BASIC THEORY OF DIGITAL COMPUTERS

1 January 1957

VOLUME I

IBM®

BASIC THEORY OF DIGITAL COMPUTERS

VOLUME 1

Contract No.

AF 30(635)-1404

1 January 1957

MILITARY PRODUCTS DIVISION
INTERNATIONAL BUSINESS MACHINES CORPORATION
KINGSTON, NEW YORK

BASIC THEORY OF DIGITAL COMPUTERS

DC1

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

AND

LIST OF TABLES

Draft No. 2

INTERNATIONAL BUSINESS MACHINES CORPORATION
KINGSTON, NEW YORK

UNCLASSIFIED

TABLE OF CONTENTS

PART 1

INTRODUCTION

PART 2

CHAPTER 1

INTRODUCTION

CHAPTER 2

DECIMAL ARITHMETIC

CHAPTER 3

BINARY ARITHMETIC

TABLE OF CONTENTS (cont.)

PART 2

CHAPTER 4

OCTONARY ARITHMETIC

CHAPTER 5

RADIX CONVERSION

CHAPTER 6

PRECISION AND ACCURACY

PART 3

CHAPTER 1

INTRODUCTION TO ELECTRICAL THEORY

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 1 (cont.)

CHAPTER 2

MAGNETISM

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 2 (cont.)

CHAPTER 3

ALTERNATING CURRENT

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 3

CHAPTER 4

MOTORS AND GENERATORS

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 4

CHAPTER 5

ELECTRONICS

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 5

TABLE OF CONTENTS (cont.)

PART 3

CHAPTER 5

PART 4

CHAPTER 1

ARITHMETIC COMPONENTS

TABLE OF CONTENTS (cont.)

PART 4

CHAPTER 2

STORAGE COMPONENTS

CHAPTER 3

MISCELLANEOUS CIRCUITS

TABLES OF CONTENTS (cont.)

PART 4

CHAPTER 3

PART 5

CHAPTER 1

SYSTEM CONSIDERATIONS

CHAPTER 2

ARITHMETIC ELEMENT

TABLE OF CONTENTS (cont.)

PART 5

CHAPTER 2

LIST OF ILLUSTRATIONS

PART 1

PART 2

CHAPTER 1

CHAPTER 2

PART 2

CHAPTER 3

CHAPTER 4

PART 2

CHAPTER 5

PART 3

CHAPTER 1

PART 3

CHAPTER 2

CHAPTER 3

PART 3

CHAPTER 3

CHAPTER 4

PART 3

CHAPTER 4

CHAPTER 5

PART 4

CHAPTER 1

CHAPTER 2

CHAPTER 3

PART 5

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

LIST OF TABLES

BASIC THEORY OF DIGITAL COMPUTERS

DC1

PART 1

INTRODUCTION

Draft No. 2

INTERNATIONAL BUSINESS MACHINES CORPORATION
KINGSTON, NEW YORK

PART 1

INTRODUCTION

1.1 THE NEED FOR COMPUTERS

The collective character of modern civilization has created the need for recording and processing enormous amounts of information. A man may carry in his wallet a driver's license, a car registration, a social security card, a draft card, a union card, club membership cards, a hospitalization card. At home he may have several insurance policies, a number of unpaid bills, cancelled checks, time payment coupon books, a birth certificate for each member of his family, some kind of record of his earnings, a dog license and a marriage license. Each year he may have to fill out renewal forms for several of his licenses and he will certainly have to file an income tax return. While this last chore may place a considerable burden upon the individual, his data processing and computational problems are trivial compared to those of government and industry. Each document that the individual holds implies the existence of vast files in some government or business office. The individual's social security card, for example, would be meaningless if the federal government did not keep complex records concerning the social security status of more than sixty million citizens.

Not only has the quantity of information requiring processing increased as civilization has become more complex, but also new reasons for speed in handling that information

have arisen. The warrior of old aiming an arrow at a slow-moving adversary had no particular calculations to perform. On the other hand, in order to shoot down one of today's fast moving aircraft it is necessary to solve a complex problem in trigonometry involving the aircraft's range, course and speed as well as wind velocity and ballistic data. Moreover, this is what is called a "real-time" problem which means that its solution must be obtained quickly enough to be used to control a process, which in this case is the process of aiming the gun. The gun must be aimed while the aircraft is still within range which is a matter of seconds. Otherwise, the result of the calculation is useless. Clearly, a human with pencil and paper is not going to be able to handle this type of calculation satisfactorily.

As a matter of fact, for problems having any considerable number of stops, a man provided only with pencil and paper is not a successful computer for several reasons. Not only is man an inherently slow calculator, but also he tends to make mistakes. He becomes bored or fatigued.

When a process is mechanized it can be repeated any number of times and the same results obtained. Everyone is familiar with the repeatability of machine processes. In some fields handmade products are highly prized just because each unit is different from every other unit while machine made products are scorned just because every unit is identical to every other unit. But this is certainly not true in the field of information handling. It is desirable to place card A in

front of card B every time it is returned to file. It is equally desirable that 2 x 2 should always yield 4. It has been estimated that an experienced clerk makes one error for every hundred operations he performs. Par for present day computers, on the other hand, is on the order of one error for every 10,000,000 operations. And where the expense can be justified even greater reliability can be built into machines. As to speed, there is no comparison between men and machines. Today's computers can multiply two six digit numbers at speeds as high as 70,000 multiplications per second.

1.2 THE NEEDS OF COMPUTERS

While men are slow and not very accurate they are extremely versatile. On the other hand, it is very expensive to build versatility into machines. Therefore, when a process is to be mechanized, the first task of the machine designer is to reduce that process to a series of simple operations. These should be as few in number and as straightforward as possible. It is obvious that the libraries and files to be consulted by machines will have to be different then those used by men. It is perhaps not so obvious that the methods of computation employed by humans are not readily adaptable to mechanization. Yet this turns out to be the case. Not only are the routines used by humans in performing arithmetic operations unsuitable for mechanization, but, in addition, even the language of human calculation turns out to be somewhat inconvenient for use in high speed computing devices. This language, which is the decimal number system, requires

the use of the ten Arabic numerals 0,1,2,3,4,5,6,7,8,9 for its representation. It is possible to represent any number using only the two digits 0 and 1. This representation is known as binary notation. It is much easier to represent numbers in a machine if only two digits are required because there are any number of devices which can assume either one of two states. For example, a switch can be open or closed or a voltage can be positive or negative. The open switch can then represent 1 while the closed switch represents 0 or the positive voltage can represent 1 while the negative voltage represents 0.

Since large scale digital computers are employed in the solution of any problem largely to save time, it is reasonable that they should be built using those components which afford the greatest advantages in speed of operation. Such components are always electronic or electro-magnetic. Thus today the term high speed computer is synonomous with electronic computer.

1.3 PURPOSE AND PLAN OF THIS BOOK

The purpose of this book is to familiarize the reader with the general theory of digital computers prior to his detailed study of a specific computing system.

In order to understand the large scale digital computing systems of today, it is important to be familiar with the binary system, for this is the number language upon which most computer operations are based. Part 2 of this book, accordingly deals with arithmetic. The decimal number system

is reviewed in this part with the emphasis placed on calling attention to the significance behind some of the operational routines which the average person learns early in childhood and performs automatically thereafter. Various schemes for performing the four arithmetic operations, by means which are more adaptable to mechanization than are pencil and paper methods, are developed in terms of the decimal system. Arithmetic operations in the binary system are then discussed. Pencil and paper methods are compared with those of the decimal system in order to give the reader some familiarity with the new notation. In addition, various schemes for performing the binary operations by means which can readily be mechanized are introduced. There follows a discussion of the octonary number system which is important in computer work because of the relationship which it bears to the binary system. A chapter on radix conversation explains the means for converting numbers from one system of notation to another. Finally, there is a chapter on precision and scaling.

Since high-speed computing components are always electronic or electro-magnetic, it is necessary to have a thorough understanding of basic electrical, electro-magnetic and electronic theory. The reader is assumed to have some training in this field; however basic principles are presented for review and reference purposes in Part 3 of this book.

Part 4 introduces the types of devices which are used as computer components. There is, of course, a chapter dealing specifically with the circuits which perform the arithmetic

operations and another dealing with storage devices. In addition there is a chapter in which the many non-computing circuits required to maintain voltage and power levels and generate and shape pulses are discussed.

In Part 5, the manner in which the components of Part 4 are organized to form a computing system is discussed. Organization of a computer depends to a very large extent upon the type of problem it is intended to solve. The computer developed in Part 5 is assumed to have a mission similar to that of the AN/FSQ-7 Combat Direction Central Computer.

While any detailed exposition of digital computer theory requires knowledge of the arithmetic techniques developed in Part 2; the electrical, magnetic and electronic theory developed in Part 3; and the component theory developed in Part 4; it is still possible to gain insight into the nature of computers in general terms of information flow without any specialized knowledge. Moreover, such insight is useful in approaching the specialized information presented in this book. For this reason, a very general block level discussion of a digital computing system follows.

1.4 COMPUTING SYSTEMS

Computers are classified as either analogue or digital depending upon whether their operation is based upon measurement or upon counting. The slide rule, for example, is a simple analogue computer on which lengths are proportional

to various functions. Multiplication using this device is performed by adding lengths which are proportional to the logarithms of numbers. The adding machine, on the other hand, is a simple digital computer. This device is used to perform arithmetic operations by means of a mechanization of the counting process.

This book is concerned with the study of digital computers, not simple ones like the adding machine, but still machines whose usefulness is based upon a capacity to perform arithmetic operations.

Any computation can be defined as a sequence of individual operations and can be stated in the form of a set of instructions or program. It is fundamental to computer theory that machines can be made to follow instructions. For example, when two numbers are entered on an adding machine, the machine is capable of executing either one of the two instructions, add or subtract. However, it is important to appreciate the fact that machines respond to the form rather than to the content of instructions. Thus there is only one way to tell an adding machine to add and that is to depress the correct key. Once a set of instructions or program has been loaded into the large scale digital computer of today, a long computation involving enumerable instructions may be carried out without any human intervention whatsoever. If the program is cyclic, that is if the last instruction commands the computer to repeat the sequence, then computations can continue indefinitely without

human intervention. But it is still true that all the machine is doing is responding to the form of a set of specific instructions which it was designed to execute.

As a further example of the difference between human and machine response, consider the instruction, "Turn out the light." The meaning of this sentence is interpreted to apply to a particular light in accordance to the circumstances in which the remark is made. Perhaps there is only one light in the room or perhaps the individual issuing the command nods toward a particular light when he speaks. The human who responds to the command, locates a switch on the wall or on the lamp itself and finally turns it around or pushes it up or down depending upon the type of switch it is. Or perhaps he merely replies, "Turn it off yourself." By contrast, the change of the switch position in the instruction which is issued to the light circuit and the circuit must respond to this instruction by extinguishing a particular lamp because the form that the instruction takes is to break the circuit of that lamp.

In order to clarify the use of a program of instructions in operating upon numerical data an example will be given. First, however, it is worthwhile to classify the various functions which must be performed in order to complete any computation. These functions are here considered in terms of their implementation in a manual computing system employing a desk calculating machine and a human operator. The relationship of such a manual system to an automatic computing machine is illustrated

BASIC COMPUTER ELEMENTS

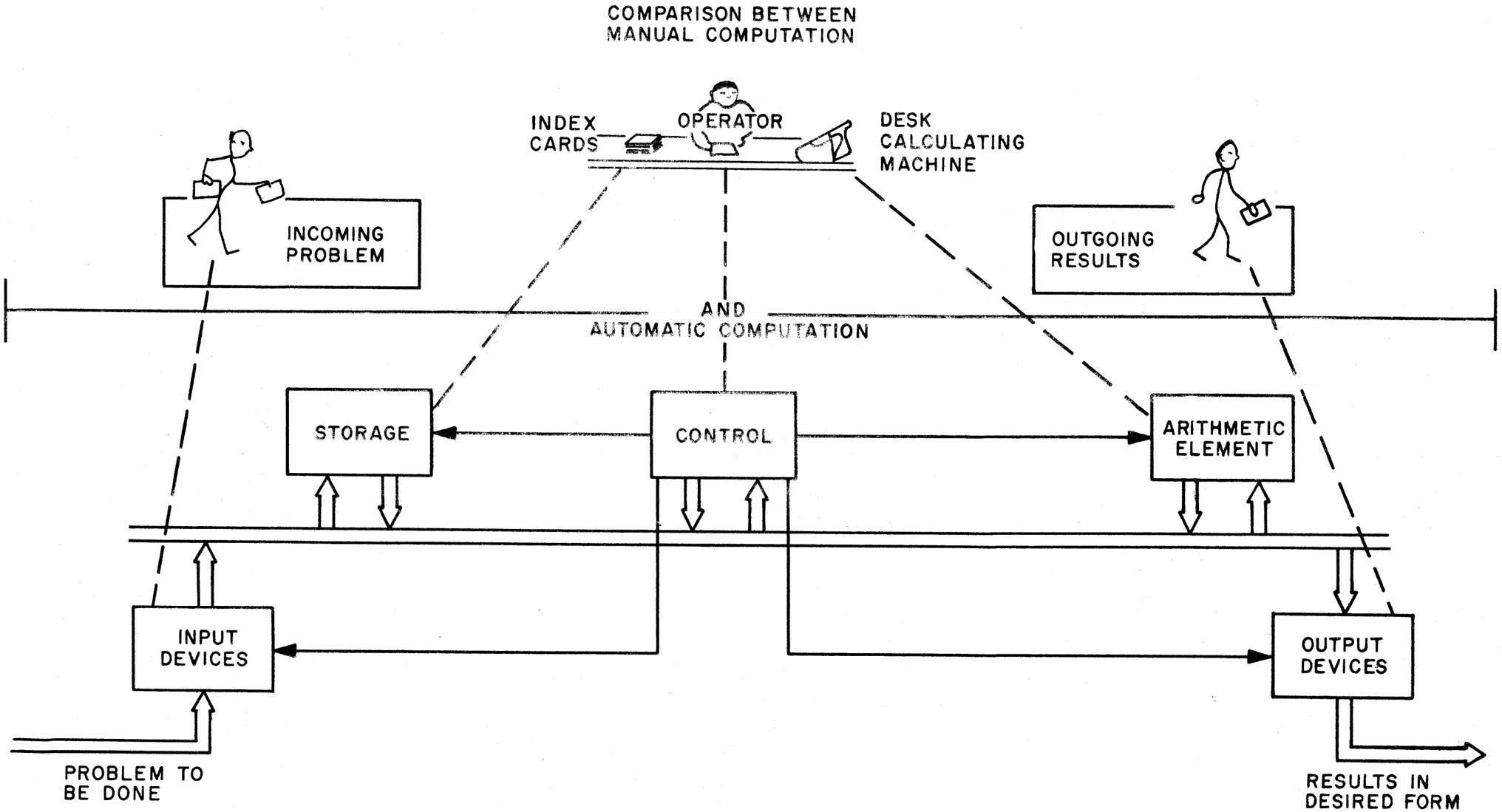


Figure 1-1

in Figure 1-1.

1.4.1 Arithmetic Function

Since a digital computer has been defined as a device which performs the arithmetic operations, it might be expected that every computer would respond to the four explicit commands; add, subtract, multiply and divide. However, this is not the case. The adding machine, for example, has no facilities for inserting explicit multiply and divide commands. Moreover, some quite large general purpose electronic computers do not have facilities for inserting explicit division commands.

The fact that a particular machine is not designed to respond to a command associated explicitly with some particular arithmetic operation does not necessarily mean that the machine cannot be used to perform that operation. For example, an adding machine can be used quite efficiently as a multiplier; however multiplication has to be performed by means of a routine that makes use of the add command. This concept of performing complex operations in terms of routines involving more simple operations is what makes computers the powerful computational tools that they are.

The arithmetic element of the manual computing system illustrated in Figure 1-1 is a desk calculator. This device can be used in fairly straightforward fashion to add, subtract, multiply and divide.

1.4.2 Storage Function

The concept of storage is implied by any computation which cannot be completed instantaneously. Pencil and paper and the human memory are storage devices used during a hand calculation. The fingers are used as a primitive storage device in counting. The desk calculator stores the numbers that are set into it and stores the result until it is cleared. The adding machine stores all operands, intermediate results and final totals on a strip of paper.

If the manual computing system of Figure 1-1 is to be used in the solution of some problem having a number of steps, it is reasonable that the instructions defining each of these steps together with the data to be operated upon should be recorded on some storage medium. Assume, for the purposes of discussion that each separate instruction and each item of data is entered on a separate index card. Assume further that the instruction cards are numbered in the order that the instructions are to be performed and that the data cards are numbered arbitrarily but successively starting with the smallest number which is not used for an instruction card. This numbering of the cards formalizes the process of referring to both instructions and data. Instructions can simply be referred to in the order specified by the card numbers, while data to be operated upon can be specified by the number of the card on which it is written. The set of cards comprises a storage device. Since any instruction or

any item of data can be specified by the number of the card upon which it is written, the card number is properly called the address or location of that instruction or item of data in storage.

1.4.3 Control Function

The human operator is the control element in the manual computing system employing the desk calculator. The function of the operator is to enter the data and instructions contained on the cards onto the calculating machine and to record the results of the arithmetic operations performed by the machine on other cards. This entails inspecting the cards in order and referring to specified cards to obtain items of data. It then entails transfer of instructions and data from the cards to the keys of the calculating machine. Finally, it entails transfer of results from the machine carriage register to numbered cards assigned to hold those results.

Before the human operator can be replaced by machine components, the instructions must be reduced to a code which can be represented by a set of mechanical or electrical states, as for example by the open or closed condition of a number of switches in a complex switching network. The cards must be replaced by some medium whose state can be sensed either mechanically or electrically such as punch tape or relays or magnetic drums. Transfer paths by means of which the conditions representing instructions or data can be made to establish comparable conditions in other parts of the computer must be provided. These paths must be such that they can

be completed or interrupted in accordance with the program of instructions.

1.4.4 Input Output Function

In the manual computing system employing the desk calculator, the input output function is trivial. The only requirement is that the necessary information be delivered to the operator (input) and that the results (output) be collected from him. However, when the machine takes over the control functions from the human operator, then the input output function includes communication between man and machine. Here, then, is where instructions and data are converted into the forms that the computer recognizes. For example, a human operator using a conventional typewriter keyboard may type information into a machine which converts that information into a pattern of holes punched in a card. Here, also is where computer results are converted into a form that can be interpreted by humans. An output device may convert patterns of holes in a card into typewritten information just reversing the translation performed by the input device previously mentioned.

When a computer is used to provide continuous solutions of real time problems then the burden on the input devices is greatly increased. Input data is constantly changing and so must be constantly renewed as the solution continues. Control and guidance information must continually be delivered to output equipment.

1.4.5 Demonstration Problem

In order to demonstrate how specific instructions are used to perform a computation, the familiar data processing problem of completing an income tax return is considered. Here the program is the tax form itself but, before it can be processed by a computer, it must be coded in terms of the instructions which the particular computer is designed to execute. For the purpose of this demonstration problem, an automatic computer capable of executing the instructions listed in Table 1-1 is assumed.

TABLE 1-1

INSTRUCTIONS EXECUTED BY DEMONSTRATION MACHINE

Code	Action
CAD 00	Clears the accumulator which is the chief operational unit of the arithmetic element and sets in the number stored in the address indicated by the address part of the instruction, (in this case 00).
ADD 00	Adds the number stored in the location specified by the address part of the instruction (in this case 00) to the contents of the accumulator.
FST 00	Transfers the contents of the accumulator to the storage location specified in the address part of the instruction (in this case 00).
SUB 00	Subtracts the number stored in the location specified by the address part of the instruction (in this case 00) from the contents of the accumulator.
MUL 00	Multiplies the number stored in the location specified by the address part of the instruction (in this case 00) by the contents of the accumulator.

TABLE 1-1 (Continued)

Code	Action
HLT 00	Stops the machine. Here the address part of the instruction is meaningless

Items of initial data required for the solution of the problem, i.e. the completion of the form, are listed in Table 1-2 while intermediate and final results to be calculated are listed in Table 1-3.

TABLE 1-2

INITIAL DATA REQUIRED FOR DEMONSTRATION PROBLEM

Code	Item	Storage Location
a	salary	21
b	additional income	22
c	medical and dental deductions	23
d	contributions to deduct	24
e	miscellaneous deductions	25
f	exemption per dependent	26
g	number of dependents	27
h	tax rate	28
k	amount withheld	29

TABLE 1-3

INTERMEDIATE AND FINAL CALCULATED RESULTS

DEMONSTRATION PROBLEM

Code	Item
a / b	adjusted gross income
c / d / e	total deductions

TABLE 1-3 (Continued)

Code	Item
(a + b) - (c + d + e)	net income
(a + b) - (c + d + e) - fg	taxable income
((a + b) - (c + d + e) - fg)h	tax
((a + b)- (c + d + e) - fg)h - k	payment to government

The program for the solution of the problem is presented in Table 1-4. It is assumed that the instructions have already been loaded into the storage locations indicated in Table 1-4 and that the initial data has been loaded into the storage locations indicated in Table 1-2. It is further assumed that the control element of the computer examines each of the storage locations in order, interprets the condition found in each location as an order and attempts to execute the order. This makes the stop instruction HLT a necessity. Otherwise, the control element, having exhausted the supply of instructions, would continue to examine succeeding storage locations and would attempt to interpret data found in those locations as instructions.

TABLE 1-4

PROGRAM FOR DEMONSTRATION PROBLEM

Storage Location	Contents	Result of Executing Instruction
1	CAD 26	f in accumulator
2	MUL 27	fg in accumulator
3	FST 30	fg in storage location 30
4	CAD 23	c in accumulator
5	ADD 24	c + d in accumulator
6	ADD 25	c + d + e in accumulator
7	FST 31	c + d + e in storage location 31
8	CAD 21	a in accumulator
9	CAD 22	a + b in accumulator
10	SUB 31	(a + b) - (c + d + e) in accumulator
11	SUB 30	(a + b) - (c + d + e) - fg in accumulator
12	MUL 28	((a + b) - (c + d + e) - fg)h in accumulator
13	SUB 29	((a + b) - (c + d + e) - fg)h k in accumulator
14	HLT	Computer stops

It should be understood that all data represented here by lower case letters of the alphabet are specific numbers; that is, the computer does arithmetic rather than algebra. The letters are used here merely to simplify the discussion.

The first instruction executed by the computer (i.e. the instruction in storage location 1) calls for clearing the accumulator (making it contain zero) and then adding into the accumulator the data contained in storage location 26 (the number f). This instruction is coded in the form; CAD 26.

The second instruction executed by the computer (i.e. the instruction in storage location 2) calls for multiplying the number in storage location 27 (the number g) to the number in the accumulator (the number f). This instruction is coded in the form; MUL 27.

The third instruction executed by the computer (i.e. the instruction in storage location 3) calls for storing the contents of the accumulator (the product fg) in storage location 30. This instruction is coded in the form; FST 30.

The program continues in this manner, each instruction being executed in the order of its storage location number. The sum, c + d + e is formed by execution of instructions 4, 5 and 6, and is placed in storage location 31 by the execution of instruction 7. The sum, a + b is formed by the execution of instructions 8 and 9. The difference, (a + b) - (c + d + e) is formed by the execution of instruction 10. The difference, (a + b) - (c + d + e) - fg is formed by the execution of instruction 11. The product, $\big((a + b) - c + d + e) - fg\big)h$ is formed by the execution of instruction 12. Finally, the difference, $\big((a + b) - (c + d + e) - fg\big)h - k$ is formed by the execution of instruction 13. The computer is then stopped by the execution of instruction 14. At this time, the solution of the problem; that is, the amount of tax to be paid or (if the solution is a negative number) the refund to be claimed, is contained in the accumulator.

This demonstration problem has indicated the steps by which data would be processed in response to a particular coding of a program. Of course, other instructions would be required in order to insert instructions data in the machine, in order to store intermediate results (which are destroyed in the course of the program outlined above), and in order to read out intermediate and final results.

1.4.6 Summary

The demonstration machine of paragraph 1.4.5 solves the problem of the tax return by means of its ability to execute six explicit instructions. Much longer and more complex problems could be solved by other programs employing only these six explicit instructions. However, the versatility of the machine would certainly be increased if its design allowed it to execute a greater number of explicit instructions. Just one example will serve to illustrate this point.

The versatility of the computer considered in the demonstration problem is limited by the fact that it must execute the instructions in the order of their storage location

numbers. Suppose that the machine were capable of carrying out one routine of instructions in response to one contingency and an entirely different routine in response to an alternative contingency. Then the computer would be able to handle problems which can be stated as follows: "If condition A exists, and if condition B exists, and if condition C exists, ..., and if condition N exists, then a particular action P will be taken."

It happens that this contingent type of response can be built into a computer without sacrificing the condition that, in general, instructions should be examined according to a pre-ordered sequence (determined by the numbers of their storage locations). This is done by adding what is called a conditional branch instruction to the set of instructions which the computer can execute. Such an instruction works in the following manner: Suppose that storage location 34 contains a conditional branch instruction specifying that storage location 45 be consulted next if the contents of the accumulator is negative. Notice that this implies a means for sensing the sign of the number in the accumulator. When this sensing operation has been performed, if the contents of the accumulator is found to be negative, then the computer will examine the instruction contained in storage location 45. When the instruction in location 45 has been executed, the computer will next examine the instruction in location 46 (unless 45 contained a second branch instruction). On the other hand, if the contents of the accumulator is found

to be positive then the computer will continue its normal sequence by examining the contents of storage location 35.

An ability to respond to a greater number of instructions usually implies more complexity of equipment. For this reason, any particular computer is designed to respond only to those instructions which are required for the solution of the particular type of problem it is expected to handle. A machine capable of responding to forty or fifty instructions is an extremely flexible computing tool.

BASIC THEORY OF DIGITAL COMPUTERS

DC1

PART 2

MACHINE ARITHMETIC

Draft No. 2

INTERNATIONAL BUSINESS MACHINES CORPORATION
KINGSTON, NEW YORK

PART 2

CHAPTER 1

INTRODUCTION

1.1 NUMBER SYSTEMS

Number words differ from language to language. Where the American says four, the Frenchman says quatre and the German says vier. However, the symbol 4 means the same thing to all three. In fact, so universal is the use of the familiar decimal number system employing the successive digits 0,1,2,3,4, 5,6,7,8,9 that it comes as something of a surprise to learn that it is just one of the many possible systems of number representation. However, the evidence indicates that man developed the decimal system, rather than one of many other possible systems, chiefly because of his habit of counting on his fingers.

When finger counting is compared to counting by means of the decimal system, there appears to be the following rather basic difference: in finger counting, a person can count from zero(closed hands) through ten(ten raised fingers) before he must begin again; in counting by means of the decimal system, it is possible only to count from zero(0) through(9) and then it is necessary to begin again. However, this difference is more apparent than real, since both finger counting and decimal counting are essentially methods of counting by ten's, as will be seen.

In decimal counting, it is necessary to begin again after the digits 0 through 9 have been used. However the

decimal system affords an ingenious scheme for keeping track of the number of complete counts (0through 9) that have been made. This may be compared to a finger counting scheme involving a team of counters in which the first member of the team counts units, the second member counts tens, the third member counts hundreds and so on.The scheme works as follows: each time that the units counter raises his tenth finger(that is, each time he has completed a count of ten units) the tens counter raises a finger. Simultaneously, the units counter closes his hands to indicate zero. Since ten raised fingers merely act as the signal for a transfer of a complete count to the tens counter and since the complete count is indicated by a raised finger belonging to the tens counter, ten raised fingers may be interpreted as a second representation of zero(the other being closed hands). Each time that the tens counter raises his tenth finger (that is each time that he completes a count of ten groups of ten), the hundreds counter raises a finger. Simultaneously the tens counter closes both hands to indicate zero. The size of the group of counters can be extended indefinitely so that any number can be counted off in this manner.

