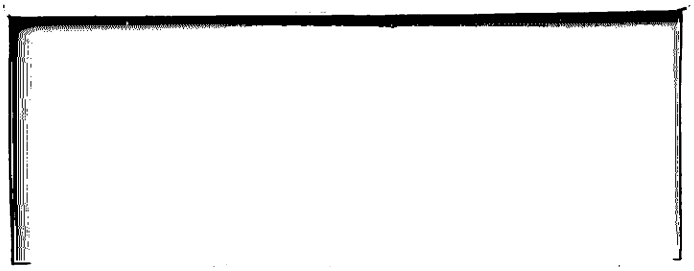
digital

C. J. Karzmark

INTRODUCTION TO PROGRAMMING

the

PDP-11



C. J. Maynard

INTRODUCTION TO PROGRAMMING

the

PDP-11

Donald S. Lawrence, Jr.

DIGITAL EQUIPMENT CORPORATION

Preliminary Printing, May, 1973

(Chapters I-III)

NOTE

This handbook is for information purposes and is subject to change without notice

Associated Documents:

PDP-11 Processor Handbook

PDP-11 Peripherals and Interfacing Handbook

PDP-11 Paper Tape Software Programming Handbook

PREFACE

The primary purpose of this book is to serve as an introduction to the PDP-11 family of computers, although it will meet the needs of a general readership as well. It assumes little or no previous computer experience on the part of the reader, and thus contains introductory information of a general nature and discussion of fundamental concepts in addition to supplying material pertinent to the PDP-11.

It is intended to provide an understanding of computers in general and the PDP-11 family in particular, and to serve as a prelude to more advanced documentation.

Donald S. Lawrence, Jr.

May, 1973

TABLE OF CONTENTS

Chapter 1

INTRODUCTION TO COMPUTERS

1.1 PERSPECTIVE

The transition from man's first desire to count and measure to your PDP-11 computer is indeed a great one, and the reader is heartily encouraged to pursue the fascinating topic of computer history. For the purpose of this text, it is sufficient to say that the development and continuing evolution of the computer has brought about a dramatic change in our lives and promises even greater change in the years to come.

During the last twenty years especially, there has been an explosive proliferation of computing machines designed to meet a vast range of applications. Perhaps of even more importance than the ever-improving developmental technology is the unceasing discovery of new ways in which we may use computers. In fact, it would seem that we have reached the point where the machine capability and the task are present, and it is only our lack of applicational insight that limits us.

1.2 DEFINITION

The majority of everyday users, as well as the novice, view the computer as a "black box." That is to say, they know what is done (the performance characteristics) but not how it is done (the components and/or means of operation). Though this knowledge is superficial, it is often sufficient; rarely is anyone required to fully comprehend all the details of any computer system. It is in fact customary for the individual to accept a "black box" description of a computer or computer function at some level, and then deepen his understanding when motivated by desire or necessity. As indicated in the preface, this text assumes that you presently regard the computer itself as a "black box," and will attempt to bring you beyond that level.

The computer may be defined as a machine, devised and used by man because (like other machines) it can perform certain tasks better than man himself.

Technically, it is an electronic device capable of accepting information, applying prescribed processes to that information, and supplying the results of those processes. Very basically then, the computer can:

(1) accept INPUT (information to be processed)

(2) PROCESS the information (manipulate it in a prescribed way)

(3) produce OUTPUT (the results)

Based on this definition, we may represent the computer by means of the following block diagram (Figure 1-1):

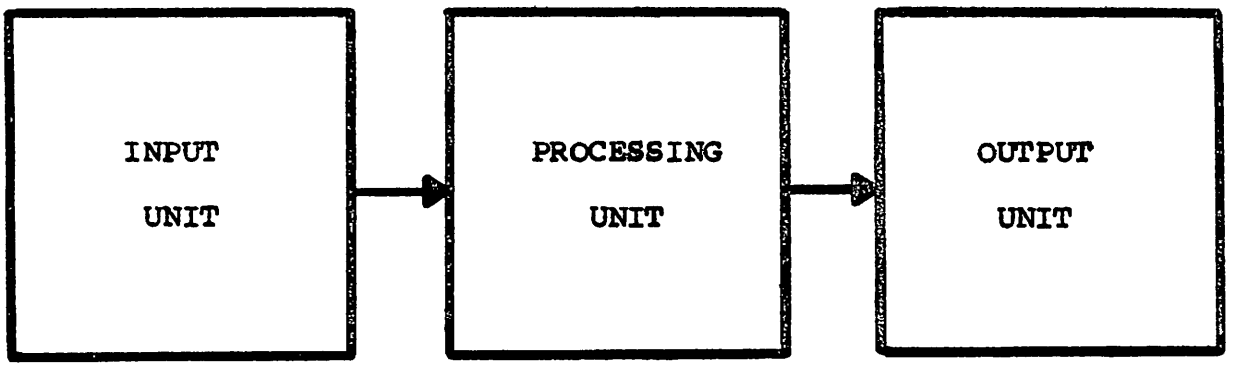


Figure 1-1 Simplified Computer Block Diagram

At this point, our definition and diagram might well encompass other devices, such as the electric adding machine and desk calculator. To differentiate, we will note that the computer posseses two additional (and distinctive) characteristics:

(1) It is capable of manipulating a variety of symbols, and is not restricted to numbers only. It processes data.

(2) It processes automatically, with only initial human intervention required. The sequence of operations to be performed (called the program) is first stored in the computer.

A deeper observation of computer operation will help illustrate these aspects of your Programmed Data Processor.

Let us approach this more detailed representation by using you as an example. You are given the verbal directive, "Mentally add the numbers fifty-four, eighty-seven, and thirteen." After an individually dependent computational pause, you orally respond, "The sum is one hundred and fifty-four."

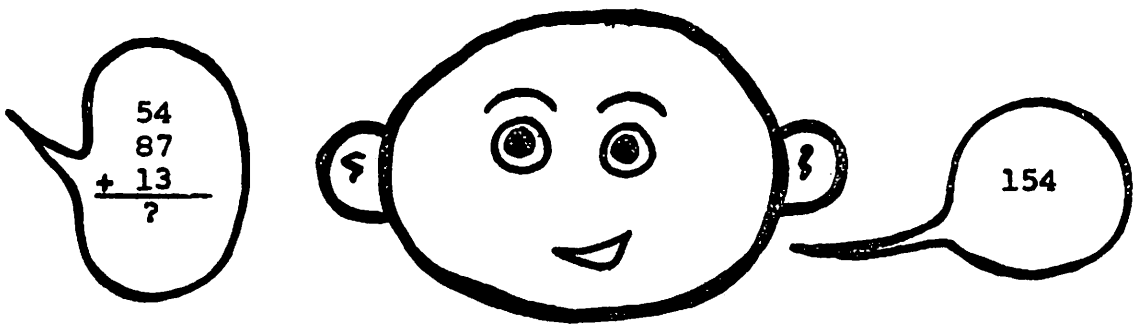


Figure 1-2 Analogous Computer Block Diagram

What has happened? Your aural and vocal anatomy has served as the means of INPUT and OUTPUT respectively; your brain has been used to PROCESS the information. For the present, the terms INPUT and OUTPUT sufficiently describe the operations performed, but the term PROCESS appears to be somewhat obscure. Let us examine what has taken place here a little more closely.

(1) You remembered the values given and called upon skills previously learned and retained - therefore, MEMORY was required

(2) The operations were ordered, with the values being manipulated in a prescibed manner - thus, some element of CONTROL was present

(3) A mathematical calculation was performed - hence, an ARITHMETIC function was involved

We may directly associate these features with units in the basic computer block diagram (Figure 1-3) to complete this general definition.

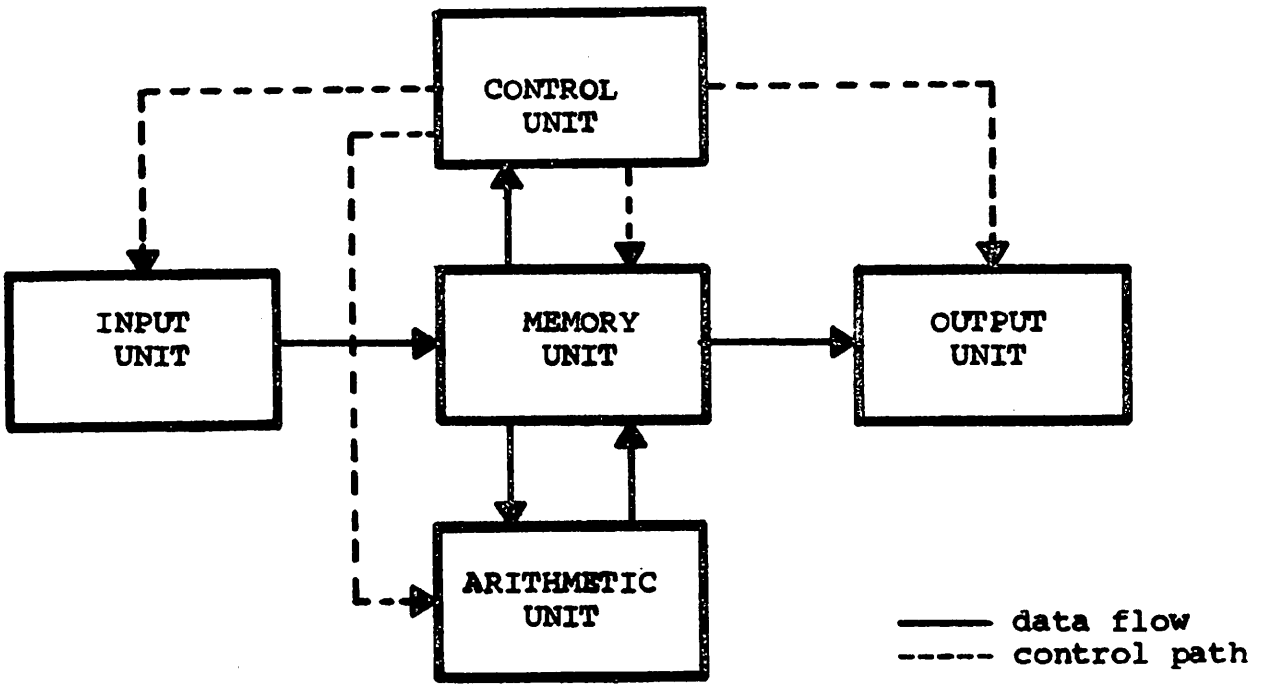


Figure 1-3 Basic Computer Block Diagram

INPUT UNIT - Under the direction of the CONTROL UNIT, it supplies the computer with all the information needed to accomplish a given task; the values to be operated upon (data) and the operations to be performed upon those values (program)

MEMORY UNIT - Contains information for the CONTROL UNIT (program) and the ARITHMETIC UNIT (data); holds intermediate and final results

CONTROL UNIT - Directs the entire process by specifying to the ARITHMETIC UNIT what operations are to be performed, in which order they are to be performed, and where to get/put the data involved

ARITHMETIC UNIT - Under the direction of the CONTROL UNIT, it performs the actual operations; the "working area"

OUTPUT UNIT - Under the direction of the CONTROL UNIT, it records the results of computer operations

1.3 CLASSIFICATION

As specified by the characteristics in Table 1-1, every computer may be basically categorized as either ANALOG or DIGITAL (there are hybrid computers that have both analog and digital properties). We have been discussing (and will continue to discuss) only the digital computer, for your PDP-11 belongs to that class. This comparison is made for reasons of completeness and further definition.

Table 1-1 Comparison of Analog and Digital Computers

ANALOG	DIGITAL
(1) Variable electrical or mechanical quantities used to represent data	(1) Discrete numerical values used to represent data
(2) Varying and continuous level of input yields varying and continuous level of output	(2) Set and discontinuous level of input yields set and discontinuous level of output
(3) Calculates by means of a measuring process	(3) Calculates by means of a counting process
(4) Example: speedometer	(4) Example: odometer

We may also categorize computers according to their design capability as being either SPECIAL PURPOSE or GENERAL PURPOSE, terms that are very self-explanatory. The special purpose machine is constructed to perform one task, or a closely related group of tasks. The single sequence of operations it is to perform (its program) is "built in." If ever a program change becomes necessary, a hardware modification (physical restructuring) is required. Conversely, the general purpose machine is designed to be capable of performing many varied tasks. The many possible operation sequences (programs) are kept in the memory unit of the general purpose computer, and for this reason it is sometimes referred to as a stored program machine. To perform a given task, the user simply calls upon the appropriate program. The change from one task to another is accomplished by selecting another program already in the memory or entering it by means of the input unit.

It should be apparent that previous discussion has been of the general purpose computer, and we will continue to confine ourselves to this type. Your PDP-11 is classed as both digital and general purpose.

Now that you have a basic understanding of what the computer is, let us generally discuss why and in what manner it is used.

1.4 APPLICATION

If given unlimited time to complete a task, factors such as volume of data, complexity of calculation, and degree of accuracy become immaterial. For example, you alone could certainly process the payroll of a large corporation or perform all the calculations necessary to launch a missile. The chance of you accurately doing either in a matter of hours or minutes, however, is rather remote. It is then speed of operation which is the ultimate consideration in both cases. This element of speed, coupled with accuracy and reliability, is the underlying advantage of the computer; it is the major reason for its existence and use.

We have said that the computer is used because it performs certain tasks "better" than man. The interpretation of this term is dependent upon the task, and may imply any combination of the following features: speed, accuracy, precision, reliability, economy, efficiency, feasibility. Where then is the computer to be used? Wherever its attributes enable the task to be done "better."

To list the wide and ever-expanding range of specific applications would be an arduous chore (surely requiring the use of a computer!). If, by way of example, the results of a PDP-11 applicational survey were immediately available to be given here, the variety of response would easily fill the remainder of the book. And this would be for only one computer model of one corporation! Keeping this in mind, we may denote four general areas of application:

Business - Computers used in business applications are usually involved with record keeping; automating the many tedious, repetitious tasks associated with classifying, processing, and maintaining information of all kinds. As a rule, the business computer is required to perform only a few simple calculations, but it must be capable of handling a great volume of data.

Scientific - In the scientific application, the computer is primarily used for problem solving; the repeated evaluation of expressions with different values. It has made practical the extrapolation of immensely complex algorithms. In contrast to the business computer, there is usually a small amount of data involved, but a great deal of calculation.

Control - The capability of the computer to make precise calculations and evaluations at a high rate of speed causes it to be used in control environments ranging from national defense to the industrial production line. Here the computer receives information, uses it in calculations, and based upon the result "decides" what to do as an appropriate response.

Simulation - Any given task may be too dangerous, costly, or intricate for man to attempt. It may not be feasible for him at all. In such situations, the computer is used to simulate all conditions and interactions, yielding knowledge without risk.

Chapter 2

NUMBERS AND OTHER STUFF

2.1 INTRODUCTION

In writing a book such as this, it is very often desirable to explain several things simultaneously. This is one of those times!

The first chapter has defined the general purpose digital computer, and shown that it manipulates data according to a program of instructions. A logical continuation could therefore be a detailed look at the PDP-11 in terms of organization and unit interaction. On the other hand, since we have mentioned the program and indicated its significance, fundamentals of programming could just as reasonably follow. Then too, a discussion of programming languages and data representation might serve as a likely sequel.

In developing any of these topics, however, there is an inescapable involvement with number concepts. Numerical references must be made in describing the PDP-11 and its operation; program instructions and data are ultimately represented in numerical code. This chapter will then concern itself with those number concepts and operations required for you to fully appreciate subsequent discussion of programming the PDP-11.

The subject of computer math, numbers and "other stuff," is interruptive regardless of when it is introduced. For this reason, the reader may wish to move past it for the present and make backward references where necessary.

2.2 NUMBER SYSTEMS

2.2.1 Basic Principles

Man's earliest form of notation was the tally mark, where there existed a one-to-one correspondence between the marker and the object to be counted. The aggregate of the scratch marks, pebbles, or notches was the "number" he wished to record. This principle of repetition proved cumbersome for even moderately large numbers, however, and so there evolved various number systems to meet the increasing demands of civilization.

The number system is a standard means of representing quantity. It consists of a finite set of symbols, called numerals or digits, and rules which specify how the symbols are arranged to form numbers. The early number systems offered an improvement over recording each unit in that they combined unique quantities of units into groups and assigned discrete symbols to represent those groups. They featured the principle of addition, where the value of an entire number is determined by adding the values of the individual symbols that comprise it, irrespective of position. (MMMCCCXXXIII = 1ØØØ+1ØØØ+1ØØØ+1ØØ+1ØØ+1ØØ+1Ø+1Ø+1Ø+1+1+1 = 3333). Though later development introduced subtractive (IV=4) and multiplicative ($\bar{M}$=1Ø,ØØØ) principles, these systems still served primarily for quantification and record keeping, and had little provision for calculation (even the most mathematically adept of Romans removed his sandals for MCVII times CLXXVIII).

The positional number systems which followed contain two additional and distinctive features which greatly simplify the operations needed to manipulate numbers: the concept of position and the inclusion of the zero symbol. Like their early counterparts, these systems have discrete symbols with unique values and follow the principles of addition and multiplication. The major distinction is the principle of place value, which specifies that there is not only a unique value for the symbol but also a unique value for the position. Thus the value associated with a symbol is determined by both its absolute value and the value of its relative position within the number (3333 = 3ØØØ+3ØØ+3Ø+3 = 3333).

The importance of the zero symbol in a positional number system is illustrated by the application of the count and carry (or regrouping) principle. For example, we count from zero to ten in the familiar decimal system as follows:

Ø
1
2
3
4
5
6
7
8
9
1Ø

We cycle through the digits to nine, but do not create another symbol in counting an additional unit. We instead carry the one to the next (tens) place, and record it there. To indicate that there are no units, and to "hold" the units place, we record a zero in that position.

The three number systems that we will discuss in some detail are presented in Table 2-1. These are all positional number systems which demonstrate the principles we have previously mentioned: addition, multiplication, place value, count and carry.

Table 2-1 Digits and Bases of Selected Number Systems

NUMBER SYSTEM	DIGITS	BASE
Decimal	Ø,1,2,3,4,5,6,7,8,9	1Ø
Binary	Ø,1	2
Octal	Ø,1,2,3,4,5,6,7	8

The term base, introduced in the Table, is commonly used to name or describe a number system. The decimal system, for example, is often referred to as the base ten system. For any positional number system, the base (or radix) is the number of digits it contains.

2.2.2 Decimal Number System

One of the few assumptions made in this text is that you are familiar with the decimal number system. It is the mathematical language of the "real world;" a language that you use on a daily basis. You have memorized the rules and operational procedures to the point that they are automatically applied, and performing any calculation is straightforward. The purpose of this chapter is to have you become equally well acquainted with the binary and octal systems. We will briefly reintroduce the decimal system here in relation to our previous discussion of basic principles, and later reference it to help illustrate those aspects it has in common with the less familiar systems.

The decimal or base ten number system is comprised of the digits zero through nine (Ø,1,2,3,4,5,6,7,8,9). It is a positional number system, so that in progressing from right to left within a decimal number, the value associated with each position is an increasing power or multiple of the base. This place value principle is presented in Table 2-3.

Table 2-2 Powers of Ten

10^{0}	=	1	=	1
10^{1}	=	10	=	10
10^{2}	=	10x10	=	100
10^{3}	=	10x10x10	=	1,000
10^{4}	=	10x10x10x10	=	10,000
10^{5}	=	10x10x10x10x10	=	100,000
10^{6}	=	10x10x10x10x10x10	=	1,000,000

Table 2-3 Positional Notation with Powers of Ten

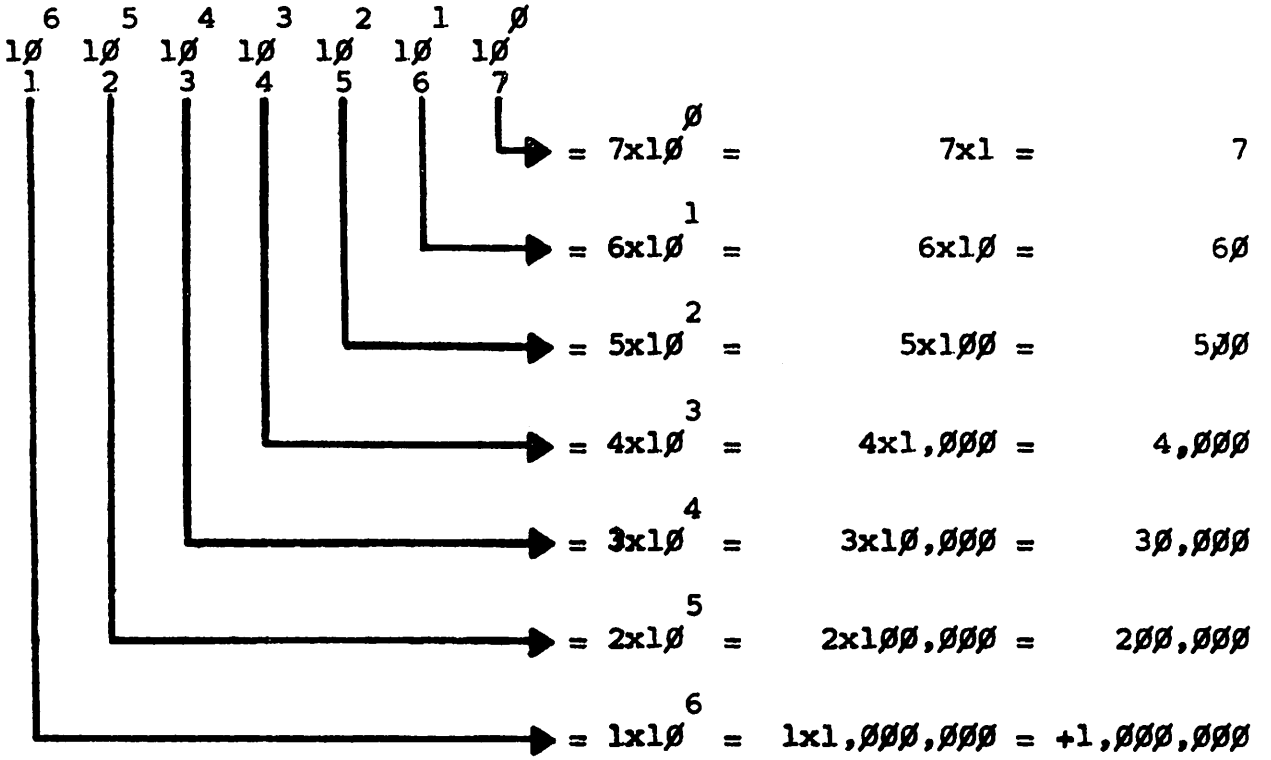
Powers of Ten

10^{6}	10^{5}	10^{4}	10^{3}	10^{2}	10^{1}	10^{0}
1,000,000	100,000	10,000	1,000	100	10	1

Place Values

As shown in Table 2-4, determining the total value of a decimal number is accomplished by applying the principles of place value, multiplication, and addition: multiplying the discrete value of the digit by the value of the position in which it is placed, and then adding the resulting products.

Table 2-4 Decimal Number as the Sum of Powers



The count and carry principle, to be examined in more detail when we later discuss arithmetic operations, can be simply illustrated by counting or addition. As evidenced by the example below, presented earlier in discussing basic principles of positional number systems, we see that the terms count and carry are quite self-descriptive; count until the base is equaled, and carry that indication to the next column.

Ø
1
2
3
4
5
6
7
8
9
carry: 1 Ø

When performing addition, the procedure is as follows: (1) Add the digits in the column, (2) If the base is neither equaled nor exceeded, record the sum; (3) If the base is equaled or exceeded, divide by the base, record the remainder, and carry the quotient to the next column.

carries:	1	1	1		1	
		5	1	4	5	9
	+	7	8	6	2	4
	1	3	Ø	Ø	8	3

Note the presence of the zero symbol in the sum, indicating "no hundreds" and "no thousands," and also "holding" those places within the number.

2.2.3 Binary Number System

The binary or base two number system is comprised of only two digits, zero and one, commonly referred to as bits (binary digits). As illustrated in Figure 2-1, this system is capable of representing but two conditions, and thus lends itself to the decision-making process; ideally practical for the computer.

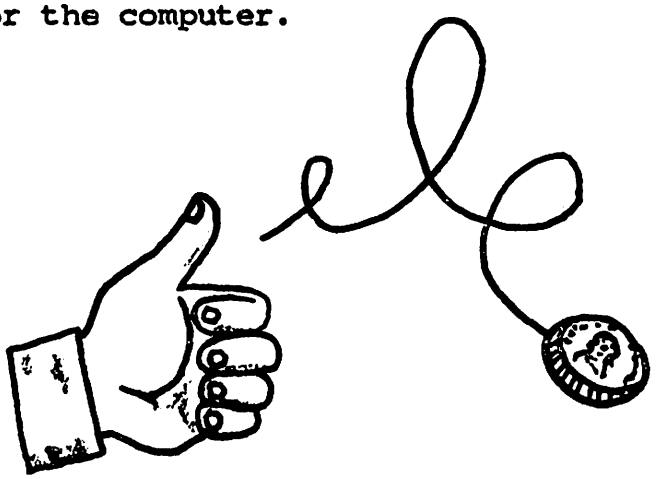


Figure 2-1 Popular Binary Device in Action

Since physical and electrical entities have but two states (i.e., switch open/switch closed, current/no current), internal components of a computer can be easily designed to accommodate data in binary form. Computers have been built to operate with other number systems, but the increased number of digits along with the proportionally increased number of possible conditions make these computers overly complex in design and difficult to manufacture. For this reason, the PDP-11 and a majority of computers operate with the binary number system, considered the "language of the computer."

Though the computer works internally with the binary number system, this does not mean that all information input must be so represented. In fact, rarely is the data initially in binary form. If strictly numerical, it is generally octal or decimal, but it is even more commonly expressed in one of many alphanumeric computer languages. As we will later discuss, there are several methods by which information in any of these forms is converted to binary before it is processed by the computer.

We noted earlier that the binary or base two number system is comprised of the digits zero and one (Ø,1). Like the decimal system, it too is a positional number system. Progressing from right to left within a binary number, the value associated with each position is an increasing power or multiple of the base. This place value principle for the binary system is presented in Table 2-6.

Table 2-5 Powers of Two

$2^{Ø}$	=	1	=	1
2^{1}	=	2	=	2
2^{2}	=	2x2	=	4
2^{3}	=	2x2x2	=	8
2^{4}	=	2x2x2x2	=	16
2^{5}	=	2x2x2x2x2	=	32
2^{6}	=	2x2x2x2x2x2	=	64
2^{7}	=	2x2x2x2x2x2x2	=	128
2^{8}	=	2x2x2x2x2x2x2x2	=	256
2^{9}	=	2x2x2x2x2x2x2x2x2	=	512
$2^{1Ø}$	=	2x2x2x2x2x2x2x2x2x2	=	1,Ø24
2^{11}	=	2x2x2x2x2x2x2x2x2x2x2	=	2,Ø48
2^{12}	=	2x2x2x2x2x2x2x2x2x2x2x2	=	4,Ø96
2^{13}	=	2x2x2x2x2x2x2x2x2x2x2x2x2	=	8,192
2^{14}	=	2x2x2x2x2x2x2x2x2x2x2x2x2x2	=	16,384
2^{15}	=	2x2x2x2x2x2x2x2x2x2x2x2x2x2x2	=	32,768

Table 2-6 Positional Notation with Powers of Two

Powers of Two

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1

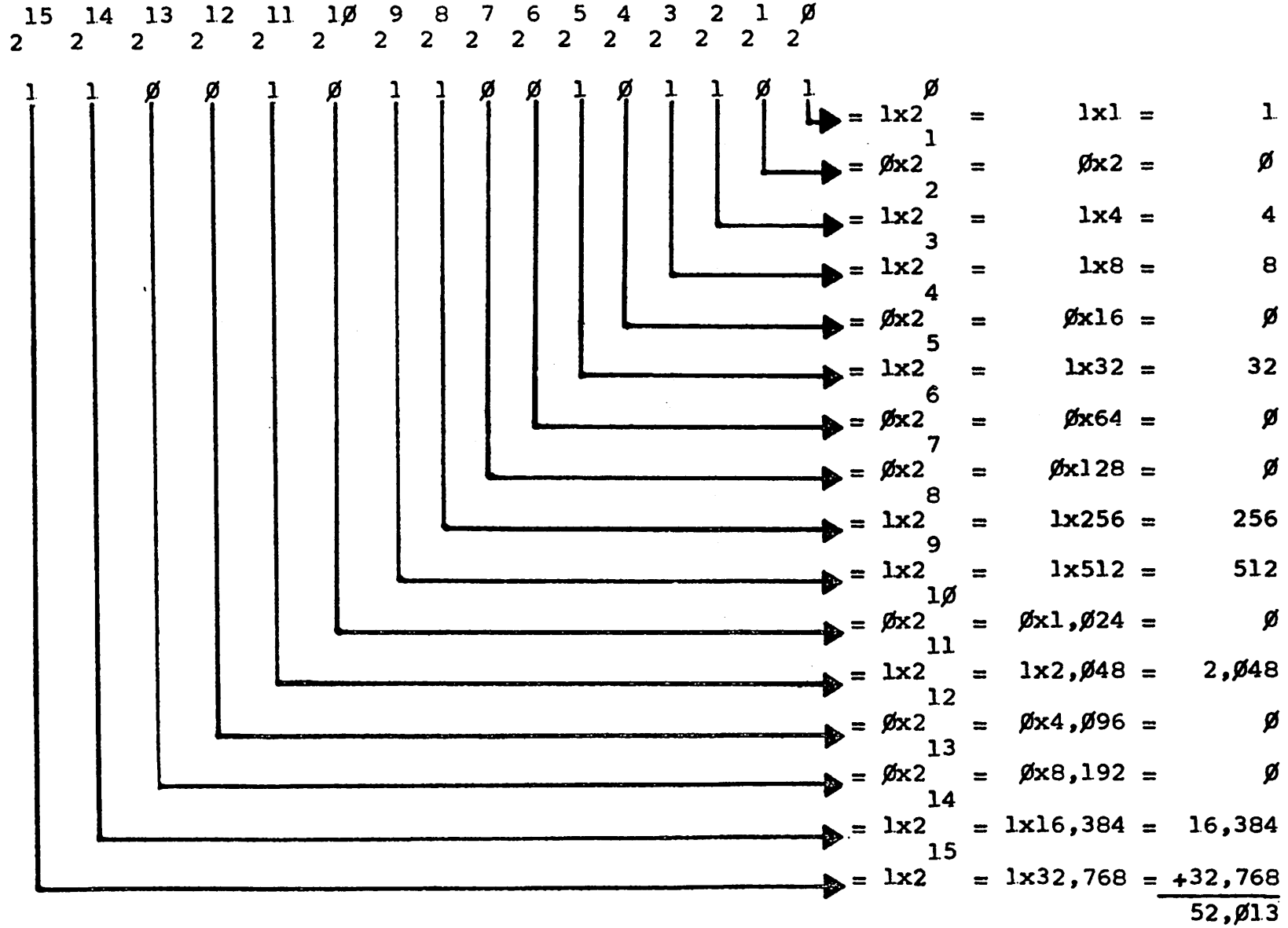
Place Values

Since the binary system is a positional number system, the rules are identical to those of the decimal and octal systems. The only difference, of course, is that the base is two rather than ten or eight.

Determining the total value of any binary number is then accomplished in the same manner used for the decimal system: applying the principles of place value, multiplication, and addition. The discrete value of the digit (Ø or 1) is multiplied by the value of the position in which it is placed, and the resulting products are added.

This operation is presented in Table 2-7.

