware. The text explains what causes a computer to compute, a printer to print, a reader to read,....

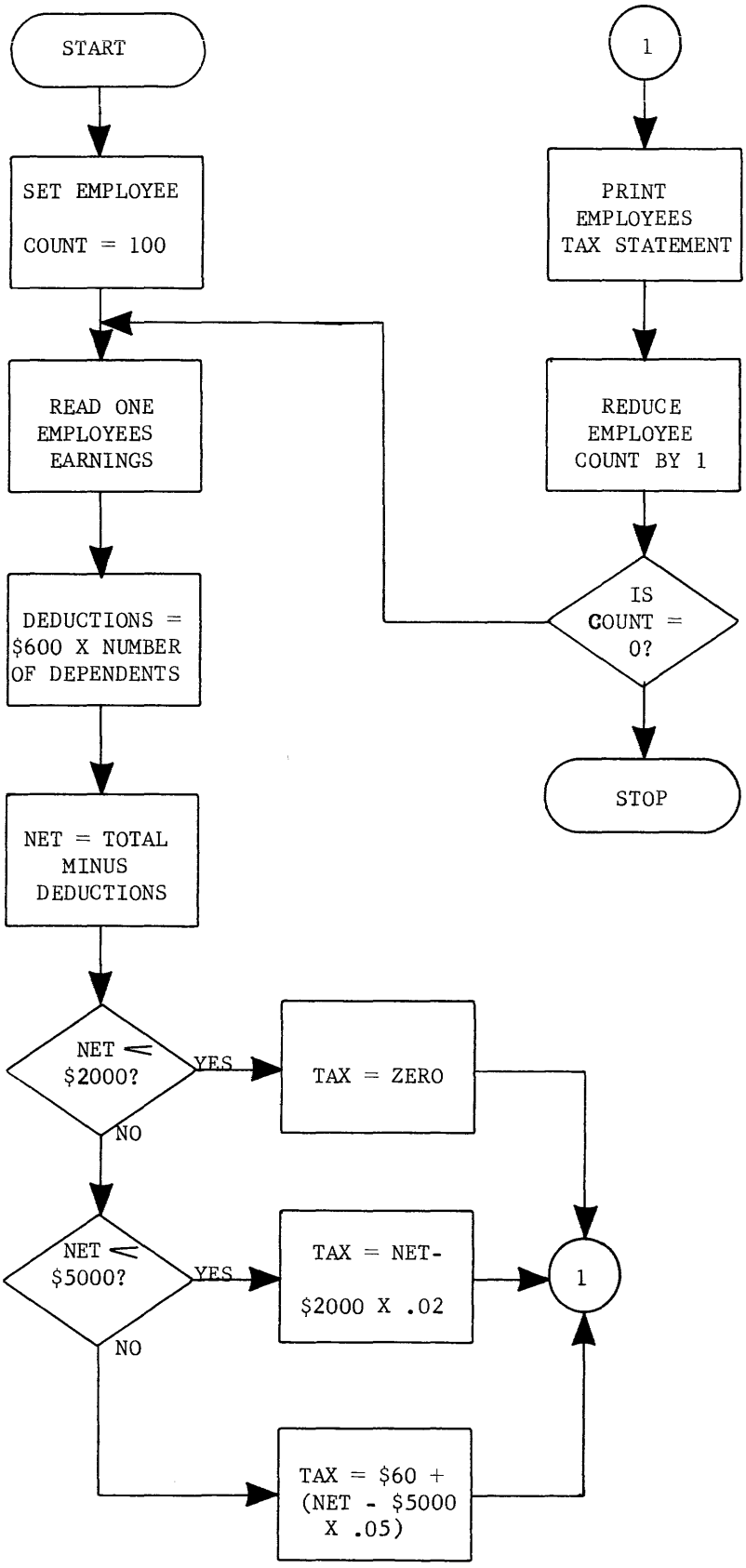
Each section of the computer and the common I/O equipments will be examined. Many more of those mysterious doors will be opened and their innermost secrets revealed.