If a number n is multiplied by itself p times, the resulting number is said to be a power of n. Thus 100 is the second power of 10 since it results from multiplying 10 x 10, and 1000 is the third power of 10 since it results from multiplying 10 x 10 x 10. Exponential notation is a convenient shorthand for indicating powers of numbers. For example, n^p indicates that n has been mutiplied by itself p times. In the same way 10^3

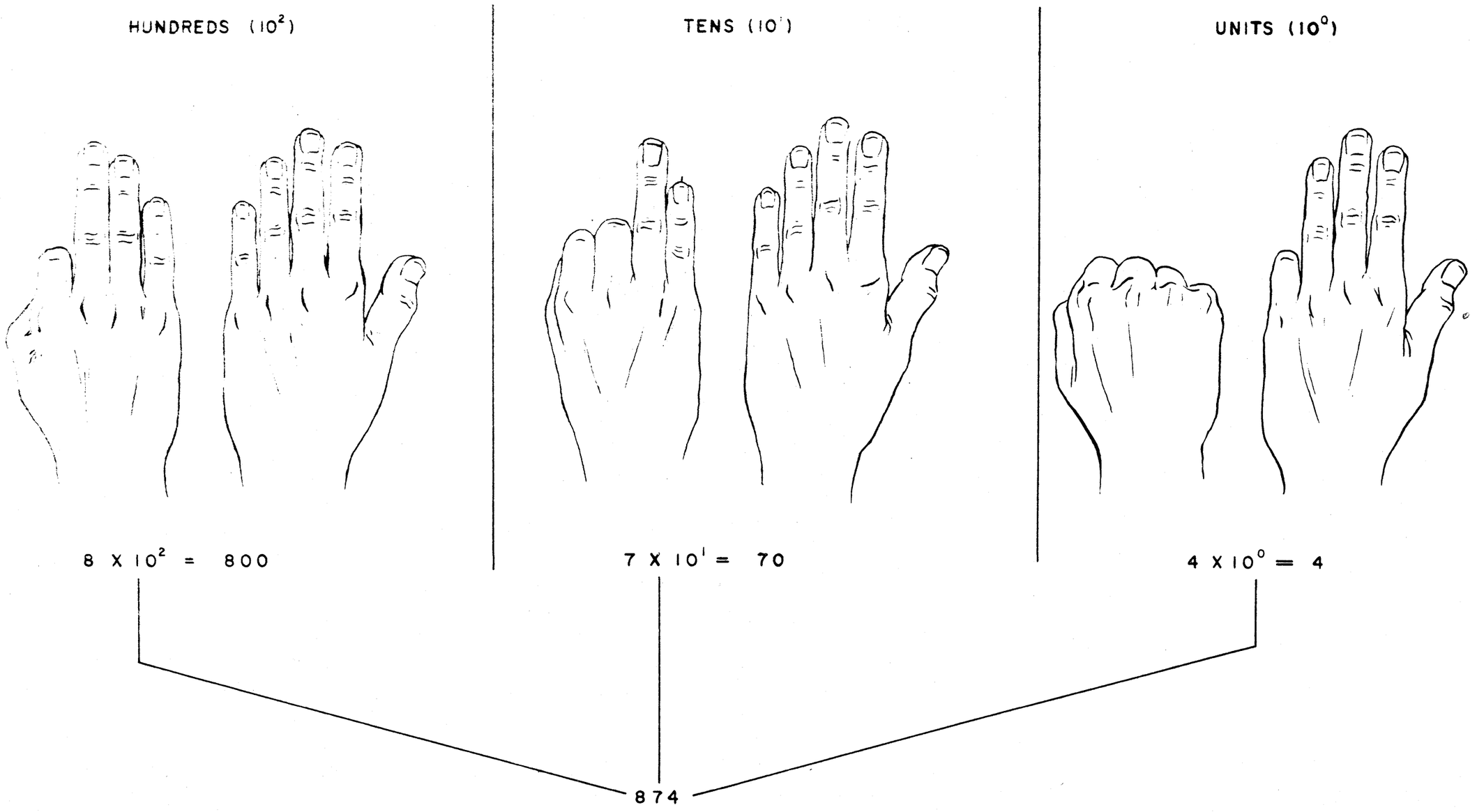

HUNDREDS (10^2)
TENS (10^1)
UNITS (10^0)
$8 \times 10^2 = 800$
$7 \times 10^1 = 70$
$4 \times 10^0 = 4$
874

Figure 2-1

indicates 10 X 10 X 10, that is the third power of 10. A special case is n^0 which is always equal to 1 regardless of the value of n.

In the expression n^p, n is called the base and p is called the exponent. In order to multiply two exponential expressions which have the same base, it is merely necessary to add the exponents. Thus:

$$10^3 \times 10^2 = (10 \times 10 \times 10)\ (10 \times 10) = 10^5$$

Moreover, in order to divide one exponential expression by a second having the same base, it is necessary to subtract the exponent of the second from the exponent of the first. Thus:

$$10^3 \div 10^2 = \frac{10 \times 10 \times 10}{10 \times 10} = 10^1 = 10$$

The activity of the finger counting team introduced above can now be restated as follows: Each member of the team counts by a particular power of ten. Thus the units counter is counting by 10^0, that is he is counting by ones. The second counter is counting by 10^1, that is he is counting by tens. The third counter is counting by 10^2, that is he is counting by hundreds. This is illustrated in Figure 2-1, in which the count represented by the extended (counted) fingers is 874.

With a pencil and paper, one person can keep track of many powers of ten by taking advantage of the positional character of decimal notation. The digits 0 through 9 can be used to represent ten distinct values. Then, it is necessary to start over again. In this case, the new start is made in a second column or place. The value shown in this place is increased by 1 each time a new start is made in the first

column. Thus the second column serves the same function as the second member of the finger counting team. This analogy between columns and members of the counting team can be continued indefinitely. It can be generalized by stating that there is a column associated with every power of ten. Thus the significance of any of the digits 0 through 9 depends upon the column in which it is found. For example, the digit 5 is more significant in the number 571 than it is in the number 751, for in the first number it appears in the 10^2 column and so represents five hundred while in the second number it appears in the 10^1 column and so represents fifty.

Thus far the discussion has involved integers, that is numbers which can be represented using only columns to the left of the familiar decimal point. Each column to the right of the decimal point is associated with the reciprocal of a power of ten. In the first column to the right of the point, 10ths are counted. Thus .3 = 3/10 .In the second column, hundreths are counted. Thus .03 = 3/100. In exponential notation: $\frac{1}{n^p} = n^{-p}$. Thus the column to the right of the decimal point is the 10^{-1} column, the second column is the 10^{-2} column and so on.

Any decimal number represents a sum of products, where each product is a digit multiplied by a power of ten. For example, the number 253 represents the sum: $2 \times 10^2 + 5 \times 10^1 + 3 \times 10^0$. Here, the digit by which each power of ten is multiplied is called the coefficient of that power of ten. Any decimal number, n, is thus of the form:

$$N_{(10)} = C_n 10^n + C_{n-1} 10^{n-1} + \ldots + C_0 10^0 (.) + C_{-1} 10^{-1} + \ldots \quad (1)$$

where each of the coefficients; C_n, C_{n-1}, .., C_o, C_{-1} ,.., can assume any of the values 0 through 9.

For example, the number 24,350.42 represents

$$2 \times 10^4 + 4 \times 10^3 + 3 \times 10^2 + 5 \times 10^1 + 0 \times 10^0$$
$$(.) + 4 \times 10^{-1} + 2 \times 10^{-2}$$

Because a decimal number is a group of digits each of which is interpreted as a coefficient of some power of ten, ten is said to be the radix of the decimal number system. In general if the radix of a number system is R, then a number is represented in that system by a group of digits which are interpreted as coefficients of powers of R. Thus the general expression for a number, N, represented in a system of radix R is as follows:

$$N = C_n R^n + C_{n-1} R^{n-1} + \ldots + C_0 R^0 (.) + C_{-1} R^{-1} + \ldots \quad (2)$$

where each of the coefficients; $C_n, C_{n-1}, \ldots, C_o, C_{-1}$ ...can assume any of the values 0 through R-1.

Notice that equation (1) is that special case of equation (2) for which R 10.

As has already been stated, the development of the decimal number system was probably related to man's habit of counting on his fingers. There is no inherent reason why positional notation must imply counting by tens. An eight-fingered race of men would naturally have counted by eights. The result, in terms of pencil and paper notation, would have been the octo-

NUMBER WORDS FOR COLUMN SIGNIFICANCE	FIVE HUNDRED AND TWELVES	SIXTY-FOURS	EIGHTS	OCTONARY POINT UNITS	EIGHTHS	SIXTY-FOURTHS	FIVE HUNDRED AND TWELFTHS	FOUR THOUSAND NINETY-SIXTHS
DECIMAL EQUIVALENT FOR COLUMN SIGNIFICANCE	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}	8^{-4}
OCTONARY NOTATION FOR COLUMN	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
SAMPLE NUMBER	1×10^3	2×10^2	3×10^1	4×10^0	2×10^{-1}	5×10^{-2}	7×10^{-3}	6×10^{-4}

1 2 3 4 • 2 5 7 6

Figure 2-2

DECIMAL	OCTONARY
0.015625	0.01
0.031250	0.02
0.046875	0.03
0.062500	0.04
0.078125	0.05
0.093750	0.06
0.109375	0.07
0.125000	0.10
0.140625	0.11
0.156250	0.12
0.171875	0.13
0.187500	0.14
0.203125	0.15
0.218750	0.16
0.234375	0.17
0.250000	0.20
0.265625	0.21
0.281250	0.22
0.296875	0.23
0.312500	0.24
0.328125	0.25
0.343750	0.26
0.359375	0.27
0.375000	0.30
0.390625	0.31
0.406250	0.32
0.421875	0.33
0.437500	0.34
0.453125	0.35
0.468750	0.36
0.484375	0.37
0.500000	0.40
0.515625	0.41
0.531250	0.42
0.546875	0.43
0.562500	0.44
0.578125	0.45
0.593750	0.46
0.609375	0.47
0.625000	0.50
0.640625	0.51
0.656250	0.52
0.671875	0.53
0.687500	0.54
0.703125	0.55
0.718750	0.56
0.734375	0.57
0.750000	0.60
0.765625	0.61
0.781250	0.62
0.796875	0.63

Figure 2-3, Sheet 1

DECIMAL	OCTONARY
0.812500	0.64
0.828125	0.65
0.843750	0.66
0.859375	0.67
0.87500	0.70
0.890625	0.71
0.90625	0.72
0.921875	0.73
0.93750	0.74
0.953125	0.75
0.96875	0.76
0.984375	0.77
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20
17	21
18	22
19	23
20	24
21	25
22	26
23	27
24	30
25	31
26	32
27	33
28	34
29	35
30	36
31	37
32	40
33	41
34	42
35	43
36	44
37	45

Figure 2-3, Sheet 2

DECIMAL	OCTONARY
38	46
39	47
40	50
41	51
42	52
43	53
44	54
45	55
46	56
47	57
48	60
49	61
50	62

Figure 2-3, Sheet 3

nary number system which is obtained by substituting the value R = 8 into the general expression for any number, N (equation (2)). Thus:

$$N_{(8)} = C_n 8^n + C_{n-1} 8^{n-1} + \ldots + C_0 8^0 + C_{-1} 8^{-1} + \ldots \qquad (3)$$

Moreover, each of the coefficients; $C_n, C_n-1, \ldots, C_0, C_{-1}, \ldots$ can assume any of the values 0 through (8-1) = 0 through 7. For example, 253 can be interpreted as an octonary number as follows:

$$253_{(8)} = 2 \times 8^2 + 5 \times 8^1 + 3 \times 8^0 = 161_{(10)}$$

Here, the subscript$_{(8)}$ indicates octonary notation while the subscript$_{(10)}$ indicates decimal notation.

A somewhat longer example of a number represented in the octonary system is given in Figure 2-2. A comparison table of decimal and octonary numbers is presented in Figure 2-3.

Aside from its relationship to finger counting, the decimal system has nothing particular to recommend it-except of cource its familiarity. This alone is probably sufficient to guarantee its continued use in hand calculations. However, from the point of view of the design of high speed computers, the decimal system has the very serious disadvantage of requiring the use of too many distinct symbols.

If ten symbols are inconveniently many from the point of view of the designer, then the question which naturally arises is, how many distinct symbols can be represented conveniently in a high speed computing device? The answer to this question

NUMBER WORDS FOR COLUMN SIGNIFICANCE	EIGHTS	FOURS	TWOS	UNITS	HALVES	FOURTHS	EIGHTHS	SIXTEENTHS
				BINARY POINT ↓				
DECIMAL EQUIVALENT FOR COLUMN SIGNIFICANCE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
BINARY NOTATION FOR COLUMN SIGNIFICANCE	10^3	10^2	10^1	10^0	10^{-1}	10^2	10^{-3}	10^{-4}
SAMPLE NUMBER	$1X10^3$	$0X10^2$	$0X10^1$	$1X10^0$	$0X10^{-1}$	$1X10^{-2}$	$1X10^{-3}$	$0X10^{-4}$

1001 ● 0110

Figure 2-4

DECIMAL	BINARY
0.015625	0.000001
0.031250	0.000010
0.046875	0.000011
0.062500	0.000100
0.078125	0.000101
0.093750	0.000110
0.109375	0.000111
0.125000	0.001000
0.140625	0.001001
0.156250	0.001010
0.171875	0.001011
0.187500	0.001100
0.203125	0.001101
0.218750	0.001110
0.234375	0.001111
0.250000	0.010000
0.265625	0.010001
0.281250	0.010010
0.296875	0.010011
0.312500	0.010100
0.328125	0.010101
0.343750	0.010110
0.359375	0.010111
0.375000	0.011000
0.390625	0.011001
0.406250	0.011010
0.421875	0.011011
0.437500	0.011100
0.453125	0.011101
0.468750	0.011110
0.484375	0.011111
0.500000	0.100000
0.515625	0.100001
0.531250	0.100010
0.546875	0.100011
0.562500	0.100100
0.578125	0.100101
0.593750	0.100110
0.609375	0.100111
0.625000	0.101000
0.640625	0.101001
0.656250	0.101010
0.671875	0.101011
0.687500	0.101100
0.703125	0.101101
0.718750	0.101110
0.734375	0.101111
0.750000	0.110000
0.765625	0.110001
0.781250	0.110010
0.796875	0.110011

Figure 2-5, Sheet 1

DECIMAL	BINARY
0.812500	0.110100
0.828125	0.110101
0.843750	0.110110
0.859375	0.110111
0.87500	0.111000
0.890625	0.111001
0.90625	0.111010
0.921875	0.111011
0.93750	0.111100
0.953125	0.111101
0.96875	0.111110
0.984375	0.111111
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101
30	11110
31	11111
32	100000
33	100001
34	100010
35	100011
36	100100
37	100101
38	100110
39	100111

Figure 2-5, Sheet 2

DECIMAL	BINARY
40	101000
41	101001
42	101010
43	101011
44	101100
45	101101
46	101110
47	101111
48	110000
49	110001
50	110010

Figure 2-5, Sheet 3

turns out to be two. This is because so many devices can be made to assume two states. A lamp, for example, may be lighted or extinguished. A switch may be open or closed.

It is the binary system, that is the system with radix two, then, that is used most commonly in the operational portion of digital computers. The form of a binary number is given by substituting the value R= 2 into the general expression for any number, N (equation 2). Thus:

$$N_{(2)} = C_n 2^n + C_{n-1} 2^{n-1} + \ldots + C_0 2^0 (\cdot) + C_{-1} 2^{-1} + \ldots \quad (4)$$

where each of the coefficients; C_n, C_{n-1}, ... can assume any of the values 0 through(2 - 1) = 0 through 1. For example, 110 can be interpreted as a binary number as follows:

$$110_{(2)} = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6_{(10)}$$

Where the subscript $_{(2)}$ indicates binary notation and the subscript $_{(10)}$ indicates decimal notation as before.

A somewhat longer example of a number represented in the binary system is given in Figure 2-4. A comparison table of decimal and binary numbers is presented in Figure 2-5.

Representation of any given number in the binary system requires many more columns or places than are necessary in the decimal system because only the two values 0 and 1 can be displayed in each column. For example decimal 1535 = binary 10000000000.

The interpretation of any unfamiliar representation of numbers is difficult for humans. In the case of binary numbers this difficulty is compounded by the length of the numbers and the repetition of the two symbols 0 and 1. It follows

that humans will continue to operate upon decimal representations of numbers while machines will operate upon binary numbers. But this does not mean that the humans who will be concerned with a computing machine can forget about the binary system. On the contrary, in order to understand the machine, it will be necessary not only to understand binary notation but also to understand the performance of the arithmetic operations upon binary numbers.

Before considering the performance of arithmetic processes in the unfamiliar binary system, it is worthwhile to analyze their performance in the familiar decimal system. For the fact is that the arithmetic operations are performed automatically in accordance with rules learned early in childhood. Moreover, the nature of these rules is such that they tend to obscure the significance behind the operations. For this reason the next chapter is devoted to a discussionof decimal arithmetic.

1.2 SHIFTING

Everyone is familiar with the fact that in the decimal number system multiplication of any number by ten is accomplished by shifting each of the digits of that number one place to the left, while division by ten is accomplished by shifting each of the digits of the number one place to the right. Thus:

125 X 10 = 1250

125 ÷ 10 = 12.5

This follows from the fact that ten is the radix of the decimal system. Thus:

$$125 = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

$$125 \times 10 = 1 \times 10^2 \times 10^1 + 2 \times 10^1 \times 10^1 + 5 \times 10^0 \times 10^1$$

$$= 1 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0$$

where the righthand term, 0×10^0, indicates that 0 must be entered in the 10^0 place when the number is written in terms of coefficients as 1250.

Multiplication or division of a decimal number by ten is a special case, then, because ten is the radix of the decimal number system. This can be generalized as follows: multiplication or division of any number in any number system by the radix of that system can be accomplished by a shift to the left (in the case of multiplication) or a shift to the right (in the case of division). In terms of the general expression for a number (equation (2)):

$$RN = C_n R^n R + C_{n-1} R^{n-1} R + \ldots + C_0 R^0 R (\cdot) + C_{-1} R^{-1} R + \ldots$$

$$= C_n R^{n+1} + C_{n-1} R^n + \ldots C_0 R^1 + C_{-1} R^0 (\cdot) + \ldots$$

and

$$\frac{N}{R} = \frac{C_n R^n}{R} + \frac{C_{n-1} R^{n-1}}{R} + \ldots + \frac{C_0 R^0}{R} (\cdot) + \frac{C_{-1} R^{-1}}{R} + \ldots$$

$$= C_n R^{n-1} + C_{n-1} R^{n-2} + \ldots (\cdot) + C_0 R^{-1} + C_{-1} R^{-2} + \ldots$$

As an example, consider the binary number $11_{(2)} = 3_{(10)}$. A shift left yields $110_{(2)} = 6_{(10)}$. A shift right, on the other hand, yields $1.1_{(2)} = 1.5_{(10)}$. Thus, a shift to the left is a multiplication by two (i.e. by the radix of the system) while a shift right is a division by two.

PART 2

CHAPTER 2

DECIMAL ARITHMETIC

2.1 ADDITION

Counting must be performed upon the units of a group in succession. One unit must be associated with the number 1, another unit with the number 2, still another unit with the number 3 and so on. If the number of elements in two collections A and B are known and if A and B are combined to form a new collection S, then there are two ways of discovering the number of members belonging to S. The first way is to count the members of S. By this method individual numbers are re-ordered to form a new group. The second way is to add the number of members of A to the number of members of B. Addition is thus a method which avoids counting.

Addition is performed in accordance with a table which establishes correspondences between pairs of collections on the one hand and single collections on the other hand. This table is, of course, the familiar addition table which is memorized early in life by most literate people and is, thereafter used automatically. With every possible combination of two numbers from 0 to 9, the addition table associates a number which is their sum. Thus the process of adding two numbers less than or equal to nine consists of referring to information which is stored in the memory of the operator.

Since the addition table associates sums only with pairs of numbers from 0 through 9, the performance of addition requires a second operation; that is, the process of carrying from one column to the next. This operation requires only an understanding of the positional significance of decimal notation. Thus, if 48 is added to 34, reference to memory first produces the sum 12. But 12 is composed of the component 2 in the units (10^0) column and the component 1 which belongs in the tens (10^1) column. The component 1 is accordingly carried to the 10^1 column where it is added to the sum of the 10^1 column components of the original numbers. The addition table only specifies sums for pairs of numbers. But this places no limitation on the process of addition, for any group of numbers can be added by first adding two, then adding a third to the sum of these two, then adding a fourth to the sum of these three and so on. In adding a column of figures using pencil and paper these successive additions are usually carried out on one column at a time. The sum being accumulated, as a column is added, is simply retained in the memory of the operator until the successive additions of all digits in that column have been completed. The component belonging to that column is then entered below the column and other components are carried and added to the respective next left columns where the process is repeated. This places no particular burden upon the human memory when

the group of numbers to be added is not too large, and is, therefore, a very convenient method for pencil and paper addition.

Notice the storage roll played by the sheet of paper. Throughout the operation, a record of the original numbers remains before the operator and as the addition in each column is completed the component belonging to that column is entered below it. It turns out that this ability to store a group of more than two numbers and operate upon them column by column cannot conveniently be duplicated by computing machines. Thus in a machine, addition of two numbers is completed before a third is brought forward to be operated upon.

Addition of any two numbers is possible no matter how large they are. This follows from the fact that there is no largest number. However, when a machine is used to perform addition, a limitation is placed upon the size of the sum, because if the sum exceeds the capacity of the machine a part of it is lost. If a decimal machine provides facilities for representing a number having 5 places, then it can represent

100,000 distinct numbers, as, for example, 0 through 99,999. In this case 100,000 is said to be the modulus of the machine and the machine is said to perform modulo 100,000 arithmetic. The significance of this is that if an addition produces a sum which equals or exceeds the modulus of the machine, then a part of that sum equal to the modulus of the machine is lost.

An example of this is furnished by the odometer which records the mileage traveled by an automobile. Generally these devices perform modulo 99,999.9 arithmetic. Thus if the car travels 100,000.0 miles the odometer resets to 00000.0; that is, an amount equal to the modulus is lost.

In an automatic computing machine it is generally necessary to provide some sort of warning device which is actuated when the modulus of the machine is exceeded.

Addition is what is called a commutative process, that is adding 6 to 5 produces the same result as adding 5 to 6. However, it is convenient to have different names for the two numbers involved in an addition, particularly when discussing a mechanization of the addition process. The number to which a second number is added is the augend. The number which is added to the augend is the addend. The result is the sum.

Before ending the examination of decimal addition it is worthwhile to consider very briefly some particular mechanization of the decimal addition process. The most straightforward scheme involves the use of a counter similar to the automobile odometer mentioned above. Here the digits 0 through 9 are displayed about the periphery of a set of counter wheels which

represent successive powers of ten. The wheels are viewed from a vantage point such that only one digit on each wheel is visible at any one time. Since the ten digits are spaced evenly about the 360 degrees of each wheel, an angular motion of the wheel through 36 degrees is necessary to pass from one digit to another. All the wheels are initially set to read zero. In order to set a number into the machine each wheel is rotated 36xd degrees, where d is the digit which belongs to the order associated with that wheel. To add a second number, the wheels are rotated through further angles determined by the digits of that second number. At this point a carry may occur in any of the orders, that is the sum developed in that order may exceed 9. For this reason facilities for carrying must be provided. However, this problem is easily solved by arranging to have each wheel turn the wheel at its left through 36 degrees as it passes from 9 to 0. Thus as the unit wheel passes from 9 to 0, the digit displayed on the tens wheel is increased by 1.

Such a device is properly called a counter because as the magnitude of any number is increased the various wheels pass through each successive position between the initial value and the final value. This mechanization of the addition operation makes no use of the addition table but depends instead on the more fundamental process of counting. Thus to add a group of numbers, each number in turn is counted into the machine and the machine accumulates the total count or sum.

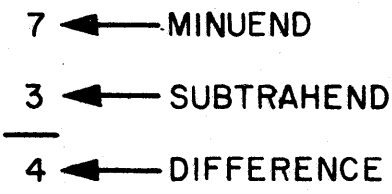


Figure 2-6

2.2 SUBTRACTION

Subtraction is the inverse of addition; that is, it is a method of determining what remains of group A after group B has been removed from it. Such a problem can be solved by counting down, that is counting in the direction of decreasing magnitude. The subtraction process avoids the necessity for counting by substituting, instead, an inverse reading of the addition table. The solution of the problem of subtracting 3 from 7, for example may be restated as follows: What number must be added to 3 in order to form 7? Since the addition table associates the sum 7 with the pair of numbers 3,4, it contains the answer to that question.

Subtraction unlike addition is not a commutative process, that is subtracting 3 from 7 does not yield the same result as subtracting 7 from 3. Therefore, even more than in the case of addition it is convenient to have names for the two numbers involved in a subtraction operation. Accordingly, the number which is subtracted is the subtrahend while the number from which the subtrahend is removed is the minuend. The result is the difference. These relationships are illustrated in Figure 2-6.

Like addition, subtraction involves another process besides reference to memory. In this case, the other process is that of borrowing from the column to the left when a minuend digit is smaller than a subtrahend digit. Since each successive column to the left represents a higher power of ten, a borrow of one from the column to the left increases the value of the minuend digit by ten. For example, in subtracting 47

from 82, a 1 must be borrowed from the tens digit of the minuend in order to perform the subtraction in the units column. Thus the digit in the tens column is reduced by 1; that is, from 8 to 7. At the same time, the units digit is increased by 10 so that it becomes 12. Next, 7 is subtracted from 12, that is reference is made to the addition table to find the answer to the question, "What number must be added to 7 in order to obtain 12?" The partial difference, 5, is entered beneath the units column. The tens digit of the subtrahend is then subtracted from the diminished tens digit of the minuend and the result, 3, is entered beneath the tens column. In performing the borrow operation automatically many people add 1 to the tens digit of the subtrahend rather than subtracting 1 from the tens digit of the minuend. This, of course, is a completely equivalent operation but it tends to obscure the logic on which the process is based.

The concept of negative number allows any number to be subtracted from any other number. Just as there is no largest number, so also there is no least (or most negative) number. This creates the same difficulty with respect to the mechanization of subtraction that it did with respect to the mechanization of addition; that is, a subtraction operation may produce a difference which is a more negative number than a given computing machine is capable of representing.

An even more fundamental problem than this is encountered in mechanizing subtraction. In order to appreciate this problem it is necessary to consider how the human operator approaches

a subtraction problem in which the minuend is smaller than the subtrahend. The rule which covers this case appears simple enough. A human operator, recognizing that the subtrahend is larger than the minuend, merely turns the process around; that is, subtracts the minuend from the subtrahend. The difference is then assigned a minus sign to indicate that it is less then zero. Thus, given 5 as a minuend and 7 as a subtrahend, 5 is subtracted from 7 and a minus sign is attached to the result, yielding -2. The negative number -2 is the answer to the subtraction question, what number must be added to 7 in order to obtain 5? Notice that this implies equivalence between the subtraction of a positive number and the addition of a negative number. Viewed in this light, the operation of subtraction is a special case of addition; that is, it is the addition of a negative number to a positive number.

While the special case of subtraction which arises when the minuend is smaller than the subtrahend offers no particular difficulty for the human operator, it should be recognized that the manner in which he handles it implies the following special operations:

a. Comparison - Since the special case is to be handled by a special routine, every minuend and subtrahend must be compared to discover which is larger, before any routine can be initiated.

b. Reversal of the roles of the minuend and subtrahend and performance of subtraction routine.

c. Tagging of resultant difference with a minus sign.

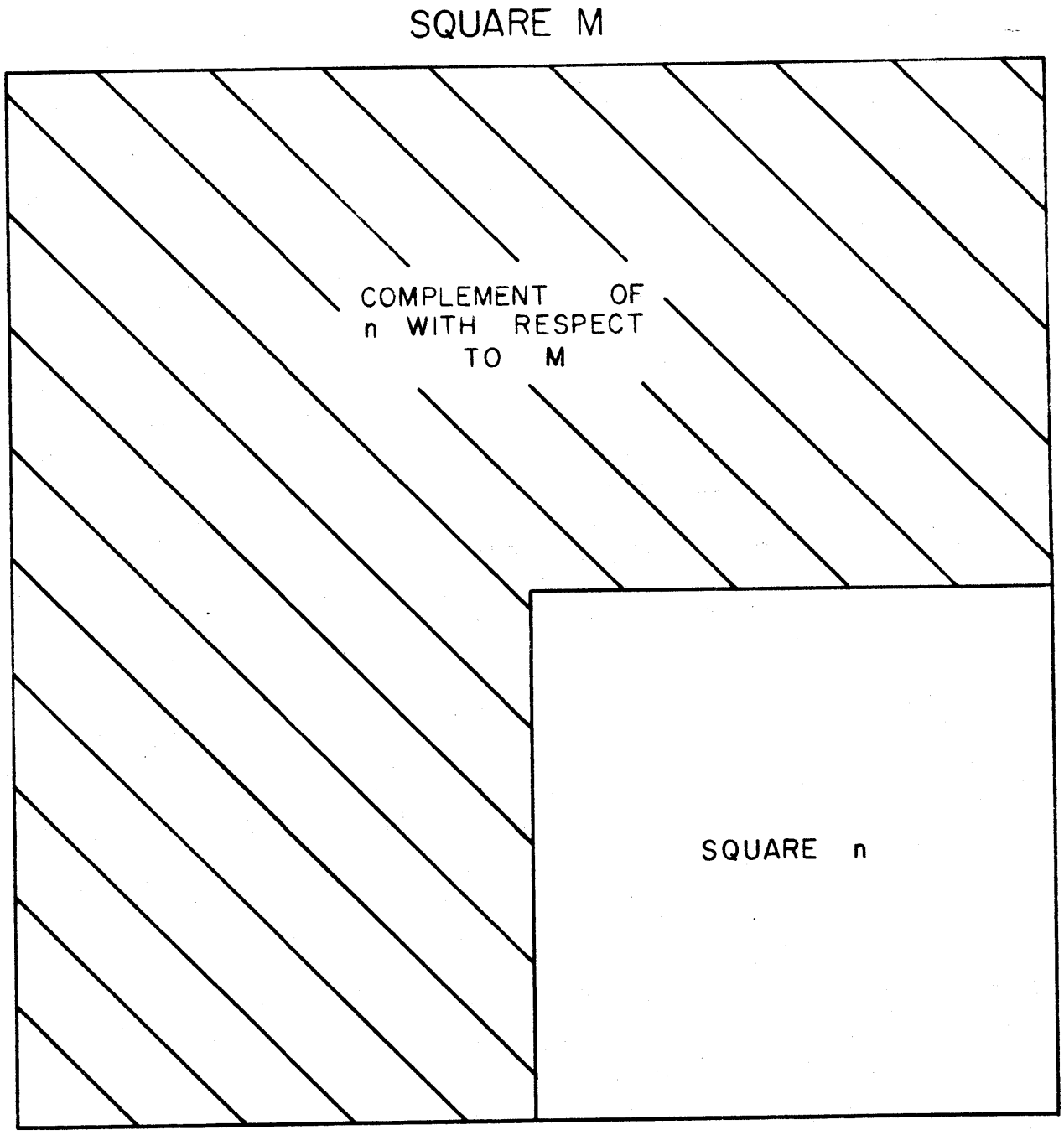


Figure 2-7

Such a routine can only be mechanized by increasing the complexity of the machine which is to handle it or by introducing addition steps which will slow down its performance. It turns out there is a better scheme for mechanizing the subtraction operation. Interestingly enough this scheme converts a limitation of machine into an advantage. It has been mentioned that either the addition or the subtraction operation yields an answer other than that anticipated if the modulus of the machine is exceeded. It is this fact which is made use of in the subtraction scheme used in most automatic computing machines.