Table 2-7 Binary Number as the Sum of Powers



The count and carry principle also applies to the binary number system, and can be simply illustrated by counting or addition. In fact, one advantage of binary notation is the simplicity of operation. Since the system consists of only the symbols zero and one, all the digits are used merely counting to one! Counting an additional unit equals the base, and is represented as 1Ø (read as "one zero," not "ten"). You have counted until the base was equaled, and then carried to the next column. As shown in Table 2-8, this occurs quite often in the binary number system!

Table 2-8 Counting in Binary with Decimal Equivalents

Binary	Decimal Equivalents
Ø	Ø
1	1
1Ø	2
11	3
1ØØ	4
1Ø1	5
11Ø	6
111	7
1ØØØ	8
1ØØ1	9
1Ø1Ø	1Ø
1Ø11	11
11ØØ	12
11Ø1	13
111Ø	14
1111	15
1ØØØØ	16
1ØØØ1	17
1ØØ1Ø	18
1ØØ11	19
1Ø1ØØ	2Ø

Binary addition illustrates both the count and carry principle and the operational simplicity of the system. As shown in Figure 2-2, there are only four possible individual conditions.

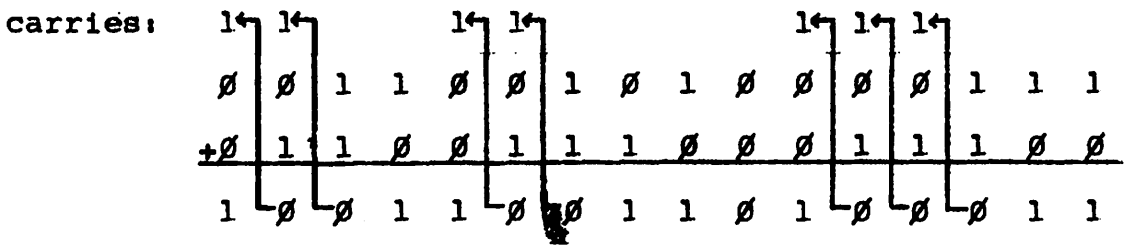
Figure 2-2 The Four Possible Conditions for Binary Addition

Addend	Addend	Sum	Carry
Ø	Ø	Ø	Ø
Ø	1	1	Ø
1	Ø	1	Ø
1	1	Ø	1

Ø	Ø	1	1
+Ø	+1	+Ø	+1
Ø	1	1	1Ø

The procedure followed in the addition operation is the same as that followed for the decimal system: (1) Add the digits in the column, (2) If the base is neither equaled nor exceeded, record the sum; (3) If the base is equaled or exceeded, divide by the base, record the remainder, and carry the quotient to the next column.

Note the importance of recording the zero remainders in the example below, "holding" those places within the sum.



2.2.4 Octal Number System

We may call the decimal number system the "numerical language of the real world" because much computer input and output data is in this form. The binary number system is considered the "numerical language of the computer" because the majority of computers are designed to work with this notation. For the machine language and assembly language user, the octal number system provides an easily handled bridge between these two, and may be called the "numerical language of the programmer."

As noted earlier, the binary system is most commonly used with computers because its simplicity yields hardware advantages; the components can be fast, yet relatively simple and inexpensive to manufacture. Computers, however, don't have to "look" at the binary numbers they manipulate, and due to speed of operation, work with them one at a time. To the programmer who must work with many cumbersome groupings, the length of the numbers and the similarity of digits makes the binary system far from ideal. As we will discuss shortly, there exists a quick and direct conversion between the binary and octal systems, and for reasons given any numerical work at the machine language or assembly language level is done with the latter system.

The octal or base eight number system is comprised of the digits zero through seven (Ø,1,2,3,4,5,6,7). Like the decimal and binary systems, it too is a positional number system. Progressing from right to left within an octal number, the value associated with each position is an increasing power or multiple of the base. This place value principle for the octal system is presented in Table 2-1Ø.

Table 2-9 Powers of Eight

8^{0}	= 1	= 1
8^{1}	= 8	= 8
8^{2}	= 8x8	= 64
8^{3}	= 8x8x8	= 512
8^{4}	= 8x8x8x8	= 4,096
8^{5}	= 8x8x8x8x8	= 32,768

Table 2-10 Positional Notation with Powers of Eight

Powers of Eight

8^{5}	8^{4}	8^{3}	8^{2}	8^{1}	8^{0}
32,768	4,096	512	64	8	1

Place Values

Since the octal system is a positional number system, the rules are identical to those of the decimal and binary systems. The only difference, of course, is that the base is <u>eight</u> rather than ten or two.

As presented in Table 2-11, determining the total value of any octal number is accomplished in the same manner used for the decimal and binary systems: applying the principles of place value, multiplication, and addition. The discrete value of the digit is multiplied by the value of the position in which it is placed, and the resulting products are added.

Table 2-11 Octal Number as the Sum of Powers

5	4	3	2	1	Ø
8	8	8	8	8	8
1	2	3	4	5	6

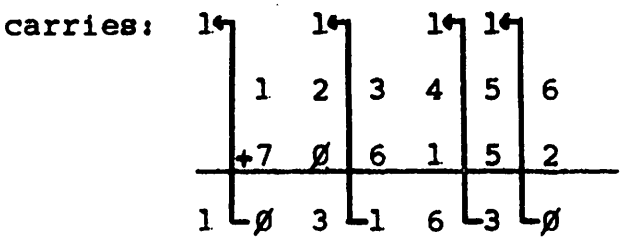
= 6x8^{Ø}	=	6x1	= 6
= 5x8^{1}	=	5x8	= 4Ø
= 4x8^{2}	=	4x64	= 256
= 3x8^{3}	=	3x512	= 1,536
= 2x8^{4}	=	2x4,Ø96	= 8,192
= 1x8^{5}	=	1x32,768	= +32,768
			42,798

The count and carry principle also applies to the octal number system, and can be simply illustrated by counting or addition. As shown in Table 2-12, you count until the base is equaled, and then carry to the next column.

Table 2-12 Counting in Octal with Decimal Equivalents

Octal	Decimal Equivalents
Ø	Ø
1	1
2	2
3	3
4	4
5	5
6	6
7	7
1Ø	8
11	9
12	1Ø
13	11
14	12
15	13
16	14
17	15
2Ø	16
.	.
.	.
3Ø	24
.	.
.	.
4Ø	32
.	.
.	.
5Ø	4Ø
.	.
.	.
6Ø	48
.	.
.	.
7Ø	56
.	.
.	.
1ØØ	64

The procedure followed for addition is the same as that followed for the decimal and binary systems: (1) Add the digits in the column, (2) If the base is neither equaled nor exceeded, record the sum; (3) If the base is equaled or exceeded, divide by the base, record the remainder, and carry the quotient to the next column.



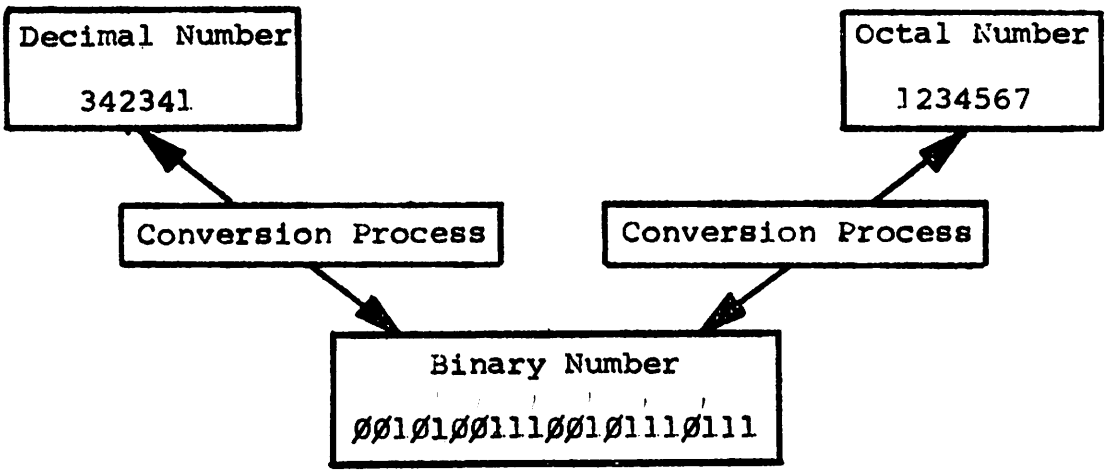
Again note the importance of the zero remainders recorded in the sum, "holding" those places within the number.

2.3 CONVERSIONS

2.3.1 Introduction

It should by now be established that the binary number system is good for computers, but little else! Therefore, numerical data written in decimal or octal form must first be converted to binary so that it can be processed by the computer, and then the results converted back from binary to decimal or octal so that they can be readily interpreted. This process is represented by Figure 2-3.

Figure 2-3 The Conversion Process



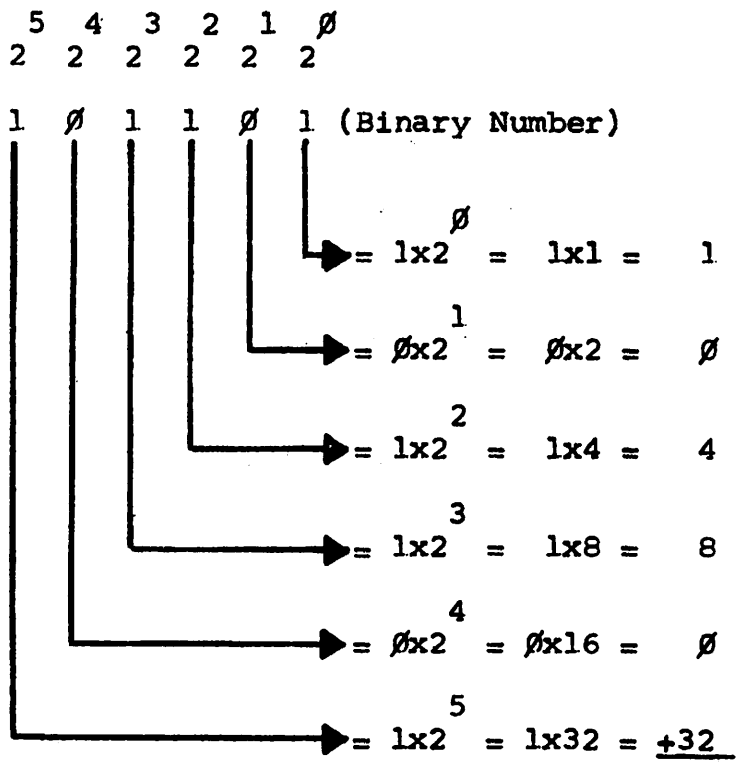
As noted earlier, these conversion processes are usually performed by programs previously written and stored in the computer. Specific conversion examples follow to provide an understanding of the processes.

2.3.2 Binary to Decimal Conversion

There are two commonly used methods for converting binary numbers to decimal equivalents: the Place Value method and the Double Dabble method.

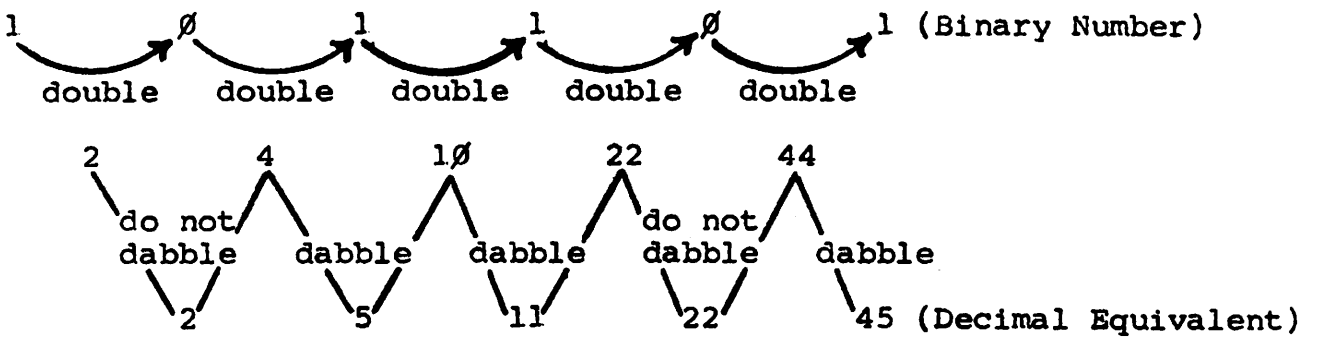
The Place Value method is simply the procedure used in representing a binary number as the sum of powers. The discrete value of each digit is multiplied by the value of the position in which it is placed, and the resulting products are added. An example of this method is presented in Table 2-13.

Table 2-13 Place Value Binary to Decimal Conversion



To convert binary numbers to decimal equivalents by means of the Double Dabble method, begin with the most significant bit (left-most one bit) of the binary number. Double that bit, and if the next lower order bit is a one, dabble (add one). If the next lower order bit is a zero, do not dabble. Moving from left to right within the binary number, repeat this process (doubling the sum if the next bit is zero, doubling the sum and dabbling if the next bit is one) until there are no more digits. An example of this method is presented in Table 2-14.

Table 2-14 Double Dabble Binary to Decimal Conversion

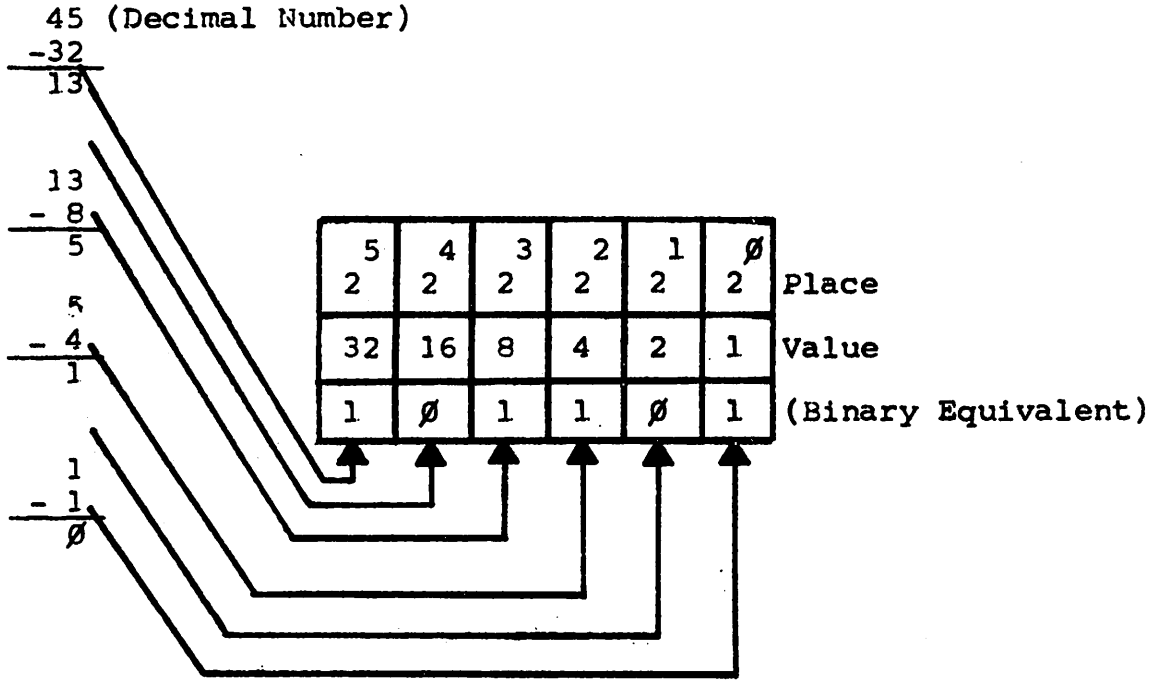


2.3.3 Decimal to Binary Conversion

There are two commonly used methods for converting decimal numbers to binary equivalents: the Subtraction of Powers method and the Division method.

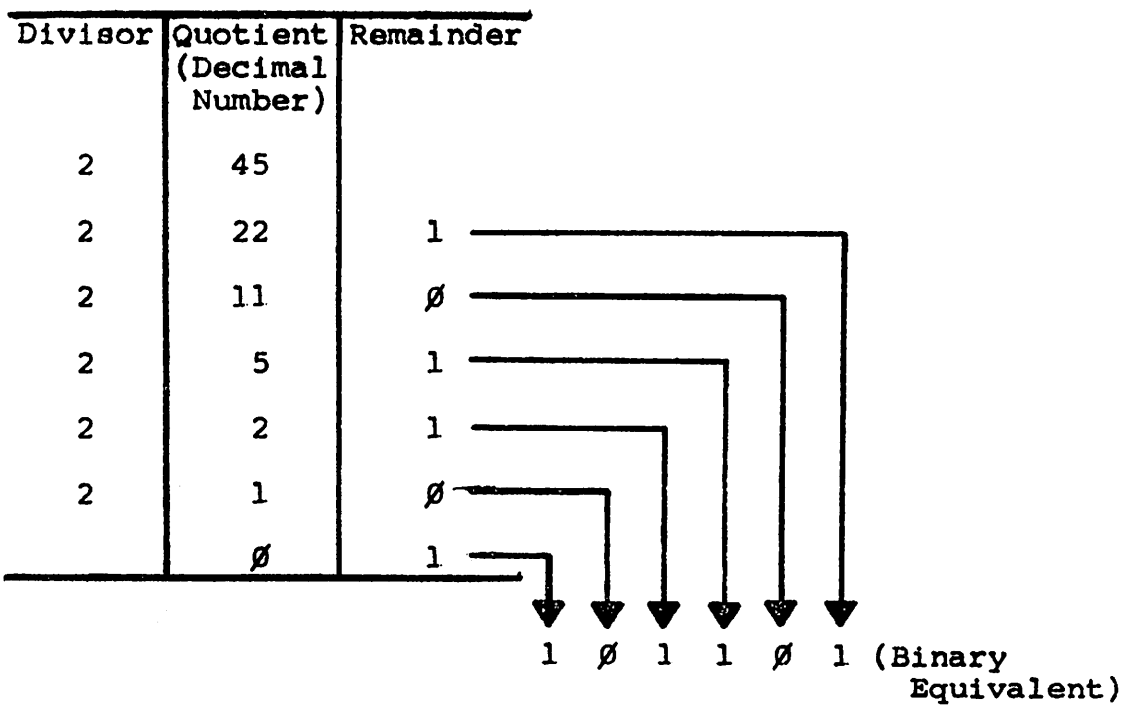
The procedure for the Subtraction of Powers method is as follows: (1) Subtract the highest possible power of two from the decimal number, and record a one in the apporpriate place within the partially completed binary equivalent, (2) Repeat this subtraction process with the resulting differences and descending powers of two (recording a one if that power can be subtracted, recording a zero if it cannot be subtracted) until the decimal number is reduced to zero. An example of this method is presented in Table 2-15.

Table 2-15 Subtraction of Powers Decimal to Binary Conversion



The procedure for the Division method of decimal to binary conversion is as follows: (1) Divide the decimal number by two; the remainder is the LSD (Least Significant Digit) of the binary equivalent, (2) Repeat this division process with the resulting quotients (recording remainders right to left within the binary equivalent) until the quotient becomes zero. An example of this method is presented in Table 2-16.

Table 2-16 Division Method Decimal to Binary Conversion

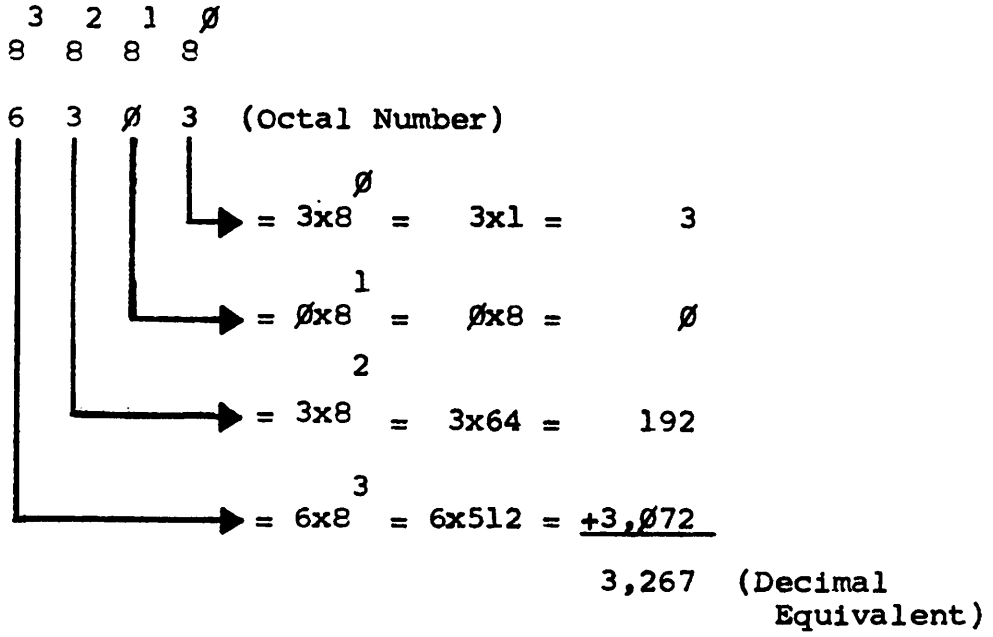


2.3.4 Octal to Decimal Conversion

There are two commonly used methods for converting octal numbers to decimal equivalents: the Place Value method, and a method similar in principle and procedure to the Double Dabble method for binary to decimal conversion.

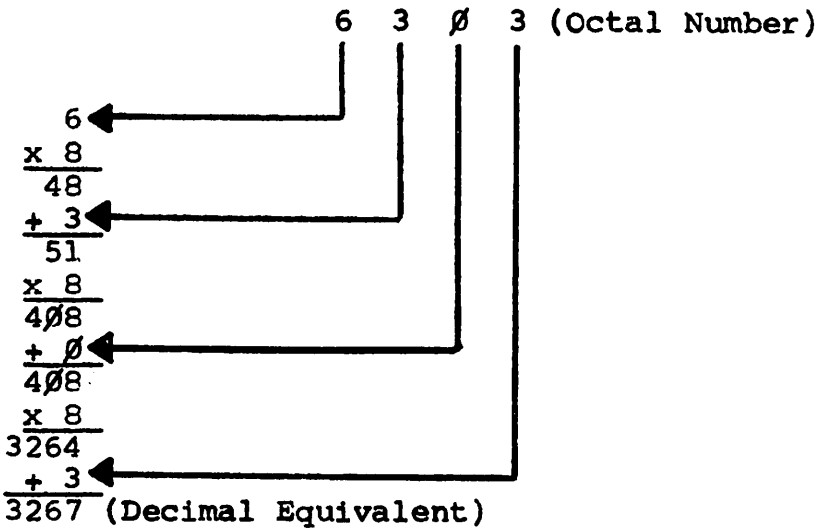
The Place Value method is simply the procedure used in representing an octal number as the sum of powers. The discrete value of each digit is multiplied by the value of the position in which it is placed, and the resulting products are added. An example of this method is presented in Table 2-17.

Table 2-17 Place Value Octal to Decimal Conversion



The octal to decimal conversion method that is similar to the Double Dabble method also begins with the MSD (Most Significant Digit). The procedure is as follows: (1) Record the MSD, (2) Multiply the MSD by eight, (3) Add the next octal digit, (4) Repeat steps two and three until the last digit of the octal number has been added. An example of this method is presented in Table 2-18.

Table 2-18 Octal to Decimal Conversion

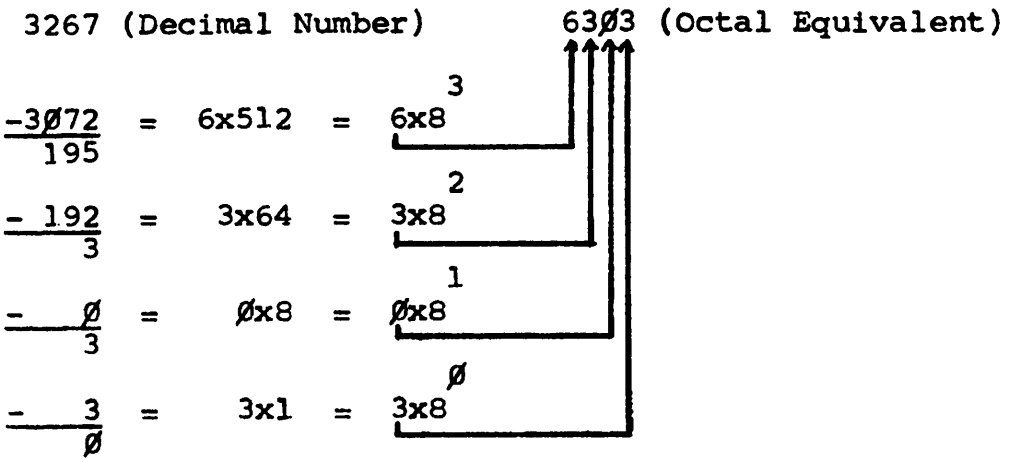


2.3.5 Decimal to Octal Conversion

There are two commonly used methods for converting decimal numbers to octal equivalents: the Subtraction of Powers method and the Division method.

Using the Subtraction of Powers method for decimal to binary conversion simply required subtracting powers of two from the decimal number. The additional digits of the octal system create the need for more work when using this method for decimal to octal conversion. We may subtract not only a power of eight, but up to seven times that power of eight from the decimal number. The procedure is then as follows: (1) Subtract the highest possible value of the form $a8^n$ (where a = Ø-7) from the decimal number, and record the value of a in the appropriate place within the partially completed octal equivalent, (2) Repeat this subtraction process with the resulting differences and descending powers of eight (recording the value of a) until the decimal number is zero. An example of this method is presented in Table 2-19.

Table 2-19 Subtraction of Powers Decimal to Octal Conversion



The procedure for the Division method of decimal to octal conversion is as follows: (1) Divide the decimal number by eight; the remainder is the LSD (Least Significant Digit) of the octal equivalent, (2) Repeat this division process with the resulting quotients (recording remainders right to left within the octal equivalent) until the quotient becomes zero. An example of this method is presented in Table 2-2Ø.

Table 2-2Ø Division Method Decimal to Octal Conversion

Divisor	Quotient (Decimal Number)	Remainder
8	3267	
8	4Ø8	3
8	51	Ø
8	6	3
	Ø	6

6 3 Ø 3 (Octal Equivalent)

2.3.6 Binary to Octal to Binary Conversion

As the numerical language of the machine language and assembly language programmer, the octal number system serves as a convenient "shorthand" notation for the binary number system, numerical language of the computer. The unwieldy strings of binary ones and zeros are converted to the more workable octal notation by inspection, with no calculation required, because eight is an integral power of two ($8=2^3$). As illustrated by Figure 2-4, three binary digits are the direct equivalent of one octal digit; one octal digit is the direct equivalent of three binary digits.

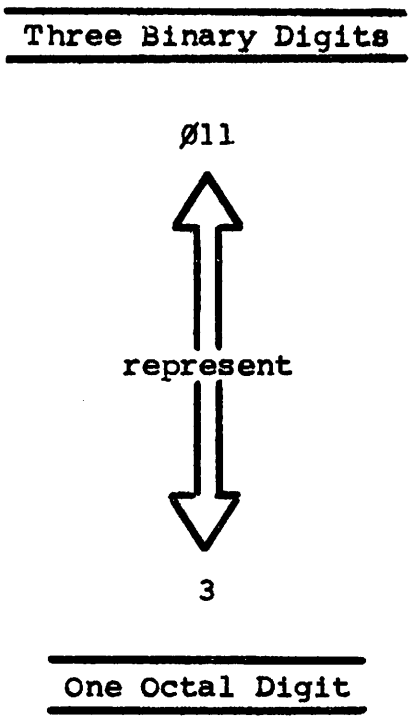


Figure 2-4 Binary to Octal to Binary Conversion

Knowing the binary and octal equivalents (Table 2-21), we can then represent any binary number as an octal number by means of the following steps: (1) Beginning with the LSD of the binary number, group the bits by threes (filling in leading zeros if necessary), (2) Convert these three bit groupings to their octal equivalents. An example of this procedure is given below:

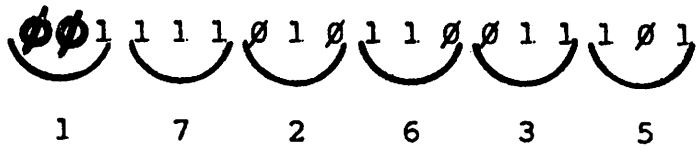


Table 2-21 Binary and Octal Equivalents

Binary	Octal
ØØØ	Ø
ØØ1	1
Ø1Ø	2
Ø11	3
1ØØ	4
1Ø1	5
11Ø	6
111	7

As noted earlier, existing programs are available for all conversion processes, but there may be several occasions when you will need to make the binary to octal and octal to binary conversion directly:

(1) Interpreting reference texts and instruction lists

Texts will often call upon the reader to make these conversions when describing the computer, illustrating the contents of various registers, and explaining instruction formats.

(2) Manually loading and verifying programs

While the machine language programmer must always do this, it should be noted that even the most advanced computer systems usually have short preparatory programs that must be so entered and/or checked.

(3) Avoiding binary notation

You may be involved in situations where you must work directly with numbers. If any of the notation is binary, convert to octal, operate, and if necessary convert the results back to binary.

2.4 ARITHMETIC OPERATIONS

2.4.1 Introduction

No matter how complex the arithmetic problem, it is eventually reduced to one of the four fundamental operations: Addition, subtraction, multiplication, division (Figure 2-5).

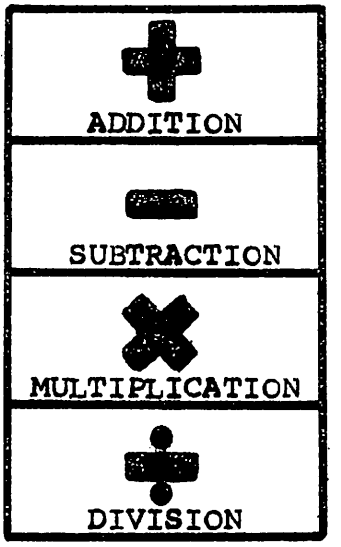


Figure 2-5 Four Fundamental Arithmetic Operations

As with the programs available to handle any conversion process, there exist several arithmetic "packages" that the computer user may call upon to perform his calculations. It is important to keep in mind, however, that any of these packages is a program comprised of instructions which are essentially the four fundamental operations. It is the program which "breaks down" the complex problem; the computer receives only the simplest of instructions.

Many computers, including the PDP-11, reduce the four fundamental arithmetic operations to one; addition. For reasons of hardware simplicity and efficiency, complementary (negative) addition is performed rather than direct subtraction. Though we will later examine other methods (rotating and shifting) when we later discuss the PDP-11 instruction set, multiplication can be accomplished by means of repeated addition; division by means of repeated complementary addition (subtraction).

We will therefore limit our discussion of arithmetic operations to the following topics: Addition, complementary addition, and (for comparision) direct subtraction. Examples in the decimal, binary, and octal number systems will be given for each operation.

2.4.2 Addition

Our discussions on the count and carry principle of positional number systems have also provided the steps required to perform decimal, binary, or octal addition. To review, the procedure is as follows: (1) Add the digits in the column, (2) If the base is neither equaled nor exceeded, record the sum; (3) If the base is equaled or exceeded, divide by the base, record the remainder, and carry the quotient to the next column.

This procedure works with any positional number system, and once the addition facts for the systems are learned (see Tables 2-22, 2-23, 2-24), binary or octal addition becomes as automatic as decimal addition.