FLOW CHART SOLUTIONS

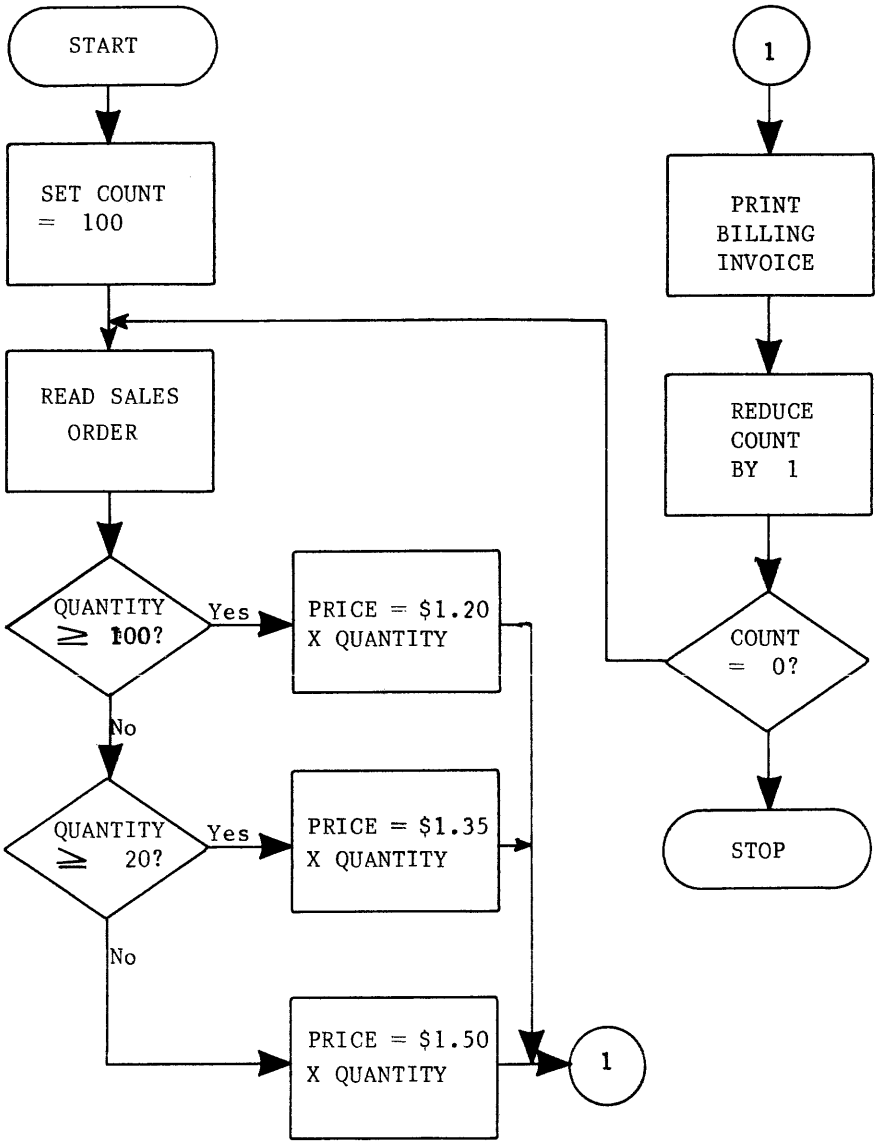
1.



2.


START
SET EMPLOYEE
COUNT = 100
READ ONE EMPLOYEES EARNINGS
DEDUCTIONS = $600 X NUMBER OF DEPENDENTS
NET = TOTAL MINUS DEDUCTIONS
NET ≤ $2000?
YES
TAX = ZERO
NO
NET ≤ $5000?
YES
TAX = NET- $2000 X .02
NO
TAX = $60 + (NET - $5000 X .05)
1
1
PRINT EMPLOYEES TAX STATEMENT
REDUCE EMPLOYEE COUNT BY 1
IS COUNT = 0?
STOP

3.


START
SET COUNT = 100
READ SALES ORDER
QUANTITY ≧ 100?
Yes
PRICE = $1.20 X QUANTITY
No
QUANTITY ≧ 20?
Yes
PRICE = $1.35 X QUANTITY
No
PRICE = $1.50 X QUANTITY
1
1
PRINT BILLING INVOICE
REDUCE COUNT BY 1
COUNT = 0?
STOP

Answers to Practice Problems

4) M = 22222

5) M = 34567

6) M = --1--

7) M = 67777

8) M = 34570 (not 34568)

9) K = 00015, $(A)_f$ = 2000 0001 (shift A left end-around)

10) K = 00004 + (B3) = 77774, $(A)_f$ = 0000 0500 (shift A right end-off)