Before introducing the subtraction scheme just referred to, it is necessary to define the term, complement. A geometric interpretation of complement is shown in Figure 2-7. Referring to the figure notice that the square n is contained in the square M. The complement of the square n with respect to the square M is defined to be all the area of the square M which remains if n is removed. Thus the area of M is the sum of the areas of n and the complement of n. But the concept of complement is by no means limited to geometry. Consider a computing machine of modulus M and consider any number n which can be represented by that machine. Then the complement of n with respect to M is defined to be all of M that remains after n has been removed. Since M and n are, in this case numbers, the complement of n can be stated in terms of arithmetic to be the difference resulting from the subtraction of n from M.

The scheme for mechanizing subtraction using complements can now be stated as follows: Let any negative number

M = MACHINE MODULUS = 1 0 0 0

P = MINUEND = 4 9

N = SUBTRAHEND = 3 2

P =	4 9	← MINUEND IN TRUE FORM
M − N =	9 6 8	← SUBTRAHEND IN COMPLEMENT FORM
(M − N) + P =	1 0 1 7	
− M =	−1 0 0 0	← MODULUS AUTOMATICALLY SUBTRACTED BY MACHINE
(M − N) + P − M = P − N =	0 1 7	← DIFFERENCE IN TRUE FORM RETAINED BY MACHINE

Figure 2-8

be represented in the machine by the complement with respect to the machine modulus of the corresponding positive number. Let subtraction be performed as a special case of addition. Since the complement of any number n with respect to modulus M is equal to M-n, the subtraction of n from p in accordance with this scheme is performed by adding (M-n) to p. The resultant of such an operation is the sum M + p - n. But as stated earlier when a sum equals or exceeds the modulus of the machine, then an amount equal to the modulus is lost. Thus the part of the sum M + p - n which is retained by the machine is p - n so that the subtraction of n from p has been successfully performed.

An example of subtraction by means of complementing the subtrahend and adding it to the minuend is illustrated in Figure 2-8. Here, the minuend is 49. the subtrahend is 32 and the machine modulus is 1000. Thus, the complement of the subtrahend with respect to the modulus is 1000 - 32 = 968. This complement is added to the true form of the minuend yielding 1017. However, as shown in the illustration, the modulus (in this case 1000) is automatically subtracted by the machine (by virtue of the fact that the machine is not equipped to represent 10^3 order digits, so that the carry which occurs from the 10^2 order is lost. Thus, what is retained by the machine is 017 which is the true form of the difference resulting from the subtraction of 32 from 49.

The principle reason for adopting this special subtraction routine is to enable the machine to handle subtraction in the same way regardless of the relative magnitudes of the minuend and subtrahend. For this reason, it is worthwhile to

```
LET  M =  1 0 0 0
     P =    3 2
     N =    4 9
     P =    3 2  <-- MINUEND IN TRUE FORM
M - N =    9 5 1  <-- SUBTRAHEND IN COMPLEMENT FORM
-------------------
(M - N) + P = 9 8 3  <-- DIFFERENCE IN COMPLEMENT FORM
```

Figure 2-9

consider a second example in which the subtrahend is larger than the minuend. Such an example is afforded by Figure 2-9. Here, the minuend is 32, the subtrahend is 49 and the machine modulus is 1000. Thus, the complement of the subtrahend with respect to the modulus is 951. This complement is added to the true form of the minuend yielding 983. Since 983 is the complement with respect to the machine modulus of 17 (i.e. 983 = 1000 - 17) it can be interpreted as - 17.

Comparing the results obtained in the examples of Figures 2-8 and 2-9 reveals that, when subtraction is performed by complementing the subtrahend and adding it to the minuend, the difference appears in true form when the minuend is larger than the subtrahend, while the difference appears in complement form when the minuend is less than the subtrahend. This corresponds to the fact that the difference in the first case is a positive number and in the second case is a negative number. Thus, in a machine which performs subtraction by this method, negative numbers are represented in complement form. With this cnnvention in mind, the operation of subtraction can be generalized to cover the addition of positive and negative numbers. The examples of Figures 2-8 and 2-9 can then be thought of as cases of addition of a negative number to a positive number (where the negative number is represented in complement form). There is nothing particularly surprising about this, since the addition of - x to + y is equivalent to the subtraction of + x from + y. However, when the special operation is thought of in terms of the addition of signed numbers, then it seems

$$M = 1000$$

$$X = -32$$

$$Y = -49$$

$$M - |X| = 968$$

$$M - |Y| = 951$$

$$1919$$

$$1000 \leftarrow \text{MODULUS AUTOMATICALLY SUBTRACTED BY MACHINE}$$

$$919 \leftarrow \text{SUM IN COMPLEMENT FORM}$$

Figure 2-10

natural to expect it to satisfy the case of the addition of two negative numbers, which in fact it does. An example of such an addition is shown in Figure 2-10. Here, the absolute value bars shown around x and y indicate that the complements are taken with respect to absolute values. (The absolute value of any positive number, $|p| = p$. The absolute value of any negative number, $|n| = -n$.) The sum of any two negative numbers is also a negative number. Thus, the sum obtained in the addition of Figure 2-10 appears in complement form. Specifically, the sum is 919 which is the complement with respect to the machine modulus of 81. Thus, the sum is - 81, which can be checked by performing the addition in the ordinary manner.

The question that naturally arises is, given any number within the machine range, as for example 983, is that number to be interpreted as a complement and hence a representation of a negative number or is it to be interpreted directly as a positive number? This question must be settled in advance of any computation in terms of the modulus of the machine. It has been seen that the modulus defines the number of distinct numbers which can be represented by the machine. Thus, if every number is to have a unique representation, the ability to represent 100 negative numbers can only be obtained by sacrificing 100 positive numbers from the range of the machine.

For example, assume that it is necessary to handle negative numbers in the range of -1 to -100 on the machine of modulus 1000. Then, using the complement representation, numbers 999 through 900 will be interpreted as complements. Thus the positive range of the machine will be from 0 through 899.

An apparent contradiction which arises in connection with complementation is as follows: Complements are introduced in order to avoid explicit performances of the subtraction process. Yet in order to obtain the complement of any number, a subtraction operation is necessary.

There are several answers to this objection. The first answer is that this special case of subtraction is much less objectionable than the general case. As already explained a difficulty in handling subtraction arises from the possibility of encountering a subtrahend which is larger then the minuend. Since any number n which can appear in a computing machine must, by definition of modulus, be smaller than the modulus M, it is impossible to encounter an n larger than M when performing the subtraction M - n which is required to obtain the complement of n. Thus one difficulty is removed. Further, the subtraction routine can be specialized by virtue of the fact that the modulus of a decimal machine is always a power of ten, that is it always takes the form of a 1 followed by a number of 0's. For this reason the subtraction of any number within the range of the machine from the modulus will result in a borrow from each order to the left of the order where the first non-zero digit of the subtrahend appears. For example, in the

subtraction of 320 from 1000, a borrow from the hundred column is necessary since the subtrahend digit 2 is larger than the minuend digit 0. This in turn entails a borrow from the thousands column, since the minuend digit in the hundreds column is also zero. Thus the thousands order digit is reduced to zero and the hundreds order digit is first replaced by 10 as a result of the borrow from the thousands order and then reduced to 9 as a result of the borrow from the tens order.

The routine which has been performed in this example can be re-stated in a form which applies to the formation of a complement with respect to any power of 10 (that is any decimal modulus). This form is as follows: Subtract the first non-zero digit of the subtrahend from ten. Subtract all succeeding digits of the subtrahend from 9. This rule anticipates the borrow from the most significant place of the modulus which reduces the 1 in that place to 0 and produces 9's in all less significant places (by virtue of a borrow from the right) until the place corresponding to the first non-zero digit of the subtrahend is reached. Here there is no borrow from the right and so the full ten borrowed from the column to the left remains. Applying the rule to a particular example, the complement with respect to 100,000 of 2430 is obtained as follows:

$$\begin{array}{r} 9999(10) \\ 00243 \\ \hline 99757 \end{array}$$

The correctness of this result can easily be verified by performing the subtraction according to the usual pencil and

paper methods.

It should be noted that in performing the subtraction indicated by the rule given above, there is no possibility of the need for a borrow occurring (since this need has been anticipated and satisfied in advance.

To summarize, complements with respect to a decimal modulus (often called 10's complements because any decimal modulus is a power of 10) can be formed by means of a subtraction routine which is much more simple than that required to handle the general use of subtraction.

One difficulty still exists in this routine and that is that the first non-zero digit of the number to be complemented is handled differently from the digits to the left. This implies, first, some means of examining the number which locates the first non-zero digit and second, two different subtraction routines, one for that digit and another for the digits to the left.

This difficulty can be avoided by forming complements, not with respect to the modulus M, but with respect to M - 1. Since M for a decimal machine is always composed of a 1 followed by a number of 0's, M - 1 is always a succession of 9's. Thus if M is 10,000, M - 1 is 9,999. Thus complements with respect to M - 1 (called 9's complements) can be formed by subtracting each digit of the number being complemented from 9. For example, the complement with respect to 9999 of 320 is 9679.

The next question to be considered is as follows: Does

M = 100 C = A + B
A = −2 = −2 − 3
B = −3 = −5

M − |A|	=	9	8	←	10'S COMPLEMENT OF |A|
M − |B|	=	9	7	←	10'S COMPLEMENT OF |B|
2M − |A| − |B|	=	1	9	5	
−M	=	−1	0	0	← MACHINE AUTOMATICALLY SUBTRACTS MODULUS
M − |A| − |B|	=		9	5	← 10'S COMPLEMENT OF C

(A) OPERATION UPON 10'S COMPLEMENTS

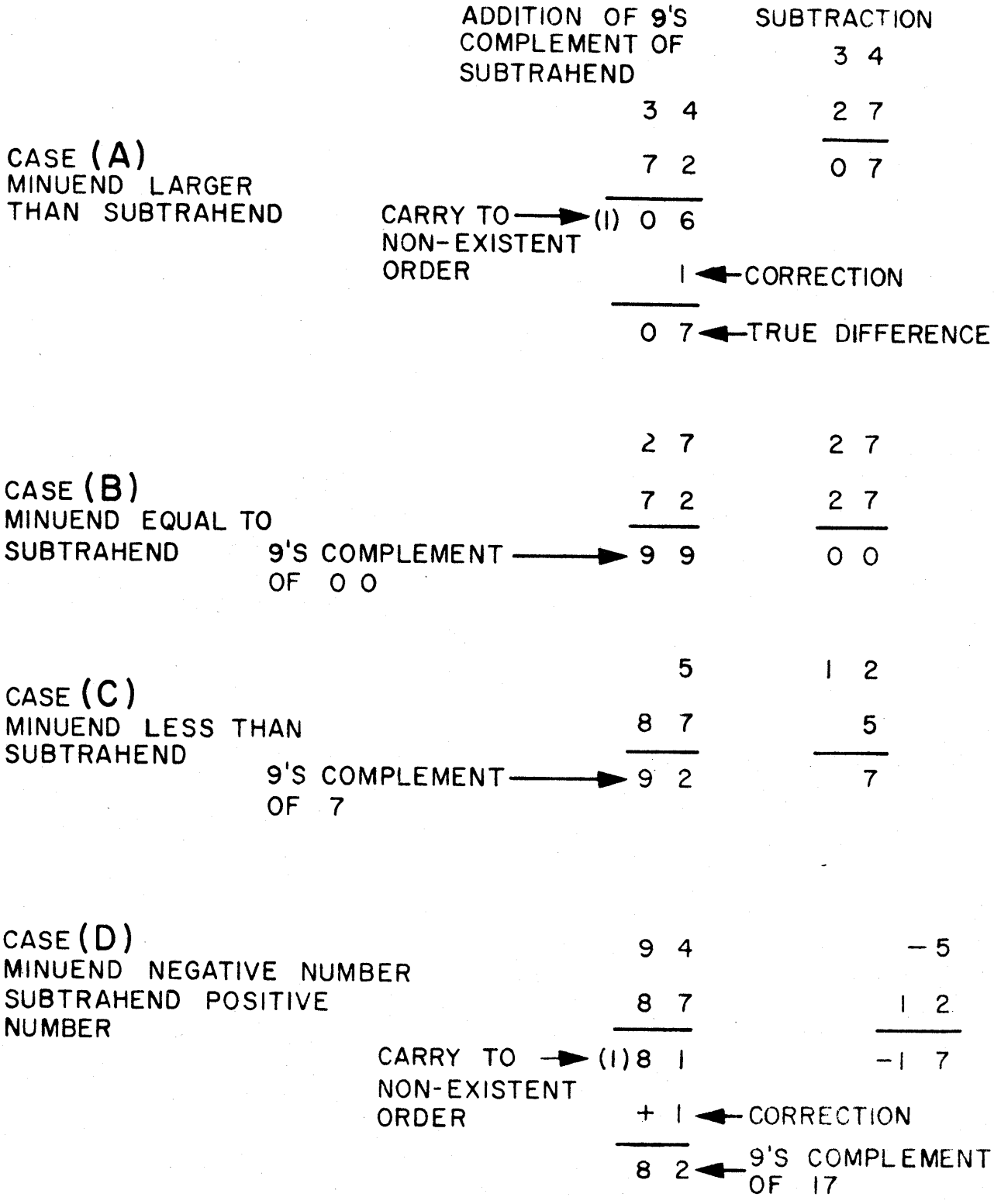
(M − 1) − |A|	=		9	7	← 9'S COMPLEMENT OF |A|
(M − 1) − |B|	=		9	6	← 9'S COMPLEMENT OF |B|
2M − 2 − |A| − |B|	=	1	9	3	
−M	=	−1	0	0	← MACHINE AUTOMATICALLY SUBTRACTS MODULUS
M − 2 − |A| − |B|	=		9	3	
+1	=		+	1	← CORRECTION
M − 1 − |A| − |B|	=		9	4	← 9'S COMPLEMENT OF C

(B) OPERATION UPON 10'S COMPLEMENTS

Figure 2-11

the use of 9's complements instead of 10's complements to represent negative numbers raise any new problems? The answer is that it does, but that they can all be solved in a fairly straightforward manner.

It has been shown that if a + b = c, then the addition of the 10's complement of a to the 10's complement of b yields the 10's complement of c. However, it is not true that the addition of the 9's complement of a to the 9's complement of b yields the 9's complement of c. Operation upon 10's complements is compared with operation upon 9's complements in Figure 2-11. Here, the machine modulus is 100, a is - 2 and b is - 3. Both 10's complements and 9's complements are taken with respect to absolute values. Each of the addition operations is worked out both as a problem in algebra and as a problem in arithmetic. Referring to (a) of the figure, which shows the 10's complement operation, notice that the algebraic result is M - a - b which is, by definition, the 10's complement of + a + b. This checks with the numerical answer, 95, which is the 10's complement of 5. On the other hand, referring to (b) of the Figure, notice that operation upon 9's complements does not yield the 9's complement of the sum . A look at the algebraic result obtained reveals that - 2 appears where, by definition of 9's complement, there should be - 1. For this reason, it is necessary to add a corrective 1 after the addition operation. This checks with the numerical result which is, after the correction has been added, 94 (i.e. the 9's complement of 5).



NOTE: MACHINE MODULUS = 100

Figure 2-12

Consider the effect of using 9's complements on the scheme of subtracting by complementing the subtrahend and adding. Here, there are essentially four cases to be studied as shown in Figure 2-12. The results obtained can be summarized as follows:

a. If both minuend and subtrahend are positive and the minuend is larger than the subtrahend, then the result obtained is 1 less than the difference in true form.

b. If both minuend and subtrahend are positive and are equal, then the result obtained is the 9's complement of zero.

c. If minuend and subtrahend are both positive and the minuend is less than the subtrahend, then the result obtained is the difference in 9's complement form.

d. If the minuend is negative and the subtrahend is positive, then the result obtained is 1 less than the difference in 9's complement form.

Notice that in cases (a) and (d) where a correction is required, a carry occurs to the non-existent order, while in case (c) where no correction is required no carry occurs. In case (b), 0 occurs in 9's complement form and no carry occurs. In general, if it is acceptable to have 0 represented in 9's complement form, then when a carry occurs a correction is required and when no carry occurs no correction is required. This suggests the method by which the correction is usually made in a machine which operates upon 9's complements. The carry to the non-existent order is applied to the units order of the machine. This so-called end-around carry automatically introduces a corrective 1 when it is required.

2.3 DECIMAL MULTIPLICATION

Multiplication can be defined as the process of repeating a quantity a specified number of times. Thus the notation 25 x 4 indicates that 25 is to be repeated 4 times. This is equivalent to the special repetitive addition operation: 25 + 25 + 25 + 25.

Like addition, multiplication is a commutative operation. Thus 4 x 25, which indicates that 4 is to be repeated 25 times, will produce the same result as 25 x 4. The number whose repetition is specified is called the multiplicand. The other number which specifies how many times the multiplicand is to be repeated is called the multiplier. The result of a multiplication is called the product.

The pencil and paper methods of multiplication is based upon memorization of tables. Just as in the case of addition, the multiplication table which is commonly committed to memory lists products for all possible pairs that can be formed using the numbers 0 through 9.

The pencil and paper multiplication routine includes reference to the memorized tables, carry from column to column, shift of partial products and summation of partial products.

Before considering this particular multiplication, a general comment should be made about multiplication by 10. As has previously been explained, decimal positional notation associates each column or place with a power of 10. Thus the number 271 may also be written $2 \times 10^2 + 7 \times 10^1 + 1 \times 10^0$. A consequence of this is that multiplication of any decimal number by ten simply has the effect of shifting each digit of the

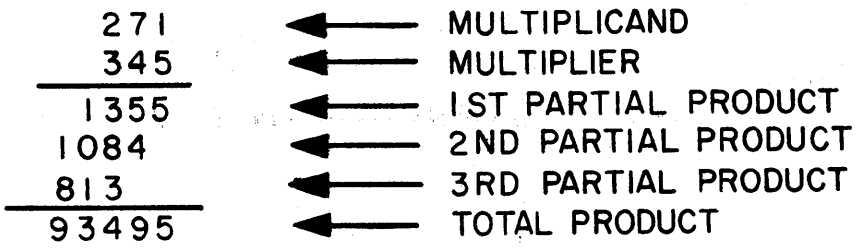

271
345
1355
1084
813
93495
MULTIPLICAND
MULTIPLIER
1ST PARTIAL PRODUCT
2ND PARTIAL PRODUCT
3RD PARTIAL PRODUCT
TOTAL PRODUCT

Figure 2-13

number 1 column to the left. For example 271 x 10 = 2710 = $2 \times 10^3 + 7 \times 10^2 + 1 \times 10^1 + 010^0$.

Referring now to Figure 2-13, the first partial product is produced as the digits of the multiplicand are successively operated upon by the first digit of the multiplier. The principle of carry is employed just as in the case of addition when the result of a digit multiplication exceeds 9. Thus, for example, in forming the first partial product, the multiplication of 1 x 5 produces no carry while the succeeding multiplication of 7 x 5 produces the carry 3 which is added to the product of 2 x 5.

The second partial product is formed in the same manner, as the digits of the multiplicand are successively operated upon by the second digit of the multiplier. This second partial product is shifted one column to the left to correspond to the fact that the second digit of the multiplier belongs to the 10^1 column so that it has the significance of 40. Thus the second partial product is really 271 x 40 = 10,840. It is customary not to show the final 0 but since the digits are each shifted to the left this makes no difference in the final result. No new principle is involved in the generation or notation of the third partial product. The summation of the partial products follows the rules of addition. Multiplication by this method is certainly not a difficult process, but as in the case of arithmetic operations studied earlier it turns out to be inconvenient for use in computing machines since it would require the multiplication table to be built into the machine. Since multiplication is essentially a special case of

addition, it can be handled by various addition routines. This will of course involve more steps, for the whole purpose of using the multiplication table is to avoid the performance of the successive additions which are indicated by the multiplication sign. But where humans can save time by performing a more complex routine and thus avoiding additional steps, a computing machine can generally perform a more simple routine faster even if it involves more steps.

The most obvious routine for mechanizing the multiplication operation, then, is simply to perform all the addition operations indicated. For example, the multiplication of 271 by 345 is performed simply by adding 271 to 0, then adding 271 to this sum or 271, then adding 271 to this sum or 542, then adding 271 to this sum or 813 and so on until 271 has entered into the addition process 345 times as indicated by the value of the multiplier. This process is known as over and over addition and is used in some calculating machinery.

Another multiplication scheme, often referred to as the add and shift method is a compromise between the pencil and paper method of multiplication and the over and over addition method. The multiplication of 271 by 345 proceeds in the following manner when the scheme is used: First 271 is entered into the addition process 5 times (in the manner of the over and over addition method) in accordance with the value of the right-hand multiplier digit. Next, 271 is shifted one column to the left corresponding to a multiplication by ten. The resulting 2710 is entered into the addition process 4 times, as specified

PRODUCT ACCUMULATION	MULTIPLIER DIGITS
000	
271	344
271	
271	343
542	
271	342
813	
271	341
1084	
271	340
1355	
2710	330
4065	
2710	320
6775	
2710	310
9485	
2710	300
12195	
27100	200
39295	
27100	100
66395	
27100	000
93495	

Figure 2-14

by the tens digit of the multiplier. In effect this substitutes the multiplication of 4 x 2710 for that of 40 x 271. However, each of these expressions is equivalent to $4 \times 10^1 \times 271$ so that the result is the same. When this second set of additions has been completed the sum which has been accumulated is the sum of the first two partial products. The multiplication is completed by shifting 2710 to the left to obtain 27100 and then entering this number into the addition process 3 times as specified by the hundreds digit of the multiplier. This complete routine is shown in Figure 2-14. The additions and shifts are shown in the column labeled product accumulation. The correspondence between the values of the multiplier digits and the number of additions is revealed by comparing this column with the column labeled "Multiplier Digits". Incidentally, in an actual machine routine, the multiplier digits are successively decreased as each addition is performed and successive additions are performed until each digit in turn assumes the value 0. When this method of keeping track of the process is used, the first shift left is actuated when the right-hand multiplier digit has been decreased to 0, the second shift when the second digit has been reduced to 0 and so on.

It has been seen that the representation of negative numbers in complement form is useful for purposes of subtraction. A natural question then is as follows: How will this representation of negative numbers affect the operation of multiplication?

Since multiplication is a special case of addition it seems reasonable that correct results should be obtained when numbers in complement form are entered into the multiplication

	TRUE FORM	COMPLEMENT FORM
①	(-5) X(+6) -30	95 X6 (5) 70
②	(-5) X(-6) +30	95 X94 (3)80 (85)5 (89) 30

NOTE: ASSUME
MACHINE MODULUS-100

Figure 2-15

process, and this turns out to be so. Two such multiplications are shown in Figure 2-15. In the first of these a negative number is multiplied by a positive number producing a negative result. Notice that when the negative number is represented in complement form, the negative product also appears in complement form. In the second of the multiplications shown in the figure, both multiplicand and multiplier are negative and the result appears in true form which is as it should be, since the multiplication of two negative numbers produces a positive product. The digits in parenthesis do not appear in the machine since, by definition of modulus, the machine is assumed to have no facilities for representing these orders.

The routine involved in multiplying when numbers are represented by 9's complements is not so straightforward. Here a correction must be added to the results obtained as in the case of subtraction by adding the 9's complement of the subtrahend. Suffice it to say here that the necessary corrections can be developed without too much difficulty if other considerations justify the use of 9's complements.

Representation of numbers in complement form has one distinct disadvantage which is apparent from a glance at the problem of Figure 2-15. It results in longer multiplication routines. This follows from the fact that the block of numbers which are interpreted as complements are always larger than the block which are interpreted as true values. For this reason, it is advantageous to transform negative numbers into true form before they are entered into the multiplication

operation. This implies that the sign of the product must be determined on the basis of a comparison of the signs of the multiplier and multiplicand prior to the multiplication operation. It also implies that if the sign or the product is found to be negative, the result obtained in the multiplication operation must be complemented, in order to conform to the convention that negative numbers are represented in complement form.

Multiplication of any two numbers is possible. As in the case of addition and subtraction, this implies that the multiplication operation may produce a result which exceeds the capacity of the machine. It is therefore necessary to provide some sort of device for sensing such an overflow.

2.4 DECIMAL DIVISION

Division is the inverse of multiplication, just as subtraction is the inverse of addition. Thus, dividing 12 by 3 answers the following question: if 3 is the multiplicand and 12 is the product, then what is the multiplier? Since 3 x 4 = 12, 4 is the multiplier. In the terminology of division, 3 is the **divisor**, 12 is the dividend and 4 is the quotient.

Since multiplication is that special case of addition in which one number (the multiplicand) is entered into the addition operation a number of times specified by a second number (the multiplier), the division question just asked can be rephrased as follows: how many times must 3 be entered into the addition operation to obtain 12? This is the kind of question which must be answered by a subtraction operation. Thus, subtraction is fundamental in any division routine.

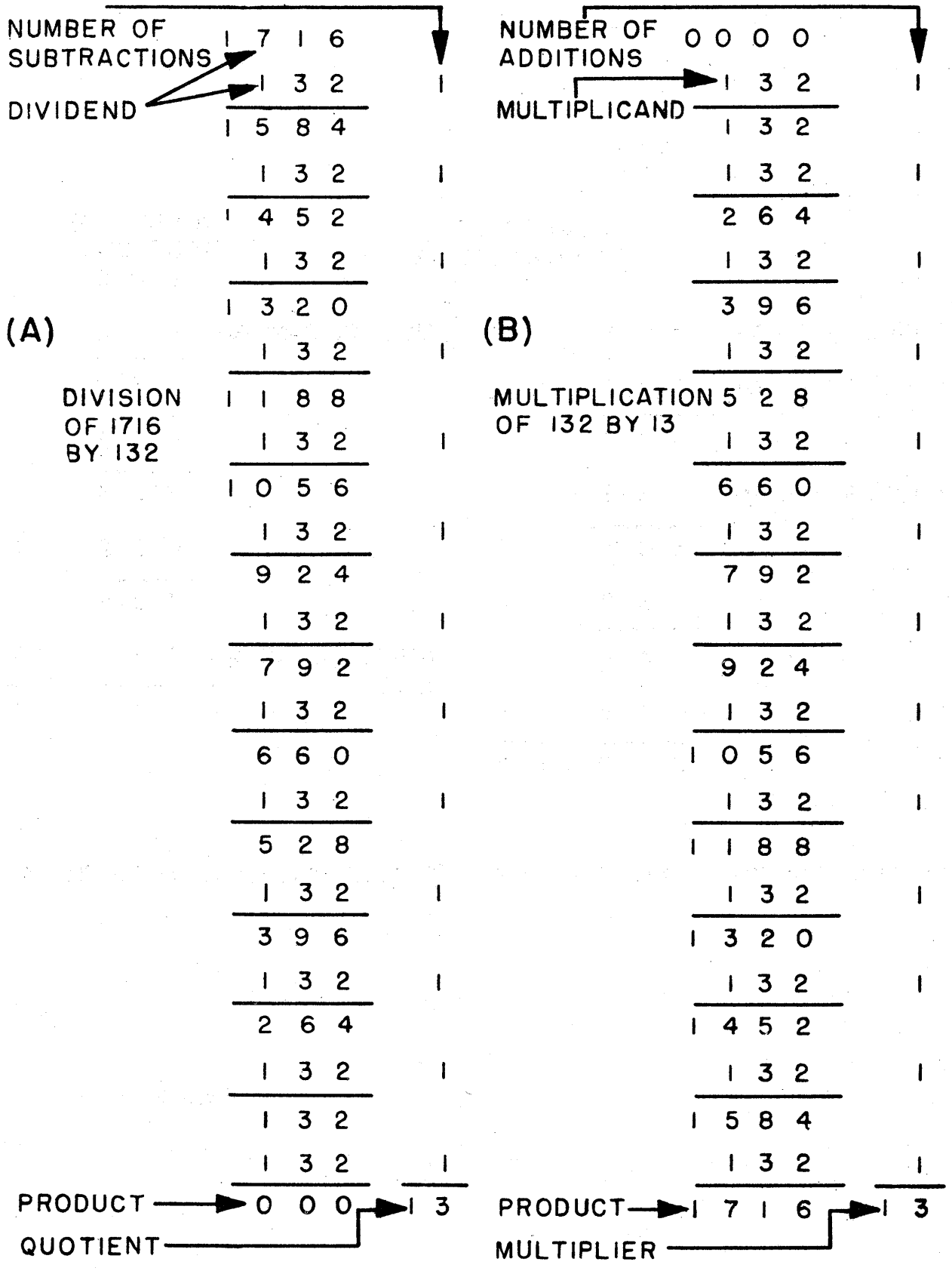

NUMBER OF SUBTRACTIONS
DIVIDEND
(A)
DIVISION OF 1716 BY 132
PRODUCT
QUOTIENT
NUMBER OF ADDITIONS
MULTIPLICAND
(B)
MULTIPLICATION OF 132 BY 13
PRODUCT
MULTIPLIER

Figure 2-16

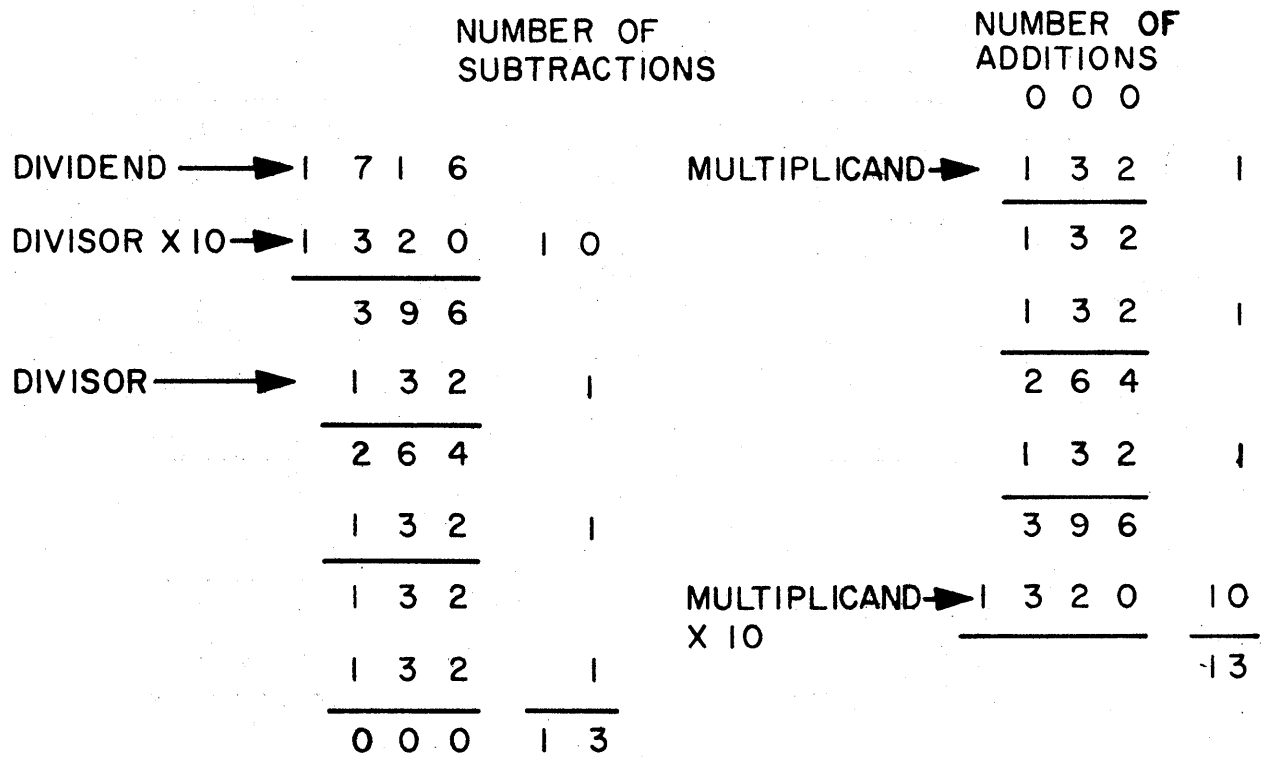

NUMBER OF SUBTRACTIONS
DIVIDEND → 1 7 1 6
DIVISOR X 10 → 1 3 2 0 1 0
3 9 6
DIVISOR → 1 3 2 1
2 6 4
1 3 2 1
1 3 2
1 3 2 1
0 0 0 1 3
(A) DIVISION OF 1716 BY 132
NUMBER OF ADDITIONS
0 0 0
MULTIPLICAND → 1 3 2 1
1 3 2
1 3 2 1
2 6 4
1 3 2 1
3 9 6
MULTIPLICAND X 10 → 1 3 2 0 10
13
(B) MULTIPLICATION OF 132 BY 13

Figure 2-17

The most straightforward method of performing division is by means of over-and-over subtraction. A sample division performed by this method is shown in Figure 2-16 - (a). Here the dividend is 1716 and the devisor is 132. The first step of the routine is to subtract the devisor from the dividend; the second step is to subtract the devisor from the difference obtained in the first step; the third step is to subtract the divisor from the difference obtained in the second step. The process is continued until the dividend has been diminished to a difference (or remainder) which is less than the divisor. The number of subtractions performed up to this point corresponds to the integral portion of quotient. Since, in the example of Figure 2-16-(a), the dividend is an integral multiple of the divisor, a remainder of 000 is obtained as shown. The multiplication of the divisor by the quotient (by over-and-over addition) is shown in (b) of Figure 2-16.in order to clarify the relationship between multiplication and division.