An addition problem is solved below in the decimal, binary and octal systems. Note that when working with more than one number system, the base number is subscripted to differentiate.

carries:	1	1 1 1 1 1 1 1 1	1 1 1
	1 1 8$_{(10)}$	Ø Ø 1 1 1 Ø 1 1 1$_{(2)}$	1 6 7$_{(8)}$
	+5 Ø 2$_{(10)}$	+1 1 1 1 1 Ø 1 1 1$_{(2)}$	+7 6 7$_{(8)}$
	6 2 Ø$_{(10)}$	1 Ø Ø 1 1 Ø 1 1 1 Ø$_{(2)}$	1 1 5 6$_{(8)}$

Table 2-22 Binary Addition

+	Ø	1
Ø	Ø	1
1	1	1Ø

Table 2-23 Octal Addition

+	Ø	1	2	3	4	5	6	7
Ø	Ø	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	1Ø
2	2	3	4	5	6	7	1Ø	11
3	3	4	5	6	7	1Ø	11	12
4	4	5	6	7	1Ø	11	12	12
5	5	6	7	1Ø	11	12	13	14
6	6	7	1Ø	11	12	13	14	15
7	7	1Ø	11	12	13	14	15	16

Table 2-24 Decimal Addition

+	Ø	1	2	3	4	5	6	7	8	9
Ø	Ø	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	1Ø
2	2	3	4	5	6	7	8	9	1Ø	11
3	3	4	5	6	7	8	9	1Ø	11	12
4	4	5	6	7	8	9	1Ø	11	12	13
5	5	6	7	8	9	1Ø	11	12	13	14
6	6	7	8	9	1Ø	11	12	13	14	15
7	7	8	9	1Ø	11	12	13	14	15	16
8	8	9	1Ø	11	12	13	14	15	16	17
9	9	1Ø	11	12	13	14	15	16	17	18

2.4.3 Direct Subtraction

The operation of direct subtraction is performed in the same manner for all positional number systems, regardless of the base. The procedure is as follows: (1) For each column, subtract the subtrahend from the minuend (if the subtrahend is greater than the minuend, "borrow" a power of the base from the next column and then subtract), (2) Record the difference.

As with addition, binary or octal subtraction should become as automatic for you as decimal subtraction. The only difference is the base, and you should keep this in mind; that when you "borrow," you borrow a power of that base. Reference the example problems below.

borrows:

$$\begin{array}{r} {}^{3}\not{4}\ {}^{12}\not{3}\ {}^{1}2_{(10)} \\ -\ 2\ 3\ 4_{(10)} \\ \hline 1\ 9\ 8_{(10)} \end{array}$$

borrows:

$$\begin{array}{r} {}^{\phi}\not{1}\ {}^{1\phi}\not{\phi}\ {}^{1}\not{\phi}\ 1\ {}^{\phi}\not{1}\ {}^{1}\not{\phi}\ {}^{1}\not{\phi}\ {}^{1}\not{\phi}\ \phi_{(2)} \\ -\ \phi\ 1\ 1\ 1\ \phi\ 1\ \phi\ 1\ \phi_{(2)} \\ \hline \phi\ 1\ 1\ \phi\ \phi\ \phi\ 1\ 1\ \phi_{(2)} \end{array}$$

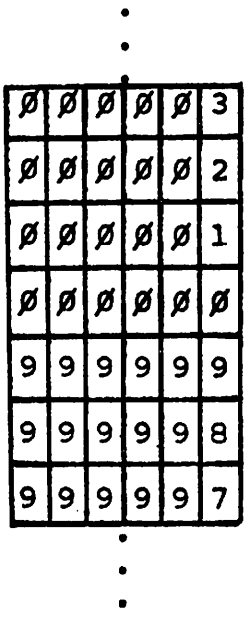
borrows:

$$\begin{array}{r} 6\ {}^{5}\not{6}\ {}^{1}\not{\phi}_{(8)} \\ -\ 3\ 5\ 2_{(8)} \\ \hline 3\ \phi\ 6_{(8)} \end{array}$$

easier to octal convert to subtract & convert binary

2.4.4 Complementary Addition

To understand complements, and thus the way in which negative numbers are commonly handled in the computer, consider again the odometer of the automobile. If the mileage indicator is rotated backwards, it will eventually approach and pass through zero, as shown below.



Considering zero to be a "boundary," we see that the number 999998 corresponds to -2. Applying this relational concept to the operation of complementary addition, we add the numbers 5 and 999998.

$$\begin{array}{rr} & \text{ØØØØØ5} \\ + & 999998 \\ \hline 1 & \text{ØØØØØ3} \end{array}$$

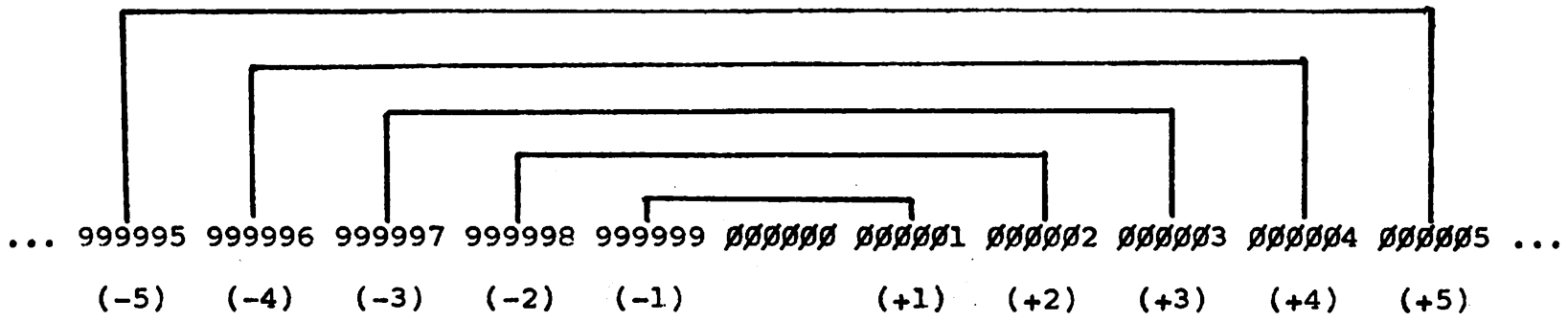
If we ignore the last generated carry, we have effectively performed the operation of subtraction (5-2=3).

The number 999998 in the above example is referred to as the ten's complement of 2. For the complementary addition operation, the term radix complement is defined as either of two numbers which when added will result in a sum or zero (last generated carry disregarded). This concept is illustrated by the example below and by Table 2-25. (It should be noted that the term radix complement can by definition apply to either a positive or negative number, but that it is most commonly used to describe the negative quantity.)

$$\begin{array}{rr} & 999998 \\ + & \text{ØØØØØ2} \\ \hline 1 & \text{ØØØØØØ} \end{array}$$

We can thus do away with direct subtraction; instead of subtracting a positive number, we add the negative representation of that number. In using a system of complements, however, we omit the minus sign, and must therefore establish what is and what is not a negative number. For example, is 123456 a positive 123456 or a negative 876544? With the odometer as an arbitrary example, we have the ability to represent one million numbers (Ø to 999999), and it would be reasonable to use half for positive and half for negative. Thus, by convention, we would regard Ø to 499999 as positive and 5ØØØØØ to 999999 as negative. And this in fact is exactly what is done with the computer; with a finite range of binary numbers to represent, half are designated as positive and half as negative.

Table 2-25 Radix Complements for the Decimal System



We have established the following points concerning the radix complement:

(1) It is the negative representation of a positive number.

(2) It is used because complementary notation can be efficiently and simply handled by the computer. All numbers can be treated alike (added) in arithmetic operations; complementary addition (add the negative) rather than subtraction (subtract the positive) can be performed.

(3) Signs are not required. The computer works with a finite range of binary numbers, and a convention can be established such that the number itself designates whether it is positive or negative.

We will examine the radix complement, the radix minus one complement, and the complementary addition operation first with the familiar decimal system, and then with the languages of the computer (binary) and programmer (octal).

The radix complement, which commonly takes the name of the base, is called the 1Ø's complement in the decimal system. The procedure for radix (1Ø's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base; the difference is the radix (1Ø's) complement, (2) Add the radix (1Ø's) complement to the minuend, (3) Record the sum, (4) Disregard the last generated carry (the next highest power of the base was introduced in step one and is "tossed out" here); this is the final result. Two examples are presented below.

$237_{(1Ø)}$ $-\ 125_{(1Ø)}$ $112_{(1Ø)}$	direct subtraction (as a check)	$84_{(1Ø)}$ $-\ 59_{(1Ø)}$ $25_{(1Ø)}$
$1ØØØ_{(1Ø)}$ $-\ 125_{(1Ø)}$ $875_{(1Ø)}$	subtract the subtrahend from next highest power of the base the difference is the radix (1Ø's) complement	$1ØØ_{(1Ø)}$ $-\ 59_{(1Ø)}$ $41_{(1Ø)}$
$875_{(1Ø)}$ $+\ 237_{(1Ø)}$ $(1)\ 112_{(1Ø)}$	add the 1Ø's complement to the minuend and record the sum disregard the last generated carry; this is the result	$41_{(1Ø)}$ $+\ 84_{(1Ø)}$ $(1)\ 25_{(1Ø)}$

The radix minus one complement in the decimal system is called the 9's complement. The procedure for radix minus one (9's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base minus one; the difference is the radix minus one (9's) complement, (2) Add the radix minus one (9's) complement to the minuend, (3) Record the sum, (4) Bring the last generated carry around to the least significant digit position and add it to the sum; this is the final result. The same examples worked with the radix (1Ø's) complement are repeated below using the radix minus one (9's) complement.

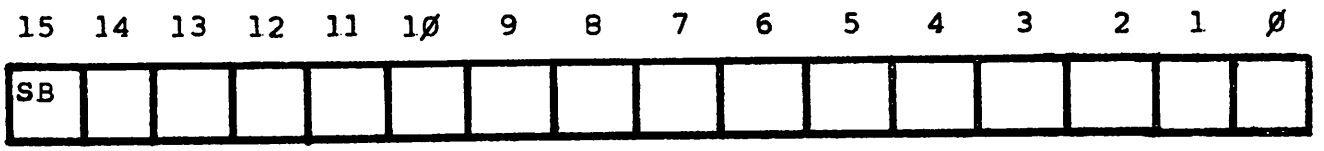
237(1Ø) - 125(1Ø) 112(1Ø)	direct subtraction (as a check)	84(1Ø) - 59(1Ø) 25(1Ø)
999(1Ø) - 125(1Ø) 874(1Ø)	subtract the subtrahend from next highest power of the base minus one the difference is the radix minus one (9's) complement	99(1Ø) - 59(1Ø) 4Ø(1Ø)
874(1Ø) + 237(1Ø) (1) 111(1Ø) + 1(1Ø) 112(1Ø)	add the 9's complement to the minuend and record the sum bring the last generated carry around to the LSD position and add it to the sum; this is the result	4Ø(1Ø) + 84(1Ø) (1) 24(1Ø) + 1(1Ø) 25(1Ø)

The radix complement in the binary (base 2) number system is called the 2's complement. But before we take up the subject of 2's complement addition, let's relate our previous discussion to the PDP-11 and the binary number system.

The PDP-11 is a variable word length machine, capable of handling both 16 bit words and 8 bit bytes. For the purpose of our discussion, let us consider it to be like many other computers, a fixed word length machine. This means that all data processed by the computer will be in the form of words (binary numbers) of the same length. It should be noted that from the programmer's standpoint words may be in varied formats and lengths; we are here veiwing words as the computer will ultimately receive them - in the form of fixed length binary numbers.

Viewing the PDP-11 as a 16 bit fixed word length machine, it has a binary number range of

$\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset_{(2)}$ to $1\ 111\ 111\ 111\ 111\ 111_{(2)}$.



By convention, half of these numbers are designated as positive ($\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset$ - $\emptyset$ 111 111 111 111 111) and half as negative (1 $\emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset\ \emptyset\emptyset\emptyset$ - 1 111 111 111 111 111) (Figure 2-6). The bit 15 position then assumes the role of sign bit; the number is considered positive if bit 15 = $\emptyset$ and negative if bit 15 = 1.

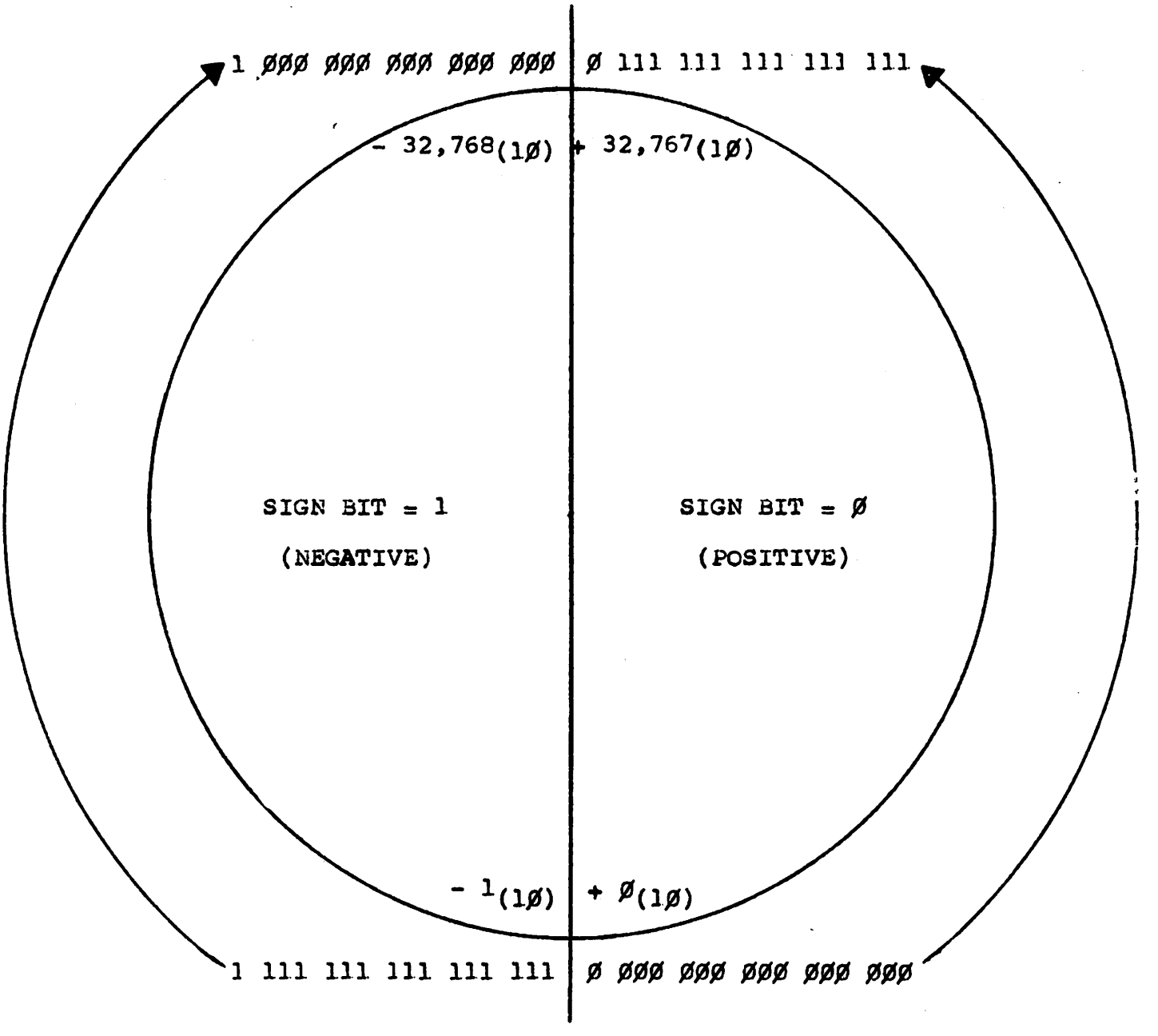


Figure 2-6 PDP-11 Positive and Negative Number Ranges (Binary)

Now that we know the application, let's look at the operation. The procedure for radix (2's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base; the difference is the radix (2's) complement, (2) Add the radix (2's) complement to the minuend, (3) Record the sum, (4) Disregard the last generated carry (the next highest power of the base was introduced in step one and is "tossed out" here); this is the final result. An example is given below.

$\begin{array}{r} \text{Ø ØØØ Ø11 1ØØ 11Ø ØØ1}_{(2)} \\ \underline{- \text{Ø ØØØ Ø1Ø ØØØ 1ØØ 1Ø1}_{(2)}} \\ \text{Ø ØØØ ØØ1 1ØØ ØØ1 1ØØ}_{(2)} \end{array}$	direct subtraction (as a check)
$\begin{array}{r} \text{1Ø ØØØ ØØØ ØØØ ØØØ ØØØ}_{(2)} \\ \underline{- \text{Ø ØØØ Ø1Ø ØØØ 1ØØ 1Ø1}_{(2)}} \\ \text{1 111 1Ø1 111 Ø11 Ø11}_{(2)} \end{array}$	subtract the subtrahend from next highest power of the base the difference is the radix (2's) complement
$\begin{array}{r} \text{1 111 1Ø1 111 Ø11 Ø11}_{(2)} \\ \underline{+ \text{Ø ØØØ Ø11 1ØØ 11Ø ØØ1}_{(2)}} \\ \text{(1) Ø ØØØ ØØ1 1ØØ ØØ1 1ØØ}_{(2)} \end{array}$	add the 2's complement to the minuend and record the sum disregard the last generated carry; this is the result

The radix minus one complement in the binary system is called the 1's complement. The procedure for radix minus one (1's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base minus one; the difference is the radix minus one (1's) complement, (2) Add the radix minus one (1's) complement to the minuend, (3) Record the sum, (4) Bring the last generated carry around to the least significant digit position and add it to the sum; this is the final result. The example worked with the radix (2's) complement is repeated below using the radix minus one (1's) complement.

Ø ØØØ Ø11 1ØØ 11Ø ØØ1 (2) - Ø ØØØ Ø1Ø ØØØ 1ØØ 1Ø1 (2) Ø ØØØ ØØ1 1ØØ ØØ1 1ØØ (2)	direct subtraction (as a check)
1 111 111 111 111 111 (2) - Ø ØØØ Ø1Ø ØØØ 1ØØ 1Ø1 (2) 1 111 1Ø1 111 Ø11 Ø1Ø (2)	subtract the subtrahend from next highest power of the base minus one the difference is the radix minus one (1's) complement
1 111 1Ø1 111 Ø11 Ø1Ø (2) + Ø ØØØ Ø11 1ØØ 11Ø ØØ1 (2) (1) Ø ØØØ ØØ1 1ØØ ØØ1 Ø11 (2) + 1 (2) Ø ØØØ ØØ1 1ØØ ØØ1 1ØØ (2)	add the 1's complement to the minuend and record the sum bring the last generated carry around to the LSD position and add it to the sum; this is the result

Have you noticed something unsettling about our complementary addition processes? The reason given for the use of complementary addition was that direct subtraction could not be performed with the PDP-11, and yet direct subtraction was used in all previous cases to obtain the complements! Let's see how the computer gets around this.

Note below that the bit patterns for any binary number and its 1's complement are exact opposites, and that the 2's complement is equal to the 1's complement plus 1.

```
                                  Ø ØØ1 Ø1Ø Ø11 1ØØ 1Ø1  (binary number)

  1 111 111 111 111 111
- Ø ØØ1 Ø1Ø Ø11 1ØØ 1Ø1
  1 11Ø 1Ø1 1ØØ Ø11 Ø1Ø     1 11Ø 1Ø1 1ØØ Ø11 Ø1Ø  (1's complement)

 1Ø ØØØ ØØØ ØØØ ØØØ ØØØ
- Ø ØØ1 Ø1Ø Ø11 1ØØ 1Ø1
  1 11Ø 1Ø1 1ØØ Ø11 Ø11     1 11Ø 1Ø1 1ØØ Ø11 Ø11  (2's complement)
```

The PDP-11 performs 2's complement addition, and therefore all negative numbers must be represented in 2's complement form. The PDP-11 2's complements any binary number without direct subtraction; it obtains the 1's complement by simply changing all bits to their opposites and then adds 1.

In the octal (base 8) number system, the radix complement is called the 8's complement and the radix minus one complement is called the 7's complement. Here too, the octal system serves the programmer as shorthand notation for the binary system; the 1's complement is to the 7's complement as the 2's complement is to the 8's complement. Again it should be stressed that the programmer rarely works in the binary number system; that any numerical work he must perform is done in the octal system and only if necessary converted to binary. If the 2's complement is required, for example, the programmer obtains the 8's complement (or the 7's complement plus 1) and then converts to binary.

Using the direct conversion that exists between the binary and octal number systems, the 16 bit PDP-11 word may be represented by 6 octal digits.

n nnn nnn nnn nnn nnn $_{(2)}$

N N N N N N $_{(8)}$

The octal representation of the PDP-11 fixed length number range is then Ø ØØØØØ $_{(8)}$ to 1 77777 $_{(8)}$; the leading octal digit will specify whether the bit 15 (sign bit) position contains a zero or a one. By convention, the numbers from Ø ØØØØØ $_{(8)}$ to Ø 77777 $_{(8)}$ are designated as positive, and the numbers from 1 ØØØØØ $_{(8)}$ to 1 77777 $_{(8)}$ are designated as negative (Figure 2-7).

The procedure for radix (8's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base; the difference is the radix (8's) complement, (2) Add the radix (8's) complement to the minuend, (3) Record the sum, (4) Disregard the last generated carry (the next highest power of the base was introduced in step one and is "tossed out" here); this is the final result. The example worked with the 2's complement is repeated below using the 8's complement.

$\begin{array}{r} \text{Ø ØØ3461}_{(8)} \\ \underline{- \text{Ø ØØ2Ø45}_{(8)}} \\ \text{Ø ØØ1414}_{(8)} \end{array}$	direct subtraction (as a check)
$\begin{array}{r} \text{1Ø ØØØØØØ}_{(8)} \\ \underline{- \text{Ø ØØ2Ø45}_{(8)}} \\ \text{7 75733}_{(8)} \end{array}$	subtract the subtrahend from next highest power of the base the difference is the radix (8's) complement
$\begin{array}{r} \text{7 75733}_{(8)} \\ \underline{+ \text{Ø ØØ3461}_{(8)}} \\ \text{(1) Ø ØØ1414}_{(8)} \end{array}$	add the 8's complement to the minuend and record the sum disregard the last generated carry; this is the result

The procedure for radix minus one (7's) complement addition is as follows: (1) Subtract the subtrahend from the next highest power of the base minus one; the difference is the radix minus one (7's) complement, (2) Add the radix minus one (7's) complement to the minuend, (3) Record the sum, (4) Bring the last generated carry around to the least significant digit position and add it to the sum; this is the final result. The example worked with the 1's complement is repeated below using the 7's complement.

$\emptyset\ \emptyset3461_{(8)}$ $-\ \emptyset\ \emptyset2\emptyset45_{(8)}$ $\emptyset\ \emptyset1414_{(8)}$	direct subtraction (as a check)
$7\ 77777_{(8)}$ $-\ \emptyset\ \emptyset2\emptyset45_{(8)}$ $7\ 75732_{(8)}$	subtract the subtrahend from next highest power of the base minus one the difference is the radix minus one (7's) complement
$7\ 75732_{(8)}$ $+\ \emptyset\ \emptyset3461_{(8)}$ $(1)\ \emptyset\ \emptyset1413_{(8)}$ $+\ \quad\quad 1_{(8)}$ $\emptyset\ \emptyset1414_{(8)}$	add the 7's complement to the minuend and record the sum bring the last generated carry around to the LSD position and add it to the sum; this is the result

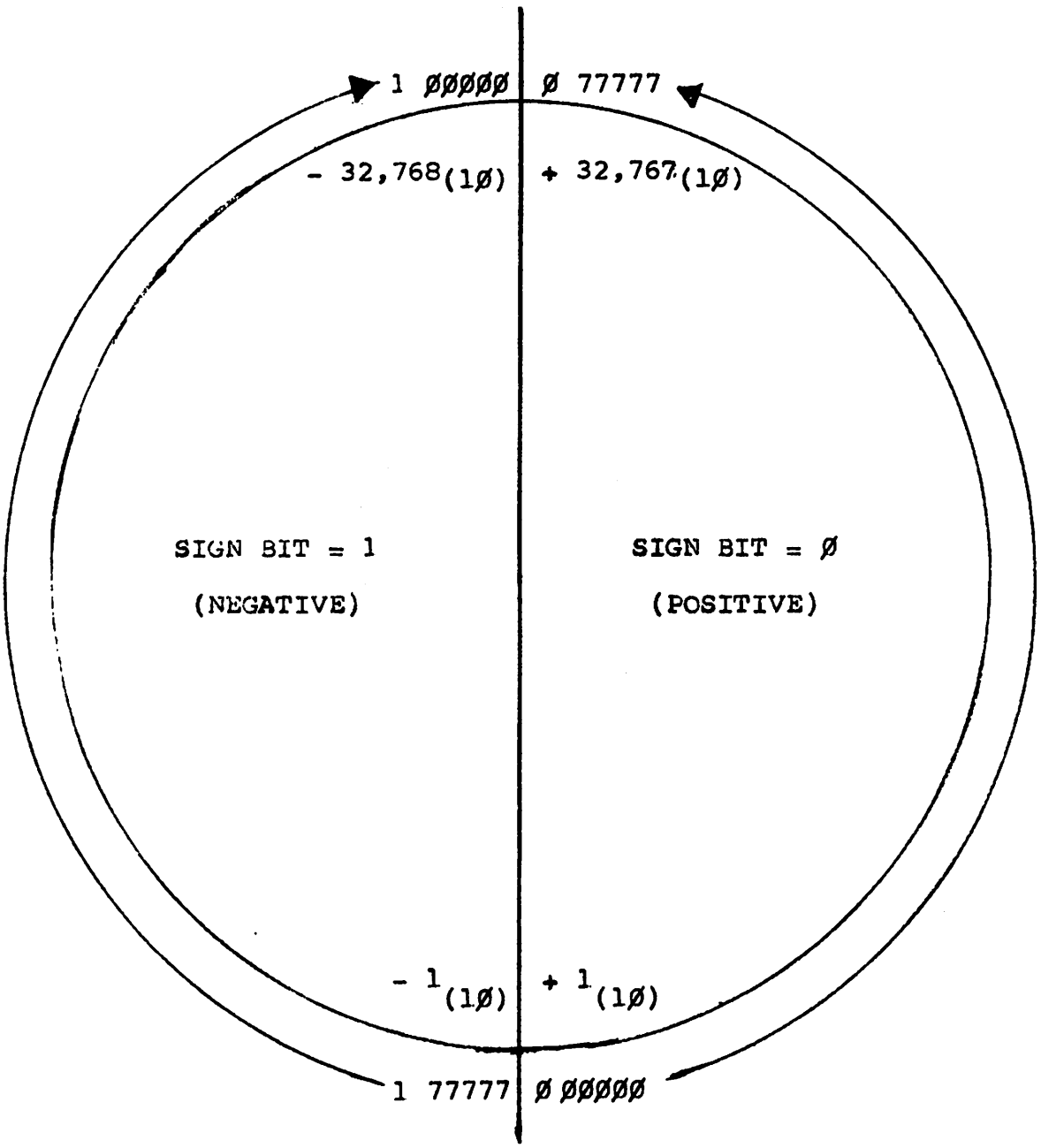


Figure 2-7 PDP-11 Positive and Negative Number Ranges (Octal)

2.5 LOGIC OPERATIONS

2.5.1 Introduction

Computers use logic operations as well as arithmetic operations in the execution of programs. The logic operations we will discuss have a direct relationship to an algebraic system used to represent logic statements known as Boolean algebra (named in honor of George Boole, English mathematician and logician). We will be concerned with the application of two Boolean axioms to computer circuitry. These are the two basic connectives used to express the relationship between two statements, the AND and the OR.

We will specifically examine the AND, INCLUSIVE OR, and EXCLUSIVE OR operations. Simple circuit diagrams, truth tables, and applicational examples will be given to help illustrate each of the operations.

2.5.2 The AND Operation

The diagram below (Figure 2-8) is that of a simple circuit with two switches. Current is allowed to flow through the switch if it is closed, and is not allowed to flow through the switch if it is open. Therefore, in order for the Function to occur, current must be allowed to flow through the entire length of the circuit; both switch A and switch B must be closed.

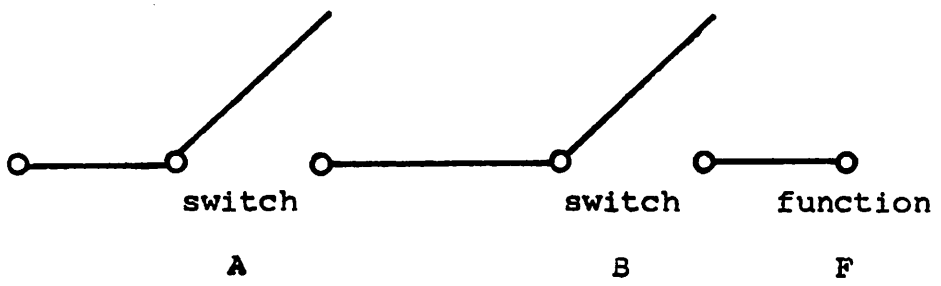


Figure 2-8 AND Circuit Diagram

In computer logic, the closed switch (or true condition) is represented as a 1, and the open switch (or false condition) is represented as a Ø. Expressing the AND axiom in terms of our variables, we can say that A & B = F (when using PDP-11 symbolic language, the ampersand specifies the AND operation). If A is 1 (true), and B is 1 (true), then F will be 1 (true); any other combination of the variables will result in a Ø (false) condition. The relationship between the variables and the resulting value of F is summarized in Table 2-26 below.

Table 2-26 Truth Table for the AND Operation

A	B	F
Ø	Ø	Ø
Ø	1	Ø
1	Ø	Ø
1	1	1

When the AND operation is applied to binary numbers, a binary 1 will appear in the result wherever a binary 1 appeared in the corresponding positions of the two numbers. A binary Ø will appear in the result wherever a binary Ø appeared in either (or both) of the corresponding positions of the two numbers. The AND operation is commonly used to extract (or mask) a portion of a 16 bit number. In the example below, it is used to extract the two least significant octal digits (mask the ten most significant bits) of the number.

```
  Ø ØØ1 Ø1Ø Ø11 1ØØ 1Ø1  (16 bit number)
& Ø ØØØ ØØØ ØØØ 111 111  (mask number)
  Ø ØØØ ØØØ ØØØ 1ØØ 1Ø1  (result)
```

2.5.3 The INCLUSIVE OR Operation

The diagram below (Figure 2-9) is that of a parallel circuit with two switches. Recalling that current is allowed to flow through a switch if it is closed and not allowed to flow through a switch if it is open, we see that current will flow through the entire length of this circuit if switch A or switch B or both are closed. The Function will be able to occur as long as both switches are not open.

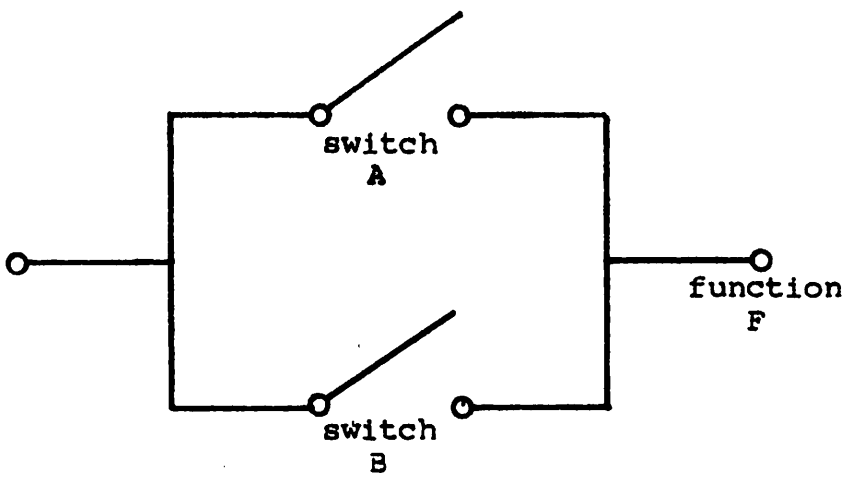


Figure 2-9 INCLUSIVE OR Circuit Diagram

Recall that in computer logic the closed switch (or true condition) is represented as a 1, and the open switch (or false condition) is represented as a Ø. Expressing the IOR axiom in terms of our variables, we can say that A!B = F (when using PDP-11 symbolic language, the exclamation point specifies the IOR operation). If A is 1 (true), or B is 1 (true), or both are 1 (true), then F will be 1 (true); only when both are Ø (false) will the result be Ø (false). The relationship between the variables and the resulting value of F is summarized in Table 2-27 below.