11) K = 77766 + (B2) = 00003, $(A)_f$ = not affected, $(Q)_f$ = 0000 0004 (Q left)

12) K = 00001 + (B3) = 77771, $(A)_f$ = not affected, $(Q)_f$ = 7740 0000 (Q right)

13) K = 00014, $(A)_f$ = 5000 4000, $(Q)_f$ = 0000 0000 (AQ left)

14) K = 00003 + (B3) = 77773, $(A)_f$ = 0000 0240, $(Q)_f$ = 0200 0000 (AQ right)

15) Program results

00100	ENA,S	Enters A register with 00 0 00050
00101	ENQ	Enters Q register with 00 0 77777
00102	ENI	Enters B3 with 12345 (15 bit register)
00103	ENI	Enters B2 with 11225
00104	ENI	Enters B1 with 34567
00105	ADA	Adds (00114) to (A) and leaves sum (00 0 00115) in A
00106	MUA	Multiplies (M) by (A), product of (+115)x(-3) to QA (Q = 777 77777, A = 777 77430)
00107	SHAQ	K = 00030, shifts AQ left and exchanges register contents
00110	DVA	Divides -347 by -6 (contents of location 00116) (A = 000 00046, Q = 777 77774)
00111	SHQ	Shifts (Q) left 21_{10} places. Q = 477 77777
00112	SBA	Indirectly addresses location 00117 for subtrahend. After subtraction, (A) = 000 00146

00113 HALT Stops program execution

00114 Contains addend for instruction at address 00105

00115 Contains multiplicand for instruction at address 00106

00116 Contains divisor for instruction at address 00110

00117 Contains subtrahend for instruction at address 00112

00120 Contains new lower 18 bits (0 00 115) for instruction at address 00106

00121 Contains new lower 18 bits (00117) for subtract instruction at address 00112

P = 00113 address of last instruction

F = Halt instruction (not yet explained)

B1 = 34567

B2 = 11225

B3 = 12345

A = 00 00 01 46

Q = 47 77 77 77

16) form the sum of four operands in the A register, enter zeros into Q, interchange contents of A and Q, divide by 4, and store answer and remainder in consecutive storage locations.

17) The sum of the four memory locations would now be 7777 7707 (-70) yet the Enter Q instruction enters Q with all zeros. When (A) and (Q) are interchanged, the register contents would be $0000^{A}0000 \quad 7777^{Q}7707$. Because the A register determines the sign of the dividend, the divide instruction would assume that the divident (AQ) was equal to positive 7777 7707 instead of -70. To achieve the correct results, the Q register should have been entered with 77777777. The dividend would then have been interpreted as -70 and the correct answer would have been derived.

18) If the sign of the A register could have been examined to detect negative operands, Q could be correctly entered with either zeros or ones.

NOTE:

The register sensing instructions needed for the preceding program will be explained in the next group of instructions.

19) A Shift AQ instruction (left 30_8 places) between the multiply and divide instructions would be the only requirement. A multiply forms the product in QA and would condition the (Q) register.

20) Not enough information. AZJ,EQ examines entire contents of A

21) Yes. (A) = -1 which is $\neq$ zero

22) No. (A) actually represents the quantity -37777777, less than zero.

23) Yes. Negative zero is less than positive zero only because the sign bit alone is being examined.

24)

a. Enter B3 with 33000, B2 with 77776

b. Obtain indirect address (M = (B2) + 00105 = 00104)

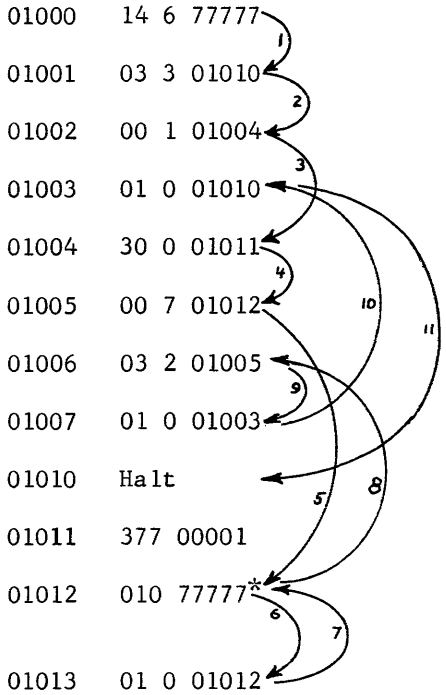
c. Indirect address storage location 00104 and transfer lower 18 bits to F register. Instruction now becomes 01 7 45102.

d. Obtain new indirect address (M = (B3) + 45102 = 00103)

e. Indirect address from storage location 00103. Lower 18 bits to F register. Instruction now becomes 01 3 45670

f. Perform address modification to obtain jump address 00671

25)



* This instruction is not executed on the first pass

26) A two instruction subprogram exists at addresses 01012 and 01013.