It has been shown, that for the case of multiplication, an add-and-shift routine generally requires fewer steps than an over-and-over addition routine. In the same way, it can be shown that for the case of division, a subtract-and-shift routine requires fewer steps than an over-and-over subtraction routine. This is illustrated in Figure 2-17-(a). Here the dividend and the divisor are the same as in Figure 2-16(a). However, the routine is started with the divisor lined up left with the dividend so that, in the first step, ten times the divisor is subtracted from the dividend as indicated. Since

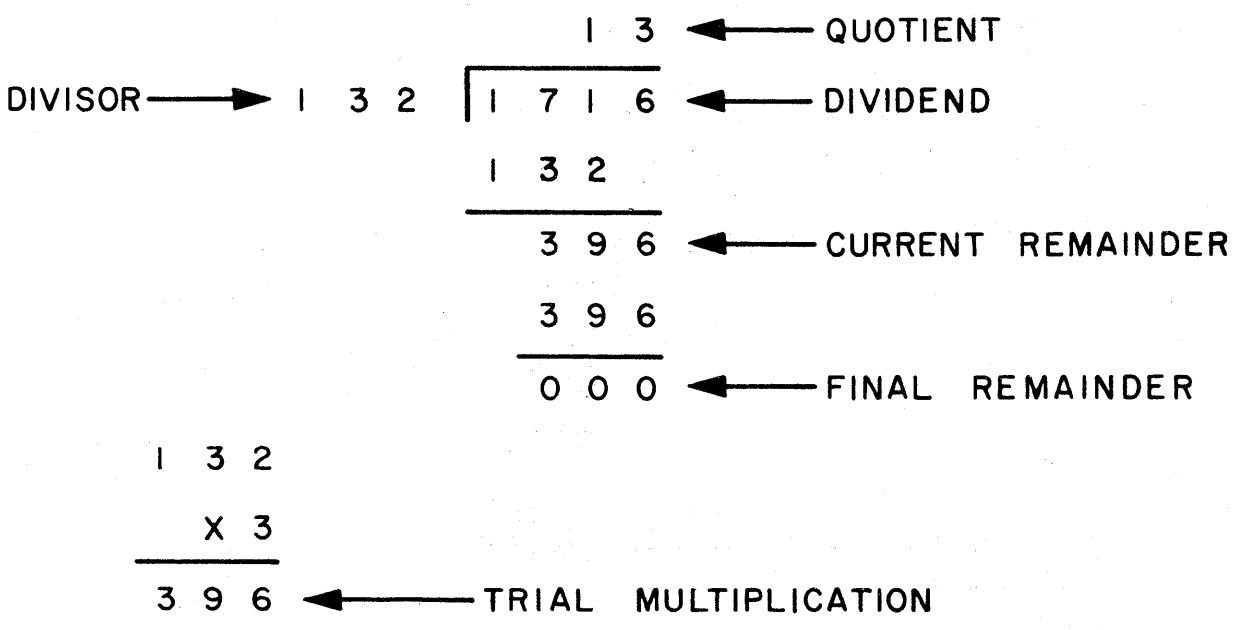

1 3
QUOTIENT
DIVISOR
1 3 2
1 7 1 6
DIVIDEND
1 3 2
3 9 6
CURRENT REMAINDER
3 9 6
0 0 0
FINAL REMAINDER
1 3 2
X 3
3 9 6
TRIAL MULTIPLICATION

Figure 2-18

the remainder obtained as a result of this step is less than ten times the divisor, the divisor is now shifted to the right one place. Thus, the next three subtractions are subtractions of the divisor from the diminishing remainder. In this example, nine steps have been saved by lining up the divisor to the left in the first step. In a case where the dividend is more than one hundred times as large as the divisor, the saving is on the order of one hundred steps. The multiplication of the divisor by the quotient (by means of an add-and shift routine) is shown in (b) of Figure 2-17 in order to clarify the relationship between multiplication and division.

Division by means of the subtract-and-shift routine of Figure 2-17-(a) is very similar to the pencil and paper method of long division. The latter method, however, employs one additional device for minimizing the number of subtraction steps required. This is the device of trial multiplication which is illustrated in the pencil and paper routine of Figure 2-18. In this example, as in the two preceding ones, the dividend is 1716 and the divisor is 132. The human operator begins the routine by comparing the size of the dividend and divisor. The divisor is seen to be about one-tenth the size of the dividend (i.e. it is seen that the divisor can be lined up under the three left-hand digits of the dividend and subtracted from it in this position without producing a negative remainder). The subtraction is performed and a 1 entered in the 10^1 order of the quotient (indicating that ten times the divisor has been subtracted from the dividend). The current remainder

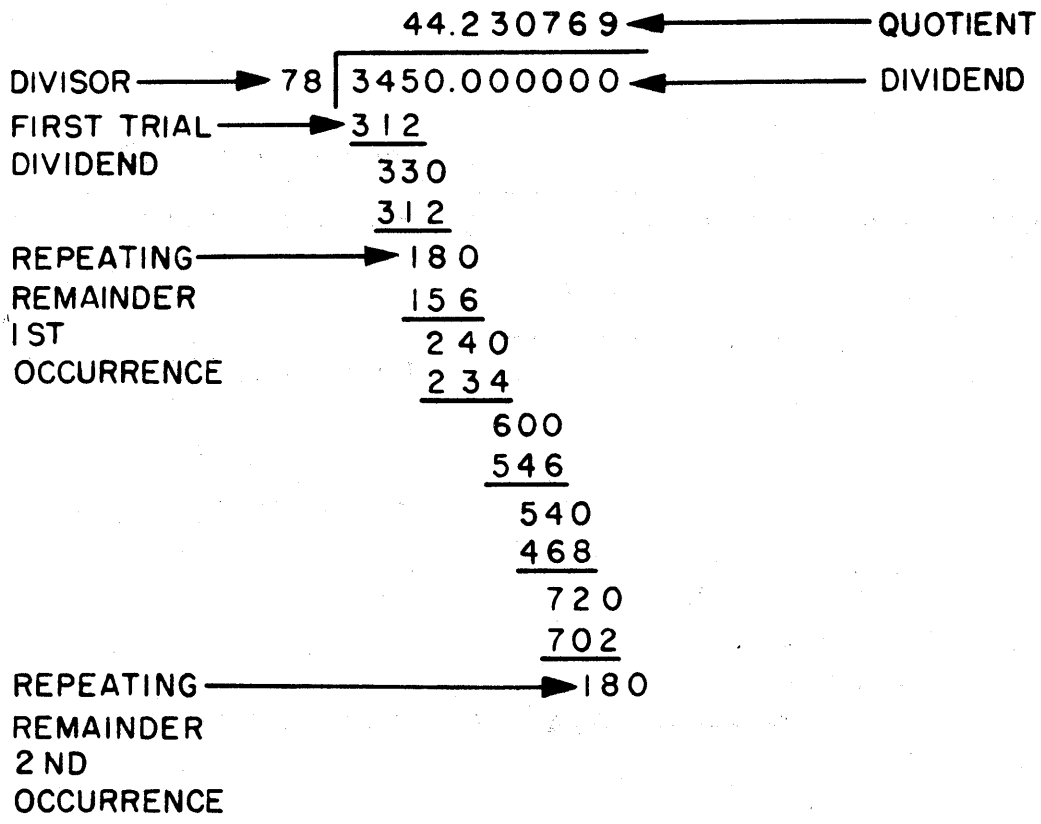

44.230769
QUOTIENT
DIVISOR
78
3450.000000
DIVIDEND
FIRST TRIAL DIVIDEND
312
330
312
REPEATING REMAINDER 1ST OCCURRENCE
180
156
240
234
600
546
540
468
720
702
REPEATING REMAINDER 2ND OCCURRENCE
180

Figure 2-19

(i.e. 396) is now inspected and found to be smaller than ten times the divisor. Accordingly, the divisor must be shifted right before the next subtraction. At the same time it can be seen that some integral multiple of the divisor larger than 1 X the divisor can be subtracted from the current remainder without producing a negative result. Thus, the next step is to find by trial multiplication the largest such multiple. In the case of the example under consideration this is 3 x 132 = 396. Accordingly, this is subtracted from the current remainder and a 3 is entered in the 10^0 order of the quotient, to indicate that 3 x 132 has been subtracted by this step. Notice that the last three subtraction steps of the subtract and shift routine have been replaced by a trial multiplication operation and a single subtraction operation.

The division which has been considered thus far has been the very simple case of a dividend which is an integral multiple of the quotient. In general, however, there is no reason to expect such a simple solution. It is more apt to happen that an infinite number of digits are required to represent a quotient of two numbers. In this case it is obvious that the division process must stop short of a complete solution. This is illustrated by the long division example of Figure 2-19. Referring to the Figure, notice that after the division has been carried out to eight decimal places the remainder repeats. Thus, if the division is carried on, the entire sequence which is shown between the first and second occurrences of the repeating remainder will be repeated,

and this process can be continued indefinitely. Although it cannot be written down because it contains an infinite number of digits, the complete solution is known. It is a number in which the sequence 230769 is repeated indefinitely. A machine with facilities for handling a quotient of eight decimal places would have been required to discover the repeating sequence in the case of this particular division. Moreover, the number of places required depends upon the length of the repeating sequence and this varies from problem to problem. For example the division indicated by 345 + 87 must be carried out to 27 decimal places before a remainder repeats. Thus in a machine division operation, the capacity of the machine is usually exhausted before the character of the quotient is completely known. However, the significance of all the unknown digits can only change the value of the result by an amount equal to a change of 1 in the value of the least significant known digit. Thus the correct solution can be approximated to a known degree of precision (See Chapter 7).

Division is the most difficult of the four arithmetic operations to mechanize. The most troublesome feature of division from the point of view of mechanization is the comparison of the relative magnitudes of dividend and divisor, on the basis of which they are lined up for the initial subtraction operation. So difficult is this problem, that permissible machine division is usually limited to some range of relative magnitudes such that the initial line-up of dividend and divisor is always the same. Even with this limitation

ROUTINE # 1

```
                 0 4 4 .
          78 | 3 4 5 0 . 0
1   SUB      7 8 0 0
       -   4 3 5 0
    ADD      7 8 0 0
             3 4 5 0
1   SUB        7 8 0
             2 6 7 0
2   SUB        7 8 0
             1 8 9 0
3   SUB        7 8 0
             1 1 1 0
4   SUB        7 8 0
               3 3 0
    SUB        7 8 0
       -       4 5 0
    ADD        7 8 0
               3 3 0
1   SUB          7 8
             2 5 2
2   SUB          7 8
             1 7 4
3   SUB          7 8
               9 6
4   SUB          7 8
               1 8
    SUB          7 8
       -         6 0
    ADD          7 8
               1 8 0
1   SUB            7 8
```

ROUTINE # 2

```
                 0 4 4 .
          78 | 3 4 5 0 . 0
(1) SUB    7 8 0 0
       -   4 3 5 0
1   ADD      7 8 0
       -   3 5 7 0
2   ADD      7 8 0
       -   2 7 9 0
3   ADD      7 8 0
       -   2 0 1 0
4   ADD      7 8 0
       -   1 2 3 0
5   ADD      7 8 0
       -     4 5 0
6   ADD      7 8 0
             3 3 0
1   SUB        7 8
             2 5 2
2   SUB        7 8
             1 7 4
3   SUB        7 8
               9 6
4   SUB        7 8
               1 8
(5) SUB        7 8
       -       6 0 0
    ADD          7 8
```

Figure 2-20

placed on division, the pencil and paper routine for division cannot readily be mechanized. Instead, machine division is usually based upon a subtract-and-shift routine. The problem of sensing when a shift right is necessary; that is, when another subtraction carried out in the current position will generate a negative remainder, is most easily handled on a machine by simply continuing to subtract successively until a negative remainder is obtained. At this point the subtraction which generated the negative remainder can be cancelled by performing an addition. Such a subtract-and-shift routine is illustrated in Figure 2-20 (Routine #1). Here, it is presumed that the range of permissible division is such that the operation can start with the divisor lined up left with the dividend. However, subtraction in this position yields a negative remainder. Since this indicates that 100 x 78 is larger than the dividend, a 0 is entered in the 10^2 order of the quotient. At the same time the subtraction is cancelled by addition of the divisor in the same position. The divisor is now shifted one place to the right and is subtracted 5 successive times in this position. However, since the 5th subtraction generates a negative remainder, it is cancelled by an addition. At the same time, a 4 is entered in the 10^1 position of the quotient to indicate that 4 successful subtractions of 78 x 10^1 from the dividend have been completed. The divisor is now shifted to the right a second time and is again subtracted successively until a negative remainder is obtained. Again, the final subtraction is cancelled by an

addition. Again the count of the uncancelled or successful subtractions (which happens to be 4) is entered in the quotient, this time in the 10^0 order to indicate that 4 successful subtractions of 78 x 10^0 have been completed.

Routine #1 of Figure 2-20 is called a restoring division routine because, each time that a negative remainder is obtained, an addition is performed and the positive balance is restored before a shift to the right is made. Routine #2, on the other hand, is a non-restoring routine (i.e. each time that a negative remainder is obtained a shift right is performed without any restoring addition). Referring to the figure, Routine #2 begins with the subtraction of 10^2 x 78 from 3450 just as does the restoring routine. However, when this subtraction is seen to be unsuccessful (that is when the negative sign of the remainder is sensed) a shift right is immediately executed. Successive additions of 10^1 x 78 are performed until the remainder again becomes positive. In the example, six such additions are required. At this point 100 x 78 has been removed from the dividend (leaving a negative remainder) and 6 x 10 x 78 has been replaced. Thus the net quantity removed is 4 x 10 x 78. Accordingly, a 4 is entered in the 10^1 order of the quotient. Notice that this result (i.e. 4) is the Complement with respect to 10 of the count of additions performed (i.e. 6). Thus, a means is available for generating the quotient bit on the basis of the number of additions required to generate a positive balance. When the positive balance is obtained, another shift right is executed and

successive subtractions are performed until the remainder again becomes negative. The successful subtractions are counted; the final subtraction which produces the negative remainder is not. The count of successful subtractions is entered directly in the appropriate order of the quotient just as in the case of Routine #1. The process of shifting and subtracting, then shifting and adding, then shifting and subtracting continues until the division has been carried out to the desired number of places.

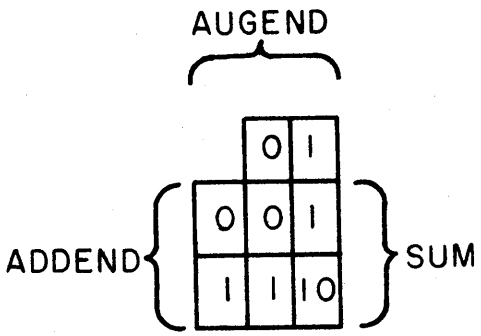


Figure 2-21

PART 2

CHAPTER 3

BINARY ARITHMETIC

3.1 BINARY ADDITION

Binary numbers can be added by a method of involving reference to a memorized addition table and carry from column to column which is similar to the normal method of adding decimal numbers.

Because the binary system employs only the two bits 1 and 0, the binary addition table (shown in Figure 2-21) is trivially simple. It can, in fact, be summarized as follows:

0 + 0 = 0

1 + 0 = 1

0 + 1 = 1

1 + 1 = 10

While the brevity of the addition table may certainly be regarded as an advantage, the use of only two symbols to represent numbers has another consequence which is not so fortunate from the point of view of the human operator. This is that the representation of any given whole number, with the exception of 1, requires more places in the binary system than in the decimal system. Thus the human operator in adding two binary numbers must operate upon more individual symbols than would be necessary if he were performing the same operation upon the equivalent decimal numbers.

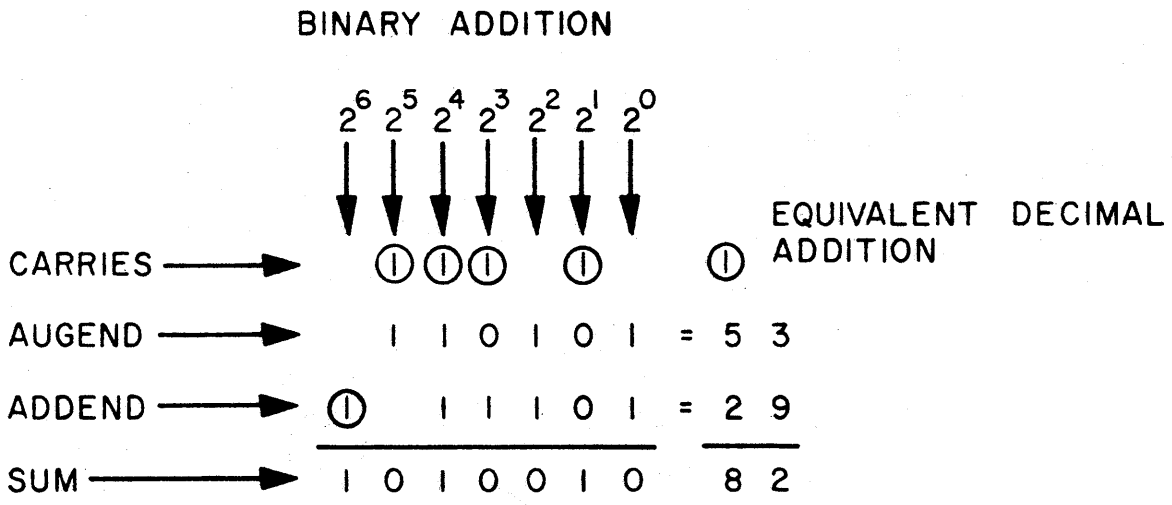

BINARY ADDITION
2^6 2^5 2^4 2^3 2^2 2^1 2^0
EQUIVALENT DECIMAL ADDITION
CARRIES
AUGEND
ADDEND
SUM
1 1 0 1 0 1 = 5 3
1 1 1 1 0 1 = 2 9
1 0 1 0 0 1 0 8 2

Figure 2-22

The addition of two binary numbers is illustrated in Figure 2-22. For purposes of comparison, the addition of the equivalent decimal numbers is also shown. The binary addition proceeds as follows: First the 2^0 order is considered. Here, both the augend and addend bits are 1. By the binary table, Figure 2-21, 1 + 1 = 10. Thus, a 0 is entered in the 2^0 order and a carry is entered in the 2^1 order. Next, the 2^1 order is considered. Here, both addend and augend bits are 0. By the binary addition table, 0 + 0 = 0. However, there is the carry from the 2^0 order to be added to the sum of the augend and addend. By the binary addition table, 0 + 1 = 1. Thus, a 1 is entered in the 2^1 order of the sum. Next, the 2^2 order is considered. Here, as in the first order, both the augend and addend bits are 1. Thus, a 0 is entered in the 2^2 order of the sum and a carry is entered in the 2^3 order. Next, the 2^3 order is considered. First, the addend is added to the augend, yielding 1; then this sum is added to the carry, yielding 10. Thus, a 0 is entered in the 2^3 order of the sum and a carry is entered in the 2^4 order. Next, the 2^4 order is considered. The addition of augend to addend yields 10. The addition of the carry to this sum yields 11. Thus, a 1 is entered in the 2^4 order of the sum and a carry is entered in the 2^5 order. The addition in the 2^5 order yields 10. Thus, a 0 is entered in the 2^5 order of the sum and a carry is entered in the 2^6 order. Since there are no augend or addend bits in the 2^6 order, the carry is brought down into the 2^6 order of the sum.

While the length of binary numbers together with their unfamiliar aspect makes binary addition difficult for the human operator, the advantages of the binary system far outweigh its disadvantages when machine computations are considered.

The machine, which can be designed to perform essentially repetitive operations very rapidly, is not slowed down much by the fact that binary addition requires operation upon more individual symbols. On the other hand, the brevity of the addition table which offers, from the point of view of the human operator, only the temporary advantage that its memorization is easier, offers, from the point of view of mechanization, striking advantages.

Assume that the addition of two numbers is to be performed by a set of mechanized adders each one of which is to be associated with the addition in a particular column or order. Then each adder must be capable of generating the sum of any pair of symbols which can be formed from the set of symbols used in the number system in which the addition is being performed. Thus, a decimal adder must be capable of accepting as inputs any of the one-hundred pairs that can be formed from the ten digits 0,1,2,3,4,5,6,7,8,9; and must be capable of generating each of the nineteen distinct sums that can result from the addition of these pairs. In sharp contrast, a binary adder must accept only the four possible combinations of the two binary digits 0 and 1 (i.e. 0,0;0,1; 1,0; or 1,1) and must be capable of generating only the three

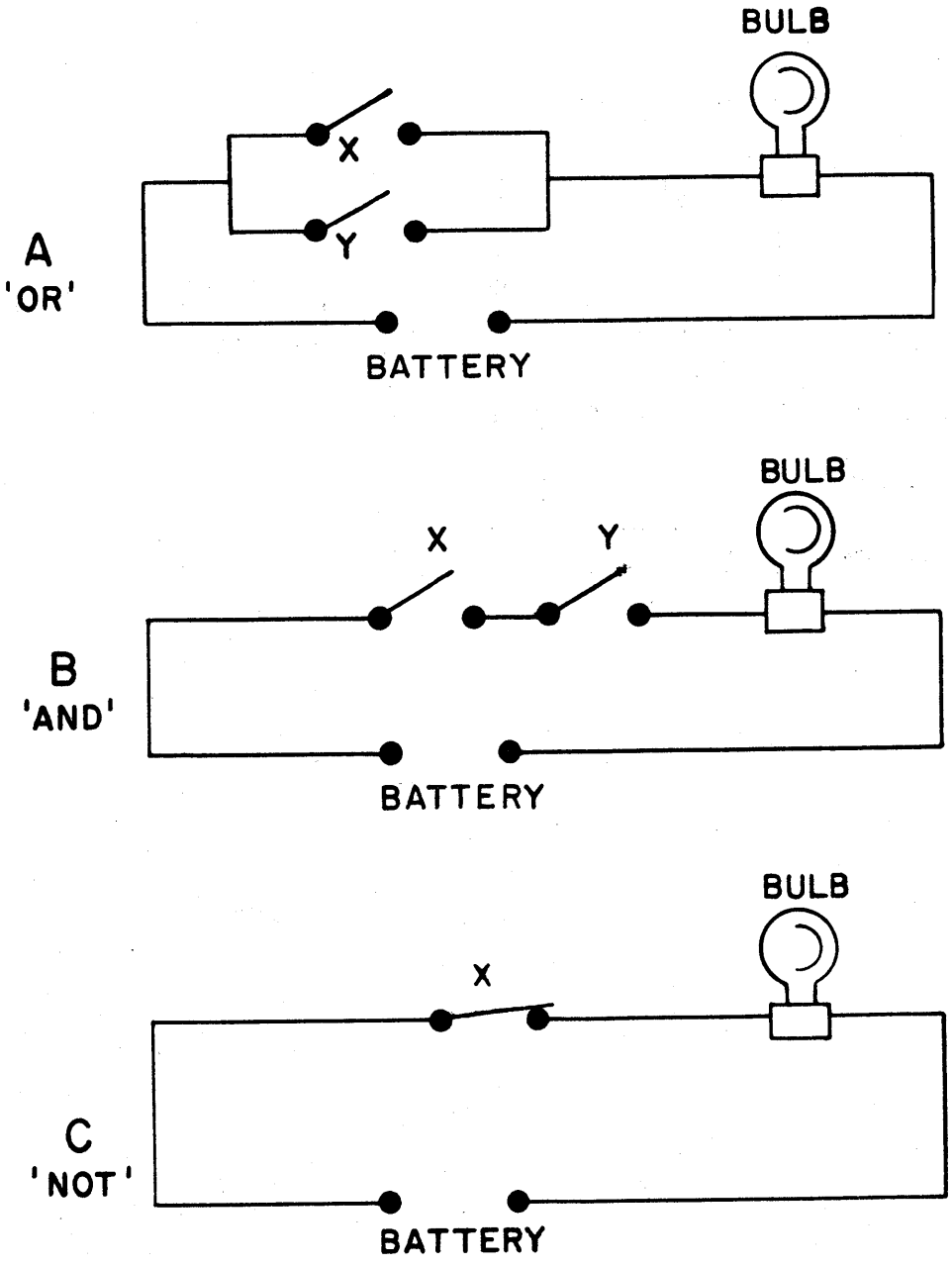


Figure 2-23

sums; 0,1 and 10. Moreover, the last sum (10) does not belong entirely to the order with which the adder is associated, so that it may be considered as a sum of 0 and a carry of 1. Accepting this definition, the results possible from the addition of two bits x and y can be described as follows:

If both x AND y are 1, then the carry is 1 and the sum is 0.

If it does NOT happen that x AND y are 1, AND if either x OR y is 1, then the sum is 1.

If it does NOT happen that either x OR y is 1 then the sum is 0 and the carry is 0.

Here AND, NOT and OR are capitalized because it turns out that they are "logical operations" which can be mechanized. It is not, in general, the function of this Part to introduce actual devices for doing arithmetic. That is done in Part 4 after the fundamentals of electrical and electronic theory have been reviewed in Part 3. However, since the concept of a "logical operation" may be new to the reader, a rather trivial example is given in Figure 2-23 for purposes of clarification. In the "light bulb logic" of the figure, x and y are represented by switches. A lighted bulb is interpreted as a 1 and an unlighted bulb as a 0. In parts (a) and (b) of the figure, a closed switch is interpreted as a 1 and an open switch as a 0. Referring to part (a) of the figure, if the bulb is lighted (indicating a 1) then either the x switch is closed (indicating that x is 1) or the y switch is closed (indicating that y is 1) or both switches are closed

(indicating that both x and y are 1). Thus, the bulb and the switches constitute an OR circuit.

Referring to part (b) of the figure, if the bulb is lighted (indicating a 1), then both the x switch and the y switch must be closed (indicating that both x AND y are 1). Thus, the bulb and the switches constitute an AND circuit.

In part (c) of the figure, the convention about the switch has been reversed, so that a closed switch is interpreted as a 0 and an open switch as a 1. The lighted bulb is still interpreted as a 1 and the unlighted bulb as a 0. Thus, referring to part (c) of the figure, if the bulb is lighted (indicating a 1), then the switch must be closed (indicating that x is 0).

Applying this "light bulb logic" to the binary addition problem: if both the AND circuit and OR circuit bulbs are lighted, then the carry is 1. This corresponds to closed x and y switches in both circuits; i.e. x AND y both 1. If the OR circuit bulb is lighted but the AND circuit bulb is NOT lighted, then there is a sum but no carry; i.e. either x OR y is 1 but NOT both. If neither the AND circuit nor the OR circuit bulb is lighted, then there is no sum and no carry; i.e. it does NOT happen that either x OR y is 1. Notice that the NOT bulb wasn't used. However, the statement that a sum existed depended as much upon the fact that the AND bulb was NOT lighted as it did upon the fact that the OR bulb was lighted.