Table 2-27 Truth Table for the IOR Operation

A	B	F
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

When the IOR operation is applied to binary numbers, a binary 1 will appear in the result wherever a binary 1 appeared in the corresponding position of either (or both) of the two numbers. A binary Ø will appear in the result wherever a binary Ø appeared in both corresponding positions of the two numbers. The IOR operation is commonly used to set bits within a 16 bit number, where the present bit pattern cannot be known. In the example below, it is used to set bits in positions Ø, 7, and 15.

Ø 111 ØØ1 11Ø Ø1Ø 1Ø1 (16 bit number)

1 1 ØØØ ØØØ Ø1Ø ØØØ ØØ1 (IOR value)

1 111 ØØ1 11Ø Ø1Ø 1Ø1 (result)

2.5.4 The EXCLUSIVE OR Operation

The diagram below (Figure 2-1Ø) is that of a parallel circuit with two switches. The dotted line between the switches indicates they are mechanically connected such that they cannot be simultaneously closed (i.e., close one, open the other). Thus one set of conditions (both closed) is excluded in this OR operation, and the Function will be able to occur only if switch A or switch B is closed.

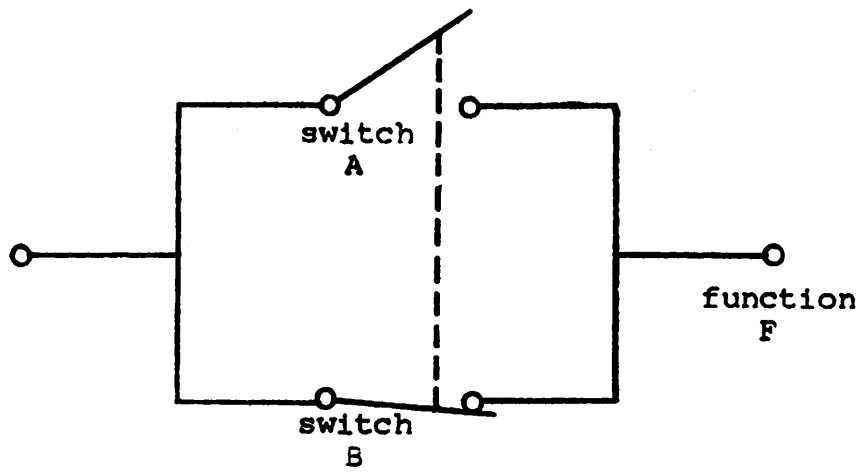


Figure 2-1Ø EXCLUSIVE OR Circuit Diagram

Again recall that in computer logic the closed switch (or true condition) is represented as a 1, and the open switch (or false condition) is represented as a Ø. Expressing the XOR operation in terms of our variables, we can say that A ⊕ B = F (when using PDP-11 symbolic language, the encircled exclamation point specifies the XOR operation). If A is 1 (true), or B is 1 (true), but not both are 1 (true), then F will be 1 (true). The relationship between the variables and the resulting value of F is summarized in Table 2-28 below.

Table 2-28 Truth Table for the XOR Operation

A	B	F
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

When the XOR operation is applied to binary numbers, a binary 1 will appear in the result wherever a binary 1 and a binary Ø appeared in the corresponding positions of the two numbers. A binary Ø will appear in the result wherever binary Ø's or binary 1's appeared in both the corresponding positions of the two numbers. The XOR operation is commonly used to set and/or clear bits within a 16 bit number, where the present bit pattern is known. In the example below, it is used to set bits in positions 7 and 15 and clear bits in positions Ø and 6.

```
  Ø ØØ1 Ø1Ø ØØ1 Ø1Ø ØØ1   (binary number)
⊕ 1 ØØØ ØØØ Ø11 ØØØ ØØ1   (XOR value)
  1 ØØ1 Ø1Ø Ø1Ø Ø1Ø ØØØ   (result)
```

2.6 EXERCISES

The following examples based upon the content of this chapter are presented as an optional exercise for the reader. The answers can be found in Appendix A.

2.6.1 Decimal to Binary Conversion

Convert the following decimal numbers to their binary equivalents

1. 1ØØ
2. 235

2.6.2 Binary to Decimal Conversion

Convert the following binary numbers to their decimal equivalents

1. ØØØ ØØØ ØØØ ØØ1 1ØØ 11Ø
2. ØØØ ØØØ ØØØ ØØØ 11Ø 11Ø

2.6.3 Decimal to Octal Conversion

Convert the following decimal numbers to their octal equivalents

1. 58Ø
2. 1ØØØ

2.6.4 Octal to Decimal Conversion

Convert the following octal numbers to their decimal equivalents

1. ØØØ742
2. ØØ1ØØØ

2.6.5 Binary to Octal Conversion

Convert the following binary numbers to their octal equivalents

1. ØØØ ØØØ ØØ1 Ø1Ø Ø11 1ØØ
2. ØØØ ØØØ ØØØ 1Ø1 111 11Ø

2.6.6 Octal to Binary Conversion

Convert the following octal numbers to their binary equivqlents

1. ØØØ736
2. ØØ5224

2.6.7 Binary Addition

Perform the indicated binary addition

1. ØØØ ØØØ 1ØØ 11Ø Ø11 1ØØ
   +ØØØ ØØØ 11Ø 11Ø 111 Ø11

2. ØØØ ØØØ Ø11 1Ø1 1ØØ 1Ø1
   +ØØØ ØØØ 11Ø 111 111 11Ø

2.6.8 Binary Subtraction

Subtract using both the direct and the complementary methods

1. ØØØ ØØØ ØØØ Ø11 11Ø ØØØ
   -ØØØ ØØØ ØØØ ØØØ 111 1Ø1

2. ØØØ ØØØ ØØØ 1ØØ ØØ1 1Ø1
   -ØØØ ØØØ ØØØ Ø11 11Ø 111

2.6.9 Octal Addition

Perform the indicated octal addition

1. Ø54362
   Ø73441
   +Ø6775Ø

2. ØØ3321
   ØØ44Ø7
   +ØØ5622

2.6.1Ø Octal Subtraction

Subtract using both the direct and the complementary methods

1. Ø13421
   -Ø12Ø54

2. Ø11234
   -Ø1Ø567

2.6.11 Logical AND

Perform the indicated AND operation

1. ØØØ ØØ1 Ø1Ø Ø11 1ØØ 1Ø1
   &ØØ1 Ø1Ø Ø11 1ØØ 1Ø1 11Ø

2.6.12 Inclusive OR

Perform the indicated OR operation

1. ØØ1 Ø1Ø Ø11 1ØØ 1Ø1 11Ø
   !Ø1Ø Ø11 1ØØ 1Ø1 11Ø 111

2.6.13 Exclusive OR

Perform the indicated OR operation

1. Ø1Ø Ø11 1ØØ 1Ø1 11Ø 111
   ⊕Ø11 1ØØ 1Ø1 11Ø 111 ØØØ

Chapter 3

The PDP-11

3.1 SYSTEM ORGANIZATION

3.1.1 Introduction

We have discussed the general organization of the computer in terms of the major units (input, memory, control, arithmetic, output), and with reference to a basic block diagram. We will now discuss these major units in more detail, and relate specifically to the elements of the Simplified PDP-11 System Organization diagram (Figure 3-1).

3.1.2 The UNIBUS

The UNIBUS is a single, common path that connects the processor, memory, and all peripheral (input and output) devices; it carries all information. Each device on the UNIBUS is assigned an address, and communicates in the same way. This means that peripheral devices may be as flexibly manipulated as memory. From the programmer's standpoint, this is the most important feature of the UNIBUS. Most computers require a separate line (and thus a special instruction subset) for input-output devices. With the PDP-11 and its UNIBUS, all of the powerful instructions that can be applied to data in memory can be applied to data in peripheral devices.

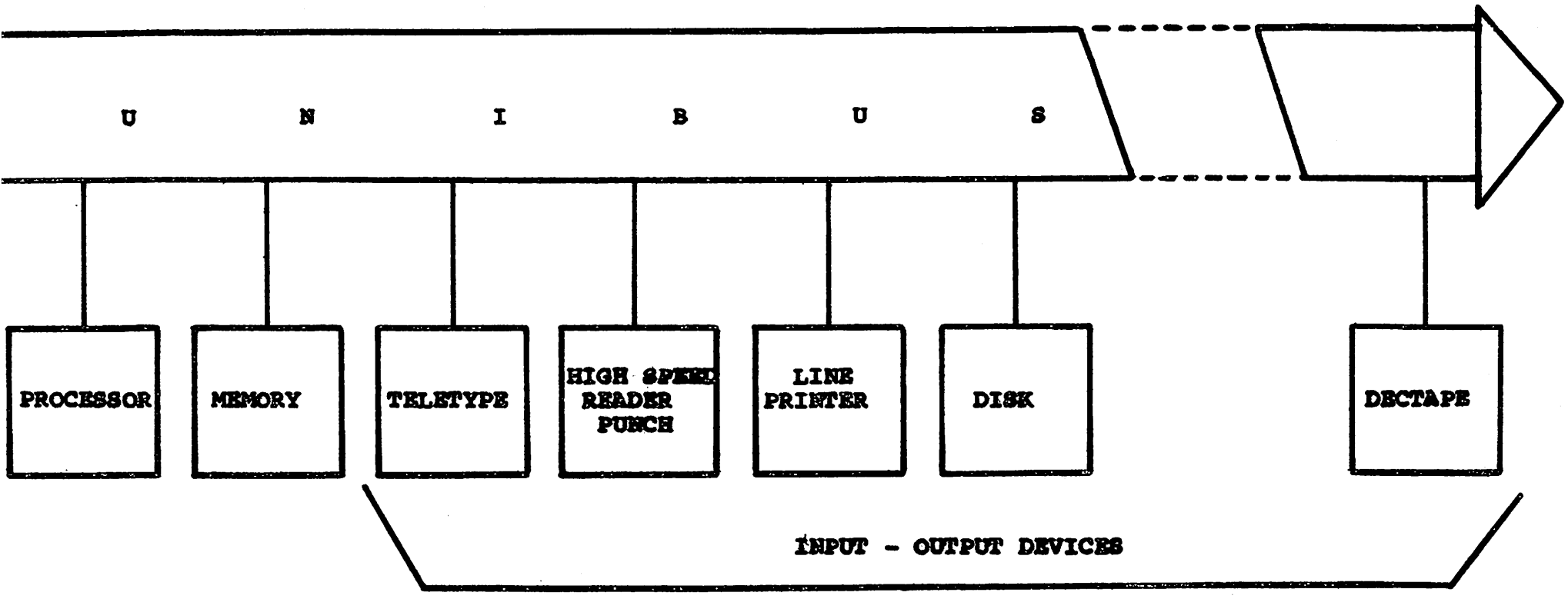


Figure 3-1 Simplified PDP-11 System Organization

3.1.3 Memory

The memory unit is used to store information until it is needed. Just as you remember facts concerning past and present events, the memory of the computer stores information for future reference.

We may conceptualize the computer memory as a series of locations, in a pigeon-hole or slot-like arrangement, where each location has a binary address and contains binary information (Figure 3-2).

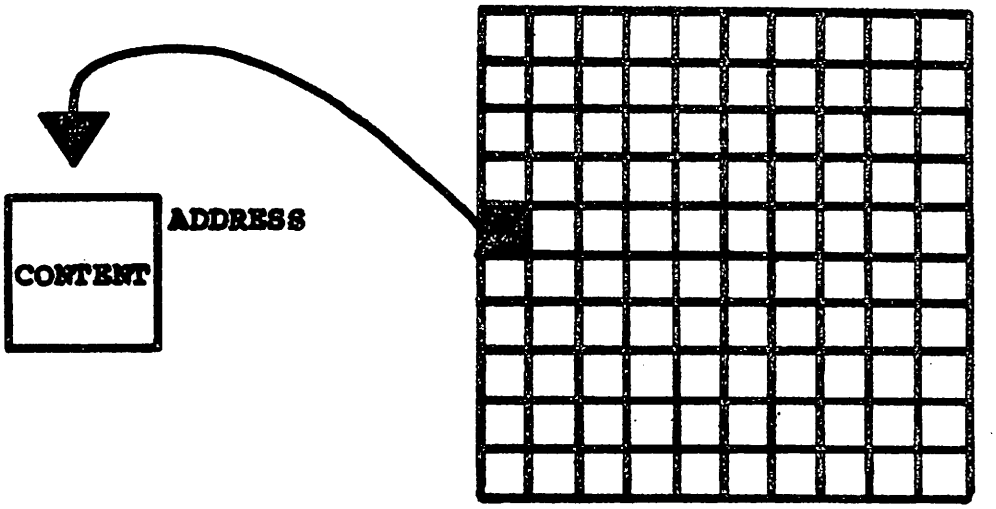


Figure 3-2 Conceptual Computer Memory Section

When the binary information in any location is accessed (used, modified, erased), that information will always be referenced by its address - never directly. As we will later discuss, that binary content may be interpreted as an instruction, another ("forwarding") address, or data. It will depend upon when (in which major state) and how (with which addressing mode) it is accessed by the computer.

The PDP-11 is a variable word length machine, working with either 16-bit numbers called words or 8-bit numbers called bytes. Any 16-bit word (bit positions ø-15) will then consist of two 8-bit bytes; the low byte (bit positions ø-7) and the high byte (bit positions 8-15).

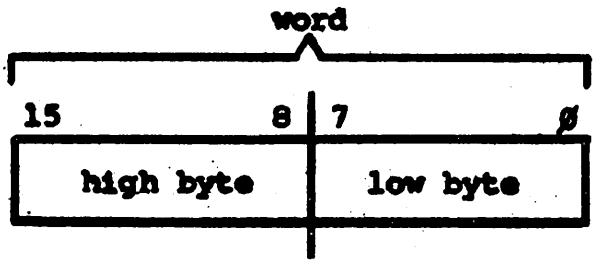


Figure 3-3 PDP-11 Word and Byte Relationship

The basic PDP-11 memory unit consists of 4,ø96 (1ø,øøø octal) word locations, and therefore 8,192 (2ø,øøø octal) byte locations. As mentioned, the machine is capable of handling either 16-bit words or 8-bit bytes, and the memory is therefore byte addressed so that both forms can be accommodated. The address range for the 2ø,øøø octal byte locations is ø-17777. As illustrated by Figures 3-4 and 3-5, the PDP-11 memory may be conceptualized as either sequential word locations or sequential byte locations. Note that words and low bytes are found at even addresses; high bytes at odd addresses.

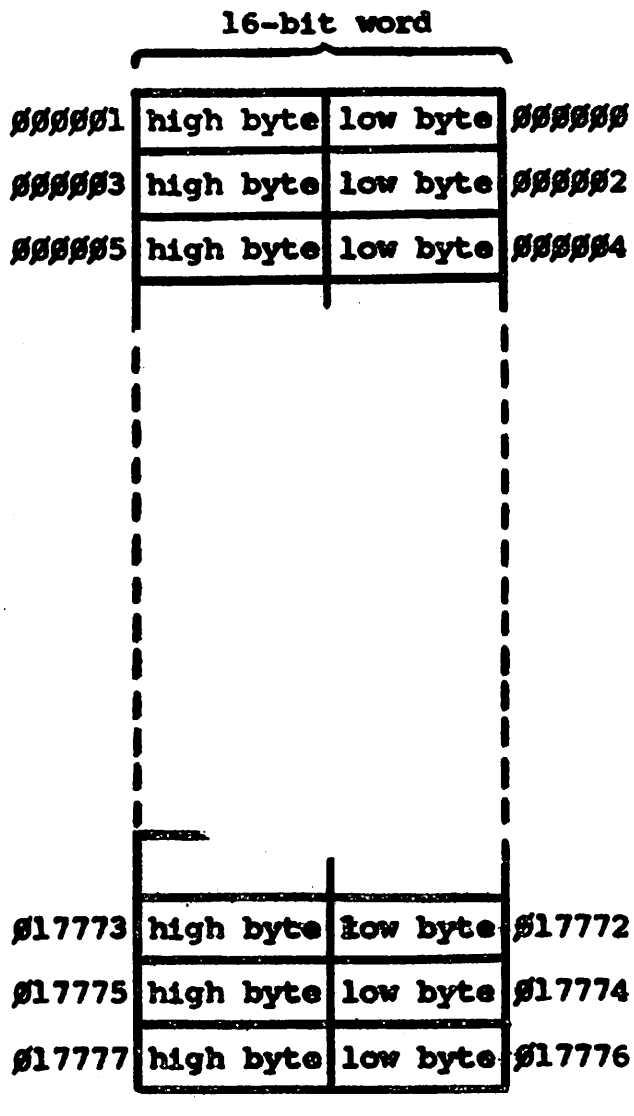


Figure 3-4 Word Organization

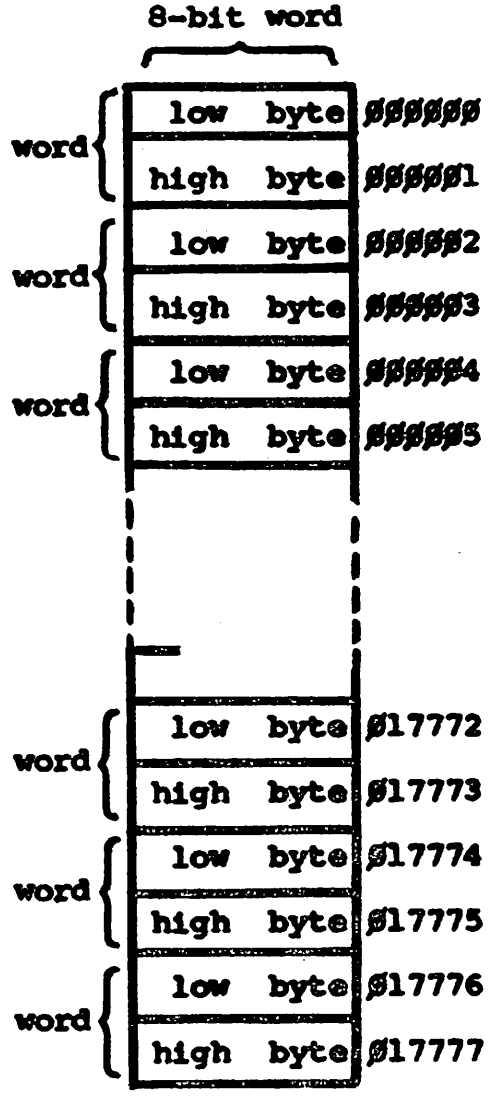


Figure 3-5 Byte Organization

3.1.4 Central Processor

The central processor (Figure 3-6) is comprised of three functional blocks: The Control Unit and Arithmetic Unit (as also given in our basic computer block diagram), and the General Purpose Registers. A figure eight is formed by the data paths connecting these units, and describes the flow of data through the processor. The total function of the processor is to process data; to execute the program, controlling operations from beginning to end.

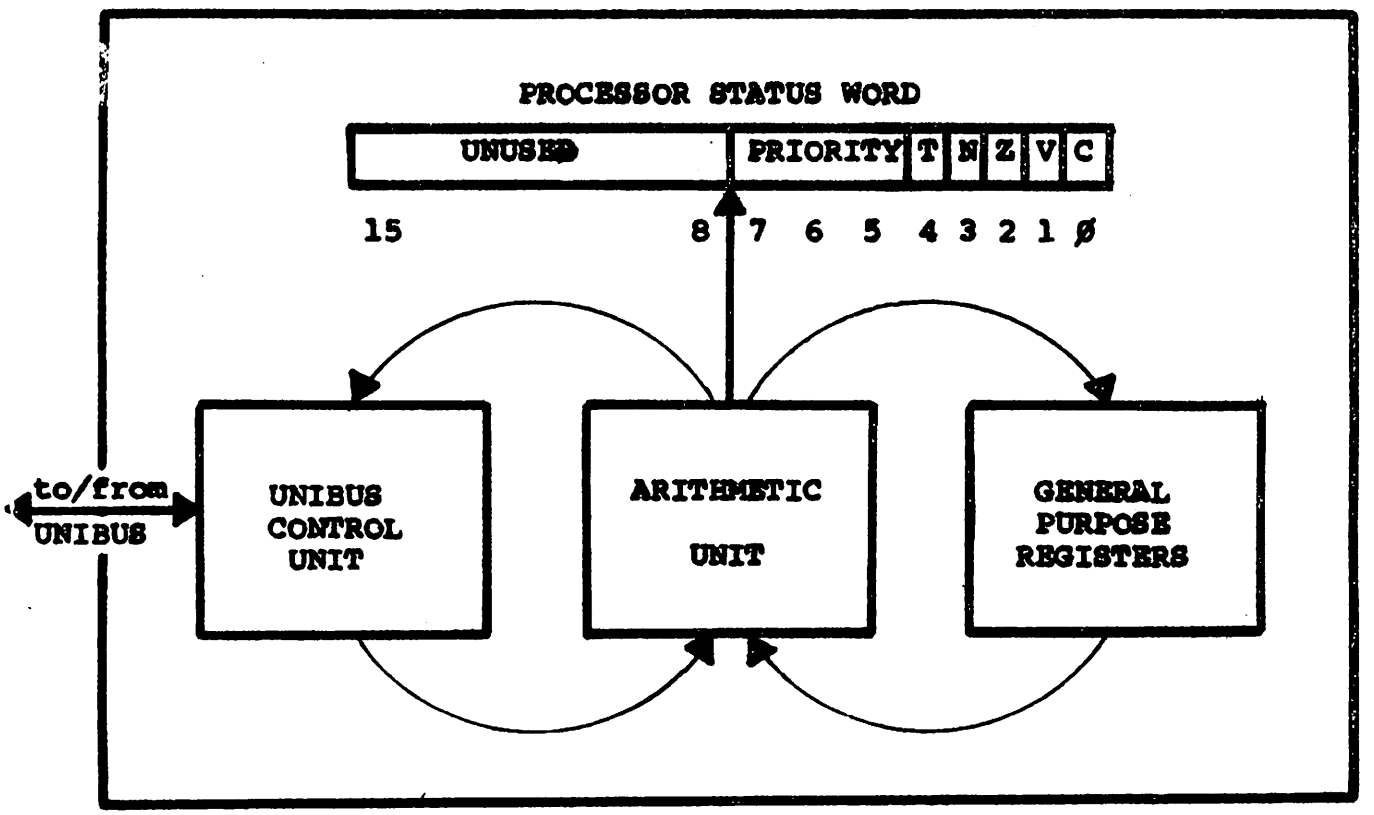


Figure 3-6 PDP-11 Central Processor

The UNIBUS Control Unit directs the processing by means of the following sequence: (1) Fetch an (the next) instruction from the program stored in the Memory Unit, (2) Decode that instruction, (3) If data is required, obtain that data from the Memory Unit or a peripheral device and bring it to the Arithmetic Unit, (4) Specify to the Arithmetic Unit what operation is to be performed upon the data, and (5) If required, store the result of the operation.

The PDP-11 processor has major states of operation, and four are listed below to help give the reader a basic description of the processor's operational flow.

FETCH - Obtain and decode an instruction. When fetch is completed, the processor enters another major state. It is possible to go from fetch to any other state, including back to fetch, depending upon the type of instruction decoded.

SOURCE - Decode the source address field of a double operand instruction (determine the address of the data), and transfer that data to the arithmetic unit. The source major state is entered only if the instruction is the double operand type.

DESTINATION - Decode the destination address field (determine the address of the data), and transfer that data to the arithmetic unit. The destination major state is entered for both single and double operand instructions.

EXECUTE - Perform the instruction. If data is to be operated upon, the arithmetic unit is directed to manipulate the data accordingly; if the result is to be stored, it is transferred from the arithmetic unit to the appropriate location.

Although the major states given follow the sequence of fetch, source, destination, and execute, not all are needed for every instruction; the processor enters only the states necessary to perform the current instruction.

Processor Status Word (Figure 3-6) is a self-descriptive title; it is an addressable word location that contains information on the status of the processing. Specifically, the low byte will indicate the following: Current priority level of the processor (bit positions 5-7), instruction trap (bit position 4), and result of the previous operation (bit positions ø-3).

The priority level of the processor, which can be manipulated by the program at any time, is an integral part of the Automatic Priority Interrupt System of the PDP-11. We will look at all of this in more detail when we later discuss input-output programming. Discussion of the trap indicator will also be postponed. Its role will be examined when we present trap instructions during discussion of the PDP-11 instruction set.

We will talk about the four least significant bits of the Processor Status Word, called the condition code bits. Upon the completion of the execute major state of an instruction, these bits are conditionally modified to reflect the result of that instruction (note the direct line from the Arithmetic Unit to the Processor Status Word). The program may then use this information to determine subsequent action. These bits are set as follows:

C bit (ø) - if there was a Carry from the most significant bit position

V bit (1) - if there was arithmetic oVerflow

Z bit (2) - if the result was Zero

N bit (3) - if the result was Negative

The central processor also contains a set of eight General Purpose Registers (Figure 3-7). These registers (commonly referred to as RØ, R1, R2...R7) are addressable word locations with special features that greatly enhance the power and flexibility of the PDP-11.

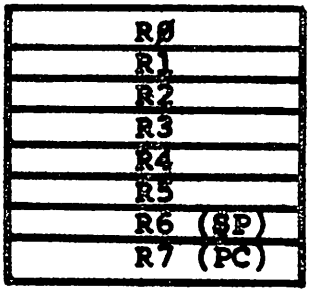


Figure 3-7 General Purpose Registers

The registers are called general purpose because each may be used as an:

ACCUMULATOR
Where a sum is accumulated in the General Purpose Register

POINTER
Where the General Purpose Register points to the operand (contains the address of the operand)

AUTOINCREMENT REGISTER
Where the General Purpose Register points to the operand (contains the address of the operand); the address is used and then automatically incremented

AUTODECREMENT REGISTER
Where the General Purpose Register points to the operand (contains the address of the operand); the address is first automatically decremented and then used

INDEX REGISTER
Where the General Purpose Register contains an index value that is added to a base address to provide the address of the operand

All addressing with the PDP-11 is accomplished through the General Purpose Registers, and they therefore play a vital role in efficient programming of the machine. We have only listed the addressing features of the registers here, and will examine them in more detail when we later discuss addressing modes.

It should be noted here that two of the eight registers have unique capabilities; R7 serves as the Program Counter, and R6 serves as the Stack Pointer. Both will later be discussed in detail, but a brief description of each follows.

Program Counter (PC) - This register might be better named the Program Pointer; it will always contain the address of the next location to be referenced. It is automatically updated by the processor as it steps through the program (after an instruction is fetched from a location, the Program Counter is stepped to contain the address of the next sequential location).

Stack Pointer (SP) - During the running of a program, there are several circumstances that can cause a change from one sequence of instructions to another (interrupts, traps, error conditions, etc.). The processor will automatically "remember" where it was in the first sequence of instructions by saving a return address (content of the PC) on the Stack. Thus R6, as the Stack Pointer, will contain the address of that location which holds the return address.

3.1.5 Input-Output Devices

The Input Devices associated with a computer system enable data and control information to be entered into the computer. Some devices require that the input information be in a special form (a card reader, for example, accepts only punched cards); other devices do not require any previous preparation of information (the Teletype allows information to be simply typed in). In all cases, these devices translate the various forms of input information into a form which can be handled by the computer.

The Output Devices associated with a computer system enable information (intermediate and final results) to be received from the computer. This output information may be in any of several forms, depending upon the device and the controlling program.

The list of Input-Output devices for the PDP-11 system is a long one. As examples, several of the more common devices are described generally below.

The operator's console (Figure 3-8) provides function switches to control the system and indicators to monitor the status of the system.

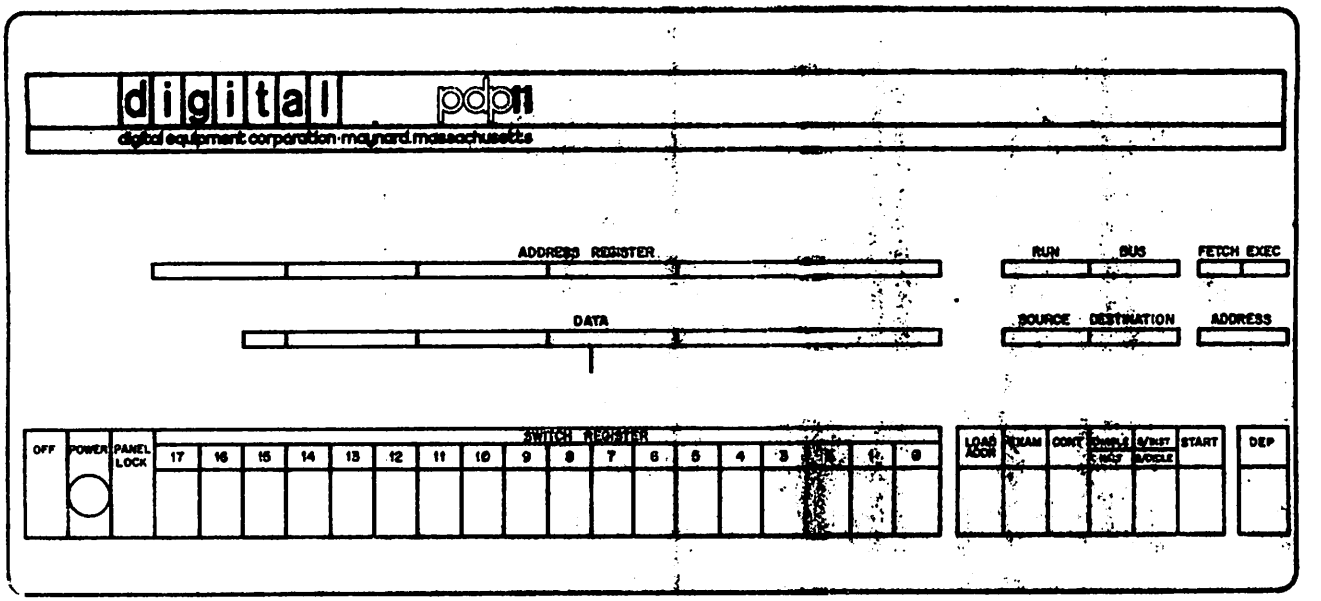


Figure 3-8 PDP-11 Console

Although it cannot be described as an input-output device, the console is discussed here because it does provide the operator with a direct means of input and output.

To input information (DEPOSIT), the procedure is as follows:

(1) Specify the 16-bit address with bit positions Ø-15 of the SWITCH REGISTER (switch UP=1, switch DOWN=Ø)

(2) Depress the LOAD ADDRESS key (transfers content of the SWITCH REGISTER to the ADDRESS REGISTER)

(3) Specify the 16-bit contents with bit positions Ø-15 of the SWITCH REGISTER

(4) Raise the DEPOSIT key (transfers content of the SWITCH REGISTER to the address specified in the ADDRESS REGISTER. Contents also displayed in DATA DISPLAY REGISTER.

The console serves as a means of output in two ways: The function keys may be used to EXAMINE locations on the UNIBUS, and the content of General Purpose Register Ø is automatically displayed in the DATA DISPLAY REGISTER upon the completion of any program.