27) The program executed 11 instructions. The instruction at address 01012 was not executed on the jump from 01005 (return address (P) + 1 written in) but was executed on the jump from 01013.

28) Only address 01011. The operand at 01000 was actually part of the instruction.

29) 00100 20 0 00205

00101 30 1 00204

00102 10 5 77773

00103 01 0 00101

00104 HALT

On the first pass, (B1) = 00000 (cleared by M/C) and (00204) are added to (00205) contained in A. The ISD compares (B1) 00000 with y (77773) and, because they are not equalities, decrements (B1) to 77776. The instruction at P + 1 causes the jump back for the second pass.

The ADA instruction has an address of 00203 (M = 00204 + 77776) and the third operand is added.
Finally, on the fifth pass, the last operand, from address 00200, is added (M = 00204 + 77773 = 00200). The ISD instructions finds equalities between (B1) and y, clears B1 and skip exits to the HALT instruction at P + 2.

30) A statement that specifies an operation and the values or locations of the operands.

31) One

32) F register

33) The ENA is an interregister transfer into A whereas the LDA obtains the operand from storage.

34) Original contents are destroyed

35) LOAD indicate transfer of from storage <u>to</u> a register; STORE is a transfer of data <u>from</u> a register to storage.

36) (A) final would equal 0000 0004

37) 0000 0003, 00000003

38) Instructions. Data (operands)

39)

40) A. Shift A left

B. Shift A right

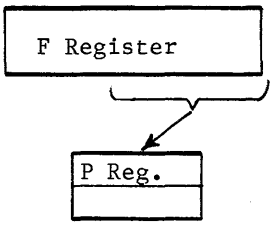
C. Enter A (extended)

D. Load Q

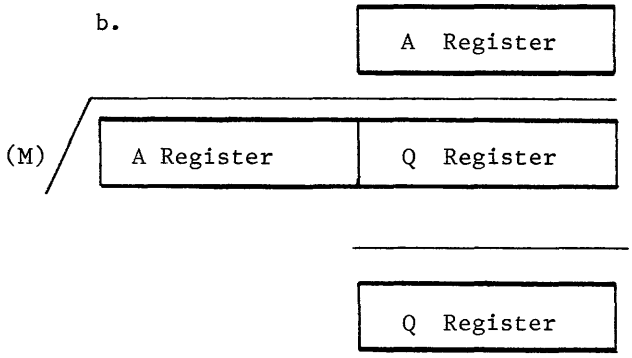
E. Store A

F. Enter Index

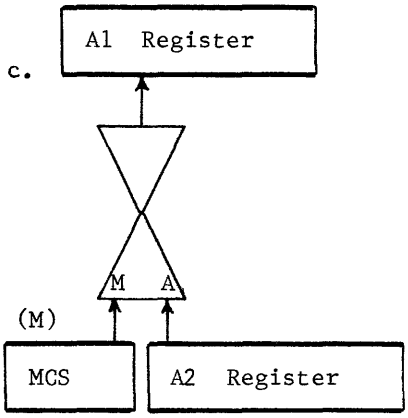
41) a.



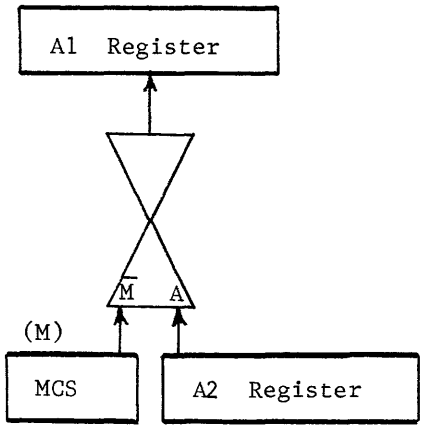
b.



c.



d.