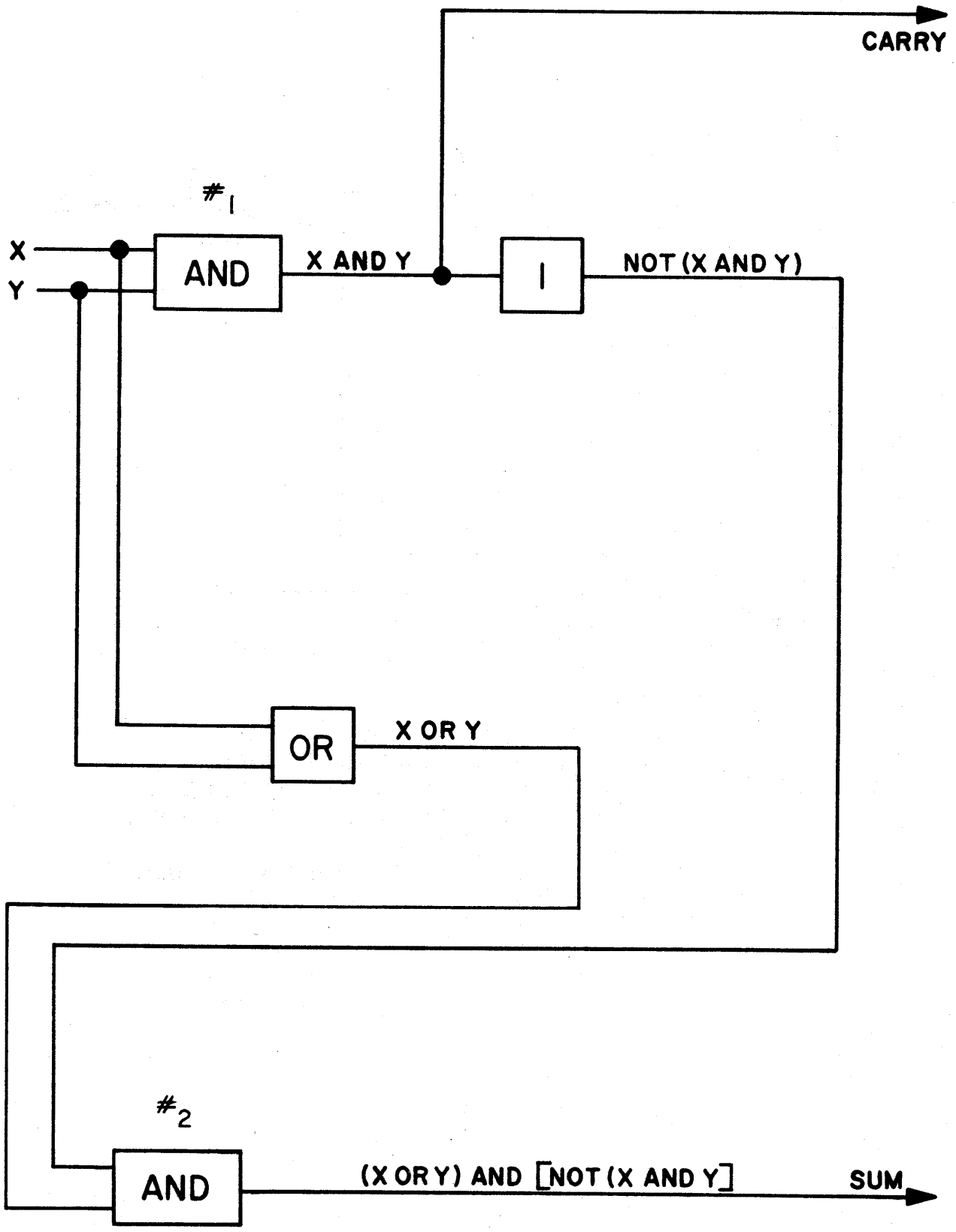

CARRY
#1
X
Y
AND
X AND Y
1
NOT (X AND Y)
OR
X OR Y
#2
AND
(X OR Y) AND [NOT (X AND Y]
SUM

Figure 2-24

Figure 2-24 shows, in block form, a more generalized device for adding two bits x and y. Such a device is called a half adder. Here, the outputs of each "logical" block are assumed to be of such a form that they are usable as inputs to other blocks. The NOT block is represented by the symbol I.

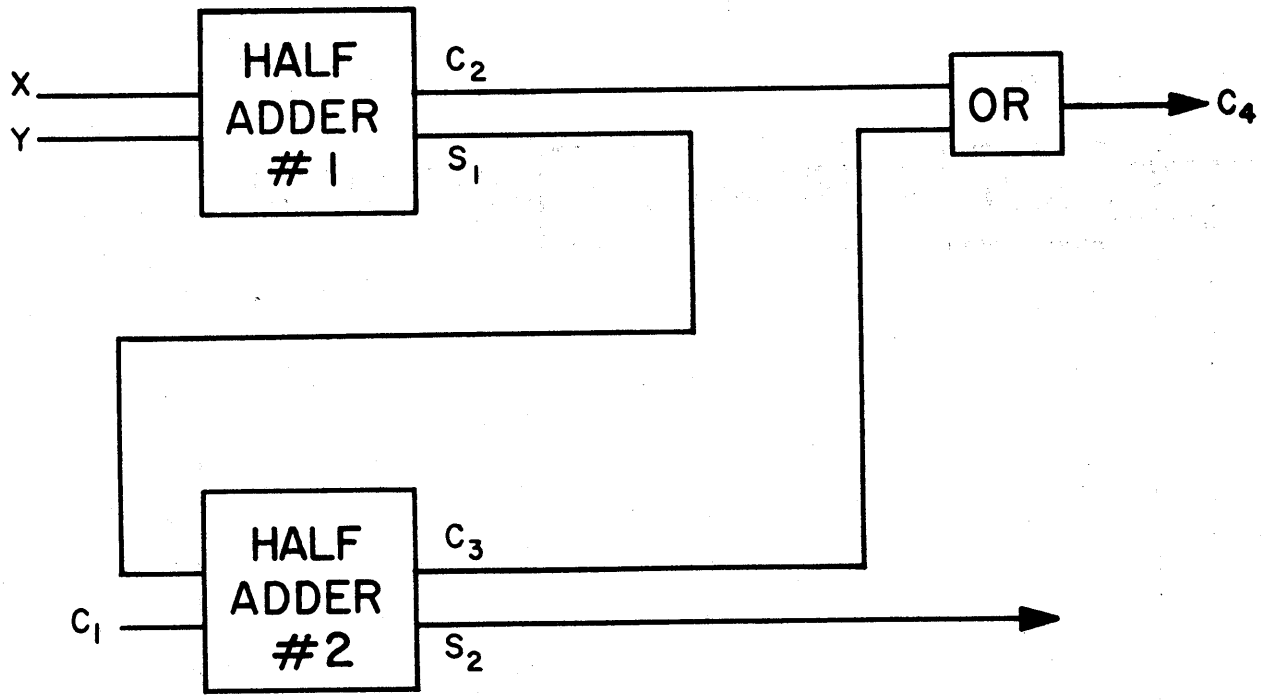
That the half adder satisfies the binary addition table can be seen by considering its response to the various combinations of inputs that can be applied to it. Suppose, for example, that x and y are both 0. Then x AND y is 0, so that the output of the first AND block is 0. Thus, the carry is 0. The output of the NOT block, on the other hand, is 1 (since NOT 0 is 1). The output of the OR block is 0 (since it does NOT happen that either x OR y is 1). Thus, the inputs to AND block #2 are a 1 (from the NOT block) and a 0 (from the OR block) so that the output of the block, which is the sum, is 0 (since it does NOT happen that both the input from the inverter AND the input from the OR circuit are 1). The configuration thus satisfies the binary addition table with respect to the entry; 0 + 0 = 0.

Consider a second combination of inputs to the configuration of Figure 2-24; for example, x = 1, y = 0. For this case, the output of AND block #1 is still 0 (since it does NOT happen that both x AND y are 1). Thus, the carry is 0 and the output of the NOT block is 1 for the same reason as before. This time, however, the output of the OR block is 1 (since it does happen that x OR y is 1; i.e. x = 1). Thus,

both inputs to AND block #2 are 1, so that the output of AND block #2, which is the sum, is 1. Again, the binary addition table has been satisfied; this time with respect to the entry; 1 + 0 = 1 (i.e. a carry of 0 and a sum of 1 are generated). In the same way, the two other possible pairs of inputs can be shown to generate outputs which satisfy the binary addition table.

In order to understand why the configuration of Figure 2-24 is called a half adder, consider again the general problem of adding two numbers. Assuming that each of the numbers has more than one bit, the operation involves performing the addition in each bit position and handling carries that arise in these additions. The half adder provides for the generation of carries to the next higher order, but does not provide for the acceptance of carries from the next lower order. It is, therefore, adequate for performing the addition in the least significant bit position (where no carry from a lower order can be encountered). However, in any of the higher bit positions, an adder which can handle a carry input as well as the x and y inputs is required. Such a device is called a full adder.

The possible combinations that can be encountered in the addition of three bits (x, y and carry) are as follows:



LEGEND:

C_1 = CARRY FROM LOWER ORDER
S_1 = SUM OUTPUT OF HALF ADDER #1
C_2 = CARRY OUTPUT FROM HALF ADDER #1
S_2 = SUM OUTPUT FROM HALF ADDER #2
= FULL ADDER SUM
C_3 = CARRY OUTPUT FROM HALF ADDER #2
C_4 = C_2 OR C_3 = CARRY TO HIGHER ORDER

Figure 2-25

INPUT COMBINATIONS			OUTPUTS FROM HALF ADDER #1		OUTPUTS FROM HALF ADDER #2		OUT-PUTS FROM OR BLOCK
X	Y	C_1	S_1	C_2	C_3	S_2	C_4
0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
0	1	0	1	0	0	1	0
0	0	1	0	0	0	1	0
1	1	0	0	1	0	0	1
1	0	1	1	0	1	0	1
0	1	1	1	0	1	0	1
1	1	1	1	1	1	1	1
						FULL ADDER OUTPUTS	

Figure 2-26

$0 + 0 + 0 = 0$ (1)

$0 + 0 + 1 = 1$ (2)

$0 + 0 + 1 = 10$ (3)

$1 + 1 + 1 = 11$ (4)

$0 + 1 + 0 = 1$ (5)

$1 + 0 + 0 = 1$ (6)

$1 + 0 + 1 = 1C$ (7)

$1 + 1 + 0 = 10$ (8)

Figure 2-25 shows how a full adder can be formed employing two of the half adders of Figure 2-24 and an extra OR block. The x, y and carry combinations that can appear at the inputs of this adder and the resulting outputs from the various "logical" blocks are tabulated in Figure 2-26.

As noted above, there are four distinct sums that can be obtained by adding possible combinations of three bits. The first of these, which arises from the addition of three 0's appears as entry 1 of the table of Figure 2-26. The second is represented by entries 2,3,4; the third by entries 5,6,8 and the fourth by entry 8. In each case, the outputs from the full adder satisfy the binary addition table. The first sum which is 0, appears as a 0 sum (S_2) and a 0 carry (C_4). The second sum, which is I, appears as a sum of 1 and a carry of 0. The third sum, which is 10, appears as a sum of 0 and a carry of 1. The fourth sum which is 11, appears as a sum of 1 and a carry of 1. Notice that outputs resulting from any particular combination of 1's and 0's are the same regardless of where the individual bits are applied to

the adder. For example, entries 2, 3 and 4 produce the same outputs although in 2 the 1 is applied to the x input, in 3 the 1 is applied to the y input and in 4 the 1 is applied to the C_1 input. This corresponds to the arithmetical fact that the sum of any particular bits is the same no matter in what order they are added.

As has already been stated, the full adder of Figure 2-25 is formed from two half adders such as are illustrated in Figure 2-24 and an extra OR block. Since it does not make use of any new "logical" operations in order to satisfy the addition table, its operation can be understood on the basis of an understanding of the half adder of Figure 2-24. As an illustration, it is worthwhile to check the operation of the full adder against one of the entries in the table of Figure 2-26. Consider, for example, entry #6. In this case, x is 1 and y is 0. Thus, the sum output of half adder #1 (S_1) is 1 while the carry output (C_2) is 0. The inputs to half adder #2 (S_1 and C_1) are both 1. Thus, its sum output (S_2) is 0 while its carry output (C_3) is 1. The inputs to the OR block (C_2 and C_3) are 0 and 1 (respectively). Thus its output is 1. These results are precisely the set of outputs shown against entry #6 of the table of Figure 2-26. It is a good exercise for the reader to check the operation of the full adder of Figure 2-25 against the remainder of the entries of the table of Figure 2-26.

What has just been demonstrated is that "logical" blocks can be combined in such a way as to satisfy the binary addi-

tion table. It is demonstrated in Part 4 that these blocks can be successfully mechanized. The important thing to understand here is that this mechanization of the binary addition table is something quite different from the mechanization of the decimal counting process which was discussed in the preceding chapter.

One of the important differences between "logical" addition and addition by counting is that the former implies simultaneous action; i.e. when certain inputs are received certain outputs are generated. Remove the inputs, and the outputs disappear. Compare this to counting which is a successive process and thus implies the storage of the accumulating sum. In a sense, the full adder "stores" the binary addition table. In the same sense, the counter "remembers" how to count. But in addition to this kind of "memory", the counter must store the accumulating count as it proceeds from step to step.

The mechanization of the binary addition table is practical because the table is so brief; i.e. the possible combinations of bits are so few. Notice that the "logic" of this mechanization is essentially binary in nature; that is, the outputs of each one of the blocks is either a 1 or a 0.

The brevity of the binary addition table is just one aspect of the simplicity, from the point of view of mechanization, offered by the binary number system. The task of mechanizing the representation of numbers, which of course must precede the task of mechanizing operations, is greatly simplified by the fact that the binary system employs only the two symbols 0 and 1.

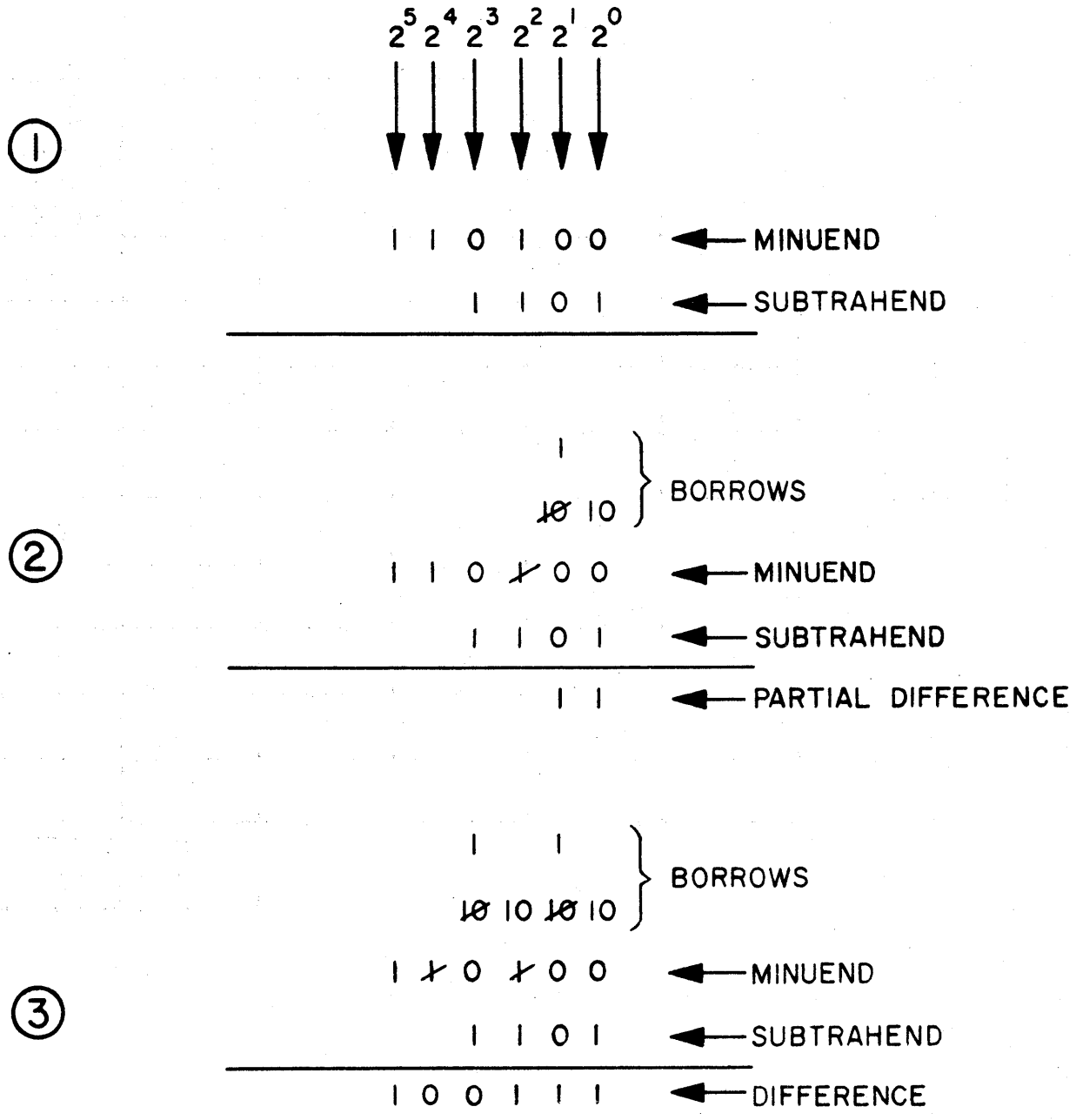

2^5 2^4 2^3 2^2 2^1 2^0
1
1 1 0 1 0 0
MINUEND
1 1 0 1
SUBTRAHEND
BORROWS
2
1 1 0 0 0 0
MINUEND
1 1 0 1
SUBTRAHEND
1 1
PARTIAL DIFFERENCE
BORROWS
3
1 1 0 0 0 0
MINUEND
1 1 0 1
SUBTRAHEND
1 0 0 1 1 1
DIFFERENCE

Figure 2-27

3.2 BINARY SUBTRACTION

Binary subtraction can be performed by a method involving reference to a memorized table and borrow from column to column which is similar to the normal method of decimal subtraction. An example of the pencil and paper binary subtraction process is illustrated in Figure 2-27. The operation is shown in three steps for purposes of clarification. In the first step, the minuend and the subtrahend have been set down. To perform the subtraction in the 2^0 order a borrow from the 2^1 order is necessary. However, since the 2^1 order bit of the minuend is 0, this in turn entails a borrow from the 2^2 order. These borrows and the subsequent subtractions in the first two orders are shown in step 2. A borrow from the 2^2 order causes the 2^2 order bit to be diminished from 1 to 0. The borrow initially appears as a 10 in the 2^1 order. However, a 1 is then borrowed for the 2^0 order diminishing the 10 to 1. The 1 that is borrowed from the 2^1 order appears as a 10 in the 2^0 order. The fact that a 1 borrowed from the nth order appears as a 10 in the (n - 1)th order is common to all number systems. The 10 always has the value of the radix of the system in use. Thus when a 1 is borrowed from the nth order in the decimal subtraction, it has the value of ten in the (n - 1)th order. However, when a 1 is borrowed from the nth order in binary subtraction, it has the value of two in the (n - 1)th order.

When the subtraction of Figure 2-27 has proceeded to the point shown in step 2, another borrow situation is encountered. Since the 2^2 bit of the minuend has been reduced to 0 by the earlier borrow, a borrow must be made from the 2^3 order. However, since the 2^3 order bit of the minuend is 0, a borrow must first be made from the 2^4 order. These two borrows and the remainder of the subtractions are shown in step 3. This subtraction, like any other, can be checked by adding the subtrahend and the difference. If the subtraction has been correctly performed, the check addition will yield the minuend (since subtraction answers the question, "what number must be added to the subtrahend in order to obtain the minuend?").

Since the table that is consulted in performing binary subtraction is the binary addition table, it follows that binary subtraction should offer the same advantages from the point of view of mechanization that are offered by binary addition. In fact, this proves to be the case. A half subtractor similar to the half adder of Figure 2-24 can be formed by combining logical blocks. Moreover, a full subtractor similar to the full adder of Figure 2-25 can be formed by combining two half subtractors together with some extra AND and OR blocks. However, subtraction can be approached in a slightly different manner. As explained in Chapter 2 of this Part, complementing the subtrahend and adding is the equivalent of subtracting. Thus, if the complement of the subtrahend can be made available, the full adder of Figure 2-25 can be used to perform subtraction, so that no explicit subtraction device is necessary.

Subtraction in the decimal system can be performed by using either 10's complements or 9's complements as explained in Chapter 2. In the binary number system 2's complements or 1's complements are used. Complements in the binary number system are defined exactly as are complements in the decimal system; i.e. they are defined with respect to the machine modulus. (As was stated in Chapter 2, the modulus of any machine is the number of distinct numbers it can represent.) If a binary machine has modulus M, then the 2's complement of any number x with respect to that modulus is M - x. For the same machine, the 1's complement of x is (M - 1) - x. The 2's and 1's complements of binary numbers are thus analogous to the 10's and 9's complements of decimal numbers.

The modulus of a binary machine is of the same form as the modulus of a decimal machine, i.e. it is a 1 followed by a sequence of 0's. When 1 is subtracted from this modulus, the result is a sequence of 1's. This is comparable to the modulus minus 1 for a decimal machine which is a sequence of 9's. The rules for forming 2's complements and 1's complements are thus analogous to the rules for forming 10's complements and 9's complements. That is, the 2's complement of a number, n, can be formed by subtracting the least significant non-zero bit of that number from 10 (decimal 2) and subtracting all more sifnificant bits from 1. In the same way, the 1's complement of n can be formed by subtracting each bit of n from 1. Moreover, if a bit, x, is subtracted from 1, only two results are possible. If x is 1, then 1 - x is 0.

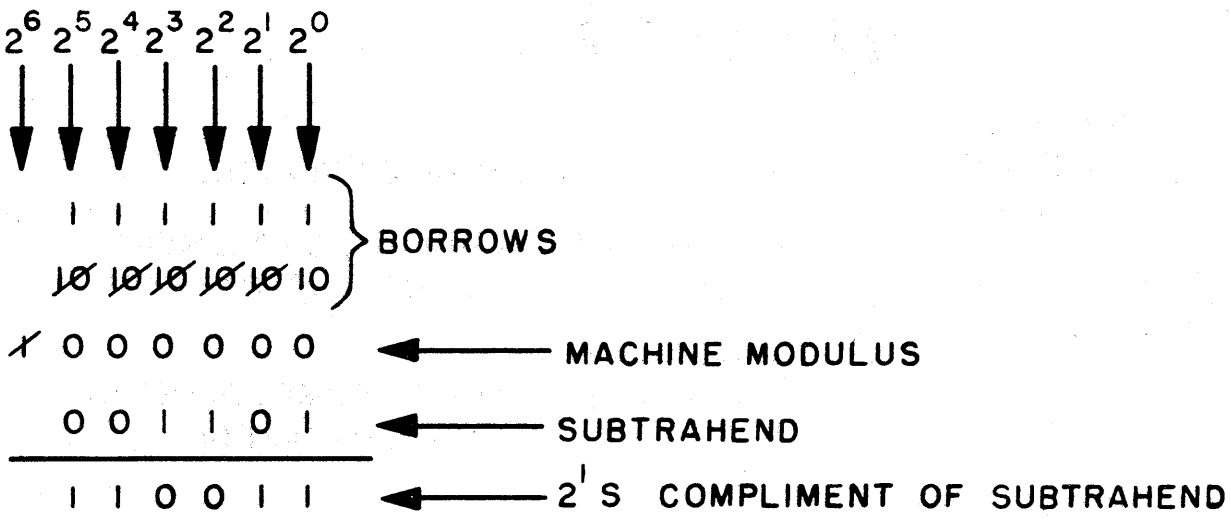

2^6 2^5 2^4 2^3 2^2 2^1 2^0
1 1 1 1 1 1
BORROWS
1Ø 1Ø 1Ø 1Ø 1Ø 10
1 0 0 0 0 0 0
MACHINE MODULUS
0 0 1 1 0 1
SUBTRAHEND
1 1 0 0 1 1
2'S COMPLIMENT OF SUBTRAHEND

Figure 2-28

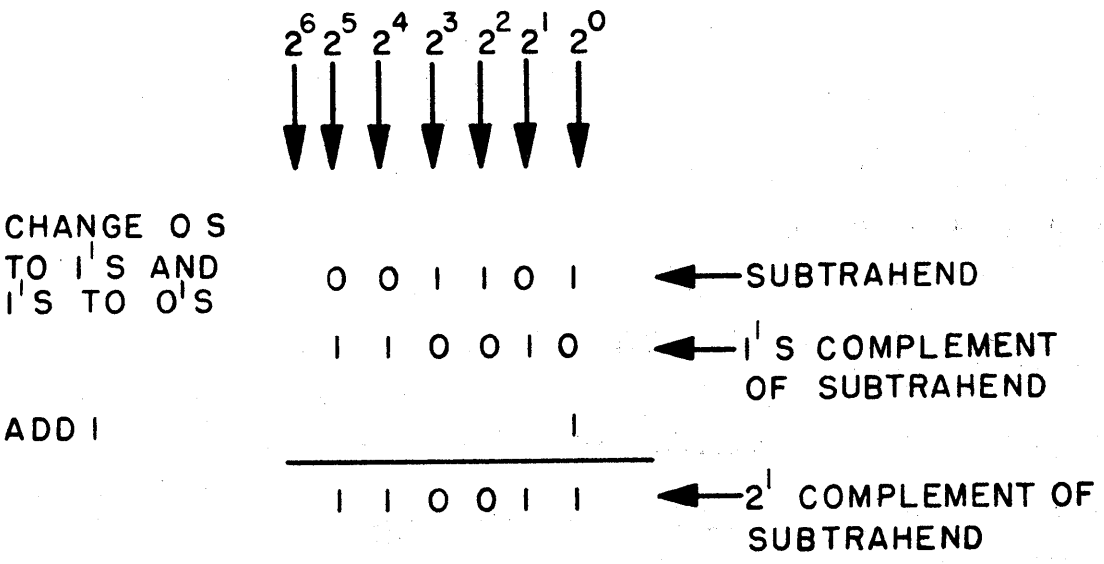

2^6 2^5 2^4 2^3 2^2 2^1 2^0
CHANGE 0 S TO 1 'S AND 1 'S TO 0 'S
0 0 1 1 0 1
SUBTRAHEND
1 1 0 0 1 0
1 'S COMPLEMENT OF SUBTRAHEND
ADD 1
1
1 1 0 0 1 1
2^1 COMPLEMENT OF SUBTRAHEND

Figure 2-29

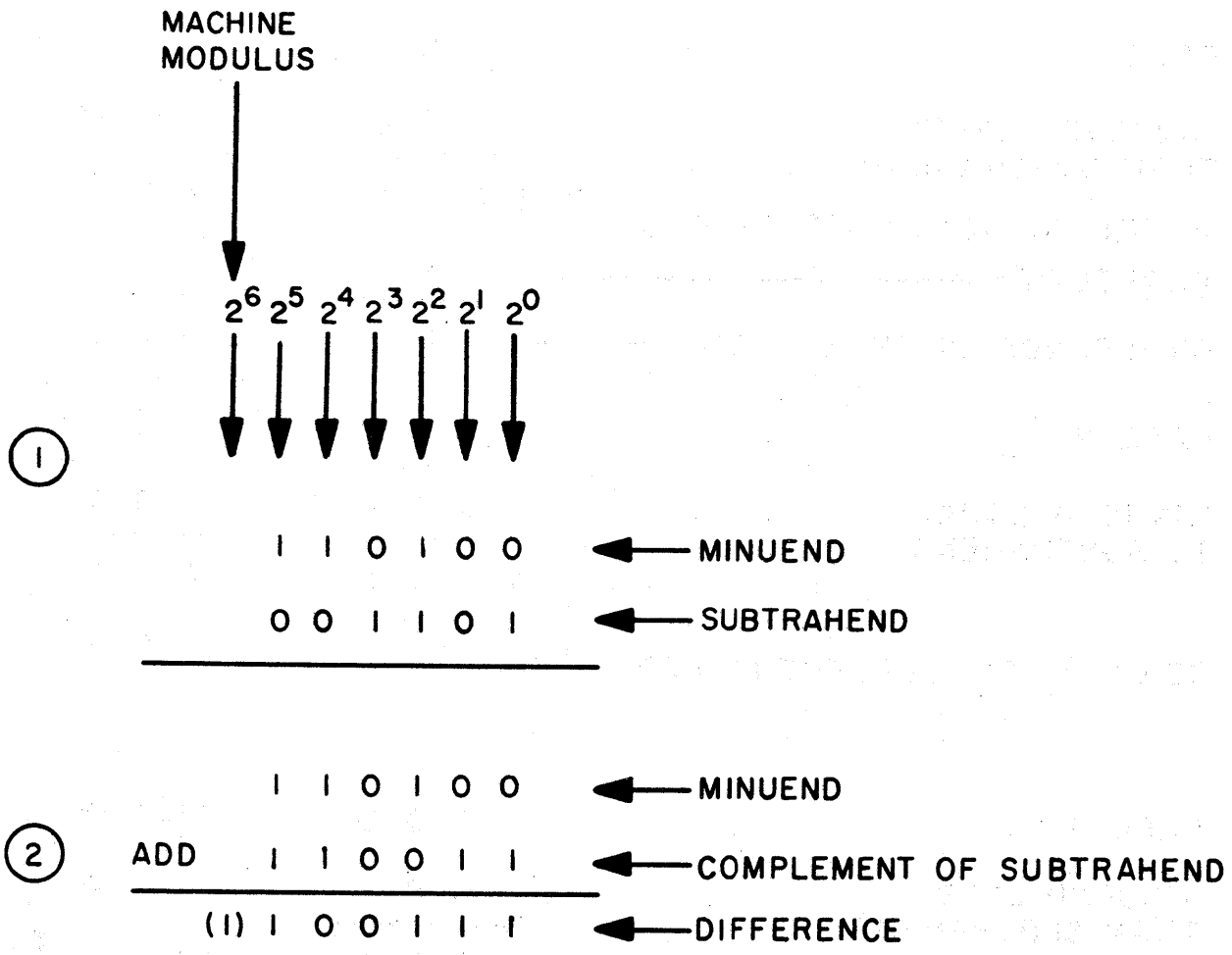

MACHINE
MODULUS
2^6 2^5 2^4 2^3 2^2 2^1 2^0
1
1 1 0 1 0 0
MINUEND
0 0 1 1 0 1
SUBTRAHEND
1 1 0 1 0 0
MINUEND
2
ADD 1 1 0 0 1 1
COMPLEMENT OF SUBTRAHEND
(1) 1 0 0 1 1 1
DIFFERENCE

Figure 2-30

On the other hand, if x is 0, the 1 - x is 1. Thus, the 1's complement of a binary number can be formed by the simple process of changing all the 1's of that number to 0's and changing all the 0's of that number to 1's. This is an extremely important simplification when considered from the point of view of mechanization of the complementation process. Moreover, since the 2's complement of a number n is always 1 more than the 1's complement, 2's complements can be formed by first taking 1's complements and then adding 1.