The EXAMINE procedure is as follows:

(1) Specify the 16-bit address with bit positions Ø-15 of the SWITCH REGISTER

(2) Depress the LOAD ADDRESS key (transfers content of the SWITCH REGISTER to the ADDRESS REGISTER)

(3) Depress the EXAMINE key (transfers content of the address specified in the ADDRESS REGISTER to the DATA DISPLAY REGISTER)

It should be noted that the operator must LOAD ADDRESS only initially if DEPOSITing or EXAMINing sequential locations. The content of the ADDRESS REGISTER is automatically updated with each DEPOSIT or EXAMINE function.

The procedure for running a program which has been input is as follows:

(1) Specify the starting address in the SWITCH REGISTER

(2) Depress the LOAD ADDRESS key

(3) Set the ENABLE/HALT key to the ENABLE position (transfers control to the processor)

(4) Depress the START key (begins processor operation)

When the program is completed, the address of the HALT instruction will be in the ADDRESS REGISTER, and the content of General Purpose Register Ø (which can be the result) will be displayed in the DATA DISPLAY REGISTER.

The Model 33 Automatic Send-Receive Teletype Unit (Figure 3-9) is an input-output device provided as standard equipment with most PDP-11 systems.

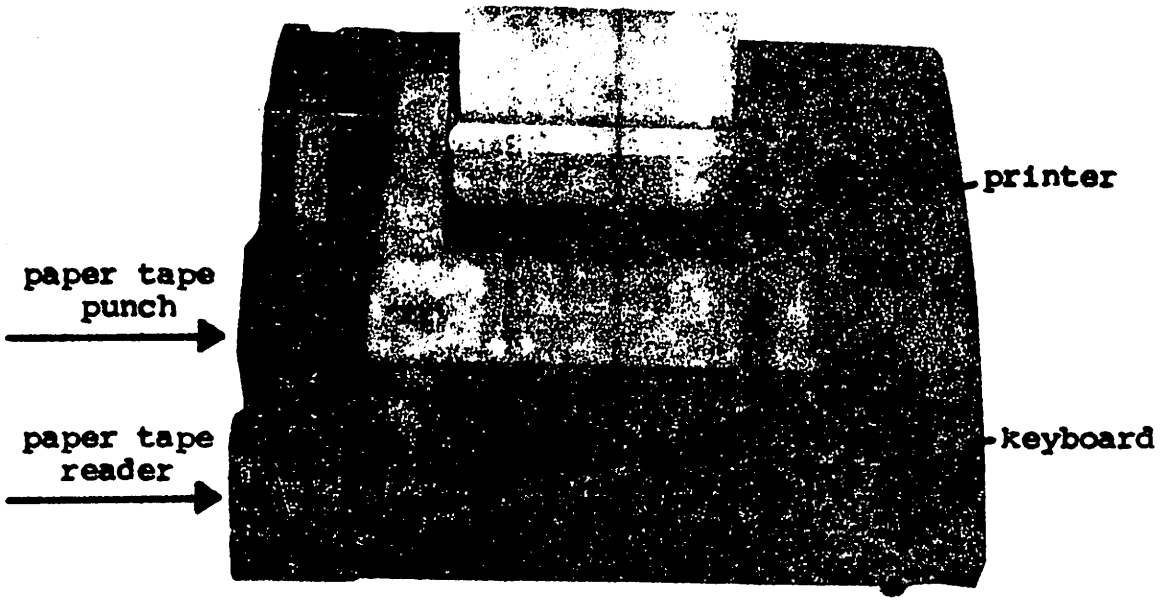


Figure 3-9 ASR-33 Teletype

Information is input in either of two ways: Typed in by means of the keyboard (10 characters per second), or read in by means of the low speed paper tape reader (10 characters per second).

Information is also output in either of two ways: printed out by means of the teleprinter (10 characters per second), or punched out by means of the low speed paper tape punch (10 characters per second).

The High Speed Paper Tape Reader and Punch (Figure 3-1Ø) is an input-output device available for those users who require faster paper tape reading and punching speeds than those of the standard ASR-33 Teletype.

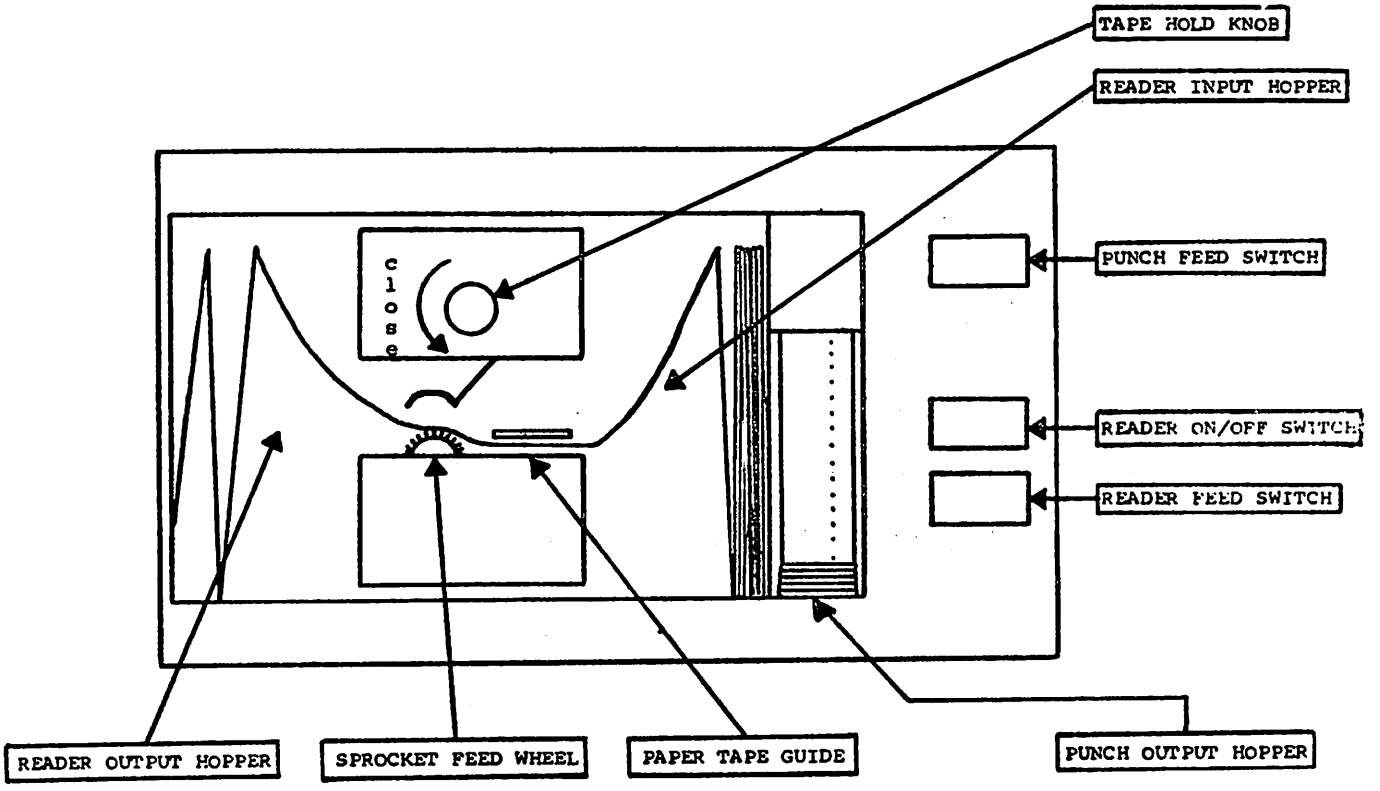


Figure 3-1Ø High Speed Reader and Punch

Information is input by means of the high speed photo-electric paper tape reader at the rate of 3ØØ characters per second.

Information is output by means of the high speed paper tape punch at the rate of 5Ø characters per second.

The High Speed Line Printer (Figure 3-11) is an output device available in several models for the user who requires a faster printing speed than that of the standard ASR-33.

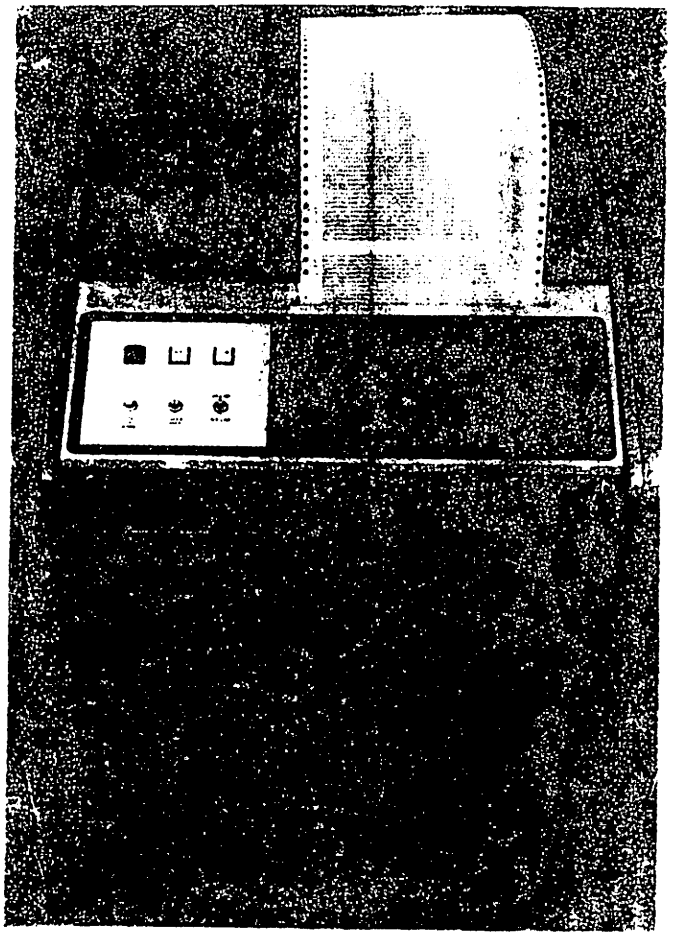


Figure 3-11 High Speed Line Printer

Using the 8Ø column, 64 character model as an example, information is printed out at the following rates:

356 lines per minute, columns 1-8Ø
46Ø lines per minute, columns 1-6Ø
65Ø lines per minute, columns 1-4Ø
111Ø lines per minute, columns 1-2Ø

The DECtape Unit (Figure 3-12) is one of two magnetic tape options available for PDP-11 systems. It is a dual-unit bidirectional magnetic tape transport system for auxiliary information storage.

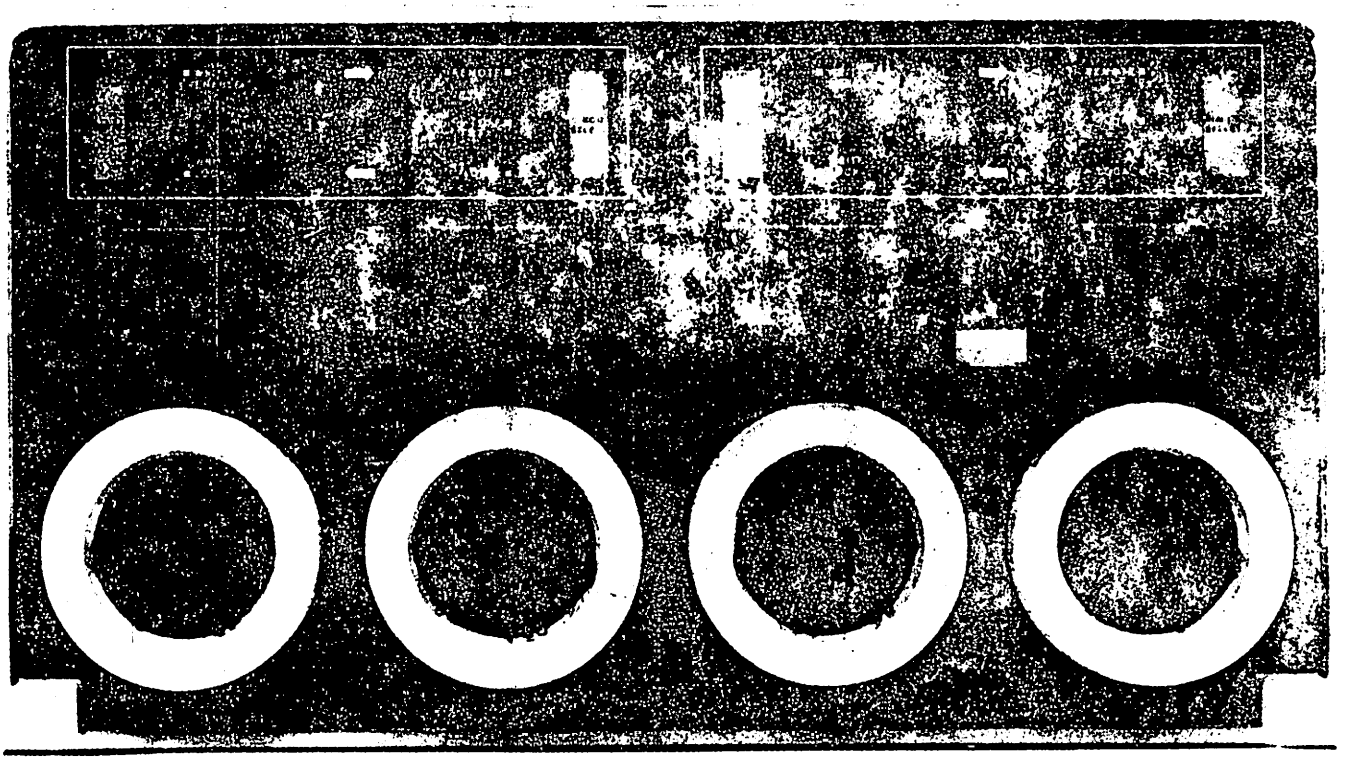


Figure 3-12 DECtape Unit

Information may be input (read) from or output (written) on this device at the rate of 5000 16-bit words per second. The system stores information at fixed positions on the magnetic tape, allowing blocks of the information to be read, written, or replaced without disturbing other previously recorded information.

The RC-11 Disk Unit (Figure 3-13) is one of many mass storage devices available for PDP-11 systems. Expandable disk mass storage systems may be used in a number of combinations, and range from the RC-11/RS-64 with a basic storage of 65 thousand words (expandable to 262 thousand) to the RP-11/RP-Ø2, which stores up to 8Ø million words in an expanded configuration.

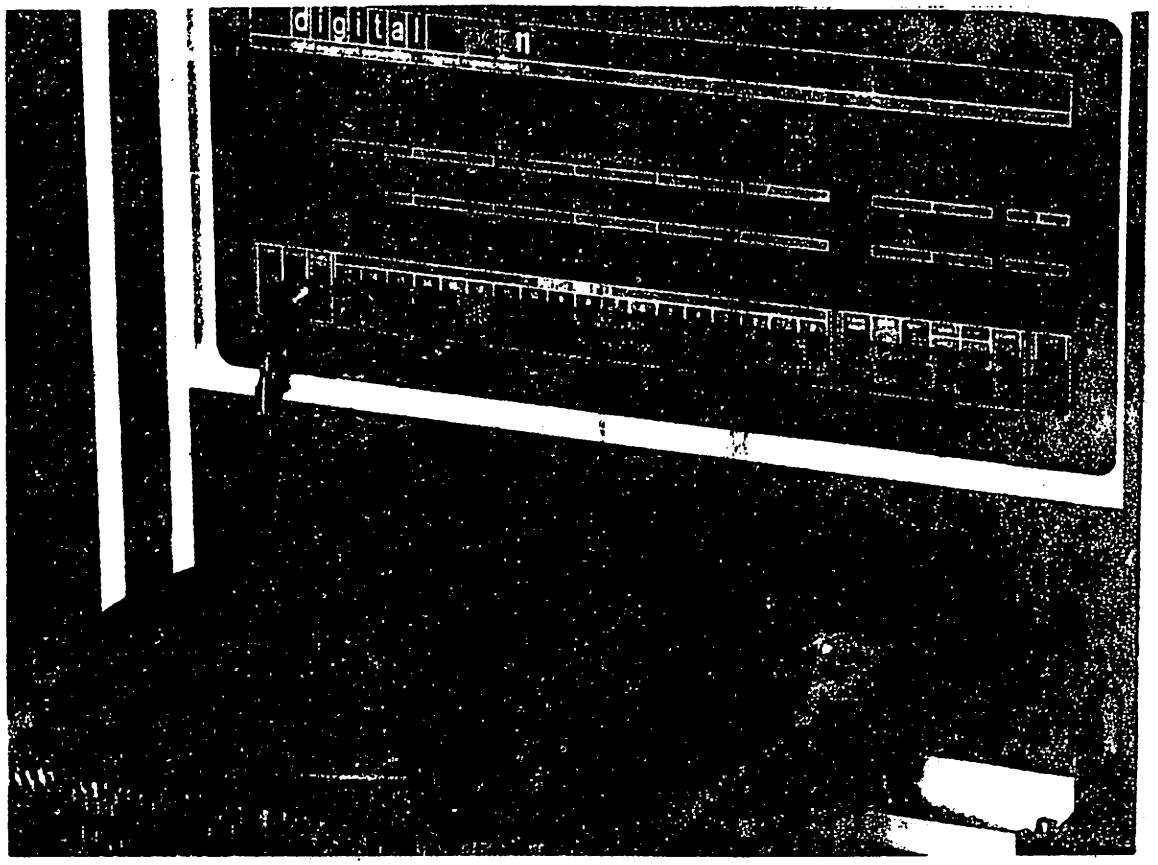


Figure 3-13 RC-11 Disk Unit

Information may be input (read) from or output (written) on the RC-11 Disk at the rate of 62,5ØØ 16-bit words per second. Information is stored at fixed positions on the disk surface, allowing blocks of the information to be read, written, or replaced without disturbing other previously recorded information.

3.2 ADDRESSING MODES

3.2.1 Introduction

A program is a series of computer words sequenced to accomplish a particular task. These computer words which comprise the program may be divided into two major categories: Data Words or operands (the values to be operated upon by the instruction words), and Instruction Words (those which access and manipulate the data words).

The Data Word (Figure 3-14) is quite straightforward; it is interpreted as a numerical value to be operated upon.

Figure 3-14 Data Word

The Instruction Word (Figure 3-15), though also numerical, must be interpreted differently. In order to manipulate the data word, the instruction word must access it, and you will recall that words in memory are always referenced by address. The instruction word is therefore of two parts; by convention, certain bits specify the operation code (how the data word is to be manipulated), and the other bits specify the address of the data word.

Figure 3-15 Instruction Word

The PDP-11 instruction set has two types of instruction words that manipulate data; the Single Operand Instruction and the Double Operand Instruction.

The Single Operand Instruction (Figure 3-16) implies one operand, and follows the general format presented earlier.

Bit positions 6-15 specify the operation code that defines the instruction to be executed.

Bit positions Ø-5 specify the destination address field (the address of the operand). This six-bit destination address field consists of two three-bit subfields:

register subfield (bit positions Ø-2) specifies which of the eight General Purpose Registers is to be used

mode subfield (bit positions 3-5) specifies how that General Purpose Register is to be used

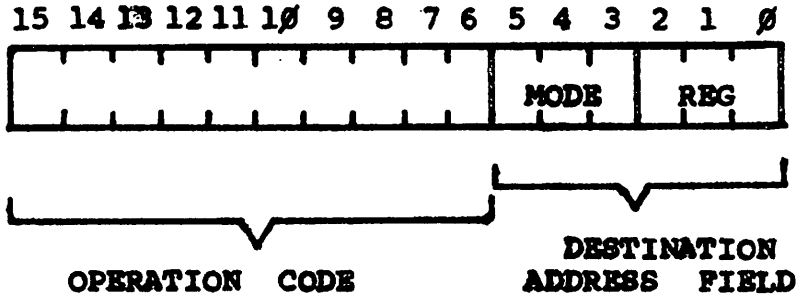


Figure 3-16 Single Operand Instruction Format

The Double Operand Instruction (Figure 3-17) implies two operands, called the source operand and the destination operand. The same general format is again followed, but here there are two address fields (one for each operand), and thus a shorter operation code.

Bit positions 12-15 specify the operation code that defines the instruction to be executed.

Bit positions Ø-5 specify the destination address field (the address of the destination operand). This six-bit destination address field consists of two three-bit subfields: the register subfield (bit positions Ø-2), and the mode subfield (bit positions 3-5).

Bit positions 6-11 specify the source address field (the address of the source operand). This six-bit source address field consists of two three-bit subfields: the register subfield (bit positions 6-8), and the mode subfield (bit positions 9-11).

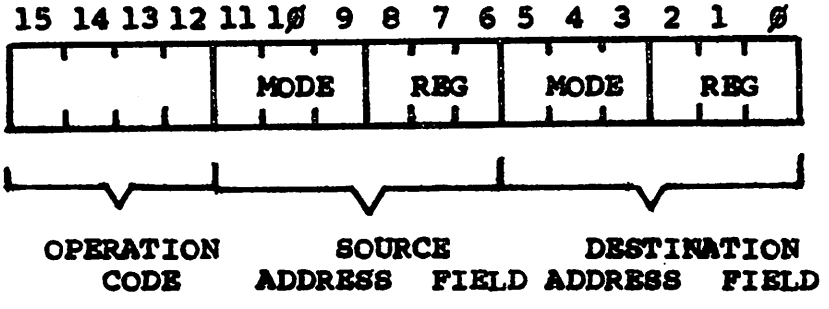


Figure 3-17 Double Operand Instruction Format

The meaning of our earlier statement - all addressing with the PDP-11 is accomplished through the General Purpose Registers - should now be clear. In specifying the address of the data (address field), one of the eight registers is selected (register subfield) along with one of several addressing modes (mode subfield).

These addressing modes enable the easy access and manipulation of data. They are especially efficient and flexible in handling of structured data (tables, lists, character strings, etc.), since a great deal of the data processed by the computer is organized in this manner.

We will examine each of the addressing modes in detail, and use the following instructions for illustration:

MNEMONIC CODE*	OCTAL CODE	DESCRIPTION
CLR DST	ØØ5ØDD	CLeaR (replace with zeros) the contents of the DeSTination location
INC DST	ØØ52DD	INCrement (add 1 to) the contents of the DeSTination location
MOV SRC,DST	Ø1SSDD	MOVe the SouRCe operand to the DeSTination location (source operand unaffected; destination operand replaced by the source operand)
ADD SRC,DST	Ø6SSDD	ADD the SouRCe operand to the DeSTination operand (source operand unaffected; DeSTination operand replaced by the sum)

*symbolic code devised for ease of recognition and retention which must be converted to machine (binary) code by some device or routine before it can be executed by the computer

3.2.2 General Register Addressing Modes

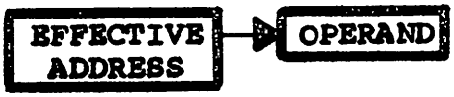
There are eight General Register Addressing Modes, and any mode may be used with any of the General Purpose Registers to access the operand. Though each of these eight modes is unique, and we will discuss the specific application of each, we may categorize them as follows according to how they use the General Purpose Register:

(1) DIRECT ADDRESSING

where the register contains the operand.

(2) INDIRECT ADDRESSING

where the register contains the address of the operand (the effective address).



(3) DEFERRED ADDRESSING

where the register contains the address of the effective address.

3.2.2.1 Direct Addressing

There is one mode where the register is used to contain the operand:

REGISTER MODE

REGISTER MODE

Assembler Syntax*	Octal Code
Rn	ø

Register Mode specifies that the register contains the operand. The register is thus used to hold data while it is manipulated.

Example: The content of R3 is incremented

Location 5øø contains the instruction code for INC R3

øøø5øø | øø52ø3

Before Execution:

R3 | øøøø13

After Execution:

R3 | øøøø14

Example: The content of R2 is added to the content of R5

Location 2øøø contains the instruction code for ADD R2,R5

øø2øøø | ø6ø2ø5

Before Execution:

R2 | øøøøø6

R5 | øøø231

After Execution:

R2 | øøøøø6

R5 | øøø237

*The percent sign (%) indicates a General Purpose Register to the PDP-11 Assembler, and may be used. Typically, however, the registers are defined as follows:

Rø = %ø
R1 = %1
R2 = %2
R3 = %3
R4 = %4
R5 = %5
SP = %6
PC = %7

These definitions will be used throughout this book.

3.2.2.2 Indirect Addressing

There three modes where the register is used to contain the address of the operand (the effective address):

REGISTER DEFERRED MODE - where the content of the register is used as a "pointer" to the operand and is not modified.

AUTOINCREMENT MODE - where the content of the register is used as a "pointer" to the operand and then is automatically stepped ahead so that it points to the next sequential operand in a table or list.

AUTODECREMENT MODE - where the content of the register is first automatically stepped back and then used as a "pointer" to an operand in a table or list.

Although INDEXED MODE does not meet our general statement exactly (register contains the effective address), it is included in this category because the register will contain part of the effective address. As we will soon discuss, the register contains an index word which is added to a base address to form the address of the operand.

REGISTER DEFERRED MODE

Assembler Syntax	Octal Code
(Rn)	1

Register Deferred Mode specifies that the register contains the address of the operand (the effective address). The content of the selected register is not affected; it is used as a "pointer" to the operand.

Example: The content of R4 is the address of the operand; replace the operand with zeros

Location 3Ø3Ø contains the instruction code for CLR (R4)

ØØ3Ø3Ø | ØØ5Ø14

Before Execution:		After Execution:	
R4	ØØ7ØØØ	R4	ØØ7ØØØ
ØØ7ØØØ	123456	ØØ7ØØØ	ØØØØØØ

Example: The content of R2 is the address of the operand; move the operand to R5

Location 5ØØ contains the instruction code for MOV (R2),R5

ØØØ5ØØ | Ø112Ø5

Before Execution:		After Execution:	
R2	ØØ1ØØØ	R2	ØØ1ØØØ
R5	111666	R5	ØØØ555
ØØ1ØØØ	ØØØ555	ØØ1ØØØ	ØØØ555

AUTOINCREMENT MODE

Assembler Syntax	Octal Code
(Rn)+	2

Autoincrement Mode specifies that the register contains the address of the operand (the effective address), just as with Register Deferred Mode. The difference is that after the content of the register is used as a pointer to the operand, it is automatically stepped so that the register then points to the next sequential operand.

Autoincrement Mode thus provides for the automatic stepping of a pointer through a table or list of operands. Although especially useful for this type of processing, this mode is completely general and may be used for a variety of purposes.

Example: R1 contains the address of the operand; increment the operand and then step the content of R1 by two so that it will point to the next sequential word operand (in a table of operands)

Location 1ØØØ contains the instruction code for INC (R1)+

ØØ1ØØØ | ØØ5221

Before Execution:

R1 | ØØ25ØØ

ØØ25ØØ | ØØØØ11

After Execution:

R1 | ØØ25Ø2

ØØ25ØØ | ØØØØ12

In typical operation, the program would loop back and repeat this same instruction a sufficient number of times to increment each of the entries in the table.

AUTODECREMENT MODE

Assembler Syntax	Octal Code
-(Rn)	4

Autodecrement Mode specifies that the register contains the address of the operand (the effective address), and as with Autoincrement Mode, this address will be modified. The difference is that with Autodecrement Mode, the content of the register is automatically stepped back, and then used as a pointer to the operand. Autodecrement Mode thus provides for the processing of structured data in an inverse direction.

We will later discuss how these post-increment (Autoincrement Mode) and pre-decrement (Autodecrement Mode) features are used to manipulate dynamic tables called stacks.

Example: Replace the operands in Table A with the operands in Table B (in inverse order)

R3 contains the address of the first operand in Table A; R4 contains the address of the next sequential location after the last operand in Table B

Location 5ØØ contains the instruction code for MOV -(R4),(R3)+

ØØØ5ØØ | Ø14423

The results of typical program operation are shown below. The program has looped back and repeated the instruction five times to accomplish the task.

BEFORE EXECUTION:

R3 ØØ1ØØØ

R4 ØØ3Ø12

Table A		Table B	
ØØ1ØØØ	Ø12345	ØØ3ØØØ	Ø22222
ØØ1ØØ2	111111	ØØ3ØØ2	Ø44231
ØØ1ØØ4	ØØØ222	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	1Ø1Ø1Ø	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1ØØ2

R4 ØØ3Ø1Ø

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	111111	ØØ3ØØ2	Ø44231
ØØ1ØØ4	ØØØ222	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	1Ø1Ø1Ø	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1ØØ4

R4 ØØ3ØØ6

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	ØØØ222	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	1Ø1Ø1Ø	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1ØØ6

R4 ØØ3ØØ4

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	Ø567ØØ	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	1Ø1Ø1Ø	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1ØØ6

R4 ØØ3ØØ4

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	Ø567ØØ	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	1Ø1Ø1Ø	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1Ø1Ø

R4 ØØ3ØØ2

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	Ø567ØØ	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	Ø44231	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1Ø1Ø

R4 ØØ3ØØ2

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	Ø567ØØ	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	Ø44231	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø1Ø1Ø1	ØØ3Ø1Ø	Ø6Ø6Ø6

R3 ØØ1Ø12

R4 ØØ3ØØØ

Table A		Table B	
ØØ1ØØØ	Ø6Ø6Ø6	ØØ3ØØØ	Ø22222
ØØ1ØØ2	Ø7Ø7Ø7	ØØ3ØØ2	Ø44231
ØØ1ØØ4	Ø567ØØ	ØØ3ØØ4	Ø567ØØ
ØØ1ØØ6	Ø44231	ØØ3ØØ6	Ø7Ø7Ø7
ØØ1Ø1Ø	Ø22222	ØØ3Ø1Ø	Ø6Ø6Ø6

INDEXED MODE

Assembler Syntax	Octal Code
X(Rn)	6

Indexed Mode specifies that the register contains an index word which is added to a base address (contained in a location following that of the instruction word) to form the address of the operand.* Neither the index word nor the base address word is affected.

Indexed Mode thus provides for the random access of operands in data structures. The index word (the content of the selected register) is modified by the program to access the desired operands in the table or list.

Example: Clear the third operand in Table A

R5 contains the index word; location 7Ø2 contains the base address of Table A

Location 7ØØ contains the instruction code for CLR 1ØØØ(R5)

ØØØ7ØØ	ØØ5Ø65
ØØØ7Ø2	ØØ1ØØØ

Before Execution:

R5	ØØØØØ4
ØØØ7Ø2	ØØ1ØØØ

Table A

ØØ1ØØØ	Ø6Ø6Ø6
ØØ1ØØ2	Ø7Ø7Ø7
ØØ1ØØ4	Ø567ØØ
ØØ1ØØ6	Ø44231
ØØ1Ø1Ø	Ø1Ø1Ø1

After Execution:

R5	ØØØØØ4
ØØØ7Ø2	ØØ1ØØØ

Table A

ØØ1ØØØ	Ø6Ø6Ø6
ØØ1ØØ2	Ø7Ø7Ø7
ØØ1ØØ4	ØØØØØØ
ØØ1ØØ6	Ø44231
ØØ1Ø1Ø	Ø1Ø1Ø1

*This is the more common usage. Realize that at the programmer's option, the selected register may hold the base address.

3.2.2.3 Deferred Addressing

There are two modes where the register is used to contain the address of the effective address:

AUTOINCREMENT DEFERRED MODE - where the content of the register is used as a pointer to the address of the operand in a table of effective addresses. It is first used, and then automatically stepped ahead.

AUTODECREMENT DEFERRED MODE - where the content of the register is used as a pointer to the address of the operand in a table of effective addresses. It is first automatically stepped back, and then used.

Although INDEXED DEFERRED MODE does not meet our general statement exactly (register contains the address of the effective address), it is included in this category because the register will contain part of the address of the effective address. The register contains an index word which is added to a base address to form the address of the effective address.

AUTOINCREMENT DEFERRED MODE

Assembler Syntax	Octal Code
@(Rn)+	3

Autoincrement Deferred Mode specifies that the register contains the address of the effective address. The content of the selected register is used as the address of the effective address, and then is automatically stepped ahead to the next sequential address.

Autoincrement Deferred Mode thus provides for automatically stepping through a table of addresses, commonly called a dispatch table, as a means of accessing operands. The effect of the instruction is dispatched through this table to the operand.