42) (A) = 000 00360

43) (A) = 000 77760

44) (A) = 000 02502 (Enter A, not Load A)

45) (A) = 000 13003 (Q) = 000 00002

46) (A) = 777 66004 (Q) = 76543210

47) (A) = 0000 0006 (Q) = 0000 0004

48) (A) = 0000 0006

49) (A) = 00000760
(Q) = 02600177
(P) = 03005

Note: The Store A instruction placed 00000760 in storage location 03005. The 000 XXXXX is a halt instruction.

50) (A) = 00000004

Did you get Octaphobia? Try octal arithmetic.

51) (A) = 77777617

(A) = 00000001

a. The product is +341 (00000000 00000341) is QA

b. The shift interchanges (A) and (Q) prior to the divide

c. The divisor is +341, dividend is -2, answer is -160, and the remainder is +1 (Answer in A, remainder in Q).

52) (A) = 7777 7776
(P) = 11010

53) (A) = 00017526
(P) = 06004
(B1) = 00000

clear when (B^b) = y

54) (A) = 7777 7775
(Q) = 0000 0002
(F) = 000 07000
(P) = 07012
(B1) = 00001

Don't forget address modification.

(B1) = 00001 after the first instruction.

55) How do you express eights and nines in a radix 8 number system?

56) (A) = 7777 5014
(Q) = 0000 0001
(P) = 10013
(F) = 000 40200
(B1) = 00000
(B2) = 00000
(B3) = 00000

57) (A) = 01 0 12010

(Q) = 03 3 12007

(P) = 12002

(F) = 000 12000

58) The A register would contain the contents of storage location 60732.

59) The instruction at 07000 is indirect by addressing itself; therefore, indirect addressing will be indicated each time the new 18 bits are read up.