Figure 2-28 shows how the 2's complement of the subtrahend can be formed by performing the subtraction indicated by its definition. Figure 2-29 shows that the same result is obtained by applying the two rules mentioned above:

a. To form the 1's complement of a binary number n, change all the 1's of the number to 0's and all the 0's to 1's.

b. To form the 2's complement of a binary number, form the 1's complement and then add 1 to it.

Notice that in forming the 1's complement of a number, the 0's to the left of the most significant non-zero bit (and within the range of the machine) must be changed to 1's.

The solution of the subtraction problem of Figure 2-27 by the technique of adding the 2's complement of the subtrahend to the minuend is shown in Figure 2-30. Step 1 shows both numbers in true form. Step 2 shows the subtrahend in complement form and shows the true form of the difference, which is obtained by addition. Here, the machine modulus is assumed to be 2^6; i.e. the machine is capable of holding the

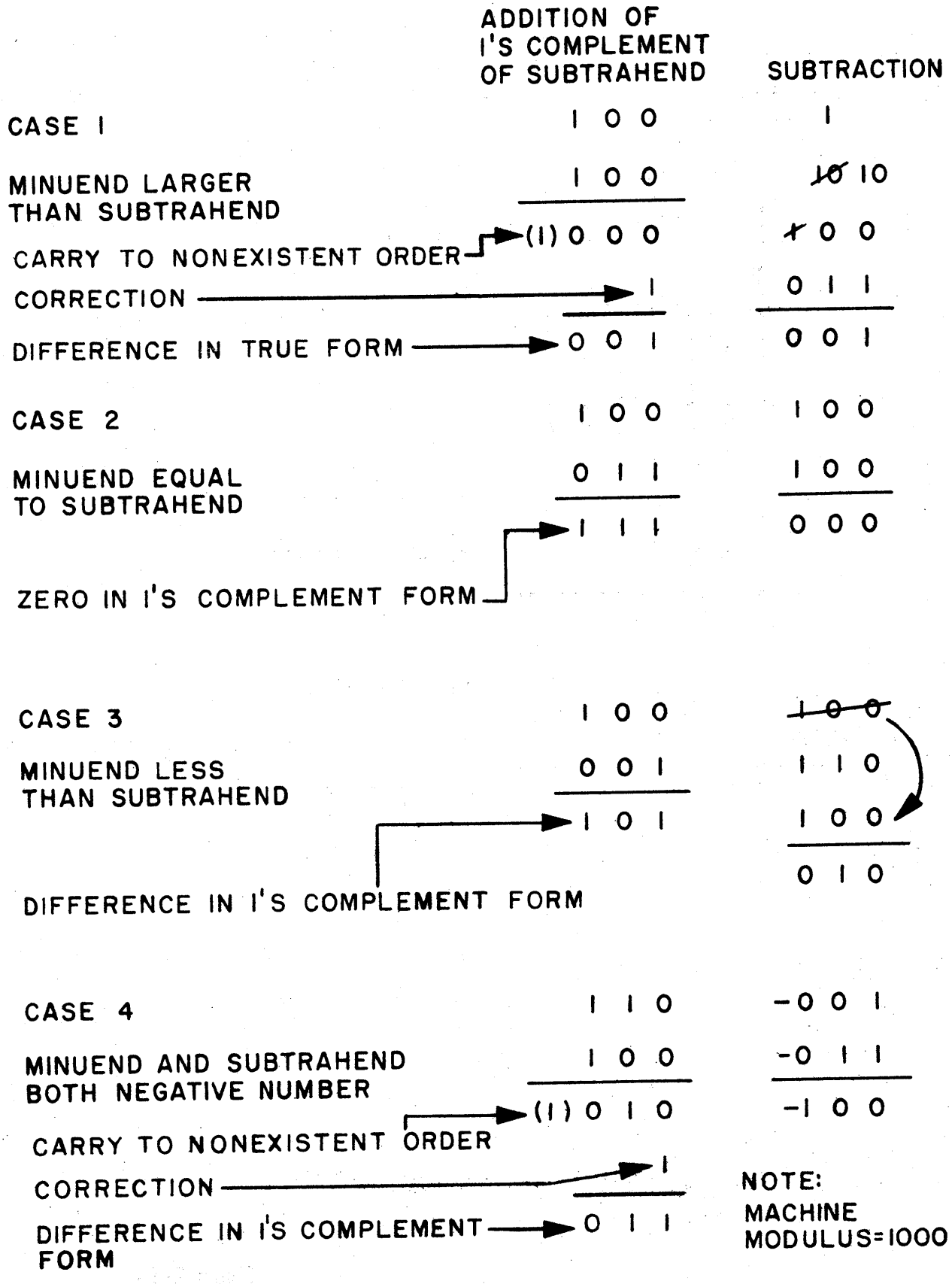

ADDITION OF
1'S COMPLEMENT
OF SUBTRAHEND
SUBTRACTION
CASE 1
1 0 0
1
MINUEND LARGER
THAN SUBTRAHEND
1 0 0
10 10
CARRY TO NONEXISTENT ORDER
(1) 0 0 0
0 0 0
CORRECTION
1
0 1 1
DIFFERENCE IN TRUE FORM
0 0 1
0 0 1
CASE 2
1 0 0
1 0 0
MINUEND EQUAL
TO SUBTRAHEND
0 1 1
1 0 0
1 1 1
0 0 0
ZERO IN 1'S COMPLEMENT FORM
CASE 3
1 0 0
1 0 0
MINUEND LESS
THAN SUBTRAHEND
0 0 1
1 1 0
1 0 1
1 0 0
0 1 0
DIFFERENCE IN 1'S COMPLEMENT FORM
CASE 4
1 1 0
-0 0 1
MINUEND AND SUBTRAHEND
BOTH NEGATIVE NUMBER
1 0 0
-0 1 1
(1) 0 1 0
-1 0 0
CARRY TO NONEXISTENT ORDER
CORRECTION
1
NOTE:
MACHINE
MODULUS=1000
DIFFERENCE IN 1'S COMPLEMENT
FORM
0 1 1

Figure 2-31

numbers 000000 through 111111. This capacity requires six orders, that is 2^0 through 2^5. Thus, there is no 2^6 order and the carry which arises during the addition is lost. It is this phenomenon, of course, upon which the whole scheme of subtraction by means of complementation and addition depends.

Use of 1's complements rather than 2's complements for subtraction has the same consequences that are encountered in decimal arithmetic when 9's complements are used instead of 10's complements; that is, corrections are required in certain cases. As is the situation in the case of 9's complements, these corrections can be taken care of by end-around carry. This can be seen from Figure 2-31 which compares the result of adding the 1's complement of the subtrahend to the minuend with the result of ordinary subtraction in four different cases. In the first case, there is a carry from the highest order. Here, the difference is a positive number in true form and a corrective addition is required. In the second case, the difference is 0 which appears in 1's complement form. There is no carry from the highest order and no correction is required. In the third case, the difference is a negative number which appears in 1's complement form when 1's complements are used. There is no carry from the highest order and no correction is required. In the fourth case, the difference is a negative number which appears in 1's complements form when complements are used. There is a carry from the highest order and a correction is required (since 011 rather than 010 is the 1's complement of 100 which is the solution obtained by the ordinary method of subtraction). In general, whenever a corrective addition

is required, there is a carry from the highest order and whenever there is no correction required, there is no carry from the highest order. Thus, in all cases, the correct solution is obtained by adding the carry (if any) from the highest order into the least significant bit position. This is called end-around carry.

At this point it is natural to raise the question of how negative numbers in complement form can be distinguished from positive numbers in true form. It turns out that in this regard, also, binary numbers offer an advantage with regard to representation. The sign of a number is binary in nature, that is a number is either positive or negative, with the exception of 0 which can be arbitrarily assigned a sign. Thus, a bit representing sign can be used in addition to the bits representing magnitude. A 0 in the sign bit position can be interpreted to mean that the number is positive and in true form while a 1 in the sign bit position can be interpreted to mean that the number is negative and in complement form. If the sign bits are assigned to the most significant bit position and are treated as a part of the number in the addition operation, then the resultant sign bit will be a true indication of the sign of the result. In order to see how this works, four cases must be considered as follows:

a. If two positive numbers are added, their sign bits are 0. Unless the capacity of the machine is exceeded, there is no carry from the highest magnitude bit order. Thus the sign bit position of the result is 0 corresponding to the fact that the result is a positive number.

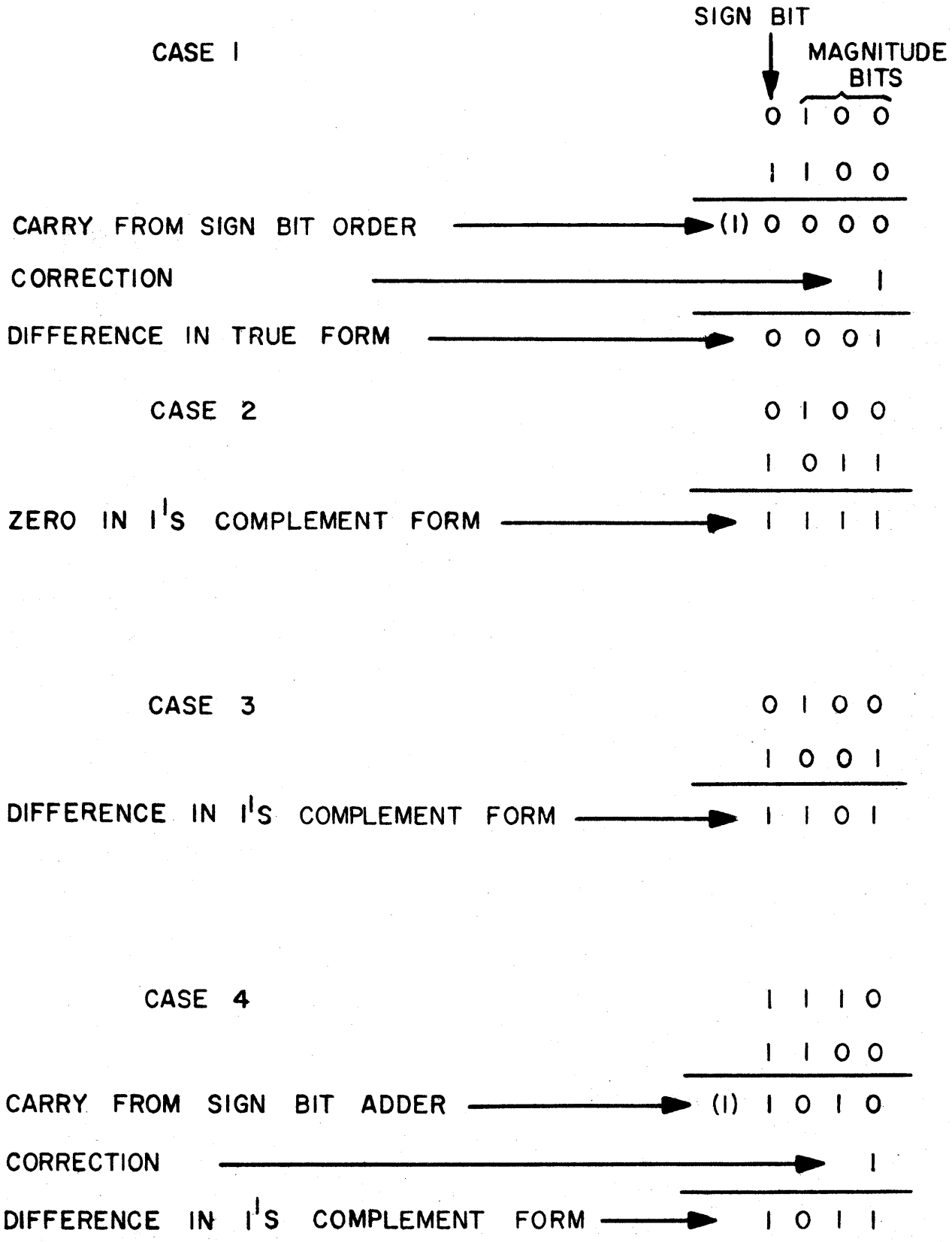

SIGN BIT
MAGNITUDE BITS
CASE 1
0 1 0 0
1 1 0 0
CARRY FROM SIGN BIT ORDER
(1) 0 0 0 0
CORRECTION
1
DIFFERENCE IN TRUE FORM
0 0 0 1
CASE 2
0 1 0 0
1 0 1 1
ZERO IN 1'S COMPLEMENT FORM
1 1 1 1
CASE 3
0 1 0 0
1 0 0 1
DIFFERENCE IN 1'S COMPLEMENT FORM
1 1 0 1
CASE 4
1 1 1 0
1 1 0 0
CARRY FROM SIGN BIT ADDER
(1) 1 0 1 0
CORRECTION
1
DIFFERENCE IN 1'S COMPLEMENT FORM
1 0 1 1

Figure 2-32

b. If a negative number is added to a smaller positive number, there is no carry from the highest magnitude bit position (see case 3, Figure 2-31). Thus, the resultant sign bit is the sum of the sign bits of the augend and addend, or 1. This corresponds to the fact that the result is a negative number.

c. If a negative number is added to a larger positive number, there is a carry from the highest magnitude bit position (see case 1, Figure 2-31). This carry added to the sign bit of the negative number yields a sum of 0 in the sign bit position (corresponding to the fact that the result is a positive number) and a carry from the sign bit order (which can be used for the end-around carry correction).

d. If a negative number is added to another negative number, both sign bits are 1. However, if the numbers are within the capacity of the machine, there is a carry from the highest magnitude position in such an addition. Thus the addition yields a sum of 1 in the sign bit position (corresponding to the fact that the result is a negative number) and a carry from the sign bit order (which can be used for the end-around carry correction).

The subtractions (by complementation and addition) of Figure 2-31 are repeated in Figure 2-32 with sign bits added, in order to verify the results of operating upon sign bits in this manner. In Figure 2-32, as in Figure 2-31, the end-around carry corrections have been made.

3.3 BINARY MULTIPLICATION

Binary multiplication can be performed by the ordinary pencil and paper method involving reference to a memorized multiplication table, shift and entry of partial products and final summation of partial products. An example of such a multiplication is illus-

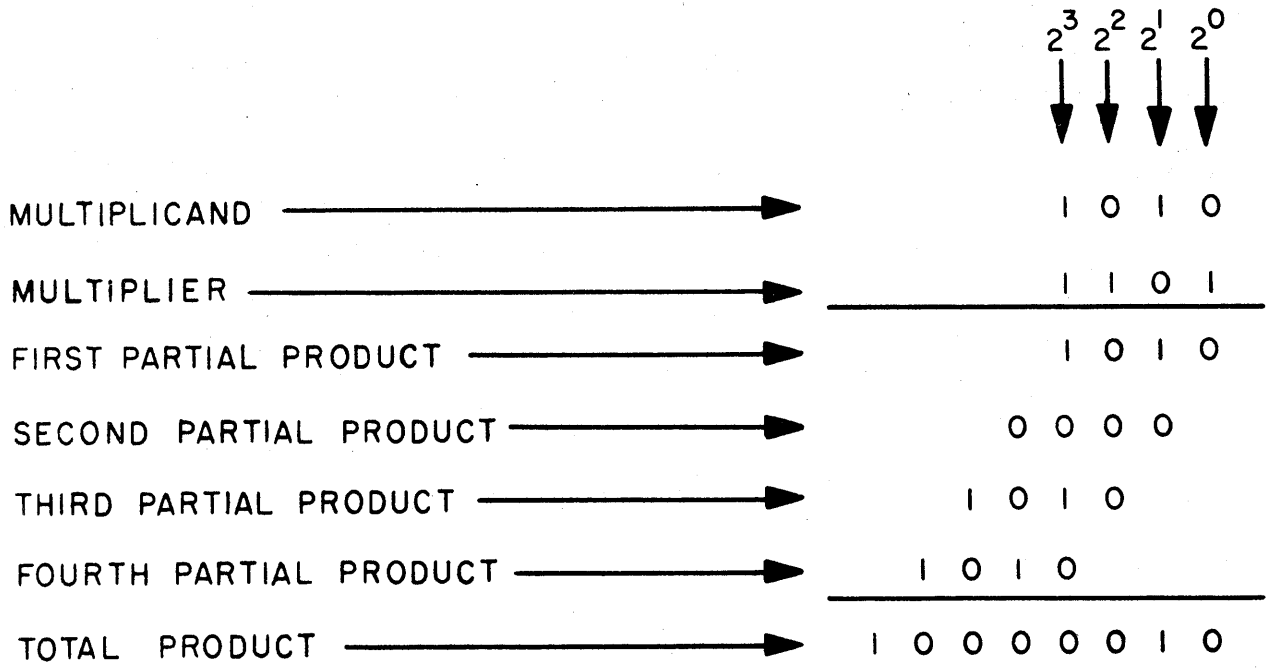

2^3 2^2 2^1 2^0
MULTIPLICAND
1 0 1 0
MULTIPLIER
1 1 0 1
FIRST PARTIAL PRODUCT
1 0 1 0
SECOND PARTIAL PRODUCT
0 0 0 0
THIRD PARTIAL PRODUCT
1 0 1 0
FOURTH PARTIAL PRODUCT
1 0 1 0
TOTAL PRODUCT
1 0 0 0 0 0 1 0

Figure 2-33

trated in Figure 2-33. However, the binary multiplication table is more trivial even than the binary addition table. It can be stated as follows:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Thus there is no problem of carries arising in the multiplication of two bits and there are only two possible products, 0 and 1. In a multiplication such as that of Figure 2-33, then, the partial products can be generated in accordance with a very simple rule which satisfies the binary multiplication table. This rule is as follows: For each multiplier bit that is 1, enter the multiplicand with its least significant bit lined up with that multiplier bit. For each multiplier bit which is 0, make no entry. Referring to Figure 2-33, it can be seen that the multiplicand (1010) has been entered once for each 1 in the multiplier as specified by the rule just given. The second partial product (0000) is the equivalent of no entry, corresponding to the 0 in the 2^1 order of the multiplier.

The only troublesome part of a binary multiplication is the summation of partial products. When hand methods are replaced by a machine routine, this summation is performed on a step-by-step basis. Thus the first partial product is generated; then the second partial product is generated and added to the first; then the third partial product is generated and added to the sum of the first two; and so on. The machine

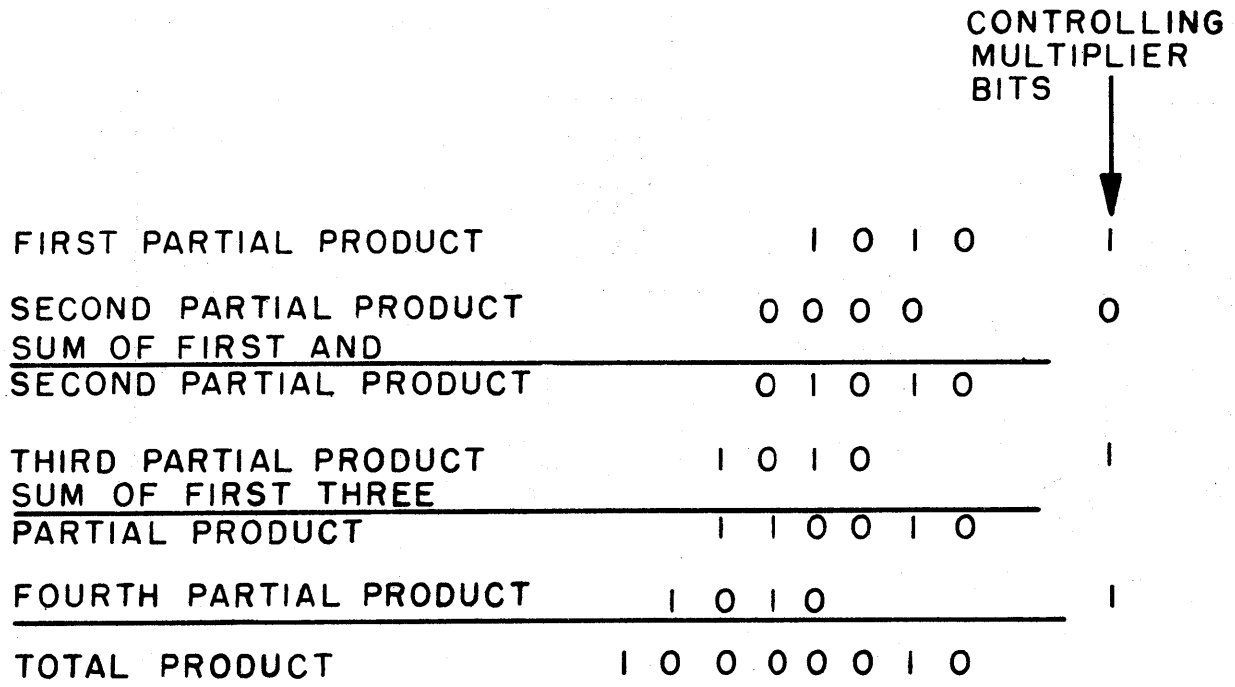

CONTROLLING
MULTIPLIER
BITS
FIRST PARTIAL PRODUCT
1 0 1 0
1
SECOND PARTIAL PRODUCT
0 0 0 0
0
SUM OF FIRST AND
SECOND PARTIAL PRODUCT
0 1 0 1 0
THIRD PARTIAL PRODUCT
1 0 1 0
1
SUM OF FIRST THREE
PARTIAL PRODUCT
1 1 0 0 1 0
FOURTH PARTIAL PRODUCT
1 0 1 0
1
TOTAL PRODUCT
1 0 0 0 0 0 1 0

Figure 2-34

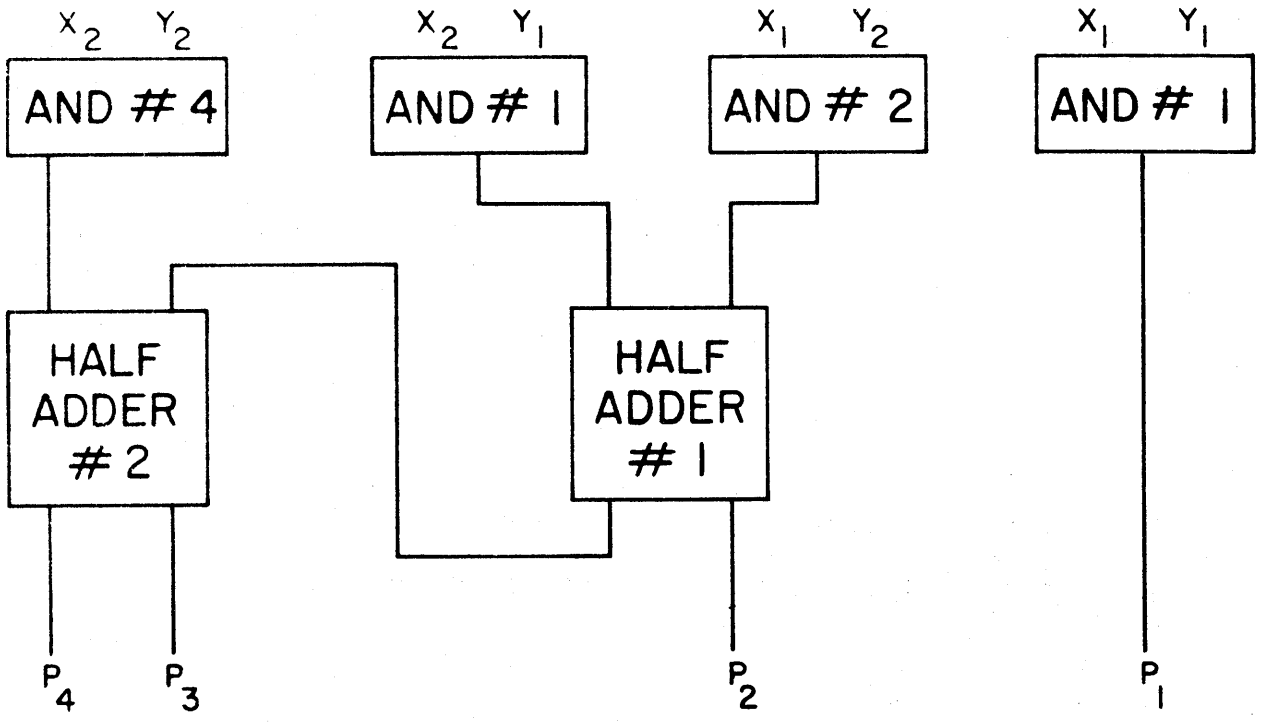

X2 Y2
AND # 4
X2 Y1
AND # 1
X1 Y2
AND # 2
X1 Y1
AND # 1
HALF ADDER # 2
HALF ADDER # 1
P4
P3
P2
P1

Figure 2-35

routine for multiplication where the multiplier has n bits is a sequence of n steps. For each 1 bit of the multiplier the step comprises a shift and an addition. For each 0 bit of the multiplier, the step comprises only a shift. The machine routine for the multiplication of Figure 2-33 is illustrated in Figure 2-34.

Binary multiplication can be performed by means of simultaneous or "logical" configurations which satisfy the trivially simple binary multiplication table. The set of "logical" operations introduced for implementing the binary addition table suffice for the performance of binary multiplication. A "logical" configuration employing two of the half adders of Figure 2-24 and four AND circuits, and capable of generating the product of any two two-bit numbers is illustrated in Figure 2-35. Here, x_1 and y_1 are the units order bits of the two numbers while x_2 and y_2 are the 2^1 order bits. P_1 through P_4 are the 2^0 through 2^3 order bits respectively of the product. The operation of the multiplier of Figure 2-26 can easily be checked against the binary multiplication table in terms of the half-add and AND operations which were explained in Section 3.1, above. For example, assume $x_2x_1 = 10$ and $y_2y_1 = 11$. Then the output of the first AND block is (since x_1 AND y_1 are NOT both 1) that P_1 is 0. The output of the second AND block is 0 (since x_1 AND y_2 are NOT both 1) but the output of the third AND block is 1 (since x_2 AND y_1 are both 1). Thus, the first half adder receives a single input of 1. Thus its sum output (P_2) is 1 while its carry output is 0. The output

of the fourth AND circuit is 1 (since x_2 AND y_2 are both 1). Thus the second half adder receives a single input of 1. Thus its sum output (P_3) is 1 while its carry output (P_4) is 0. Summarizing these results, $P_4P_3P_2P_1$ = 0110. Since this is the true product of 10 and 11, the binary multiplication table is satisfied.

As noted above, and illustrated in Figure 2-34, binary multiplication can be performed as a shift and add routine. A configuration which satisfies the binary addition table has already been introduced and it has been stated that this configuration can be successfully mechanized. Thus if the shift operation can be mechanized, and it can, multiplication can be performed by a routine employing a shifting device and an adding device.

3.4 BINARY DIVISION

The pencil and paper method of performing binary division is more simple in one important respect than the pencil and paper method of decimal division. The decimal division process starts with an inspection of the dividend and divisor to determine how the divisor can be lined up under the dividend so that it is at least as small as the dividend. This is actually a determination of how many times the divisor can be multiplied by ten without becoming larger than the dividend. When this first determination has been completed, a second investigation must be undertaken to discover the largest multiple of the shifted divisor which is smaller than or equal to the dividend.

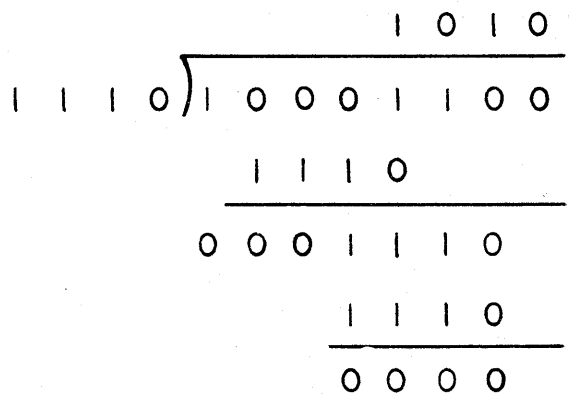


Figure 2-36

Thus, for example, 7 goes into 420 a total of 60 times. In performing this division, the first step is to establish that 7 can be multiplied by 10 once and still remain smaller than or equal to the dividend 420. The second step is to determine that 70 can be multiplied by 6 and still remain smaller than or equal to 420. In the case of binary division, this second step is unnecessary. This follows from the fact that the binary number system only allows for multiplication by 0, 1 and powers of two. Thus in binary division, trial multiplications are eliminated. The division process thus reduces to the determination of where the divisor should be entered under the dividend before each subtraction, together with the actual performance of the subtractions and the entry of the quotient bits. For each position of the least significant divisor bit that corresponds to a successful subtraction of the divisor from the dividend or from a current remainder, a 1 is entered in the quotient. For each position where no such successful subtraction can be performed a 0 is entered.

A binary long division example is illustrated in Figure 2-36. In this case, the dividend is an integral multiple of the divisor so that a remainder of 0 is obtained as a result of the second subtraction. In general, of course, binary division does not produce 0 remainders any more than decimal division does. The division process is thus, in general, left unfinished after having been carried out to an arbitrary and predetermined number of orders.