Example: Clear the first operand in Tables A, B, and C

R4 contains the address of the first effective address in the dispatch table

Location 5ØØ contains the instruction code for CLR @(R4)+

ØØØ5ØØ | ØØ5Ø34

The results of typical program execution are shown below. The program has looped back and repeated the instruction three times to accomplish the task.

R4 ØØØ7ØØ

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	Ø12345
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	122221
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	141414
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

R4 ØØØ7Ø2

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	ØØØØØØ
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	122221
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	141414
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

R4 ØØØ7Ø2

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	ØØØØØØ
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	122221
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	141414
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

R4 ØØØ7Ø4

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	ØØØØØØ
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	ØØØØØØ
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	141414
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

R4 ØØØ7Ø4

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	ØØØØØØ
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	ØØØØØØ
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	141414
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

R4 ØØØ7Ø6

Dispatch Table

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table A

ØØ1ØØØ	ØØØØØØ
ØØ1ØØ2	Ø54321
ØØ2ØØ4	123456
ØØ2ØØ6	ØØØ111
ØØ2Ø1Ø	111ØØØ

Table B

ØØ2ØØØ	ØØØØØØ
ØØ2ØØ2	133331
ØØ2ØØ4	144441
ØØ2ØØ6	155551
ØØ2Ø1Ø	166661

Table C

ØØ3ØØØ	ØØØØØØ
ØØ3ØØ2	151515
ØØ3ØØ4	161616
ØØ3ØØ6	171717
ØØ3Ø1Ø	131313

AUTODECREMENT DEFERRED MODE

Assembler Syntax	Octal Code
@-(Rn)	5

Autodecrement Deferred Mode specifies that the register contains the address of the effective address. The content of the selected register is first automatically stepped back, and then used as the address of the effective address.

Autodecrement Deferred Mode thus provides for automatically stepping through a table of addresses in an inverse direction.

Example: Increment the operand

The content of R2 is automatically stepped back, and then used as the address of the effective address (the address of the address of the operand)

Location 2ØØØ contains the instruction code for INC @-(R2)

ØØ2ØØØ | ØØ5252

Before Execution:		After Execution:	
R2	ØØ1ØØ2	R2	ØØ1ØØØ
ØØ1ØØØ	ØØ5ØØØ	ØØ1ØØØ	ØØ5ØØØ
ØØ5ØØØ	111111	ØØ5ØØØ	111112

INDEXED DEFERRED MODE

Assembler Syntax	Octal Code
@X(Rn)	7

Indexed Deferred Mode specifies that the register contains an index word which is added to a base address (contained in a location following that of the instruction) to form the address of the effective address. Neither the index word nor the base address is affected.

Indexed Deferred Mode thus provides for the random access of operands in data structures through a table of addresses.

Example: Using Dispatch Table A, add the first operand in Table C to the content of R5

RØ contains the index word; location 5Ø2 contains the base address for Dispatch Table A

Location 5ØØ contains the instruction code for ADD @7ØØ(RØ),R5

ØØØ5ØØ	Ø67ØØ5
ØØØ5Ø2	ØØØ7ØØ

Before Execution:

R5	1Ø1Ø1Ø
RØ	ØØØØØ4
ØØØ5Ø2	ØØØ7ØØ

Dispatch Table A

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table C

ØØ3ØØØ	Ø1Ø1Ø1
ØØ3ØØ2	Ø2Ø2Ø2
ØØ3ØØ4	Ø3Ø3Ø3
ØØ3ØØ6	Ø4Ø4Ø4
ØØ3Ø1Ø	Ø5Ø5Ø5

After Execution:

R5	111111
RØ	ØØØØØ4
ØØØ5Ø2	ØØØ7ØØ

Dispatch Table A

ØØØ7ØØ	ØØ1ØØØ
ØØØ7Ø2	ØØ2ØØØ
ØØØ7Ø4	ØØ3ØØØ
ØØØ7Ø6	ØØ4ØØØ
ØØØ71Ø	ØØ5ØØØ

Table C

ØØ3ØØØ	Ø1Ø1Ø1
ØØ3ØØ2	Ø2Ø2Ø2
ØØ3ØØ4	Ø3Ø3Ø3
ØØ3ØØ6	Ø4Ø4Ø4
ØØ3Ø1Ø	Ø5Ø5Ø5

3.2.3 Program Counter Register Addressing Modes

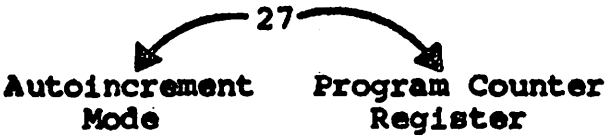
You will recall that Register 7, although a General Purpose Register, also functions as the Program Counter for the PDP-11. In this role, it always contains the address of the next location to be referenced, and is automatically updated by the processor during program operation (after an instruction is fetched from a location, the Program Counter is automatically stepped to contain the address of the next sequential location).

Although any of the eight General Register Addressing Modes we have just discussed may be used in conjunction with any of the eight General Purpose Registers, there are four of these modes with which the Program Counter can provide special advantages for the handling of unstructured data. These are called the Program Counter Register Addressing Modes.

It is important to remember that these "special effect" modes, although classed separately and given unique names, are in operation the same General Register Addressing Modes we have discussed; the only difference is that the register selected is always Register 7, the Program Counter.

IMMEDIATE MODE

Assembler Syntax	Octal Code
#n	27



IMMEDIATE MODE provides for fast access of an operand in that the operand is in a location IMMEDIATEly following that of the instruction word. The operand is actually part (a second word) of the instruction.

This mode uses to good advantage the fact that the processor automatically steps the content of the Program Counter (so that it then points to the operand) after fetching the instruction. When the instruction is executed (the address field is Autoincrement Mode with the PC), the operand is obtained and the content of the Program Counter is stepped (because of the Mode) to the next sequential location.

Example: Move the value 1ØØØ to R4

Locations 54Ø and 542 contain the code for MOV #1ØØØ,R4

ØØØ54Ø	Ø127Ø4
ØØØ542	ØØ1ØØØ

Before Execution:

R4	123456
PC	ØØØ54Ø
ØØØ54Ø	Ø127Ø4
ØØØ542	ØØ1ØØØ

After Execution:

R4	ØØ1ØØØ
PC	ØØØ544
ØØØ54Ø	Ø127Ø4
ØØØ542	ØØ1ØØØ

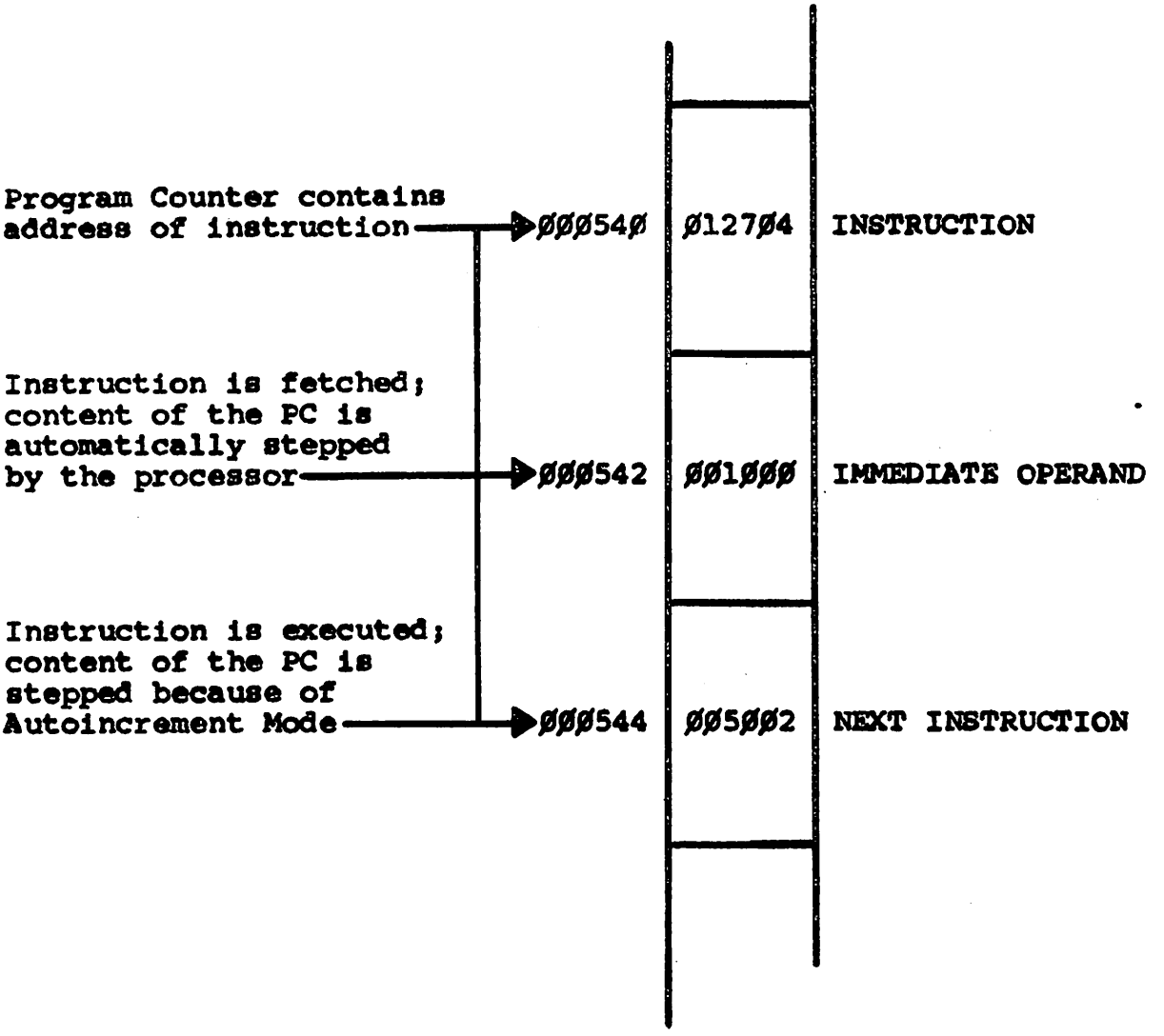


Figure 3-18 Immediate Mode

ABSOLUTE MODE

Assembler Syntax	Octal Code
@#A	37

Autoincrement Deferred Mode — Program Counter Register

ABSOLUTE MODE is Autoincrement Deferred Mode using the Program Counter, where the location immediately following that of the instruction contains the address of the operand.

The immediate data is called an ABSOLUTE address because this address remains constant no matter where in memory the instruction is located and executed. What is the implication? With the PDP-11, programs (and thus the instructions which comprise them) can be relocated in memory for subsequent execution. ABSOLUTE MODE is used to specify the address of an operand when it is desired to have that address be ABSOLUTE regardless of the program (instruction) location at execution time.

As illustrated by Figure 3-19, the same instruction is executed from different locations in memory, and because ABSOLUTE addressing is used, the address of the operand remains constant.

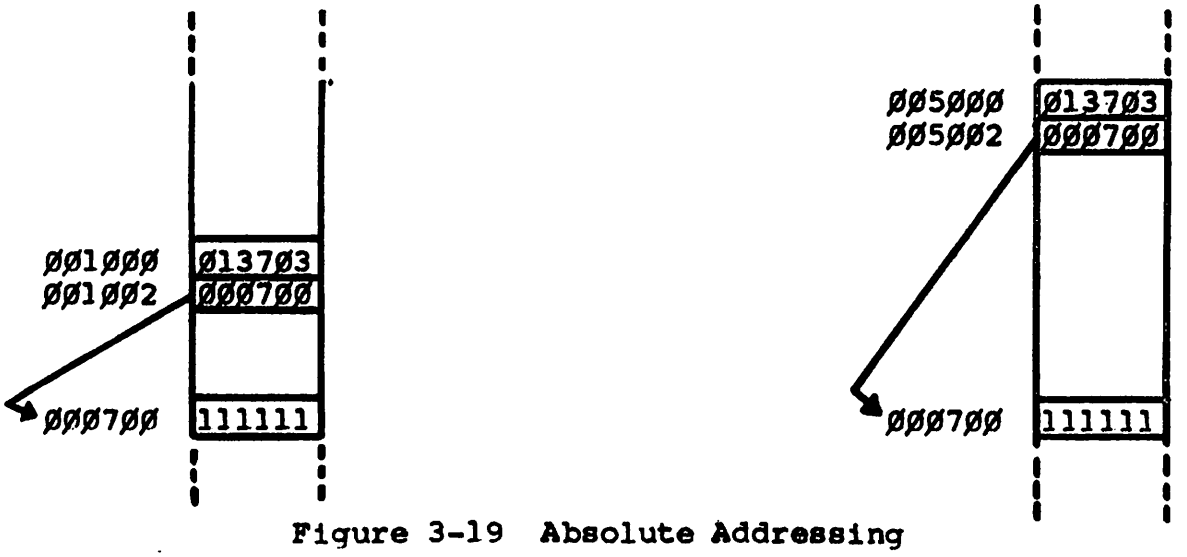


Figure 3-19 Absolute Addressing

Example: Move the content of location 7ØØ to R3

Locations 1ØØØ and 1ØØ2 contain the code for MOV @#7ØØ,R3

ØØ1ØØØ	Ø137Ø3
ØØ1ØØ2	ØØØ7ØØ

Before Execution:		After Execution:	
R3	123456	R3	111111
PC	ØØ1ØØØ	PC	ØØ1ØØ4
ØØØ7ØØ	111111	ØØØ7ØØ	111111

Example: Move the content of location 7ØØ to R3

Locations 5ØØØ and 5ØØ2 contain the code for MOV @#7ØØ,R3

ØØ5ØØØ	Ø137Ø3
ØØ5ØØ2	ØØØ7ØØ

Before Execution:		After Execution:	
R3	123456	R3	111111
PC	ØØ5ØØØ	PC	ØØ5ØØ4
ØØØ7ØØ	111111	ØØØ7ØØ	111111

RELATIVE MODE

Assembler Syntax	Octal Code
A	67

Indexed Mode — Program Counter Register

RELATIVE MODE is used whenever direct reference is made to a memory location, and is assembled as Indexed Mode using the Program Counter. Because the content of the Program Counter is used in the address calculation, the address of the operand is not absolute; it is RELATIVE to the address of the instruction.

Recall that Indexed Mode forms the effective address by adding the content of the specified register (index value) to the content of a location following that of the instruction word (base address). Relative Mode works in the same way, with two distinguishing points:

1. The specified register is the Program Counter, and its content will be updated during the execution of the instruction.

2. The address in the location following that of the instruction is here called an OFFSET, because it serves as an offset to the content of the updated Program Counter.

The following algorithms are used by the assembler:

EFFECTIVE ADDRESS = OFFSET + UPDATED PC
OFFSET = EFFECTIVE ADDRESS - UPDATED PC

Example: Increment the content of location TALLY

Locations 5ØØ and 5Ø2 contain the code for INC TALLY

ØØØ5ØØ	ØØ5267
ØØØ5Ø2	ØØØØ74

Before Execution:

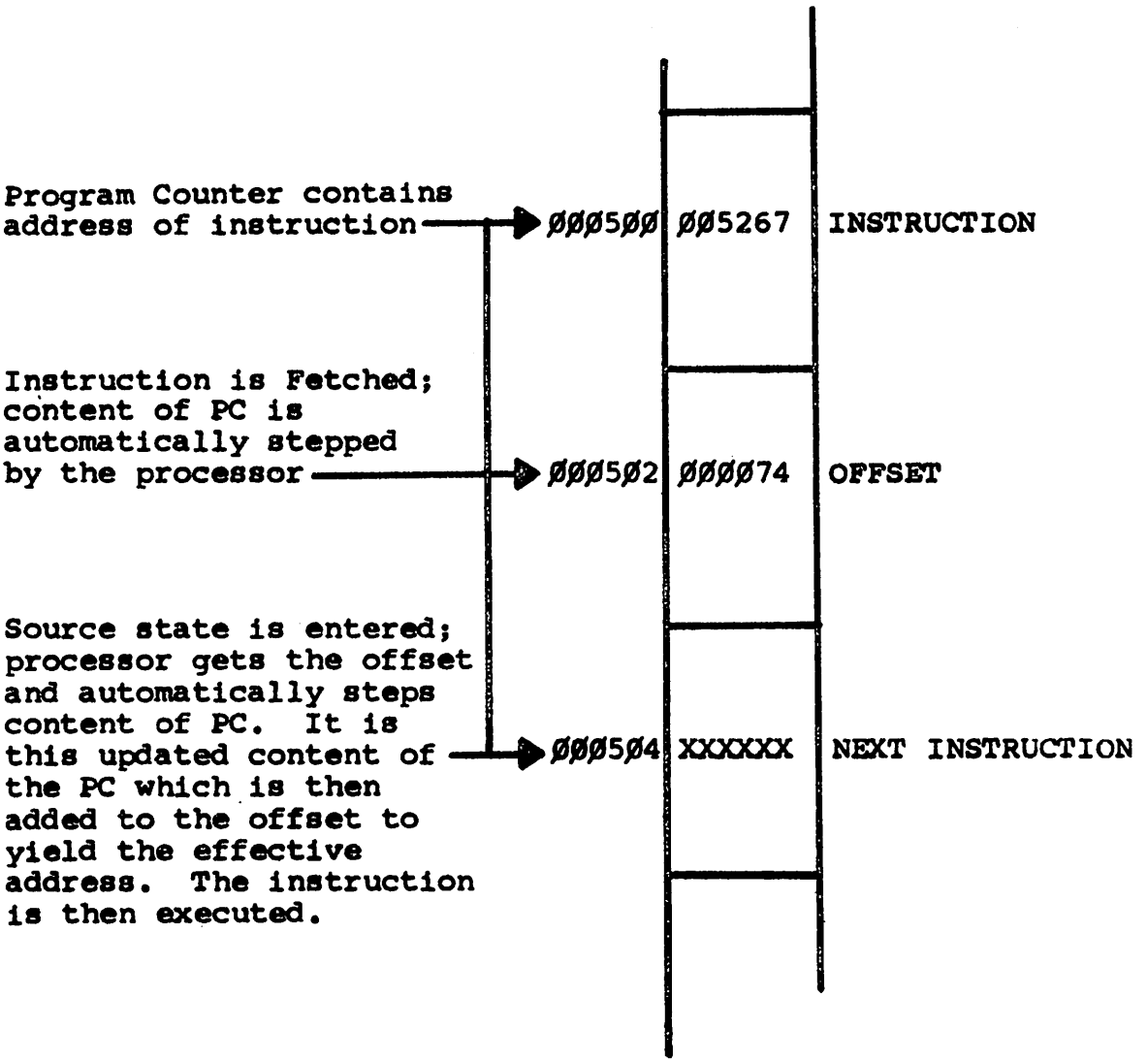
TALLY 123456

PC ØØØ5ØØ

After Execution:

TALLY 123457

PC ØØØ5Ø4



EFFECTIVE ADDRESS = OFFSET + UPDATED PC

= ØØØØ74 (8) + ØØØ5Ø4 (8)

= ØØØ6ØØ (8)

Figure 3-2Ø Relative Mode

Why have two modes (Absolute and Relative) which achieve the same purpose? Though each is used to specify the address of the operand, the method used is not the same, and that is the reason for the existence of both.

If the program is always to be loaded and executed at the same locations in memory, either mode may be used; there is no particular advantage to using one in preference to the other. If, however, the program is to be loaded and executed at various locations in memory (relocated), then there is a preference; Relative mode should be used.

The key to this usage preference is that when a program is relocated, it should be as a complete entity. This means that all locations used by the program (instructions and storage) should be relocated, and that therefore no absolute references be made.

To help clarify this point, let's look at a simple (but not terribly productive!) program wherein the same instructions are used with both Absolute and Relative modes.

```
BEGIN: CLR @#SAVE
       HALT
SAVE:  Ø
```

```
BEGIN: CLR SAVE
       HALT
SAVE:  Ø
```

As illustrated by Figure 3-21, both programs were assembled and loaded beginning at location 5ØØ, and then relocated so that they begin at location 3ØØØ. In relocating, the difference becomes clear. With Absolute mode, a location used by the program has been left "dangling" far behind; with Relative mode, our program remains a "compact" whole.

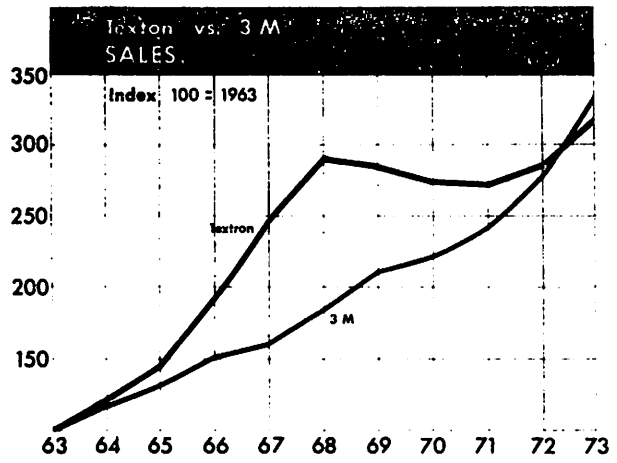


CHART # 12

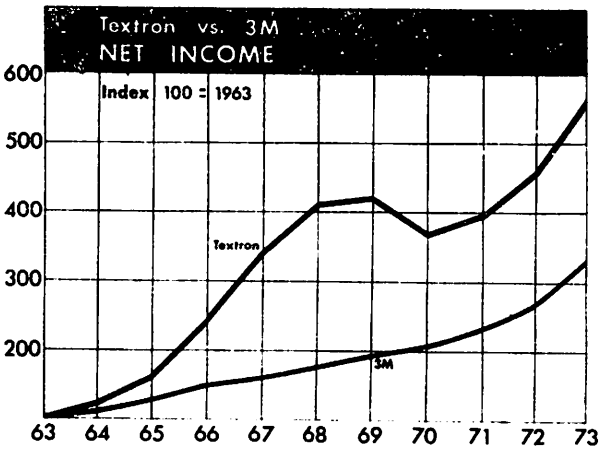


CHART # 13

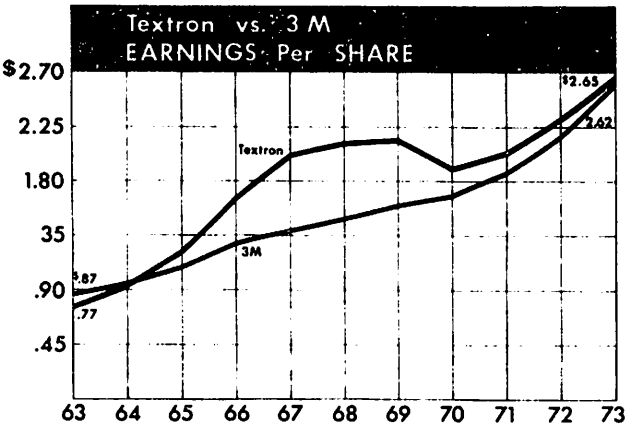


CHART # 14

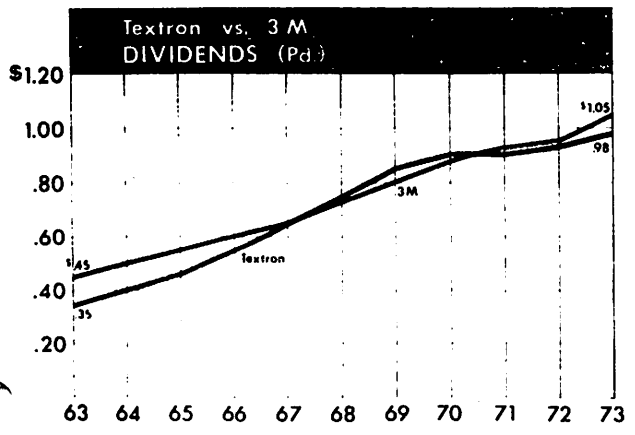


CHART # 15

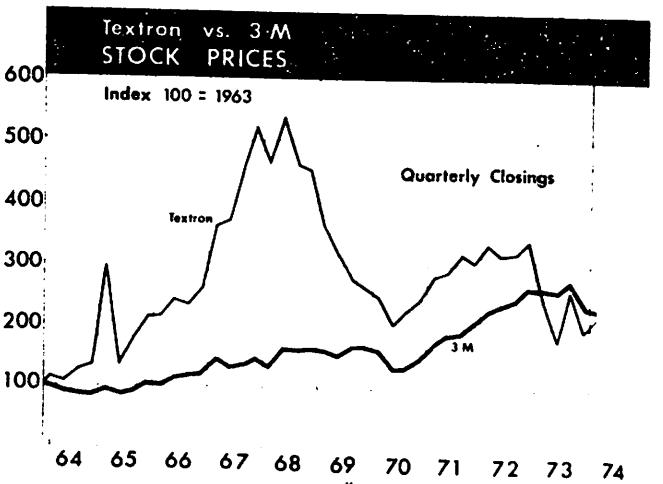


CHART # 16

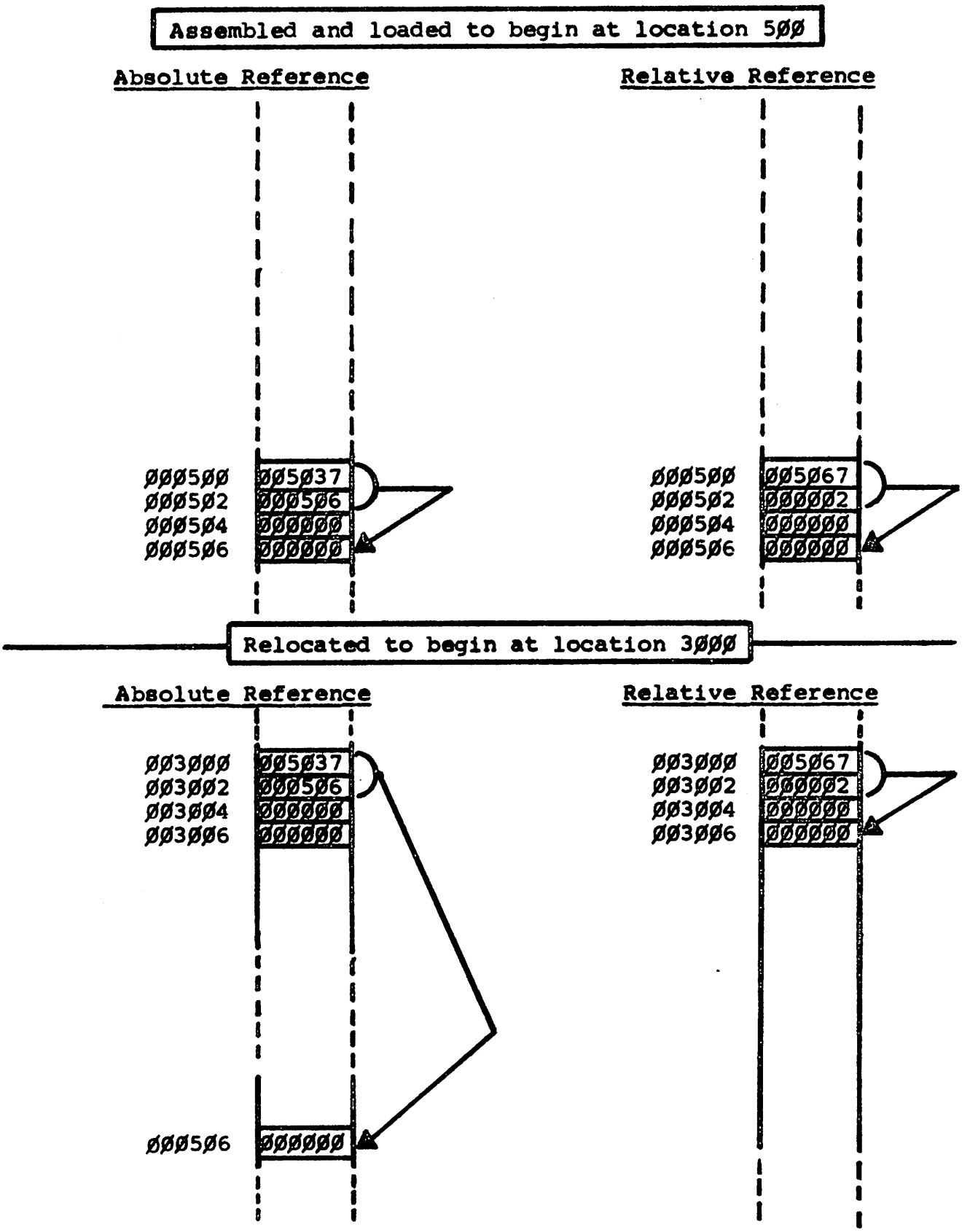
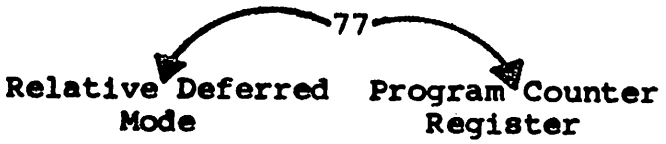


Figure 3-22 Absolute Reference vs. Relative Reference

RELATIVE DEFERRED MODE

Assembler Syntax: @A

Octal Code: 77



RELATIVE DEFERRED MODE is assembled as Indexed Deferred Mode using the Program Counter. It is similar to Relative Mode, but with the additional level of deferral, the calculation OFFSET + UPDATED PC yields the address of the effective address.

Example: Clear the location pointed to by the content of location 1ØØØ

Locations 5ØØ and 5Ø2 contain the code for CLR @1ØØØ

ØØØ5ØØ	ØØ5Ø77
ØØØ5Ø2	ØØØ274

Before Execution:

ØØ1ØØØ	ØØ2ØØØ
ØØ2ØØØ	171717
PC	ØØØ5ØØ

After Execution:

ØØ1ØØØ	ØØ2ØØØ
ØØ2ØØØ	ØØØØØØ
PC	ØØØ5Ø4

3.2.4 EXERCISES

The following examples based upon the discussion of General Purpose Registers are presented as an optional exercise for the reader. The answers can be found in Appendix B.

3.2.4.1 General Register Addressing Modes

Complete the chart below. This is an instruction list, not a program (consider the given values to be true for each instruction). Given: (R1)=1ØØØ, (R2)=2ØØØ, (2ØØØ)=6ØØØ, (1776)=5ØØØ, (21ØØ)=4ØØØ

SYMBOLIC CODE	OCTAL CODE	SOURCE EFFECTIVE ADDRESS	DESTINATION EFFECTIVE ADDRESS	(R2)
MOV R1,R2				
MOV R1,(R2)				
MOV R1,(R2)+	Ø1Ø122	R1	2ØØØ	2ØØ2
MOV R1,-(R2)				
MOV R1,1ØØ(R2)				
MOV R1,@1ØØ(R2)				
MOV R1,@-(R2)				
MOV R1,@(R2)+				

3.2.4.2 Program Counter Register Addressing Modes

Complete the chart below. This is an instruction list, not a program (consider the given values to be true for each instruction). Given: (RØ)=7ØØØ, (PC)=5ØØ, (123456)=3ØØØ, (3ØØØ)=3ØØ

SYMBOLIC CODE	OCTAL CODE	SOURCE EFFECTIVE ADDRESS	DESTINATION EFFECTIVE ADDRESS	(RØ)
MOV #123456,RØ				
MOV @#123456,RØ				
MOV 123456,RØ	Ø16700 122752	123456	RØ	3ØØØ
MOV @123456,RØ				

3.3 INSTRUCTION SET

3.3.1 Introduction

Before you can write a program, you must have a working knowledge of the instruction set which you are to use. We will then discuss the PDP-11 instruction set, but because this book is introductory in nature, we will limit our discussion to only a part of the basic set. (It is expected that you shall soon become expert and seek the power of the complete instruction set!)