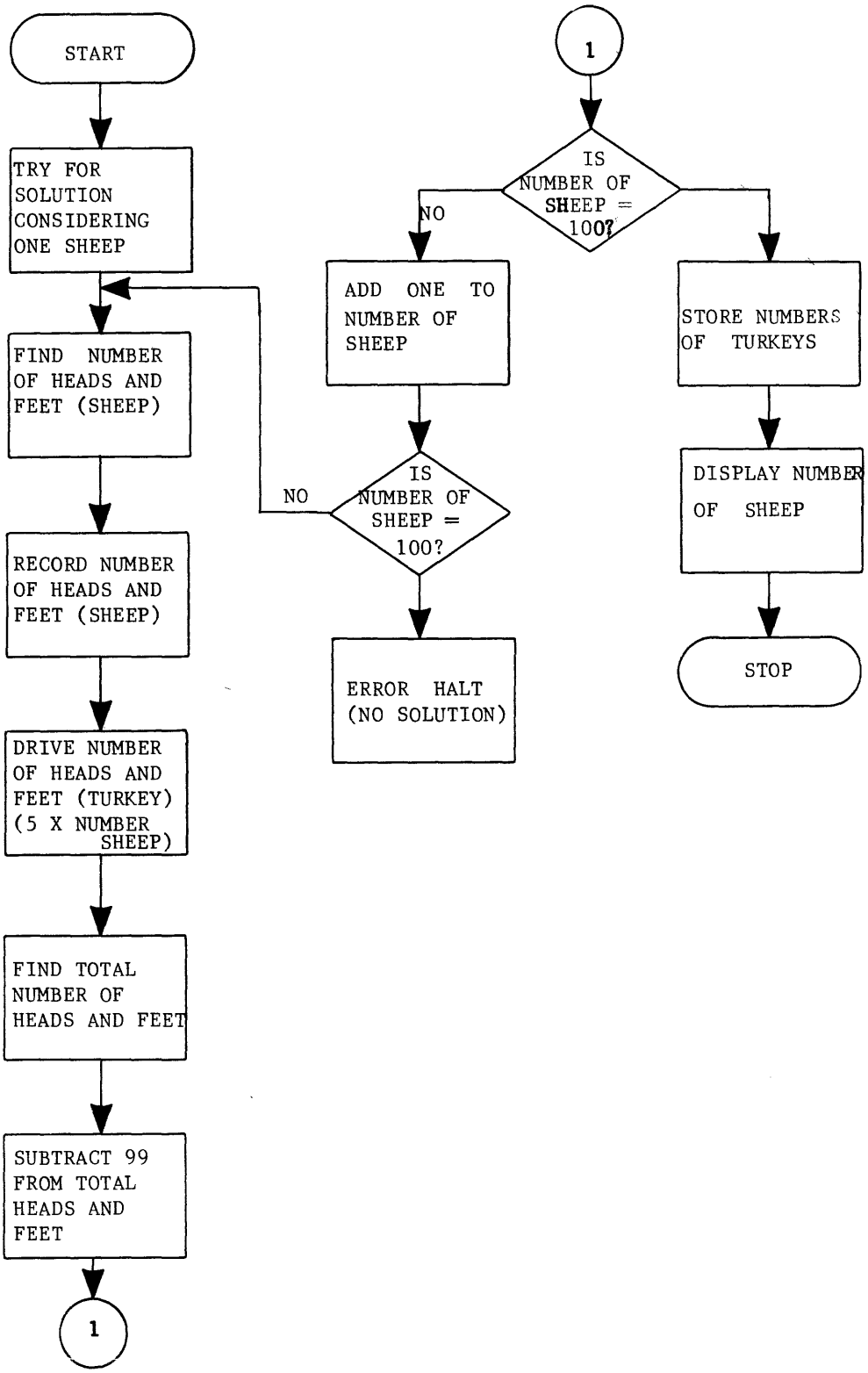
60) (P) final = 13005

(13004) = 010 13013

(13011) = 007 13013

(13013) = 010 13001

61.


START
TRY FOR SOLUTION CONSIDERING ONE SHEEP
FIND NUMBER OF HEADS AND FEET (SHEEP)
RECORD NUMBER OF HEADS AND FEET (SHEEP)
DRIVE NUMBER OF HEADS AND FEET (TURKEY) (5 X NUMBER SHEEP)
FIND TOTAL NUMBER OF HEADS AND FEET
SUBTRACT 99 FROM TOTAL HEADS AND FEET
1
NO
ADD ONE TO NUMBER OF SHEEP
IS NUMBER OF SHEEP = 100?
NO
ERROR HALT (NO SOLUTION)
1
IS NUMBER OF SHEEP = 100?
STORE NUMBERS OF TURKEYS
DISPLAY NUMBER OF SHEEP
STOP

STORAGE LOCATION						INSTRUCTION f		b	m,y,k					COMMENTS
0	0	1	0	0		1	4	6	0	0	0	0	1	Start with one sheep
0	0	1	0	1		4	0	0	0	0	1	2	3	Store number of sheep
0	0	1	0	2		5	0	0	0	0	1	2	7	Find number of heads and feet (sheep)
0	0	1	0	3		4	0	0	0	0	1	2	4	Store number of heads and feet (sheep)
0	0	1	0	4		2	0	0	0	0	1	2	3	Load A with number of sheep
0	0	1	0	5		5	0	0	0	0	1	3	0	Find number of heads and feet (turkeys)
0	0	1	0	6		3	0	0	0	0	1	2	4	Add for total heads and feet
0	0	1	0	7		3	1	0	0	0	1	3	1	Subtract given number of heads and feet
0	0	1	1	0		0	3	0	0	0	1	1	6	If difference is 0, solution has been found
0	0	1	1	1		2	0	0	0	0	1	2	3	Load A with number of sheep
0	0	1	1	2		3	0	0	0	0	1	2	5	Increase number of sheep by 1
0	0	1	1	3		1	0	1	0	0	1	4	4	Provide for no solution
0	0	1	1	4		0	1	0	0	0	1	0	1	Return to try new values
0	0	1	1	5		0	0	0	7	7	7	7	7	Error halt (wrong given number of H+F)
0	0	1	1	6		2	0	0	0	0	1	2	3	Load A with number of sheep
0	0	1	1	7		5	0	0	0	0	1	2	6	Find number of turkeys
0	0	1	2	0		4	0	0	0	0	2	0	0	Store number of turkeys (ANS)
0	0	1	2	1		2	0	0	0	0	1	2	3	Load A with number of sheep (ANS)
0	0	1	2	2		0	0	0	0	0	1	0	0	HALT
0	0	1	2	3										Reserved for current number of sheep
0	0	1	2	4										Reserved for heads and feet (sheep)
0	0	1	2	5		0	0	0	0	0	0	0	1	
0	0	1	2	6		0	0	0	0	0	0	0	2	
0	0	1	2	7		0	0	0	0	0	0	0	5	CONSTANTS
0	0	1	3	0		0	0	0	0	0	0	0	6	
0	0	1	3	1		0	0	0	0	0	1	4	3	
				2										
				3										
				4										
				5										
				6										
				7										


CONTROL DATA
CORPORATION