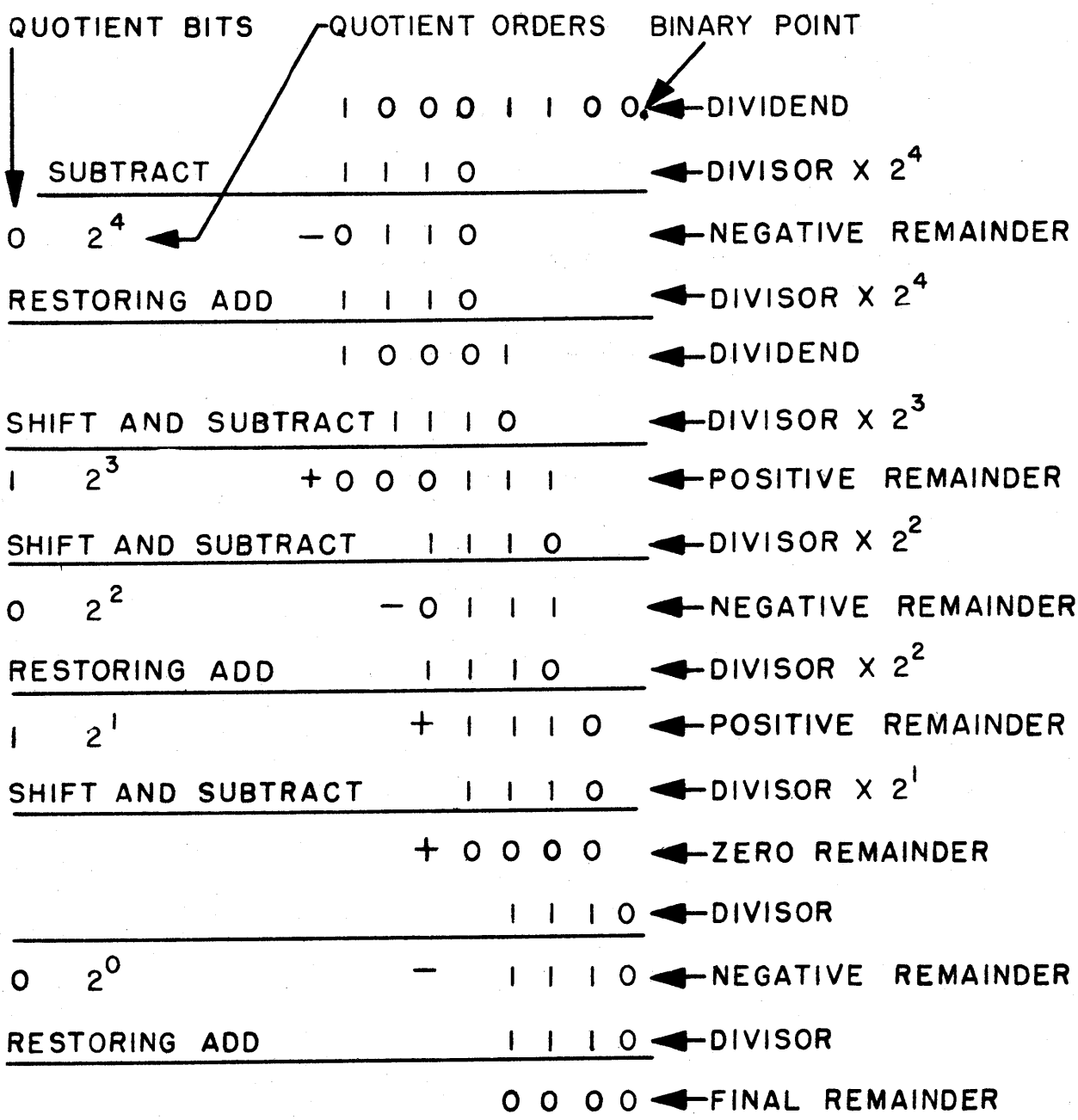

QUOTIENT BITS
QUOTIENT ORDERS
BINARY POINT
1 0 0 0 1 1 0 0. ← DIVIDEND
SUBTRACT 1 1 1 0 ← DIVISOR X 2^4
0 2^4 −0 1 1 0 ← NEGATIVE REMAINDER
RESTORING ADD 1 1 1 0 ← DIVISOR X 2^4
1 0 0 0 1 ← DIVIDEND
SHIFT AND SUBTRACT 1 1 1 0 ← DIVISOR X 2^3
1 2^3 +0 0 0 1 1 1 ← POSITIVE REMAINDER
SHIFT AND SUBTRACT 1 1 1 0 ← DIVISOR X 2^2
0 2^2 −0 1 1 1 ← NEGATIVE REMAINDER
RESTORING ADD 1 1 1 0 ← DIVISOR X 2^2
1 2^1 + 1 1 1 0 ← POSITIVE REMAINDER
SHIFT AND SUBTRACT 1 1 1 0 ← DIVISOR X 2^1
+ 0 0 0 0 ← ZERO REMAINDER
1 1 1 0 ← DIVISOR
0 2^0 − 1 1 1 0 ← NEGATIVE REMAINDER
RESTORING ADD 1 1 1 0 ← DIVISOR
0 0 0 0 ← FINAL REMAINDER

Figure 2-37

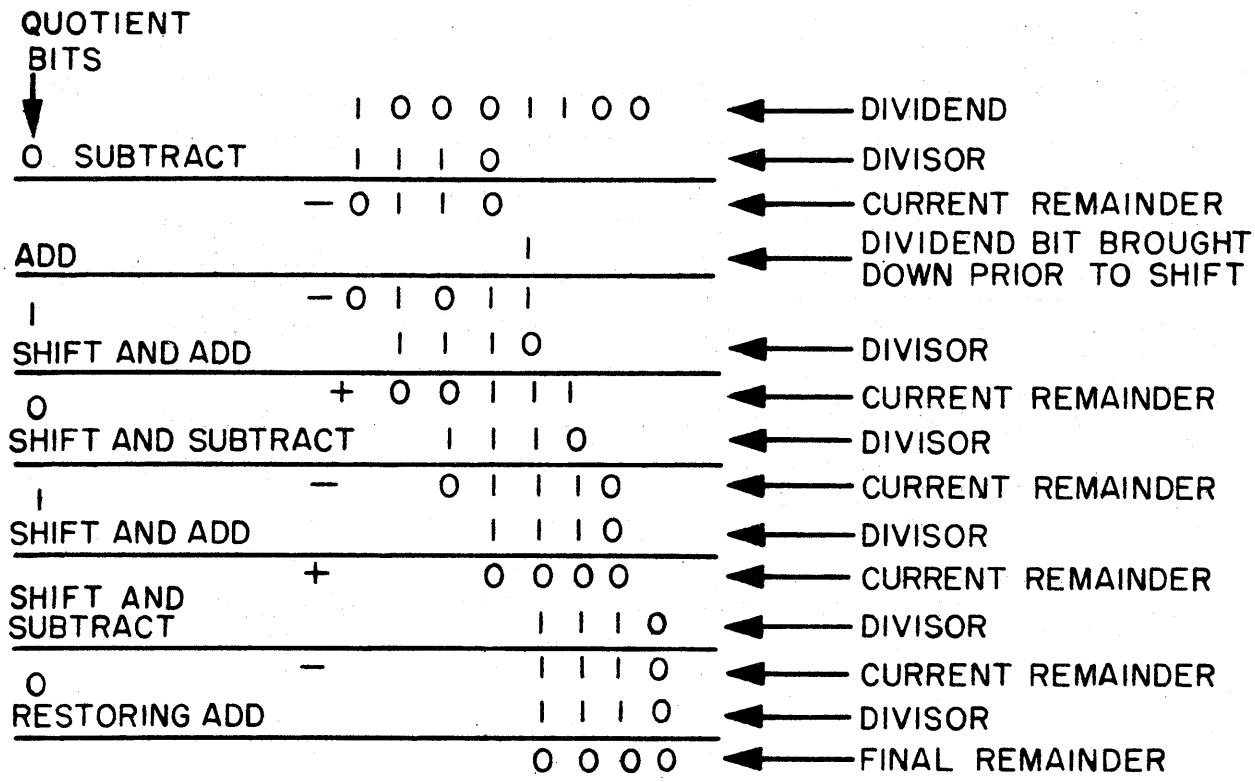

QUOTIENT BITS
1 0 0 0 1 1 0 0
DIVIDEND
0 SUBTRACT
1 1 1 0
DIVISOR
− 0 1 1 0
CURRENT REMAINDER
ADD
1
DIVIDEND BIT BROUGHT DOWN PRIOR TO SHIFT
− 0 1 0 1 1
1
SHIFT AND ADD
1 1 1 0
DIVISOR
+ 0 0 1 1 1
CURRENT REMAINDER
0
SHIFT AND SUBTRACT
1 1 1 0
DIVISOR
− 0 1 1 1 0
CURRENT REMAINDER
1
SHIFT AND ADD
1 1 1 0
DIVISOR
+ 0 0 0 0
CURRENT REMAINDER
SHIFT AND SUBTRACT
1 1 1 0
DIVISOR
− 1 1 1 0
CURRENT REMAINDER
0
RESTORING ADD
1 1 1 0
DIVISOR
0 0 0 0
FINAL REMAINDER

Figure 2-38

When it comes to translating the pencil and paper method of long division into a machine routine, the inspection function which is so readily performed by a human operator proves extremely difficult to mechanize. For this reason, division by a machine is sometimes limited to favorable cases where the complications are fewer. For example, division may be permissible only if the relative magnitudes of dividend and divisor are such that no error arises if the division routine is begun by lining up the most significant bits of dividend and divisor. Two division routines which begin in this manner are shown in Figures 2-37 and 2-38. These routines are analogoud to the two decimal routines shown in Figure 2-20. The difference is that in the binary routines, it is only necessary to subtract the divisor from the dividend or remainder once for each shift. This corresponds to the fact already noted that operating in the binary system eliminates the need for trial multiplications.

In the so-called restoring routine of Figure 2-37, an unsuccessful subtraction, that is one which produces a negative remainder is cancelled out by an addition before a shift right is executed. Referring to the figure, the routine begins with the divisor (1110) lined up left with the dividend. In this position the value of the divisor is multiplied by 2^4 as noted in the figure. When the divisor in this position is subtracted from the dividend, a negative remainder is obtained. Accordingly, a 0 is entered in the 2^4 order of the quotient (indicating that $1110_{(2)} \times 2^4$ cannot be subtracted from $10001100_{(2)}$).

At the same time the divisor, still lined up in the same position, is added to the dividend restoring it to its original magnitude. A shift right is now executed, and then the divisor is again subtracted from the dividend. The subtraction in this new position is successful (i.e. a positive remainder is obtained). Thus, a 1 is entered in the 2^3 order of the quotient (corresponding to the fact that $1110_{(2)} \times 2^3$ has just been subtracted from the dividend). The divisor is shifted right again and another subtraction is performed. This subtraction is unsuccessful; thus, a 0 is entered in the 2^2 order of the quotient and a restoring addition is performed. Another shift right is executed and another subtraction operation is performed. This operation is successful; thus, a 1 is entered in the 2^1 order of the quotient. A final shift and subtraction follows. This one is unsuccessful; thus, a 0 is entered in the 2^0 order of the quotient. A final restoring addition is performed to transform the remainder from a negative to a positive or a zero (as in the case of the example) balance.

The non-restoring routine of Figure 2-38 begins in the same way as the restoring routine of Figure 2-37; that is, the divisor is lined up left with the dividend and subtracted from it and a 0 is entered in the 2^4 order of the quotient to indicate that the subtraction is unsuccessful. However, at this point a right shift of the divisor is executed. The divisor is then added to the dividend. Since a right shift is equivalent to division by two, exactly half of what was removed by the subtraction is restored by the addition. Since, the current remainder is positive after the addition, 2^3 times the

divisor has successfully been subtracted from the dividend (by first subtracting 2^4 times the divisor and then adding 2^3 times the divisor). Thus, a 1 is entered in the 2^3 order of the quotient. The divisor is then shifted right again and subtracted from the current remainder. Since a negative remainder is obtained, a 0 is entered in the 2^2 order of the quotient. Another shift right of the divisor is executed and then another addition is performed. Since the current remainder is positive after this addition, a 1 is entered in the 2^1 order of the quotient indicating that 2^1 times the divisor has successfully been subtracted from the current remainder (by first subtracting 2^2 times the divisor and then adding 2^1 times the divisor). A final shift and a final subtraction are performed. Since this subtraction is unsuccessful, a 0 is entered in the 2^0 order of the quotient. Also, after this final unsuccessful subtraction a restoring addition is performed. This is done so that the final remainder will be positive or zero (as in the case of the example).

In the example of Figure 2-38, the steps alternate between shift-and-add and shift-and-subtract; however, this is only because the quotient bits happen to alternate between 0 and 1. The rule is that a shift-and-add step follows when the current remainder is negative while a shift-and-subtract step follows when the current remainder is positive.

As discussed in Section 3.2 of this chapter, subtraction is often performed by complementation and addition. In this case, negative remainders appear in complement form. The

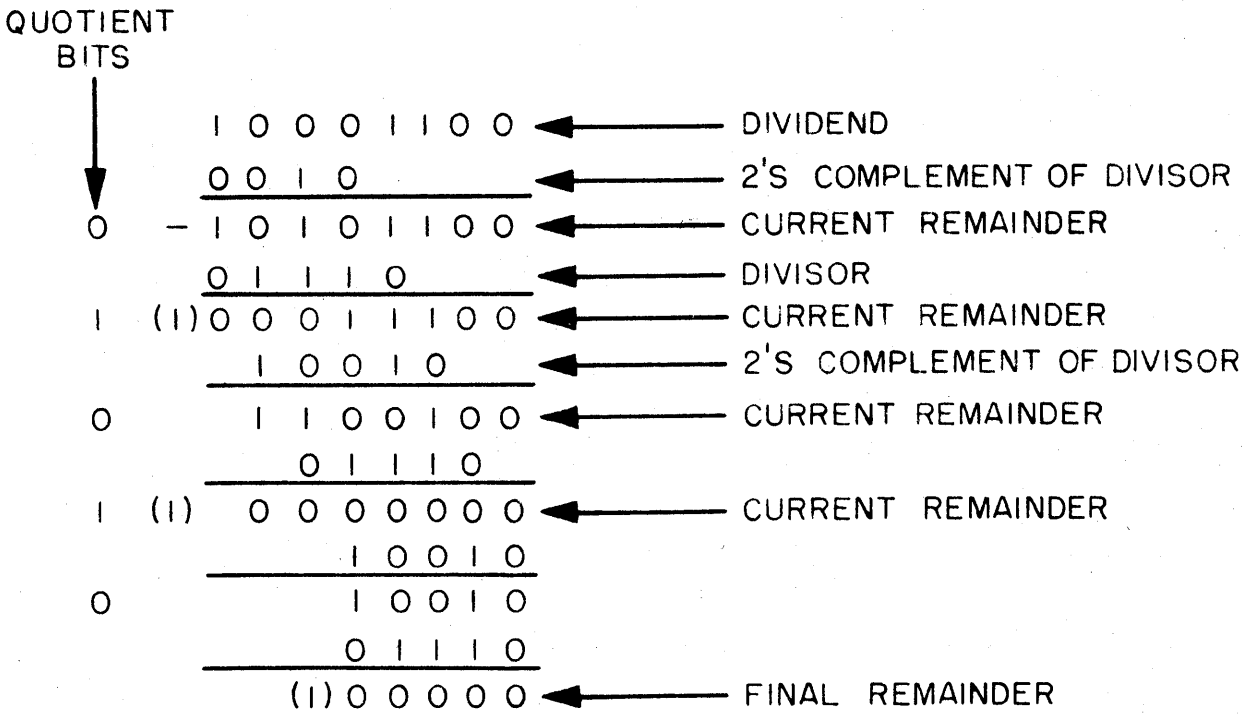

QUOTIENT BITS
1 0 0 0 1 1 0 0
DIVIDEND
0 0 1 0
2'S COMPLEMENT OF DIVISOR
0 - 1 0 1 0 1 1 0 0
CURRENT REMAINDER
0 1 1 1 0
DIVISOR
1 (1) 0 0 0 1 1 1 0 0
CURRENT REMAINDER
1 0 0 1 0
2'S COMPLEMENT OF DIVISOR
0 1 1 0 0 1 0 0
CURRENT REMAINDER
0 1 1 1 0
1 (1) 0 0 0 0 0 0 0 0
CURRENT REMAINDER
1 0 0 1 0
0 1 0 0 1 0
0 1 1 1 0
(1) 0 0 0 0 0
FINAL REMAINDER

Figure 2-39

routine of Figure 2-38 is repeated in Figure 2-39 with the subtractions handled by complementation and addition. Since the quotient in this problem is assumed to be an eight bit number while the divisor is only a four bit number some confusion may arise as to the modulus of the machine. In this regard, it should be noted that only five places are involved in each addition. Thus, in this routine modulo 2^5 addition is performed so that the carries shown in brackets are lost.

Part 2

CHAPTER 4

OCTONARY ARITHMETIC

4.1 GENERAL

The octonary number system is of interest in connection with computer work chiefly because of the relationship it bears to the binary system. This relationship can be stated as follows: since eight is the third power of two, three places in binary notation correspond to one place in octonary notation. Thus 111, which is the largest number that can be represented by three binary places, corresponds to 7, which is the largest number that can be represented by one octonary place.

The relationship between binary and octonary notation makes conversions between the two extremely simple. When conversion from octonary notation to binary notation is required, each octonary digit is replaced by the three bits which represent the equal value in binary notation. When conversion is from binary notation to octonary notation, the bits are considered in groups of three and each group is replaced by the octonary digit which represents the equivalent value. For example, 010101110 (binary) is separated into 010 101 and 110. Each of these groups is then replaced by the equivalent octonary digit to form 256 (octonary). The decimal representation of this result (174) cannot be generated by any such straightforward procedure.

Octonary numbers are much easier for human operators to work with than are binary numbers. For one thing, any given

whole octonary number has less places than the equivalent binary number. For another thing, the use of eight symbols (0,1,2,3,4,5,6,7) rather than just two (0,1) makes for a number language which is easier to read and write, and thus decreases the probability of error. These comparative advantages of the octonary number system coupled with simplicity of the conversion between octonary and binary notation, make the octonary system a good working notation for human operators who are preparing sets of numbers which eventually must be entered into a computer. For example, as is explained in Part 5, a computer solves problems by executing a sequence of instruction steps which must be entered into it in an essentially numerical form. The preparation of such a set of steps, called a program, is a task which might very well be performed by programmers using the octonary system.

If the octonary system is to be used to perform tasks such as the one just mentioned, it is convenient to be able to perform the four arithmetic operations upon octonary numbers. For this reason, a discussion of octonary arithmetic is presented in the sections which follow. It will be noted that this discussion varies from the ones involving decimal arithmetic and binary arithmetic, which appear in earlier chapters, in that it is concerned only with hand calculation techniques. This corresponds to the use of the octonary number system, in-so-far as computer work is concerned, as a working system for human operators.

OCTONARY NOTATION

8^3 8^2 8^1 8^0

(3) (2)

2 5 7

3 6 2

4 4 1

7 5 7

(2) 1 2 2

2 4 0 3

EQUIVALENT DECIMAL NOTATION

10^3 10^2 10^1 10^0

1 7 5

2 4 2

2 8 9

4 9 5

8 2

1 2 8 3

Figure 2-40

4.2 OCTONARY ADDITION

There are several ways of accomplishing the addition of a column of octonary numbers.

One method of adding a column of octonary numbers is to form the decimal sum in each order, divide this sum by eight, enter the quotient as the carry in the next higher order and enter the remainder as the sum digit in the order being operated upon. This amounts to operating in the decimal system and then converting to the octonary system on an order by order basis. An example may serve to clarify this method. Figure 2-40 compares the addition of a column of octonary numbers with the addition of the column of equivalent decimal numbers. If the method just described is applied to the addition of the octonary numbers, the operation proceeds as follows: Adding the digits in the units order, a decimal sum of nineteen is obtained. When this is divided by eight, a quotient of 2 and a remainder of 3 are obtained. Accordingly, 2 is entered as a carry above the 8^1 column and 3 is entered below the units column. The digits of the 8^1 column (including the carry) are now added and the decimal sum of twenty-four is obtained. This is divided by eight yielding a quotient of 3 and a remainder of 0.

ADDEND \ AUGEND	0	1	2	3	4	5	6	7
	SUM							
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Figure 2-41

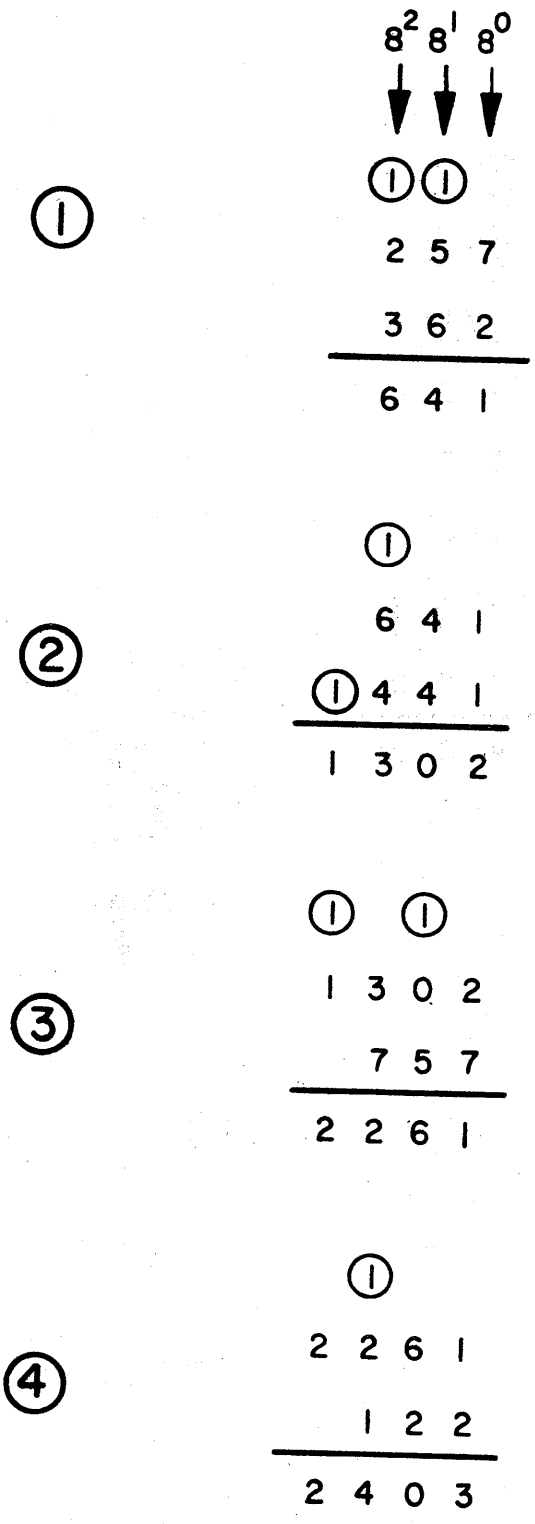

8² 8¹ 8⁰
①
1 1
2 5 7
3 6 2
6 4 1
②
1
6 4 1
1 4 4 1
1 3 0 2
③
1 1
1 3 0 2
7 5 7
2 2 6 1
④
1
2 2 6 1
1 2 2
2 4 0 3

Figure 2-42

Accordingly, 3 is entered as a carry above the 8^2 column and 0 is entered below the 8^1 column. The digits of the 8^2 column (including the carry) are now added and the decimal sum of twenty is obtained. This is divided by eight, yielding a quotient of 2 and a remainder of 4. Accordingly, 2 is carried to the 8^3 column and 4 is entered below the 8^2 column. Finally, the 2 carried to the 8^3 column is brought down as the sum digit of that column.

A second method of adding a column of octonary numbers involves reference to an octonary addition table. The most straightforward method of using this addition table, which is illustrated in Figure 2-41 is to imitate the machine method of adding a column of numbers; that is to add the second number to the first, then add the third number to the sum of the first two and so on. This technique is illustrated in Figure 2-42 which shows the addition of the same set of numbers that appeared in Figure 2-40. The operation proceeds as follows: The addition of the first two numbers begins by consulting the table (Figure 2-41) to find the sum of 2 and 7 which is 11. The units digit of this sum is entered below the units column and the 8^1 digit is entered as a carry above the 8^1 column. This carry is then added to the 5 in the 8^1 column (in accordance with the table) and the resulting 6 is added to the 6 already in the column This addition of the two 6's is handled by reference to the table which lists the sum 14. The units digit of this sum is entered below the 8^1 column and the 8^1 digit is entered as a carry above the 8^2 column.

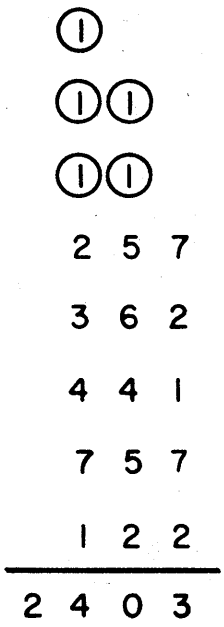


Figure 2-43

This carry is added to the two digits in the 8^2 column (still by reference to the table) producing the sum digit 6. The third number is then added to the sum of the first two that has just been formed, and so on until the entire column has been totalled. Notice that all sums less than or equal to 7 are already known by the human operator since they are identical to the sums found in the decimal addition table. This fact considerably reduces the difficulty of working with this new table.

A second method of using the octonary addition table is shown in Figure 2-43. Here the entire column is added on an order by order basis. Whenever an addition produces a two-digit sum, the 8^1 digit of that sum is entered as a carry in the next higher column. For example, starting at the bottom, the 8^0 order addition in Figure 2-43 proceeds as follows: Reference to the table reveals that the sum of 2 and 7 is 11. The 8^1 digit of this result is entered as a carry in the 8^1 order. The 8^0 digit of the result, on the other hand, is added to the next digit in the 8^0 order (1) producing the sum 2. This sum is added to the next digit (2) producing the sum 4. This sum is added to the next digit (7) producing the sum 13 of which the 1 is carried to the 8^1 column and the 3 is entered below the 8^0 column.

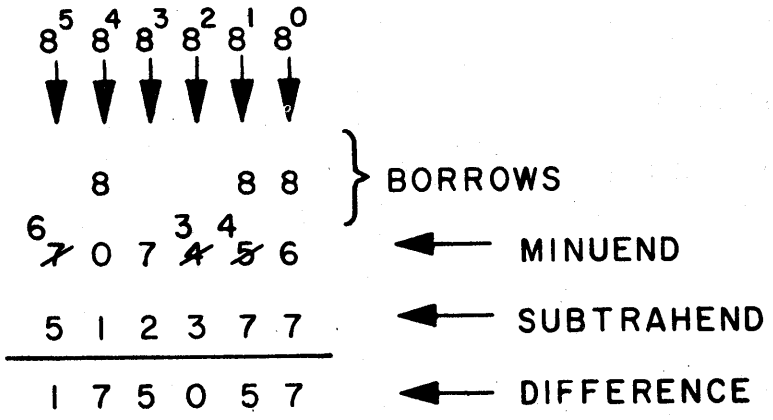

8^5 8^4 8^3 8^2 8^1 8^0
8 8 8
BORROWS
6 3 4
7 0 7 4 5 6
MINUEND
5 1 2 3 7 7
SUBTRAHEND
1 7 5 0 5 7
DIFFERENCE

Figure 2-44

4.3 OCTONARY SUBTRACTION

Octonary subtraction presents few problems to the human operator. If a minuend digit which is smaller than the corresponding subtrahend digit is encountered eight is added to it (rather than ten as in the case of the decimal system) and the subtrahend digit is subtracted from the result. At the same time the minuend digit in the next higher order is decreased by 1. The required addition of eight and the subsequent subtraction are performed by reference to the decimal addition table although no decimal notation need be written down during the process. A sample subtraction is shown in Figure 2-44. For clarification the borrows are shown in this case and, since the borrow part of the operation is performed in the decimal system, they are shown in decimal notation. The subtraction proceeds as follows: The units (8^0) order is inspected and the minuend digit is seen to be smaller than the subtrahend digit. Accordingly, eight is borrowed from the 8^1 order diminishing the minuend digit in this order from 5 to 4. The borrowed 8 is now added to the 6 in the minuend order (by decimal addition)and 7 is subtracted from the result (by decimal subtraction). The subtraction through each succeeding order proceeds in the same manner. In summary, the only difference between this routine and a decimal subtraction is that a 1 borrowed from the nth order has the value of eight (rather than ten) in the (n-1)th order. If the borrowed eights were shown in octonary notation they would appear, as does the radix in any number system, as 10's. In this case, the subtraction could be performed by referring to the octonary addition table to see what digit would

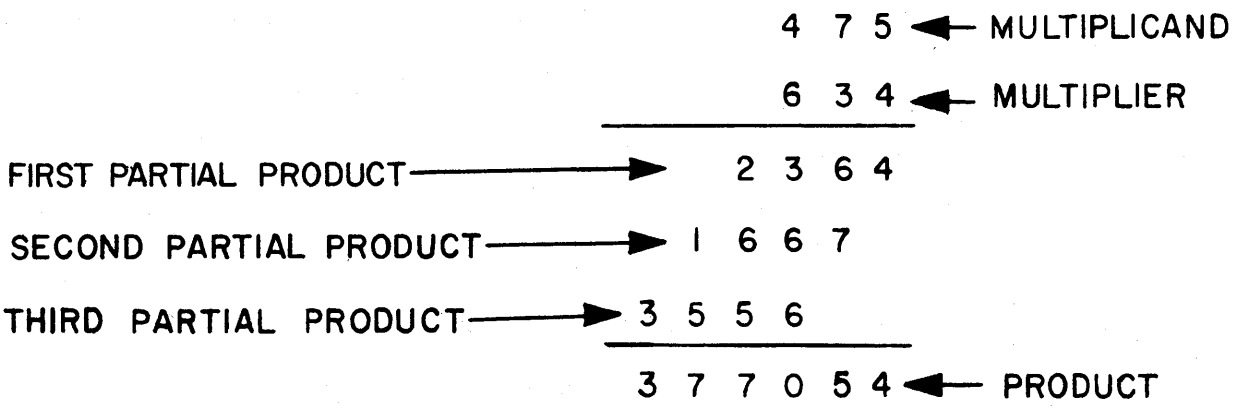

4 7 5 ← MULTIPLICAND
6 3 4 ← MULTIPLIER
FIRST PARTIAL PRODUCT → 2 3 6 4
SECOND PARTIAL PRODUCT → 1 6 6 7
THIRD PARTIAL PRODUCT → 3 5 5 6
3 7 7 0 5 4 ← PRODUCT

Figure 2-45

have to be added to the subtrahend digit in order to form 10 + the minuend digit. However, such an approach probably only increases the change of error.