Those instructions we are to discuss will be first listed by format, and then grouped according to function for a more detailed presentation.

Abbreviations, symbols, and other esoteric markings used are as follows:

R or reg = general register (3 bits), Ø-7

SS or src = source address field

DD or dst = destination address field

$\wedge$ = AND

$\vee$ = Inclusive OR

$\veebar$ = Exclusive OR

loc = arbitrary location

■ = Ø for word; 1 for byte

xxx = offset (8 bits)

CONDITION CODE LEGEND

Ø = Clear

1 = Set

* = Conditionally Set

- = Not Affected

3.3.2 Formats

3.3.2.1 Condition Code Operate Group

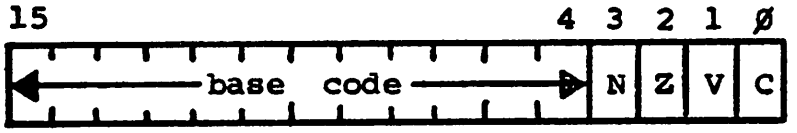


Figure 3-23 Condition Code Operate Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
CLC	CLear C	ØØØ241	-	-	-	Ø
CLV	CLear V	ØØØ242	-	-	Ø	-
CLZ	CLear Z	ØØØ244	-	Ø	-	-
CLN	CLear N	ØØØ25Ø	Ø	-	-	-
SEC	SEt C	ØØØ261	-	-	-	1
SEV	SEt V	ØØØ262	-	-	1	-
SEZ	SEt Z	ØØØ264	-	1	-	-
SEN	SEt N	ØØØ27Ø	1	-	-	-
CCC	Clear all	ØØØ257	Ø	Ø	Ø	Ø
SCC	Set all	ØØØ277	1	1	1	1

3.3.2.2 Single Operand

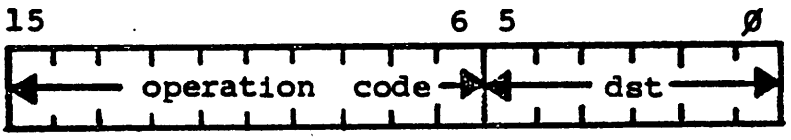


Figure 3-24 Single Operand Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
CLR(B)	CLeaR	■Ø5ØDD	Ø	1	Ø	Ø
COM(B)	COMplement	■Ø51DD	*	*	Ø	1
INC(B)	INCrement	■Ø52DD	*	*	*	-
DEC(B)	DECrement	■Ø53DD	*	*	*	-
NEG(B)	NEGate	■Ø54DD	*	*	*	*
TST(B)	TeST	■Ø57DD	*	*	Ø	Ø
ROR(B)	ROtate Right	■Ø6ØDD	*	*	*	*
ROL(B)	ROtate Left	■Ø61DD	*	*	*	*
ASR(B)	Arithmetic Shift Right	■Ø62DD	*	*	*	*
ASL(B)	Arithmetic Shift Left	■Ø63DD	*	*	*	*
JMP	JuMP	ØØØ1DD	-	-	-	-
SWAB	SWAp Bytes	ØØØ3DD	*	*	Ø	Ø

3.3.2.3 Double Operand

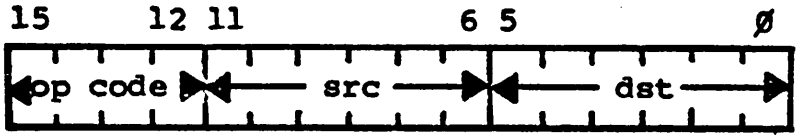


Figure 3-25 Double Operand Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
MOV(B)	MOVe	■1SSDD	*	*	ø	-
CMP(B)	CoMPare	■2SSDD	*	*	*	*
BIT(B)	BIt Test	■3SSDD	*	*	ø	-
BIC(B)	BIt Clear	■4SSDD	*	*	ø	1
BIS(B)	BIt Set	■5SSDD	*	*	ø	1
ADD	ADD	ø6SSDD	*	*	*	*
SUB	SUBtract	16SSDD	*	*	*	*

3.3.2.4 Operate Group

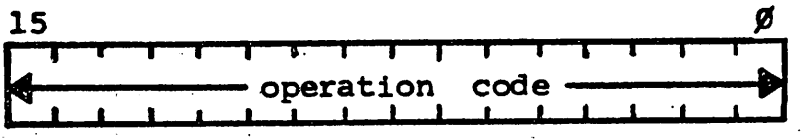


Figure 3-26 Operate Group Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
HALT	HALT	ØØØØØØ	-	-	-	-
RTI	ReTurn from Interrupt	ØØØØØ2	*	*	*	*
TRAP	TRAP	1Ø44ØØ to 1Ø4777	*	*	*	*

3.3.2.5 Branches

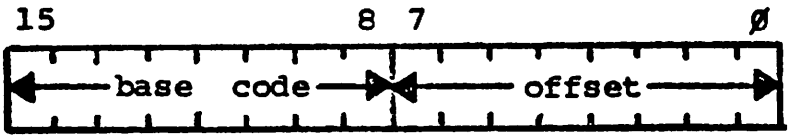


Figure 3-27 Branch Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
BR	BRanch (unconditional)	øøø4øø+xxx	-	-	-	-
BNE	Branch if Not Equal ø	øø1øøø+xxx	-	-	-	-
BEQ	Branch if Equal ø	øø14øø+xxx	-	-	-	-
BPL	Branch if PLus	1øøøøø+xxx	-	-	-	-
BMI	Branch if MInus	1øø4øø+xxx	-	-	-	-
BVC	Branch if oVerflow Clear	1ø2øøø+xxx	-	-	-	-
BVS	Branch if oVerflow Set	1ø24øø+xxx	-	-	-	-
BCC	Branch if Carry Clear	1ø3øøø+xxx	-	-	-	-
BCS	Branch if Carry Set	1ø34øø+xxx	-	-	-	-
BHI	Branch if HIgher	1ø1øøø+xxx	-	-	-	-
BLOS	Branch if LOwer or Same	1ø14øø+xxx	-	-	-	-
BHIS	Branch if HIgher or Same	1ø3øøø+xxx	-	-	-	-
BLO	Branch if LOwer	1ø34øø+xxx	-	-	-	-
BGE	Branch if Greater than or Equal ø	øø2øøø+xxx	-	-	-	-
BLT	Branch if Less Than ø	øø24øø+xxx	-	-	-	-
BGT	Branch if Greater Than ø	øø3øøø+xxx	-	-	-	-
BLE	Branch if Less tha or Equal ø	øø34øø+xxx	-	-	-	-

3.3.2.6 Subroutine Call

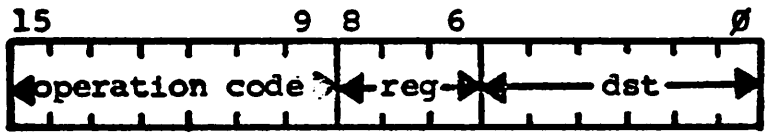


Figure 3-27 Subroutine Call Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
JSR	Jump to SubRoutine	ØØ4RDD	-	-	-	-

3.3.2.7 Subroutine Return

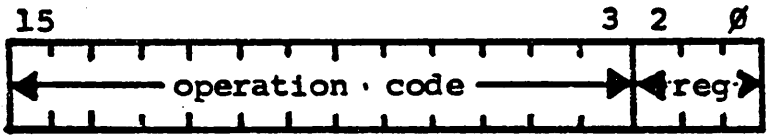


Figure 3-28 Subroutine Return Format

MNEMONIC	INSTRUCTION	OPERATION CODE	CONDITION CODES N	Z	V	C
RTS	ReTurn from Subroutine	ØØØ2ØR	-	-	-	-

3.3.3 A Word About Bytes

You have already learned (and of course retained!) some important facts about PDP-11 memory organization and addressing:

1. Memory is both byte and word addressable; each (16 bit) word is comprised of two (8 bit) bytes. (Refer to Figure 3-4)

2. Bit 15 is the Most Significant Bit and sign bit for both the word and the high (odd) byte; bit 7 is the Most Significant Bit and sign bit for the low (even) byte. (Refer to Figure 3-3)

The instruction set, then, must have the capability of dealing with both word and byte operands.

We will for the most part restrict ourselves in this book to working with words, and therefore word instructions, but it is important to remember that the PDP-11 instruction set includes a full compliment of instructions which manipulate byte operands. A good general performance guide is this: Byte instructions operate upon byte operands in the same way that word instructions operate upon word operands.

In the lists of instructions on the previous pages, you noted that those instructions used to handle both byte and word operands were presented in the following manner:

MNEMONIC	OPERATION CODE
OPR(B)	■NNNNN

The purpose was to indicate that both the mnemonic (symbolic) code and the octal (or binary) operational code differ to specify either the byte or the word operation.

As illustrated by the comparative examples below, the coding procedure is as follows:

To specify a word operation,
do not append the B to the mnemonic code;
use a zero (Ø) as the MSB of the operation code

To specify a byte operation,
do append the B to the mnemonic code;
use a one (1) as the MSB of the operation code

Example: Clear Register Ø

Location 1ØØØ contains the instruction code for CLR RØ

ØØ1ØØØ | ØØ5ØØØ

Before Execution:

RØ | 111111

After Execution:

RØ | ØØØØØØ

Example: Clear the low byte of Register Ø

Location 1ØØØ contains the instruction code for CLRB RØ

ØØ1ØØØ | 1Ø5ØØØ

Before Execution:

RØ | 111111

After Execution:

RØ | 111ØØØ

3.3.4 Condition Code Operate Group

You will recall that the four least significant bits of the Processor Status Word are referred to as the condition code bits.

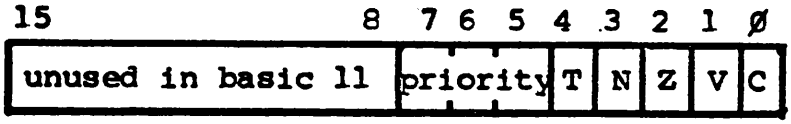


Figure 3-29 Processor Status Word

As was shown in the instruction lists presented earlier, these bits may be implicitly modified by instruction execution

N = 1 if the result was Negative

Z = 1 if the result was Zero

V = 1 if arithmetic oVerflow resulted

C = 1 if a Carry from the MSB position resulted

and will therefore in these cases reflect the result of the previously executed instruction. The information provided by these bits can then be used by other instructions (i.e., Branches) in the program.

These condition code bits may also be explicitly modified by means of the Condition Code Operate instructions. These instructions are commonly used to make sure that certain bit(s) are set (or cleared) before a given programming sequence is begun.

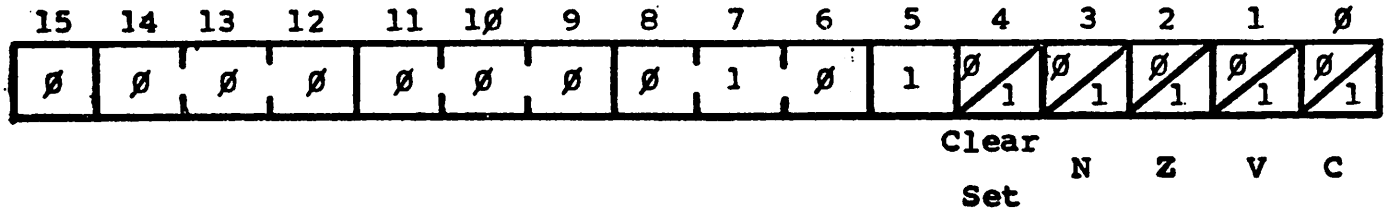


Figure 3-3Ø Condition Code Operate Instruction

Description: Set or Clear condition code bit(s)

The bit 4 position specifies whether the Condition Code Operate Instruction is a Clear (bit 4 = Ø) or a Set (bit 4 = 1)

Bit positions Ø-3 of the Condition Code Operate Instruction (corresponding to the positions of the condition code bits in the Processor Status Word) specify whether or not these bits are to receive the action of the instruction (Ø = no) (1 = yes)

Be aware that many possible combinations (other than CCC or SCC) may be selected for use (i.e., ØØØ243 Clears both C and V)

Condition Codes: All are explicitly Cleared or Set

Examples:

MNEMONIC	INSTRUCTION	OPERATION CODE
CLC	CLear C	ØØØ241
SEC	SEt C	ØØØ261
CLV	CLear V	ØØØ242
SEV	SEt V	ØØØ262
CLZ	CLear Z	ØØØ244
SEZ	SEt Z	ØØØ264
CLN	CLear N	ØØØ25Ø
SEN	SEt N	ØØØ27Ø
CCC	CLear all	ØØØ257
SEC	SEt all	ØØØ277

3.3.5 General/Arithmetic

3.3.5.1 Introduction

The group of single and double operand instructions which follows has been termed General/Arithmetic because each instruction could, depending upon a given usage, be listed in either category.

3.3.5.2 CLEAR

CLeaR DeSTination ■ØØ5ØDD

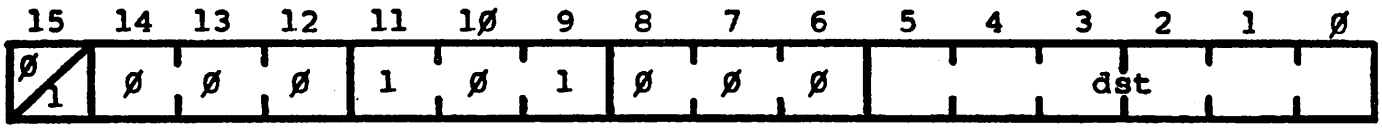


Figure 3-31 Clear Instruction

Description: Replaces the content of the destination location with zeroes

Condition Codes:

N	Cleared
Z	Set
V	Cleared
C	Cleared

Example: Clear location 1ØØØ

Locations 5ØØ and 5Ø2 contain the code for CLR @#1ØØØ

ØØØ5ØØ	ØØ5Ø37
ØØØ5Ø2	ØØ1ØØØ

Before Execution:

ØØ1ØØØ 123456

After Execution:

ØØ1ØØØ ØØØØØØ

3.3.5.3 MOVE

MOVe SouRCe,DeSTination **■1SSDD**

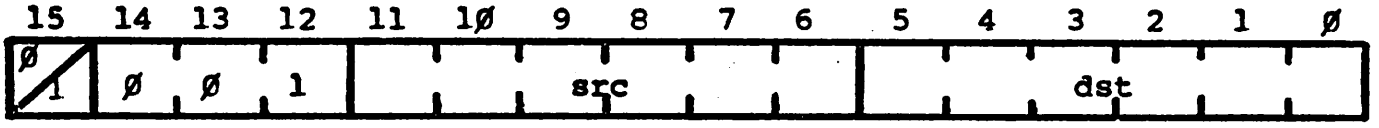


Figure 3-32 Move Instruction

Description: Moves (a copy of) the source operand to the destination location. The content of the source location is not affected; the original content of the destination location is lost (replaced by the copy of the source operand)

Condition Codes:

N	Set if source operand less than zero; Cleared otherwise
Z	Set if source operand is equal to zero; Cleared otherwise
V	Cleared
C	Not affected

Example: Move the value 123456 to Register 3

Locations 5ØØØ and 5ØØ2 contain the code for MOV #123456,R3

ØØ5ØØØ	Ø127Ø3
ØØ5ØØ2	123456

Before Execution:

R3	ØlØlØl
ØØ5ØØ2	123456

After Execution:

R3	123456
ØØ5ØØ2	123456

3.3.5.4 TEST

TeST DeSTination ■Ø57DD

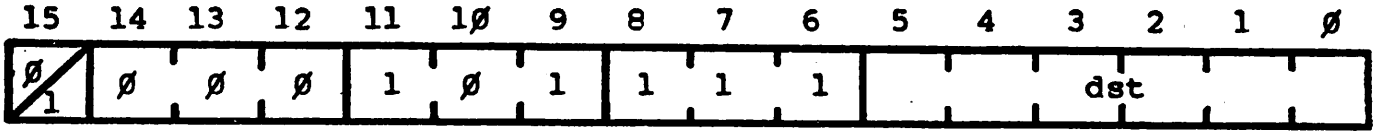


Figure 3-33 Test Instruction

Description: Tests the content of the destination (Specifically for Negative and Zero)

The content of the destination is not affected

Condition Codes:

- N Set if destination operand less than zero; Cleared otherwise
- Z Set if the destination operand is equal to zero; Cleared otherwise
- V Cleared
- C Cleared

Example: Test the content of Register 5

Location 7ØØ contains the code for TST R5

ØØØ7ØØ [ØØ57Ø5]

Before Execution:

R5 [177777]

N	Z	V	C
Ø	1	Ø	1

After Execution:

R5 [177777]

N	Z	V	C
1	Ø	Ø	Ø

3.3.5.5 COMPARE

CoMPare SouRCe,DeSTination ■2SSDD

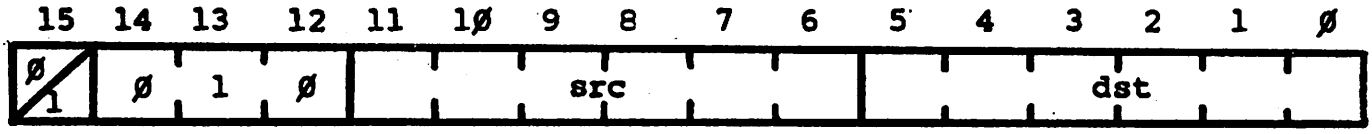


Figure 3-34 Compare Instruction

Description: Compares the source and destination operands by subtraction (source - destination)

Neither operand is affected

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Set if there was arithmetic overflow (operands of opposite signs; sign of the result same as sign of the destination); Cleared otherwise
- **C** Set if there was no carry from the MSB position of the result; Cleared otherwise

Example: Compare the contents of Register 2 and Register 3

Location 5ØØ contains the code for CMP R2,R3

ØØØ5ØØ [Ø2Ø2Ø3]

Before Execution

R2 [ØØØØØ5]

R3 [17776Ø]

N	Z	V	C
Ø	1	Ø	1

After Execution:

R2 [ØØØØØ5]

R3 [17776Ø]

N	Z	V	C
1	Ø	1	1

3.3.5.6 SWAP BYTES

SWAp Bytes DeSTination ØØØ3DD

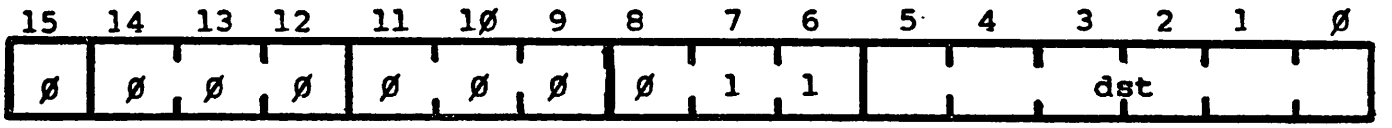


Figure 3-35 Swap Bytes Instruction

Description: Exchanges the high order byte and low order byte of the destination word

Condition Codes:
- N Set if MSB of low order byte (bit 7) of the result is set; Cleared otherwise
- Z Set if low order byte of the result is equal to zero; Cleared otherwise
- V Cleared
- C Cleared

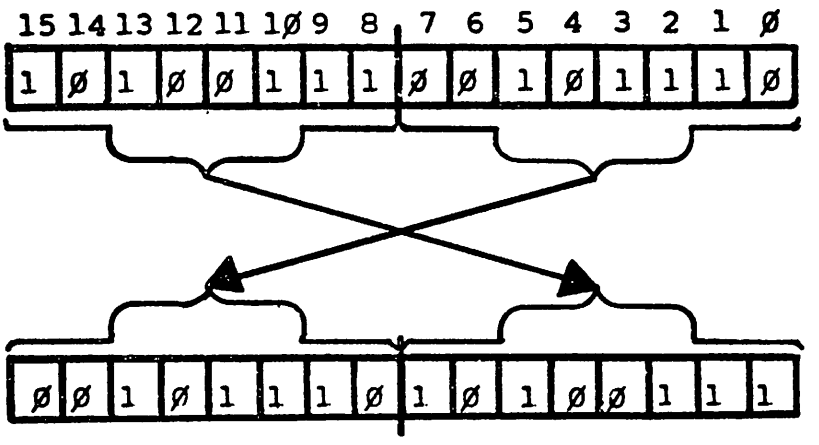
Example: Swap the high order and low order bytes of R4

Location 2ØØØ contains the code for SWAB R4

ØØ2ØØØ ØØØ3Ø4

Before Execution: R4 123456

After Execution: R4 Ø27247



3.3.5.7 ROTATE RIGHT

ROtate Right DeSTination ■Ø6ØDD

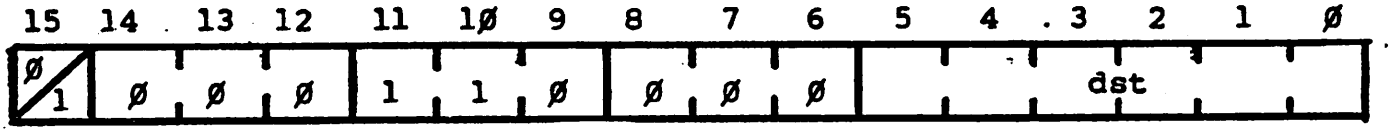
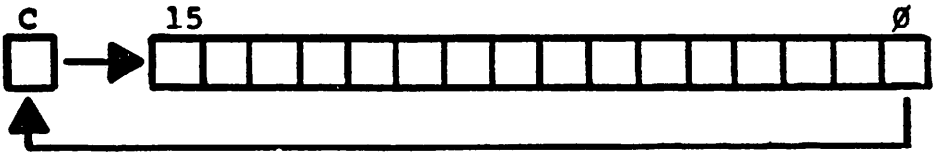


Figure 3-36 Rotate Right Instruction

Description: Rotates all bits of the destination word one place to the right. The content of the C bit of the Processor Status Word is rotated into the bit 15 position, and the content of bit Ø is rotated into the C bit position



A "17 bit connected serial shift" one position to the right which facilitates sequential bit testing and detailed bit manipulation.

Condition Codes:

- **N** Set if the high order bit of the result is set (result less than zero); Cleared otherwise
- **Z** Set if all bits of the result are zeroes; Cleared otherwise
- **V** Loaded with the Exclusive OR of the N bit and C bit (as set by the completion of the Rotate instruction)
- **C** Loaded with the low order bit of the destination

Example: Rotate Right the content of Register Ø

Location 1ØØØ contains the code for ROR RØ

ØØ1ØØØ [ØØ6ØØØ]

Before Execution:

RØ [123456]

C bit Ø

After Execution:

RØ [Ø51627]

C bit Ø

3.3.5.8 ROTATE LEFT

ROtate Left DeSTination ■Ø61DD

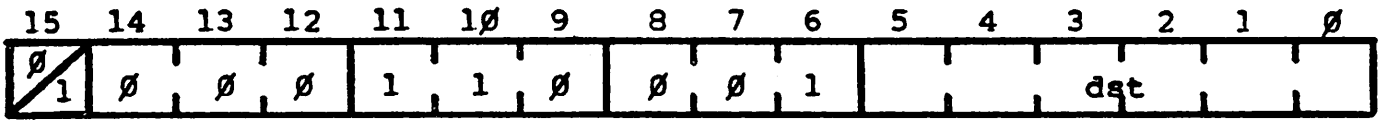
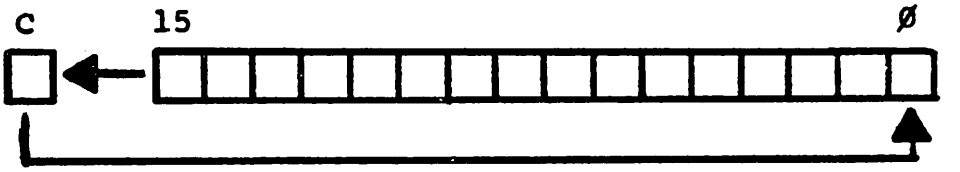


Figure 3-37 Rotate Left Instruction

Description: Rotates all bits of the destination word one place to the left. The content of the C bit of the Processor Status Word is rotated into the bit Ø position, and the content of bit 15 is rotated into the C bit position



A "17 bit connected serial shift" one position to the left which facilitates sequential bit testing and detailed bit manipulation.

Condition Codes:

- N Set if the high order bit of the result is set (result less than zero); Cleared otherwise
- Z Set if all bits of the result are zeroes; Cleared otherwise
- V Loaded with the Exclusive OR of the N bit and C bit (as set by the completion of the Rotate instruction)
- C Loaded with the high order bit of the destination

Example: Rotate Left the content of Register Ø

Location 1ØØØ contains the code for ROL RØ

ØØ1ØØØ ØØ61ØØ

Before Execution:

RØ 123456

C bit Ø

After Execution:

RØ Ø47134

C bit 1

3.3.5.9 ARITHMETIC SHIFT RIGHT

Arithmetic Shift Right DeSTination ■Ø62DD

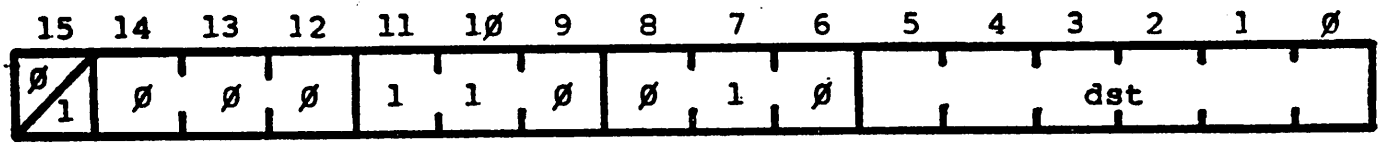
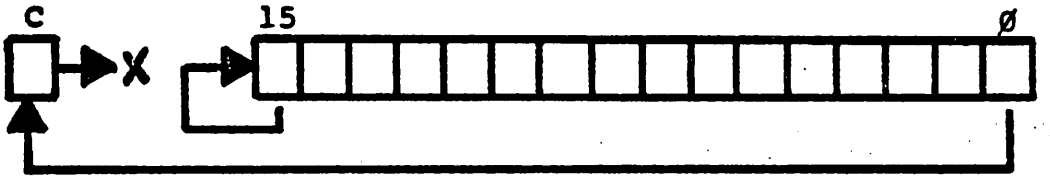


Figure 3-38 Arithmetic Shift Right Instruction

Description: Shifts all bits of the destination location one place to the right. The present content of the C bit is lost as the content of the bit Ø position is shifted in and bit 15 is replicated (to maintain the sign)



The ASR instruction performs signed division by two on the content of the destination location.

Condition Codes:

- **N** Set if the high order bit of the result is set (result less than zero); Cleared otherwise
- **Z** Set if all bits of the destination are zeroes; Cleared otherwise
- **V** Loaded with the Exclusive OR of the N bit and C bit (as set by the completion of the Shift instruction)
- **C** Loaded with the low order bit of the destination

Example: Arithmetic Shift Right the content of Register Ø

Location 1ØØØ contains the code for ASR RØ

ØØ1ØØØ ØØ62ØØ

Before Execution:

RØ 123456

C bit Ø

After Execution:

RØ 151627

C bit Ø

3.3.5.1Ø ARITHMETIC SHIFT LEFT

Arithmetic Shift Left DeSTination ■Ø63DD

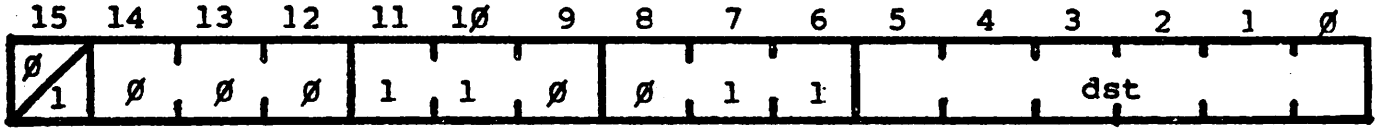


Figure 3-39 Arithmetic Shift Left Instruction

Description: Shifts all bits of the destination location one place to the left. The present content of the C bit is lost as the content of the bit 15 position is shifted in and the bit Ø position is (always) loaded with a zero

The ASL instruction performs signed multiplication by two (with overflow indication) on the content of the destination location.

Condition Codes:

- **N** Set if the high order bit of the result is set (result less than zero); Cleared otherwise
- **Z** Set if all bits of the result are zeroes; Cleared otherwise
- **V** Loaded with the Exclusive OR of the N bit and C bit (as set by the completion of the Shift instruction)
- **C** Loaded with the high order bit of the destination

Example: Arithmetic Shift Left the content of Register Ø

Location 1ØØØ contains the code for ASL RØ

ØØ1ØØØ ØØ63ØØ

Before Execution:

RØ 123456

C bit Ø

After Execution:

RØ Ø47134

C bit 1

3.3.5.11 ADD

ADD SouRCe,DeSTination **Ø6SSDD**

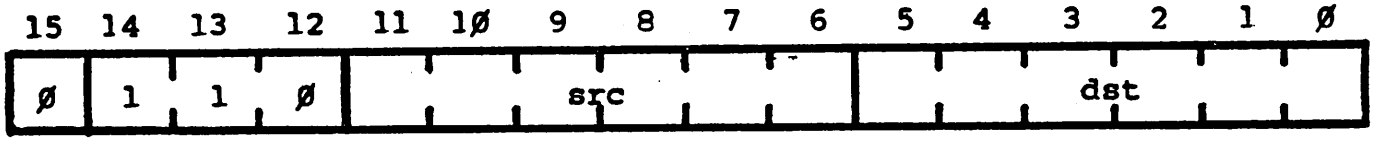


Figure 3-4Ø Add Instruction

Description: Adds the source operand to the destination operand and stores the result in the destination location.