4.4 OCTONARY MULTIPLICATION

The multiplication of octonary numbers can be performed by an application of any of the three basic approaches which are discussed in Section 3.2 above with respect to addition; i.e. numbers can be converted to the decimal system and multiplication performed in that system, digit multiplications can be performed in the decimal system and digit products mentally converted to the octonary system through division by eight, or the entire operation can be performed by reference to an octonary multiplication table. The last of these methods is probably the most straightforward. However, the second method may conveniently be used to obtain a second solution as a check against the first. Either of the second two methods would appear, on paper, as shown in Figure 2-45.

If the second method were used, the reasoning of the operator would be somewhat as follows: Four times five is twenty, which is two eights plus four. Thus, enter eight and carry two. Four times seven is twenty-eight which is three eights plus four. Thus, add four to the previous carry of two and enter the resulting six. Carry three. Four times four is sixteen which is two eights plus zero. Enter the previous carry of three and enter two in the next higher order.

If, on the other hand, the third method were used, the sums and carries resulting from each digit multiplication would be obtained directly from the octonary multiplication table.

MULTIPLIER \ MULTIPLICAND	0	1	2	3	4	5	6	7
MULTIPLIER	PRODUCT							
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	46
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Figure 2-46

```
          1.2 4 4
      ------------
573 ) 7 6 4.6 3 1
      5 7 3
      -----
      1 7 1 6
      1 3 6 6
      -------
        3 3 0 3
        2 7 5 4
        -------
          3 2 7 1
          2 7 5 4
          -------
            3 1 5
```

Figure 2-47

This table is given for reference in Figure 2-46.

4.5 OCTONARY DIVISION

Like decimal division and unlike binary division, octonary division involves trial multiplications. The easiest way to perform these is by reference to the octonary multiplication table of Figure 2-46. The octonary long division setup looks very much like the decimal setup as can be seen by referring to the sample division of Figure 2-47. The only difference lies in the way the trial multiplications and the subtractions are performed. As has been stated, the multiplications are most easily performed by reference to the octonary table. The subtractions can be performed as explained in Section 3.3 above.

PART 2

CHAPTER 5

RADIX CONVERSION

5.1 GENERAL

The decimal number system, because of its familiarity, is likely to remain as the principle number language of the human operator. At the same time, by virtue of the simplifications it offers from the point of view of mechanization, the binary number system is likely to continue as the most generally used number language of machines. The octonary system, which does not offer too many difficulties from the point of view of manipulation by the human operator and which bears such a close relationship to the binary system that conversions between the two are simple, is likely to find increasing use as a middle ground between the other two systems. Since decimal, binary and octonary systems each have their uses in computer work, the question of conversions between them is of primary interest.

The conversions between binary and octonary notation, which are extremely simple, are discussed in Section 4-1 of this Part. However, for convenience the information is repeated here as follows:

a. To convert any binary number to its octonary equivalent, first write the number in terms of groups of three bits, proceeding left from the binary point for the integral portion of the number and right from the binary point for the fractional portion of the number.

POWER OF 2	VALUE IN DECIMAL NOTATION	POWER OF 2	VALUE IN DECIMAL NOTATION
20	1048575	5	32
19	524288	4	16
18	262144	3	8
17	131072	2	4
16	65536	1	2
15	32768	0	1
14	16384	-1	.5
13	8192	-2	.25
12	4096	-3	.125
11	2048	-4	.0625
10	1024	-5	.03125
9	512	-6	.015625
8	256	-7	.0078125
7	128	-8	.00390625
6	64	-9	.001953125
		-10	.0009765625

Figure 2-48 Decimal representation of powers of 2

Replace each group of three bits by the octonary digit of equivalent value. For example, to convert 1001101101010.1101 to octonary form handle it in the following groupings;
001 001 101 101 010. 110 100
Now replace each of these groupings by its octonary equivalent as follows: 11552.64.

b. To convert any octonary number to its binary equivalent, replace each octonary digit by the grouping of three binary bits having equivalent value. For example, re-write 56473.246 as follows:

101 110 100 111 011. 010 100 110

Here the groupings are separated just to call attention to the equivalences. In actual practice there is no reason why the binary number cannot be written directly with no spacing between groupings unless it is desired to retain groupings to facilitate checking.

The simplicity of the conversion between octonary and binary numbers results from the fact that eight is the third power of two. The other conversions that are of interest, that is those between decimal and binary notation and those between decimal and octonary notation are not as easily performed. These are explained in the following sections.

5.2 CONVERSIONS BETWEEN DECIMAL AND BINARY NOTATION

5.2.1 Decimal to Binary Conversion

The most straightforward method of converting decimal numbers into binary form involves making use of a table of powers of two (in decimal notation) such as is illustrated in Figure 2-48.

2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
1	0	1	1	1	1	0	1	0	1	0	0.0	1	0	1	1	1	

3028.359375	
2048	25
980	109375
512	0625
468	046875
256	03125
212	015625
128	015625
84	0000000
64	
20	
16	
4	

Figure 2-49

A conversion performed making use of the table of Figure 2-48 is shown in Figure 2-49. The decimal number being converted in this example is 3028.359375. The equivalent binary number appears at the top of the figure, below the sequence of powers of two. The conversion proceeds as follows:

a. The integral portion of the decimal number is considered first. The table of Figure 2-48 is examined in order to find the largest power of two which is smaller than or equal to the integral portion of the decimal number. In this case that power is 2^{11} which in decimal notation is 2048. A sequence of decreasing powers of two, starting with 2^{11} and ending with 2^{0} is written at the top of the page. A 1 is entered under 2^{11} corresponding to the fact that 2^{11} is the largest power of two which is smaller than or equal to the integral portion of the decimal number.

b. The decimal form of 2^{11} (i.e. 2048) is subtracted from 3028 yielding a difference of 980. The table of Figure 2-48 is now inspected a second time to find the largest power of two which is smaller than or equal to 980. This is 2^{9} (i.e. 512). Accordingly, a 1 is entered under 2^{9} and 512 is subtracted from 980. This subtraction yields 468. The table of Figure 2-48 is now inspected a third time to find the largest power of two smaller than or equal to 468. This is 2^{8} (i.e. 256). Accordingly, a 1 is entered under 2^{8} and 256 is subtracted from 468.

c. The conversion process continues in the same manner until a difference of 1 or 0 is obtained. This difference is then entered under the 2^0. At this point, 0's can be entered beneath the powers of two which did not enter into the conversion process. For example, 2^{10} did not qualify as the largest power of two smaller than or equal to the difference at any step of the process and therefore a 0 is entered beneath it. It may be preferable to enter these 0's as the process develops. Thus, as soon as it is seen that 2^9 is the largest power of two smaller than or equal to 980, a 0 may be entered under 2^{10}.

d. The table of Figure 2-48 is now examined to find the largest negative power of two which is smaller than or equal to the fractional portion of the decimal number (.359375). This is 2^{-2} (i.e. 25). Accordingly, a 1 is entered in the 2^{-2} order and .25 is subtracted from .359375 yielding .109375. The table is examined again to find the largest negative power of two which is smaller than or equal to .109375. This is 2^{-4} (i.e. 0625). Accordingly, a 1 is entered in the 2^{-4} order and .0625 is subtracted from .109375. The process continues in the same manner until a 0 difference is obtained (which is the case in the example of Figure 2-49) or the required number of significant bits have been generated.

The conversion process that has just been described can be summarized as follows: The decimal number is separated into components which are powers of two. The sum of these powers of two is then written in binary notation.

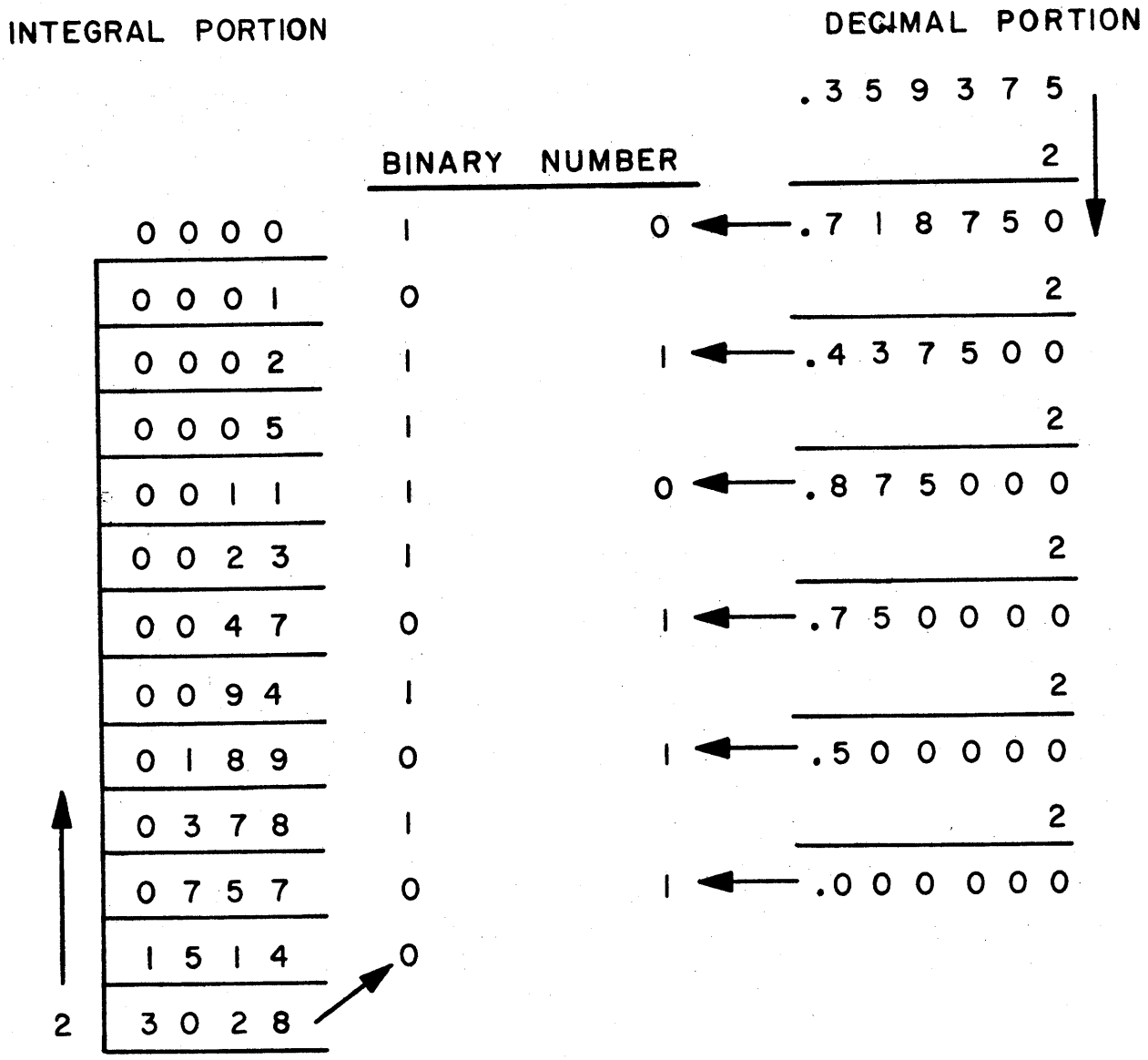

INTEGRAL PORTION
DECIMAL PORTION
.359375
BINARY NUMBER
2
0000
1
0
.718750
0001
0
2
0002
1
1
.437500
0005
1
2
0011
1
0
.875000
0023
1
2
0047
0
1
.750000
0094
1
2
0189
0
1
.500000
0378
1
2
0757
0
1
.000000
1514
0
2
3028

Figure 2-50

Step d. of the above conversion procedure reveals a complication which is, in general, encountered in converting fractional numbers from one system to another; i.e. a fractional number which can be represented by a finite number of digits in one number systed cannot, necessarily, be represented by a finite number of digits in a second number system. In fact, any fraction whose denominator contains a prime factor which is not a factor of the radix of a given number system cannot be represented by a finite number of digits in that number system (except of course by the use of a fraction bar). Thus, the conversion process, which is essentially a division process, cannot always be finished. However, since any conversion can be carried out to the required number of places, this is not too serious a difficulty.

A second method of converting from decimal to binary notation, which does not require any reference to a table of powers, is shown in Figure 2-50, for the same decimal number as before, In this method, the integral portion of the decimal number is repeatedly divided by 2 until it has been exhausted. For each division in which a remainder arises, a 1 is entered as a bit of the equivalent binary number. For each division in which no remainder arises a 0 is entered. The first division generates the 2^0 bit, the second generates the 2^1 bit and so on. Thus the integral portion of the binary number is read in an order opposite to that in which it is generated. In Figure 2-50, for example, the bits are written starting at the bottom of the sheet and proceeding upward. However, the number reads (in order of decreasing magnitude) from top to bottom.

The method of Figure 2-50 can be justified as follows: If a number is odd (i.e. if there is a remainder when it is divided by two) then its binary representation must contain a 1 in the 2^0 order. Moreover, half of any binary number, n, can be obtained by shifting each bit of n one order to the right. Thus if n/2 is odd, the binary representation of n must contain a 1 in the 2^1 order (for otherwise there will not be a 1 in the 2^0 order after a shift to the right). By the same reasoning, if n/4 is odd, then there must be a 1 in the 2^2 order of n and so on. Thus the repeated division process is merely a test to discover first if n is odd, then if n/2 is odd, then if n/4 is odd and so on.

The conversion of the fractional portion of the number has to be handled in a different manner. The fraction is repeatedly multiplied by two. Each time that the multiplication generates a product having an integral portion, a 1 is entered as a bit of the equivalent binary fraction. Each time that the product has no integral portion a 0 is entered. This process is continued until the required number of binary bits have been generated or until a product which is entirely integral is generated (as in Figure 2-50) in which case an exact conversion has been obtained. In this fractional part of the conversion process, the digits are generated in order of decreasing significance.

BINARY NO	POWER OF 2	VALUE OF POWER IN DECIMAL NOTATION	
1	11	2048	
0	10		
1	9	512	
1	8	256	
1	7	128	
1	6	64	
0	5		
1	4	16	
0	3		
1	2	4	
0	1	3028	← INTEGRAL PORTION
0	0		
•			
0	-1		
1	-2	.25	
0	-3		
1	-4	.0625	
1	-5	.03125	
1	-6	.015625	
		.359375	← FRACTIONAL PORTION

Figure 2-51

Notice that at each step, the integral portion of the product, if any, is discarded.

The justification of the handling of the fractional portion of the conversion is very similar to the justification of the handling of the integral portion. Multiplication of a number by two, causes each bit of its binary representation to be shifted 1 place to the left. Thus, if some binary fraction, n, has an integral portion, then n must have a 1 in its 2^{-1} order. By a continuation of the same reasoning, if twice the fractional portion of 2n has an integral portion then the 2^{-2} order of n must contain a 1, and so on.

5.2.2 Binary to Decimal Conversion

Binary numbers can be converted into decimal form by means of the table of powers of two of Figure 2-48. The procedure for this method of conversion is illustrated in Figure 2-51. Here, the binary number is entered at the left of the page from top to bottom in order of decreasing significance. The power of two associated with each bit of the number is then entered next to it in order to facilitate reference to the table of Figure 2-48. For each binary bit which is a 1, the decimal value of the corresponding power of 2, as obtained from the table, is entered as a component of the decimal form of the number. When these components have all been entered, they are summed to obtain the complete decimal form. (In Figure 2-51, the integral and decimal portions are shown separately so as to clarify the procedure.)

INTEGRAL PORTION

BINARY NO.	ADD	MULTIPLY BY 2
1		2
0	2	4
1	5	10
1	11	22
1	23	46
1	47	94
0	94	188
1	189	378
0	378	756
1	757	1514
0	1514	3028
0	3028	

FRACTIONAL PORTION

	ADD	DIVIDE BY 2
1		.5
1	1.5	.75
1	1.75	.875
0	.875	.4375
1	1.4375	.71875
0	.71875	.359375

Figure 2-52

POWER OF 8	VALUE IN DECIMAL NOTATION
6	262144
5	32768
4	4096
3	512
2	64
1	8
0	1
-1	.125
-2	.015625
-3	.00195325

Figure 2-53 Decimal Representation of Powers of Eight

8^3	8^2	8^1	8^0	.	8^{-1}	8^{-2}	8^{-3}	8^{-4}
5	7	2	4	.	2	7	0	0

```
3 0 2 8 . 3 5 9 3 7 5

2 5 6 0   . 2 5 0
-------   -------
  4 6 8   . 1 0 9 3 7 5
  4 4 8   . 1 0 9 3 7 5
  -----   -------------
    2 0     0 0 0 0 0 0

    1 6
    ---
      4
      4
    ---
      0
```

Figure 2-54

A second method of conversion from binary to decimal notation, which does not require the use of a table of powers, is shown in Figure 2-52. The procedure followed in this method is as follows:

The integral portion of the binary number is entered at the left of the page from top to bottom in order of decreasing significance. The most significant bit is multiplied by two and the decimal notation of the product is entered in the righthand column. This result is then added to the second most significant bit of the binary number (which in the example is 0) and the sum is entered in the center column. This sum is then multiplied by two and the product is entered in the righthand column. This second product is entered in the righthand column. This second product is added to the third most significant bit of the binary number and the resulting sum is multiplied by two, the product being entered in the righthand column. This process continues in the same manner until the 2^0 bit of the binary number has been added to the product of the preceding step. This sum is the integral portion of the equivalent decimal number. The process is just an ingenious method of generating the sum of the powers of two represented by the 1's of the binary number.

5.3. CONVERSIONS BETWEEN DECIMAL AND OCTONARY NOTATION

5.3.1. Decimal to octonary conversion

Decimal numbers can be converted into octonary notation by a method which makes use of the table of powers of eight (in decimal notation) illustrated in Figure 2-53. A conversion by this method is shown in Figure 2-54. This conversion is similar to the decimal to binary conversion shown in Figure 2-49.

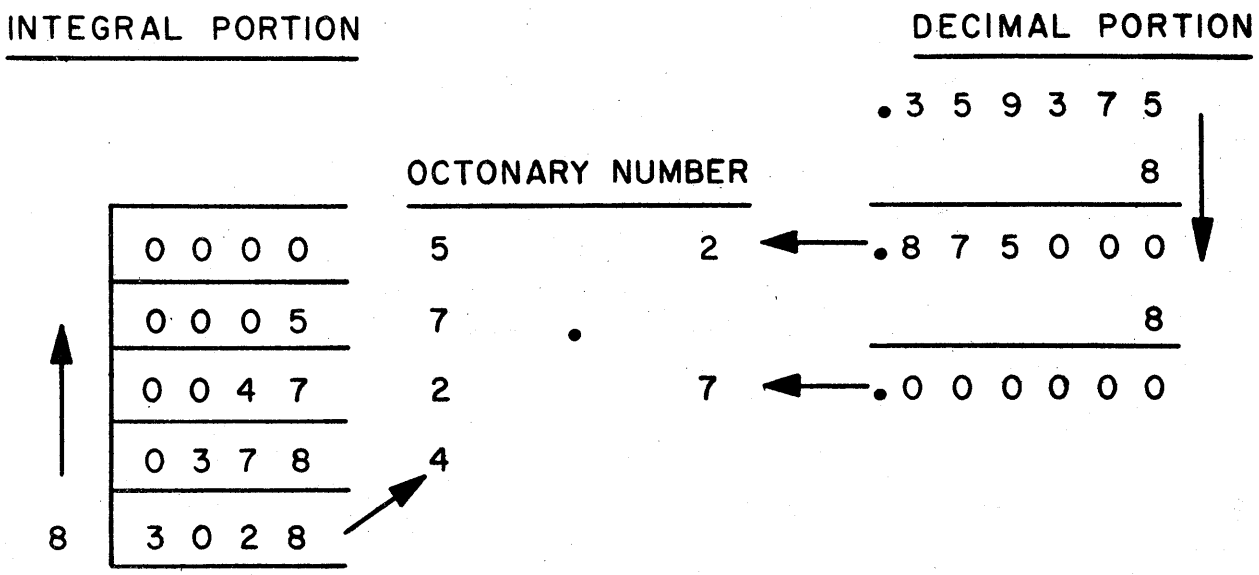

INTEGRAL PORTION
DECIMAL PORTION
.3 5 9 3 7 5
8
OCTONARY NUMBER
0 0 0 0
5
2
.8 7 5 0 0 0
0 0 0 5
7
8
0 0 4 7
2
7
.0 0 0 0 0 0
0 3 7 8
4
8
3 0 2 8

Figure 2-55

Thus the conversion begins by examining the table of powers of eight for the largest power which is smaller than or equal to the integral portion of the decimal number. However, the next step of the decimal to octonary conversion has no counterpart in the decimal to binary conversion. This step involves multiplying the selected power of eight by the largest integral factor which produces a product smaller than or equal to the integral portion of the decimal number. In the example of Figure 2-54 the largest power of eight which is smaller than or equal to the integral portion of the decimal number is 8^3 (i.e. 512). The largest multiple of 512 which is smaller than or equal to the integral portion of the decimal number is 5X512= 2560. Accordingly, a 5 is entered below 8^3 (in a sequence of powers of eight which is noted down at the top of the page) and 2560 is subtracted from 3028. The process continues in a manner analogous to that of the decimal to binary conversion of Figure 2-49 except that at each step the extra operation of finding the largest multiple of the selected power of eight must be performed. Incidentally, the decimal number in the conversion example of Figure 2-54 is the same as in Figure 2-49 It is an interesting exercise for the reader to convert the octonary number obtained in Figure 2-54 into the binary number obtained in Figure 2-49. This is accomplished by the method described in Section 5.1 of this chapter.

A second method of decimal to octonary conversion, illustrated in Figure 2-55 is analogous to the second method of decimal to binary conversion shown in Figure 2-50.

OCTONARY NUMBER	POWER OF 8	VALUE OF POWER	VALUE OF POWER IN DECIMAL NOTATION POWER VALUE X OCTONARY NO.
5	3	5 1 2	2 5 6 0
7	2	6 4	4 4 8
2	1	8	1 6
4	0	1	4
		INTEGRAL PORTION →	3 0 2 8

2	− 1	• 1 2 5	• 2 5 0
7	− 2	• 0 1 5 6 2 5	• 1 0 9 3 7 5
		FRACTIONAL PORTION →	• 3 5 9 3 7 5

Figure 2-56

The integral portion of the decimal number is repeatedly divided by eight until it is exhausted, and the remainders at each step are used as the digits of the octonary number. The fractional portion of the decimal number is repeatedly multiplied by eight until a product having no fractional part is obtained or until the required number of octonary digits have been generated. Integral portions of each product are used as the digits of the octonary number. This method can be justified in terms of the relationship between shifts and multiplication by the radix, by reasoning similar to that used to justify the analogous decimal to binary conversion process (See Section 5.2.1 above).

5.3.2 Octonary to Decimal Conversion

An octonary number can be converted to a decimal number by a method which makes use of the table of powers of eight presented in Figure 2-53. An example of such a conversion is shown in Figure 2-56. The conversion of the example proceeds as follows:

a. The octonary number is entered at the left-hand side of the page in order of decreasing significance from top to bottom. The decimal form of each power of eight associated with a non-zero digit of the octonary number is obtained from the table of Figure 2-53 and is entered to the right of that digit. Each such power of eight is multiplied by the digit of the corresponding power and the resulting products are entered in the righthand column. This column is added, yielding the decimal equivalent of the octonary number.

INTEGRAL PORTION

OCTONARY NO.	ADD	MULTIPLY BY 8
5		4 0
7	4 7	3 7 6
2	3 7 8	3 0 2 4
4	3 0 2 8	

FRACTIONAL PORTION

	ADD	DIVIDE BY 8
7		. 8 7 5
2	2 . 8 7 5	. 3 5 9 3 7 5

Figure 2-57

For example, in Figure 2-56, the first non-zero digit of the octonary number is a 5 which is found in the 8^3 order of the number. Thus this 5 is multiplied by the decimal representation of 8^3 (i.e. 512) and the product (2560) is entered in the right-hand column.

A second method of converting an octonary number to decimal form is analogous to the second method of converting a binary number to decimal form which is shown in Figure 2-52. In this method, an example of which is shown in Figure 2-57, the integral portion of the octonary number is entered in order of decreasing magnitude at the lefthand side of the page. The most significant digit is then multiplied by eight. The resulting product is added to the second most significant digit. The sum of this addition is then multiplied by eight and that product is added to the third most significant digit. This process continues until the 8^0 order digit is added to the product of the preceding step. The resultant sum is the decimal representation of the integral portion of the octonary number.

The digits of the fractional portion of the octonary number are now listed in order of increasing significance. The most significant fractional digit is divided by eight and the resulting quotient is added to the next most significant fractional digit. This sum is then divided by eight and the resulting quotient is added to the next most significant digit. This process continues until the least significant digit of the octonary number has been added and the resulting sum has been divided by eight. The quotient that results from this division is the decimal representation of the fractional portion of the octonary number.

PART 2

CHAPTER 6

PRECISION AND ACCURACY

6.1 GENERAL

Precision and accuracy are words that are often used interchangeably by the layman. However, when used as technical terms they have entirely different meanings. It is more convenient to define precision and accuracy in terms of their opposites than directly. If information is not precise, it is not very exact but it is not necessarily incorrect. For example, if the time is 10:07, and someone says that it is about 10 o'clock he is not being very precise. However, he is being perfectly accurate; for inaccuracy implies misleading information and there is nothing misleading about saying that it is about 10 o'clock if it is 10:07. However, the statement that it is 10:06, while it is more precise, is inaccurate (if it is actually 10:07). In terms of an item of data which is specified numerically, precision is defined by the number of digits or bits used. For example, 567 is more precise than 560. Accuracy on the other hand is defined in terms of the correctness of the digits which are used. A quantity which is represented by four decimal digits is said to be specified to a precision of 1 part in 10,000. This corresponds to the fact that 10,000 distinct numbers (0000 through 9999) can be represented by four decimal digits. If an accuracy to within ±.01% is specified in connection with a precision of 1 part in 10,000, this indicates that the units

digit of the number may be incorrect by 1 unit. Thus the number 4047 would imply a quantity lying somewhere between 4048 and 4046.

A quantity specified to a precision of 1 part in 10,000 by a decimal number can also be said to be specified to four significant places. In this connection, it is important to understand the concept of significance. Throughout this Part the term significance has been used freely to designate the relation between individual digits or bits of a number. In this sense, if digit x is to the left of digit y, it is more significant than digit y. In addition to this concept of relative significance, there is a concept of absolute significance. In terms of this second concept, a digit in a particular number is said to be either significant or not significant. In order to qualify as being significant in this sense, a digit must add to the precision with which a quantity is specified. For example, in the statement, "He is about 50 years old," the 0 is not siginificant since its purpose is merely to indicate that the 5 is to be associated with the 10^1 rather than 10^0 order. A somewhat different case of insignificance arises in the multiplication of two four place numbers which are accurate to ±.01%. Here, the units digits of both numbers may be off by as much as 1 unit in either direction. For example, if 5031 is multiplied by 1722 a product of 8663382 is generated. However, the specified accuracy is such that the correct values of multiplier and multiplicand may be as small as 5030 and 1721 or may be as

large as 5032 and 1723. Thus the correct value of the product may lie anywhere in the range between 8656630 (i.e. 5030 x 1721) and 8670136 (i.e. 5032 x 1723). For this reason the right-hand four digits of the product are not significant. The fifth digit from the right, on the other hand, is significant since it specifies the approximate center of the range which contains the correct product. After a calculation of this sort is completed the result must be rounded off; i.e. those digits which seem to add something to the precision of the result but actually are meaningless must be removed. For purpose of round off, the digit to the right of the least significant place does have some value, for if it is 5 or more the least significant digit should be raised 1 unit. Incidentally, when binary notation is used, a 1 in the bit position to the right of the least significant place has a value which is equivalent to the value of a 5 to the right of the least significant place in decimal notation. Thus in rounding off a binary number, a 1 is added into the least significant bit position if the position to the right contains a 1 but not if the position to the right contains a 0.

6.2 SCALING

The problem of scaling the variables which are to be operated upon in a digital computer solution is closely related to the concept of precision. Assume that a computer has a decimal modulus of 10,000. Then if a variable can be scaled so that its range falls exactly within the capacity of the

machine, it can be represented to a precision of 1 part in 10,000. If, on the other hand, it is scaled so that its maximum value is represented by just three orders of the machine, then it is only being represented to a precision of 1 part in 1000. In other words, there are two objects to keep in mind when scaling variables for a computer solution. One is to scale the variables so that they do not exceed the capacity of the machine, for if this happens meaningless results will be obtained. However, the other is to scale the variables so as to use as much of the capacity of the machine as is possible in order to obtain the maximum precision. The extent to which the full capacity of the machine can be used to represent a particular variable, depends upon the exactitude with which the range of the variable is known.

Suppose that wind velocity is to be scaled for representation in a computer and it is known that no velocity as large as 40 knots will be encountered in a particular problem. Then 40 knots can be made exactly equal to the largest number which the machine can represent. If the machine represents only fractions, then 40 knots can be equated to 1 machine unit of wind. Thus a velocity of 20 knots will appear in the machine as .5.