The source operand is unaffected; the destination operand is lost (replaced by the result)

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Set if there was arithmetic overflow (operands of same sign; result of opposite sign); Cleared otherwise
- **C** Set if there was a carry from the MSB of the result; Cleared otherwise

Example: Add the content of Register 3 to the content of location 5ØØØ

Locations 7ØØ and 7Ø2 contain the code for ADD R3,@#5ØØØ

ØØØ7ØØ	Ø6Ø337
ØØØ7Ø2	ØØ5ØØØ

Before Execution:

R3	ØØØ123
ØØ5ØØØ	ØØØ456

After Execution:

R3	ØØØ123
ØØ5ØØØ	ØØØ6Ø1

3.3.5.12 SUBTRACT

SUBtract SouRCe,DeSTination 16SSDD

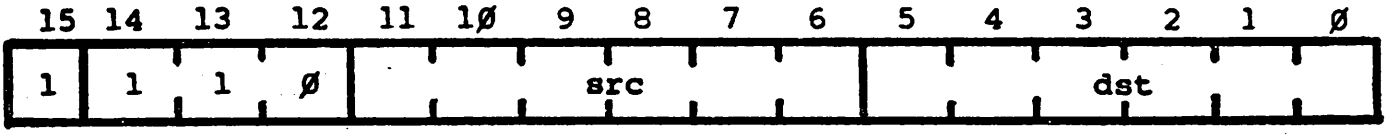


Figure 3-41 Subtract Instruction

Description: Subtracts the source operand from the destination operand (destination - source) and stores the result in the destination location

The source operand is unaffected; the destination operand is lost (replaced by the result)

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Set if there was arithmetic overflow (operands of opposite signs; sign of result same as sign of source); Cleared otherwise
- **C** Set if there was no carry from the MSB position of the result; Cleared otherwise

Example: Subtract the content of Register 3 from the content of location 5ØØØ

Locations 7ØØ and 7Ø2 contain the code for SUB R3,@#5ØØØ

ØØØ7ØØ	16Ø337
ØØØ7Ø2	ØØ5ØØØ

Before Execution:

R3	ØØØ123
ØØ5ØØØ	ØØØ6Ø1

After Execution:

R3	ØØØ123
ØØ5ØØØ	ØØØ456

3.3.5.13 INCREMENT

INCrement DeSTination ■Ø52DD

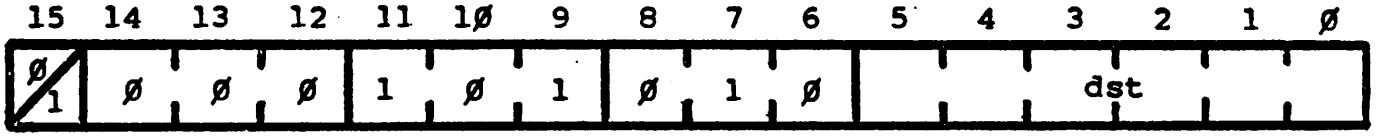


Figure 3-42 Increment Instruction

Description: Adds one to the content of the destination location

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result equal to zero; Cleared otherwise
- **V** Set if the destination operand was Ø77777; Cleared otherwise
- **C** Not affected

Example: Increment the content of Register 5

Location 3ØØØ contains the code for INC R5

ØØ3ØØØ ØØ52Ø5

Before Execution:

R5 123456

After Execution:

R5 123457

3.3.5.14 DECREMENT

DECrement DeSTination ■Ø53DD

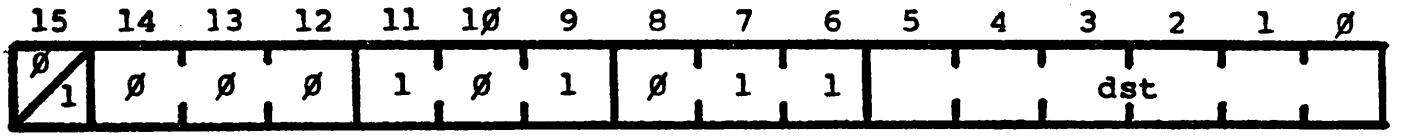


Figure 3-43 Decrement Instruction

Description: Subtracts one from the content of the destination location

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Set if the destination operand was 1ØØØØØ; Cleared otherwise
- **C** Not affected

Example: Decrement the content of Register 5

Location 3ØØØ contains the code for DEC R5

ØØ3ØØØ ØØ53Ø5

Before Execution:

R5 123457

After Execution:

R5 123456

3.3.5.15 COMPLEMENT

COMplement DeSTination **■Ø51DD**

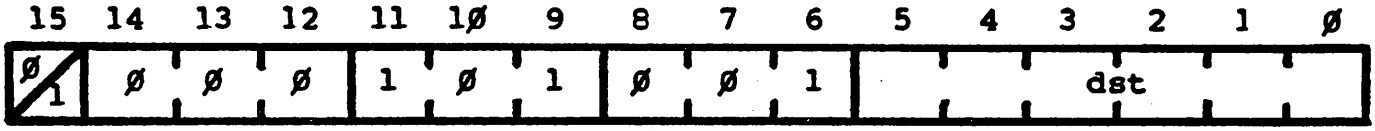


Figure 3-44 Complement Instruction

Description: Replaces the content of the destination location with its one's complement

Condition Codes:

- **N** Set if the MSB of the result is set; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Cleared
- **C** Set

Example: Complement the content of Register 5

Location 3ØØØ contains the code for COM R5

ØØ3ØØØ | ØØ51Ø5 |

Before Execution:

R5 | 123456 |

After Execution:

R5 | Ø54321 |

3.3.5.16 NEGATE

NEGate DeSTination ■Ø54DD

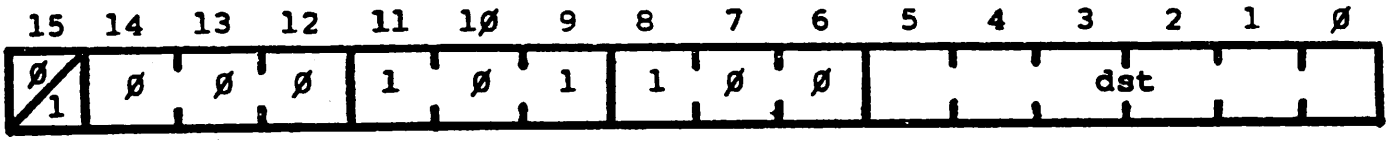


Figure 3-45 Negate Instruction

Description: Replaces the content of the destination location with its two's complement

Condition Codes:

- **N** Set if the result is less than zero; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Set if the result is 1ØØØØØ; Cleared otherwise

Example: Two's complement the content of Register 5

Location 3ØØØ contains the code for NEG R5

ØØ3ØØØ ØØ54Ø5

Before Execution:

R5 123456

After Execution:

R5 Ø54322

3.3.6 Logical

3.3.6.1 Introduction

The group of double operand instructions which follows has been termed Logical because the instructions are based on the logic operations discussed earlier (Section 2.5).

These instructions permit operations on data at the bit level.

3.3.6.2 BIT TEST

BIt Test SouRCe,DeSTination ■3SSDD

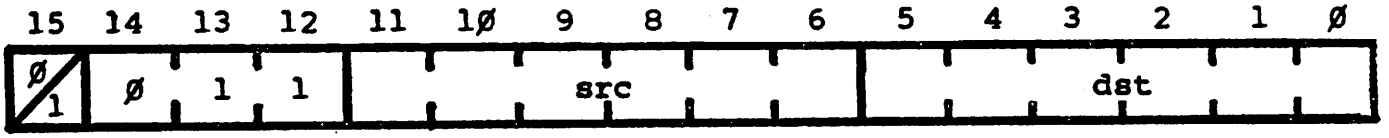


Figure 3-46 Bit Test Instruction

Description: Performs a Logical AND operation between the source and destination operands

Neither operand is affected

This instruction is commonly used for status checking; to determine whether or not certain bit(s) are set (cleared) in a specified word

Condition Codes:

- **N** Set if the MSB of the result is set; Cleared otherwise
- **Z** Set if the result is equal to zero; Cleared otherwise
- **V** Cleared
- **C** Not affected

Example: Location 17756Ø contains the status word for an Input Device; bit 7 is set when a transfer of information is complete. Check this "done bit."

Locations 5ØØØ, 5ØØ2, and 5ØØ4 contain the code for BIT #2ØØ,@#17756Ø

ØØ5ØØØ	Ø32737
ØØ5ØØ2	ØØØ2ØØ
ØØ5ØØ4	17756Ø

Before Execution:

ØØ5ØØ2	ØØØ2ØØ
17756Ø	ØØØØØØ

N	Z	V	C
1	1	Ø	Ø

After Execution:

ØØ5ØØ2	ØØØ2ØØ
17756Ø	ØØØØØØ

N	Z	V	C
Ø	1	Ø	Ø

3.3.6.3 BIT CLEAR

BIt Clear SouRCe,DeSTination ■4SSDD

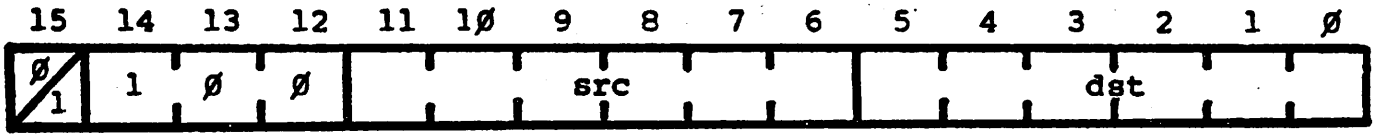


Figure 3-47 Bit Clear Instruction

Description: Clears each bit in the destination operand which corresponds to a set bit in the source operand

The source operand is unaffected. The original destination operand is lost (receives the action of the instruction); replaced by the result

This instruction is commonly used for a function called Masking (getting rid of unwanted bits) or Extracting (saving wanted bits)

Condition Codes:

- N Set if the MSB of the result is set; Cleared otherwise
- Z Set if the result is equal to zero; Cleared otherwise
- V Cleared
- C Not affected

Example: Extract the two Least Significant Digits of the content of location 1ØØØ for future action

Locations 6ØØØ, 6ØØ2, and 6ØØ4 contain the code for BIC #1777ØØ,@#1ØØØ

ØØ6ØØØ	Ø42737
ØØ6ØØ2	1777ØØ
ØØ6ØØ4	ØØ1ØØØ

Before Execution:

ØØ6ØØ2	1777ØØ
ØØ1ØØØ	123456

N	Z	V	C
1	1	Ø	Ø

After Execution:

ØØ6ØØ2	1777ØØ
ØØ1ØØØ	ØØØØ56

N	Z	V	C
Ø	Ø	Ø	Ø

3.3.6.4 BIT SET

BIt Set SouRCe,DeSTination ■5SSDD

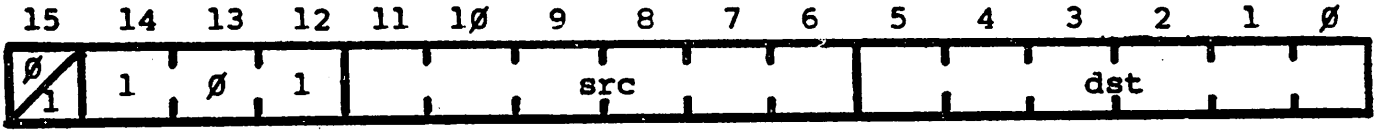


Figure 3-48 Bit Set Instruction

Description: Performs an Inclusive OR operation between the source and destination operands

The source operand is unaffected. The original destination operand is lost (receives the action of the instruction); replaced by the result

This instruction is commonly used when it is desired to set certain bit(s) within a given word without affecting the other bits

Condition Codes:

- N Set if the MSB of the result is set; Cleared otherwise
- Z Set if the result is equal to zero; Cleared otherwise
- V Cleared
- C Not affected

Example: Location 177560 contains the status word for an Input Device; setting bit 6 enables the Program Interrupt Facility. Do this.

Locations 7000, 7002, and 7004 contain the code for BIS #100,@#177560

007000	052737
007002	000100
007004	177560

Before Execution:

007002	000100
177560	000000

N	Z	V	C
1	1	0	0

After Execution:

007002	000100
177560	000100

N	Z	V	C
0	0	0	0

3.3.7 Program Control

3.3.7.1 Introduction

The group of instructions which follows has been termed Program Control because these instructions control the flow of the program. Typically, such instructions cause a change from one sequence of instructions to another.

3.3.7.2 JUMP

JuMP DeSTination ØØØ1DD

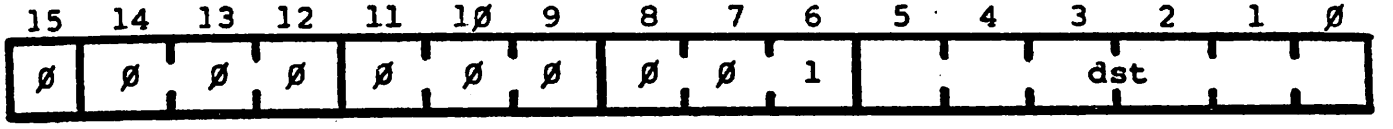


Figure 3-49 Jump Instruction

Description: Transfers program control to any location in memory (destination address calculated and loaded into the Program Counter)

Condition Codes:

N	Not affected
Z	Not affected
V	Not affected
C	Not affected

Example: Transfer program control to location LOOP (arbitrarily defined as location 1ØØØ)

Locations 5ØØ and 5Ø2 contain the code for JMP LOOP

ØØØ5ØØ	ØØØ167
ØØØ5Ø2	ØØØ274

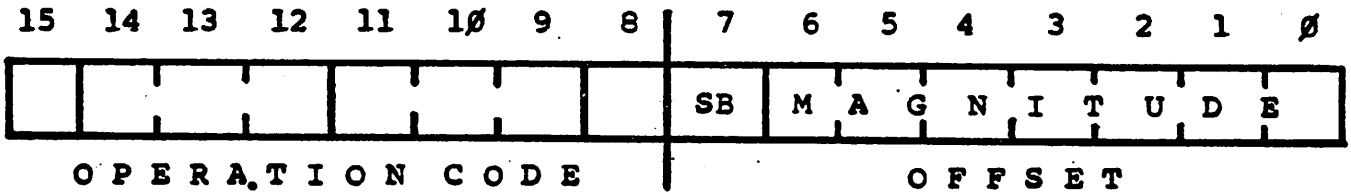
Before Execution:

PC ØØØ5ØØ

After Execution:

PC ØØ1ØØØ

BRANCH INSTRUCTIONS



OPERATION

Unconditional Branch
Transfer program control to the location defined by the offset

Conditional Branches
Check the appropriate Condition Code bit(s)
- -If the condition(s) met, transfer program control to the location defined by the offset
- -If the condition(s) not met, pass control to the next sequential location

OFFSET

The offset is a signed (two's complement) displacement to the PC within the low order 8 bits of every branch instruction

It specifies the number of words from the updated PC to the desired location

CALCULATION

The PC expresses a byte address. Because the offset is expressed in words, it is first converted to bytes and then added to the PC to effect the transfer

The algorithms used are:

LOC=(OFFSET x 2) + UPDATED PC OFFSET=(LOC - UPDATED PC)/2

RANGE

The range of any branch instruction is limited by the offset (forward 177_8 words; backward 200_8 words), but advantages are that: (1) All branch instructions (except for BRANCH) are conditional, and (2) The instruction itself will always require only one word (versus the usual two for the Jump instruction)

RANGE (WORDS)	OFFSET (LOW 8 BITS)
-200	200
⋮	⋮
-2	376
-1	377
0	000
+1	001
+2	002
⋮	⋮
+177	177

3.3.7.3 BRANCH

BRanch (to) loc ØØØ4xxx

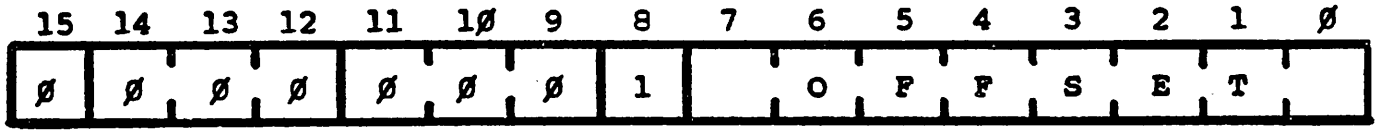


Figure 3-5Ø Branch Instruction

Description: Transfers program control unconditionally to the location defined by the offset

(See next page for general operational information)

Condition Codes: Not affected

Example: Transfer program control to location LOOP (arbitrarily defined as location 1ØØØ)

Location 5ØØ contains the code fo for BR LOOP

ØØØ5ØØ ØØØ537 *

Before Execution:

PC ØØØ5ØØ

After Execution:

PC ØØ1ØØØ

* OS=(LOC-UPC)/2
OS=($1ØØØ_8$-$5Ø2_8$)/2
OS=$276_8/2$
OS=137_8

3.3.7.4 BRANCH IF EQUAL ZERO

Branch if EQual zero (to) loc **ØØ14xxx**

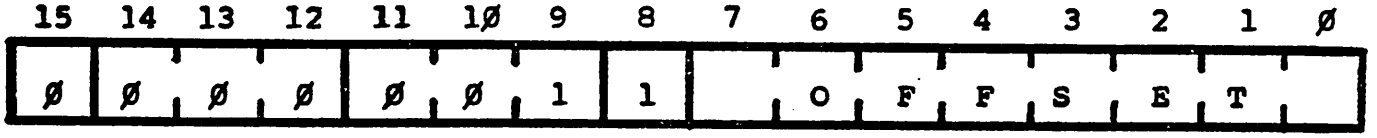


Figure 3-51 Branch if Equal Instruction

Description: Transfers program control to the location defined by the offset IF the condition (Z bit set) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Compare the contents of Register 1 and Register 2. Branch to location SAME (arbitrarily 1Ø5Ø) if the contents are equal

Locations 1ØØØ and 1ØØ2 contain the code for
CMP R1,R2
BEQ SAME

ØØ1ØØØ	Ø2Ø1Ø2
ØØ1ØØ2	ØØ1422 *

Before Execution:

R1	ØØØ5ØØ
R2	ØØØ5ØØ
PC	ØØ1ØØØ

After Execution:

R1	ØØØ5ØØ
R2	ØØØ5ØØ
PC	ØØ1Ø5Ø

* OS=(LOC-UPC)/2
OS=($1Ø5Ø_8$-$1ØØ4_8$)/2
OS=$44_8/2$
OS=22_8

3.3.7.5 BRANCH IF NOT EQUAL ZERO

Branch if Not Equal zero (to) loc ØØ1ØxxxØ

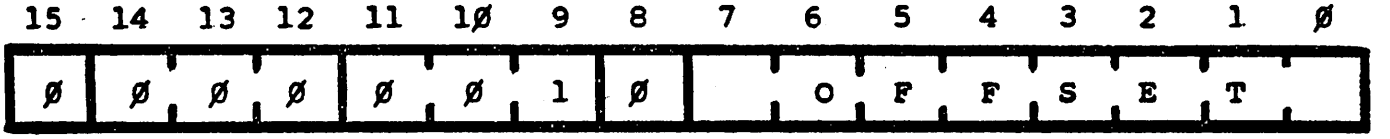


Figure 3-52 Branch if Not Equal Instruction

Description: Transfers program control to the location defined by the offset IF the condition (Z bit clear) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Compare the contents of Register 1 and Register 2. Branch to location DIFF (arbitrarily 1Ø5Ø) if the contents are not equal

Locations 1ØØØ and 1ØØ2 contain the code for

```
CMP R1,R2
BNE DIFF
```

ØØ1ØØØ	Ø2Ø1Ø2
ØØ1ØØ2	ØØ1Ø22 *

Before Execution:

R1	ØØØ5ØØ
R2	ØØØ5ØØ
PC	ØØ1ØØØ

After Execution:

R1	ØØØ5ØØ
R2	ØØØ5ØØ
PC	ØØ1ØØ4

* OS=(LOC-UPC)/2
OS=($1Ø5Ø_8$-$1ØØ4_8$)/2
OS=44_8/2
OS=22_8

3.3.7.6 BRANCH IF PLUS

Branch if PLus (to) loc **1ØØØxxx**

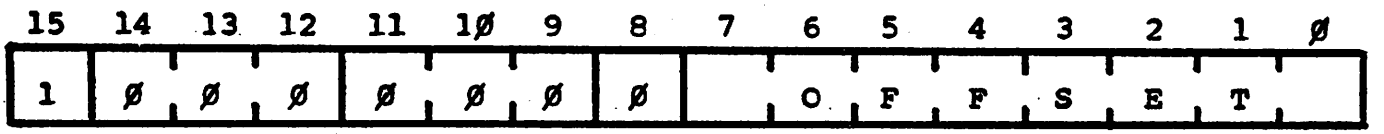


Figure 3-53 Branch if Plus Instruction

Description: Transfers program control to the location defined by the offset IF the condition (N bit clear) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Test the content of Register 4; branch to location POS (arbitrarily 2Ø2Ø) if the content is positive

Locations 2ØØØ and 2ØØ4 contain the code for
TST R4
BPL POS

ØØ2ØØØ ØØ57Ø4
ØØ2ØØ2 1ØØØØ6 *

Before Execution:

R4 111111
PC ØØ2ØØØ

After Execution:

R4 111111
PC ØØ2ØØ4

* OS=(LOC-UPC)/2
OS=(2Ø2Ø$_8$-2ØØ4$_8$)/2
OS=14$_8$/2
OS=6$_8$

3.3.7.7 BRANCH IF MINUS

Branch if MInus (to) loc 1ØØ4xxx

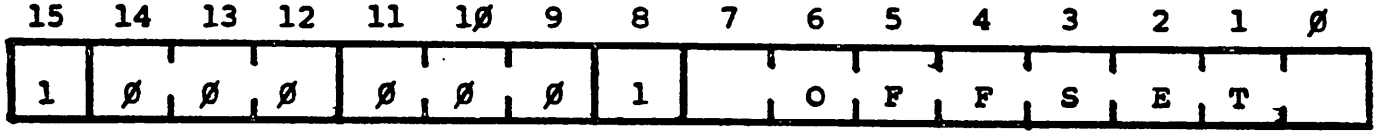


Figure 3-54 Branch if Minus Instruction

Description: Transfers program control to the location defined by the offset IF the condition (N bit set) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Test the content of Register 4; branch to location NEG (arbitrarily 2Ø2Ø) if the content is negative

Locations 2ØØØ and 2ØØ4 contain the code for

```
TST R4
BMI NEG
```

```
ØØ2ØØØ  ØØ57Ø4
ØØ2ØØ2  1ØØ4Ø6  *
```

Before Execution:

R4 111111

PC ØØ2ØØØ

After Execution:

R4 111111

PC ØØ2Ø2Ø

* OS=(LOC-UPC)/2
OS=($2Ø2Ø_8$-$2ØØ4_8$)/2
OS=14_8/2
OS=6_8

3.3.7.8 BRANCH IF CARRY CLEAR

Branch if Carry Clear (to) loc **1Ø3Øxxx**

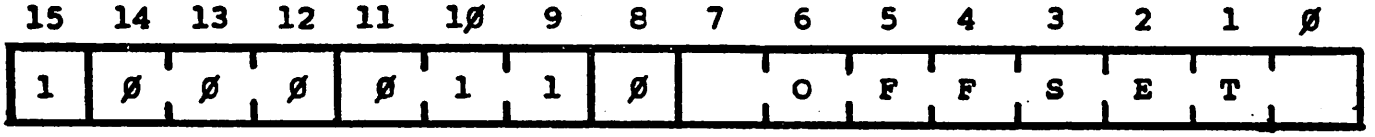


Figure 3-55 Branch if Carry Clear Instruction

Description: Transfers program control to the location defined by the offset IF the condition (C bit clear) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Add the contents of Register 3 and Register 4, then check to see if a carry resulted. If there was no carry, branch to CNO (arbitrarily 5ØØ)

Locations 54Ø and 542 contain the code for
ADD R3,R4
BCC CNO

ØØØ54Ø Ø6Ø3Ø4
ØØØ542 1Ø3356 *

Before Execution:

R3 Ø76543
R4 123456
PC ØØØ54Ø

After Execution:

R3 Ø76543
R4 Ø22221
PC ØØØ544

* OS=(LOC-UPC)/2
OS=($5ØØ_8$-544_8)/2
OS=-44_8/2
OS=-22_8
OS=356_8

3.3.7.9 BRANCH IF CARRY SET

Branch if Carry Set (to) loc **1Ø34xxx**

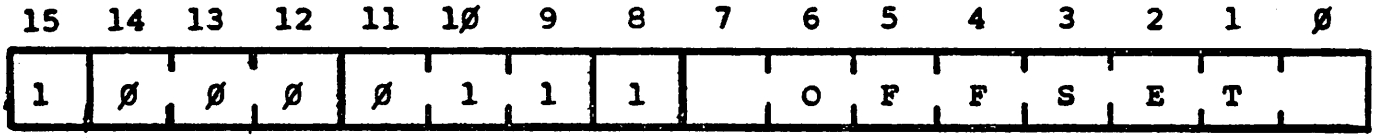


Figure 3-56 Branch if Carry Set Instruction

Description: Transfers program control to the location defined by the offset IF the condition (C bit set) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Add the contents of Register 3 and Register 4, then check to see if a carry resulted. If there was a carry, branch to CYES (arbitrarily 5ØØ)

Locations 54Ø and 542 contain the code for
ADD R3,R4
BCS CYES

ØØØ54Ø Ø6Ø3Ø4
ØØØ542 1Ø3556*

Before Execution:

R3 Ø76543
R4 123456
PC ØØØ54Ø

After Execution:

R3 Ø76543
R4 Ø22221
PC ØØØ5ØØ

* OS=(LOC-UPC)/2
OS=($5ØØ_8$-544_8)/2
OS=-44_8/2
OS=-22_8
OS=356_8

3.3.7.1Ø BRANCH IF OVERFLOW CLEAR

Branch if Overflow Clear (to) loc **1Ø2Øxxx**

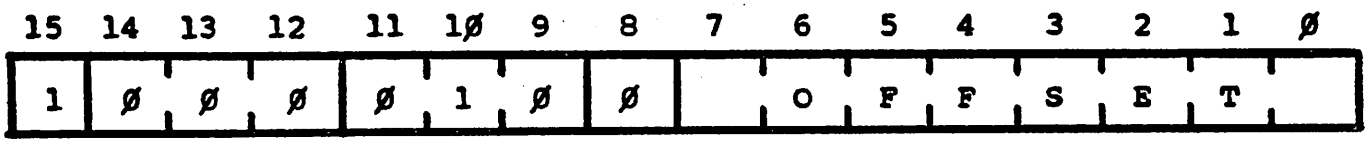


Figure 3-57 Branch if Overflow Clear Instruction

Description: Transfers program control to the location defined by the offset IF the condition (V bit clear) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Rotate right the content of Register 3, and branch to location OK (arbitrarily 5Ø3Ø) if there was no arithmetic overflow

Locations 5ØØØ and 5ØØ2 contain the code for
ROR R3
BVC OK

ØØ5ØØØ	ØØ6ØØ3
ØØ5ØØ2	1Ø2Ø12 *

Before Execution:

C bit Ø

R3 Ø12345

PC ØØ5ØØØ

After Execution:

C bit 1

R3 ØØ5162

PC ØØ5ØØ4

* OS=(LOC-UPC)/2
OS=($5Ø3Ø_8$-$5ØØ4_8$)/2
OS=24_8/2
OS=12_8

3.3.7.11 BRANCH IF OVERFLOW SET

Branch if oVerflow Set (to) loc **1Ø24xxx**

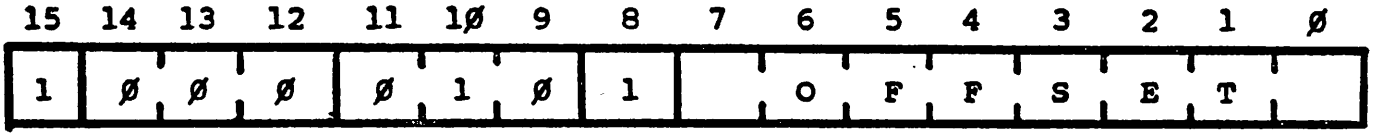


Figure 3-58 Branch if Overflow Set Instruction

Description: Transfers program control to the location defined by the offset IF the condition (V bit set) is met. If the condition is not met, control passes to the next sequential location

Condition Codes: Not affected

Example: Rotate right the content of Register 3, and branch location RESTOR (arbitrarily 5Ø3Ø) if there was arithmetic overflow

Locations 5ØØØ and 5ØØ2 contain the code for
ROR R3
BVS RESTOR

ØØ5ØØØ ØØ6ØØ3
ØØ5ØØ2 1Ø2412 *

Before Execution:

C bit Ø

R3 Ø12345

PC ØØ5ØØØ

After Execution:

C bit 1

R3 ØØ5162

PC ØØ5Ø3Ø

* OS=(LOC-UPC)/2
OS=($5Ø3Ø_8$-$5ØØ4_8$)/2
OS=24_8/2
OS=12_8

APPENDIX A

Answers to selected exercises

Chapter 2

2.6.1

1. ØØ1 1ØØ 1ØØ
2. Ø11 1Ø1 Ø11

2.6.2

1. 1Ø2
2. 54

2.6.3

1. 11Ø4
2. 175Ø

2.6.4

1. 482
2. 512

2.6.5

1. ØØ1234
2. ØØØ576

2.6.6

1. ØØØ ØØØ ØØØ 111 Ø11 11Ø
2. ØØØ ØØØ 1Ø1 Ø1Ø Ø1Ø 1ØØ

2.6.7

1. ØØØ ØØ1 Ø11 1Ø1 Ø1Ø 111
2. ØØØ ØØ1 Ø1Ø 1Ø1 1ØØ Ø11

2.6.8

1. ØØØ ØØØ ØØØ Ø1Ø 11Ø Ø11
2. ØØØ ØØØ ØØØ ØØØ Ø1Ø 11Ø

2.6.9

1. 237773
2. Ø15552

2.6.1Ø

1. ØØ1345
2. ØØØ445

2.6.11

1. ØØØ ØØØ Ø1Ø ØØØ 1ØØ 1ØØ

2.6.12

1. Ø11 Ø11 111 1Ø1 111 111

2.6.13

1. ØØ1 111 ØØ1 Ø11 ØØ1 111

APPENDIX B

Answers to selected exercises

3.2.4.1 General Register Addressing Modes

SYMBOLIC CODE	OCTAL CODE	SOURCE EFFECTIVE ADDRESS	DESTINATION EFFECTIVE ADDRESS	(R2)
MOV R1,R2	Ø1Ø1Ø2	R1	R2	1ØØØ
MOV R1,(R2)	Ø11Ø112	R1	2ØØØ	2ØØØ
MOV R1,(R2)+	Ø1Ø122	R1	2ØØØ	2ØØ2
MOV R1,-(R2)	Ø1Ø142	R1	1776	1776
MOV R1,1ØØ(R2)	Ø1Ø162 ØØØ1ØØ	R1	21ØØ	2ØØØ
MOV R1,@1ØØ(R2)	Ø1Ø172 ØØØ1ØØ	R1	4ØØØ	2ØØØ
MOV R1,@-(R2)	Ø1Ø152	R1	5ØØØ	1776
MOV R1,@(R2)+	Ø11Ø132	R1	6ØØØ	2ØØ2

3.2.4.2 Program Counter Register Addressing Modes

SYMBOLIC CODE	OCTAL CODE	SOURCE EFFECTIVE ADDRESS	DESTINATION EFFECTIVE ADDRESS	(RØ)
MOV #123456,RØ	Ø127ØØ 123456	5Ø2	RØ	123456
MOV @#123456,RØ	Ø137ØØ 123456	123456	RØ	3ØØØ
MOV 123456,RØ	Ø167ØØ 123452	123456	RØ	3ØØØ
MOV @123456,RØ	Ø177ØØ 122752	3ØØØ	RØ	3ØØ

digital

DIGITAL EQUIPMENT CORPORATION, Maynard, Massachusetts. Telephone: (617) 897-5111 • ARIZONA, Phoenix • CALIFORNIA, Sunnyvale, Santa Ana, Los Angeles, San Diego and San Francisco (Mountain View) • COLORADO, Engelwood • CONNECTICUT, Meriden • DISTRICT OF COLUMBIA, Washington (Riverdale, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • ILLINOIS, Northbrook • INDIANA, Indianapolis • LOUISIANA, Metairie • MARYLAND, Riverdale • MASSACHUSETTS, Cambridge and Waltham • MICHIGAN, Ann Arbor and Detroit (Southfield) • MINNESOTA, Minneapolis • MISSOURI, Kansas City and Maryland Heights • NEW JERSEY, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Huntington Station, Manhattan, New York, Syracuse and Rochester • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland, Dayton and Euclid • OKLAHOMA, Tulsa • OREGON, Portland • PENNSYLVANIA, Bluebell, Paoli and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Bellevue • WISCONSIN, Milwaukee • ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Crows Nest, Melbourne, Norwood, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BRAZIL, Rio de Janeiro, Sao Paulo and Porto Alegre • CANADA, Alberta, Vancouver, British Columbia; Hamilton, Mississauga and Ottawa, Ontario; and Quebec • CHILE, Santiago • DENMARK, Copenhagen and Hellerup • FINLAND, Helsinki • FRANCE, Grenoble and Rungis • GERMANY, Cologne, Hannover, Frankfurt, Munich and Stuttgart • INDIA, Bombay • ISRAEL, Tel Aviv • ITALY, Milano • JAPAN, Osaka and Tokyo • MEXICO, Mexico City • NETHERLANDS, The Hague • NEW ZEALAND, Auckland • NORWAY, Oslo • PHILIPPINES, Manila • PUERTO RICO, Miramar and Santurce • REPUBLIC OF CHINA, Taiwan • SCOTLAND, West Lothian • SPAIN, Barcelona and Madrid • SWEDEN, Solna and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Edinburgh, London, Manchester, Reading and Warwickshire • VENEZUELA, Caracas

PRINTED IN USA 0302 00174 3534/E 14 1.2