INSTRUCTION
MANUAL

DIGI-COMP II®

MECHANICAL
BINARY
DIGITAL
COMPUTER

*It's more fun than you think
when you think for fun*

EDUCATION · SCIENCE · RESEARCH
ESR
INCORPORATED

TABLE OF CONTENTS

FOREWORD

HOW TO USE THIS INSTRUCTION MANUAL

There are, of course, many ways to use the material in this manual with your DIGI-COMP II. After you have assembled DIGI-COMP II we suggest the following possibilities for starting out. Note the binary-decimal conversion table on your Programmers Card.

I. BEGINNERS

A. Read through Chapter 1.
B. Learn how to INITIALIZE your DIGI-COMP II using the NEW OPERATIONS Check List in Chapter 2, Page 5.
C. Work out the five TESTS in Chapter 2, from Page 6 to Page 8.
D. Learn how to read your DIGI-COMP II in binary and decimal numbers in Chapter 3.
E. Learn how to COUNT, CLEAR, ADD, MULTIPLY, SUBTRACT and DIVIDE from Chapter 4.
F. Learn to understand the COMPILER Language codes in Chapter 4.
G. Review the OPERATING INFORMATION given in Chapter 12.
H. Try to do Problems 1, 2, 3, 4, 5, and 13 of Chapter 13.
I. You will then no longer be a beginner and can go back and after reviewing Chapters 3 and 4, study the operation in more detail from Chapters 5 through 10. This should help you do problems 6, 7, 8, 9, 10, 11, 12, 14, and 15.
J. If you haven't already done so, try our DIGI-COMP I, and learn more about computer machine organization, and logic.

II. INTERMEDIATE

A. Go through Chapters 1, 2, 3 and 4 thoroughly.
B. Review the rules for binary addition of Chapter 7, Page 26.
C. Review the OPERATING INFORMATION of Chapter 12.
D. Try a few problems of Chapter 13, perhaps 1, 2, 3, 4, 5, 6, 7 and 13.
E. Review Chapters 5 through 11 and try doing the rest of the problems of Chapter 13.
F. Review Chapter 14 on electronic computer FORTRAN Language.
G. TRY our DIGI-COMP I with its Advanced Manual.

III. ADVANCED

A. Simply go right through the Manual from Chapter 1 through 14. Do related problems of Chapter 13 as mentioned above.
B. Try our DIGI-COMP I with its Advanced Manual and you will have fairly covered most of the basic theory of organization and operation of Digital Computers.

CHAPTER 1

ABOUT YOUR DIGI-COMP II

DIGI-COMP II is an educational computer which actually works in a manner analogous to an electronic digital computer. It performs a surprising number (but of course not all) of the operations performed by the electronic computer, but in a visible and mechanical manner rather than electronically.

As an educational device the primary purpose of DIGI-COMP II is to show *how* a computer works in binary arithmetic by mechanical simulation. To explain how a computer works, considerable attention must be devoted to an understanding of the binary number system. The operation of a computer, its logical design and many other important underlying concepts may be much better explained and understood by the aid of this mechanical model. Chief among the benefits of this approach is that no understanding of electronic circuitry is demanded.

The various operations that DIGI-COMP II performs are executed in slow motion as compared with an electronic machine. Where the electronic computer requires only a few millionths of a second to complete an operation, DIGI-COMP II requires a few seconds to complete the same task. Thus, the speed of the electronic computer has been reduced by a factor of a million to one. This enables the person operating DIGI-COMP II to comprehend what is happening as it occurs.

Further, the programming of DIGI-COMP II will be developed using computer languages similar to those in use in electronic computers. You may recognize words like FORTRAN or COBOL as languages used in many computers. Because DIGI-COMP II is a real binary digital computer its languages will be of the same type but, of course, specialized for it. As a climax to this Instruction Manual, in Chapter 14 you will be shown how to convert your DIGI-COMP II language to the IBM Computer FORTRAN language so that if you should have available an IBM computer of the models stated, *you will actually be able to insert simple problems into it with no further help required!!!*

All of the computer language words used in this Manual are summarized and defined at the end of this Manual, and on your programmers card.

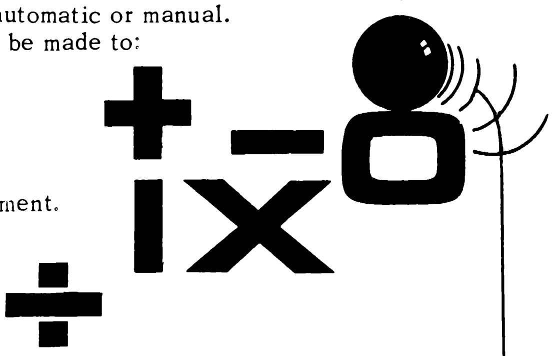
In DIGI-COMP II an electronic current impulse is simulated by a rolling ball. The ball rolls down an inclined surface under the force of gravity in an exposed manner. Thus, gravity provides the source of power for DIGI-COMP II rather than an electrical power supply. In the course of its journey the ball is directed by guides (which simulate the wires which guide electrons in the electronic computer) to the entry points of successive binary digital logic elements. The ball upon entering a logic element "questions" or "interrogates" it for its "state" and possibly changes its "state" in the process of going through it.

The logic elements in DIGI-COMP II simulate "FLIP-FLOP" circuits in an electronic computer. These elements are simply pivoted levers which have two possible positions. In entering the element from the top the ball is directed to exit along one of two possible paths according to the present setting of the element. The path of the ball out of the element is determined by the setting of the element and reveals the setting of the element. This is the sense in which the ball "interrogates" or questions the element as to its prior state. The "state", of course, refers to its left or right setting.

Thus, a single ball in its trip from the INPUT GUIDE at the top of the exposed surface to the COLLECTOR GUIDE at the bottom may take any one of many possible paths, questioning and changing the settings or states of the logic elements in the process. If the desired operation is only partially completed by the ball when it reaches the bottom it will be automatically directed to "gate" or trigger a successor ball from the top. This successor will in turn travel down the surface by a different path and gate still another successor ball if required, which will behave in like manner, etc., until the entire operation is complete. The machine will then shut itself off by directing the last ball to by-pass the gating mechanism.

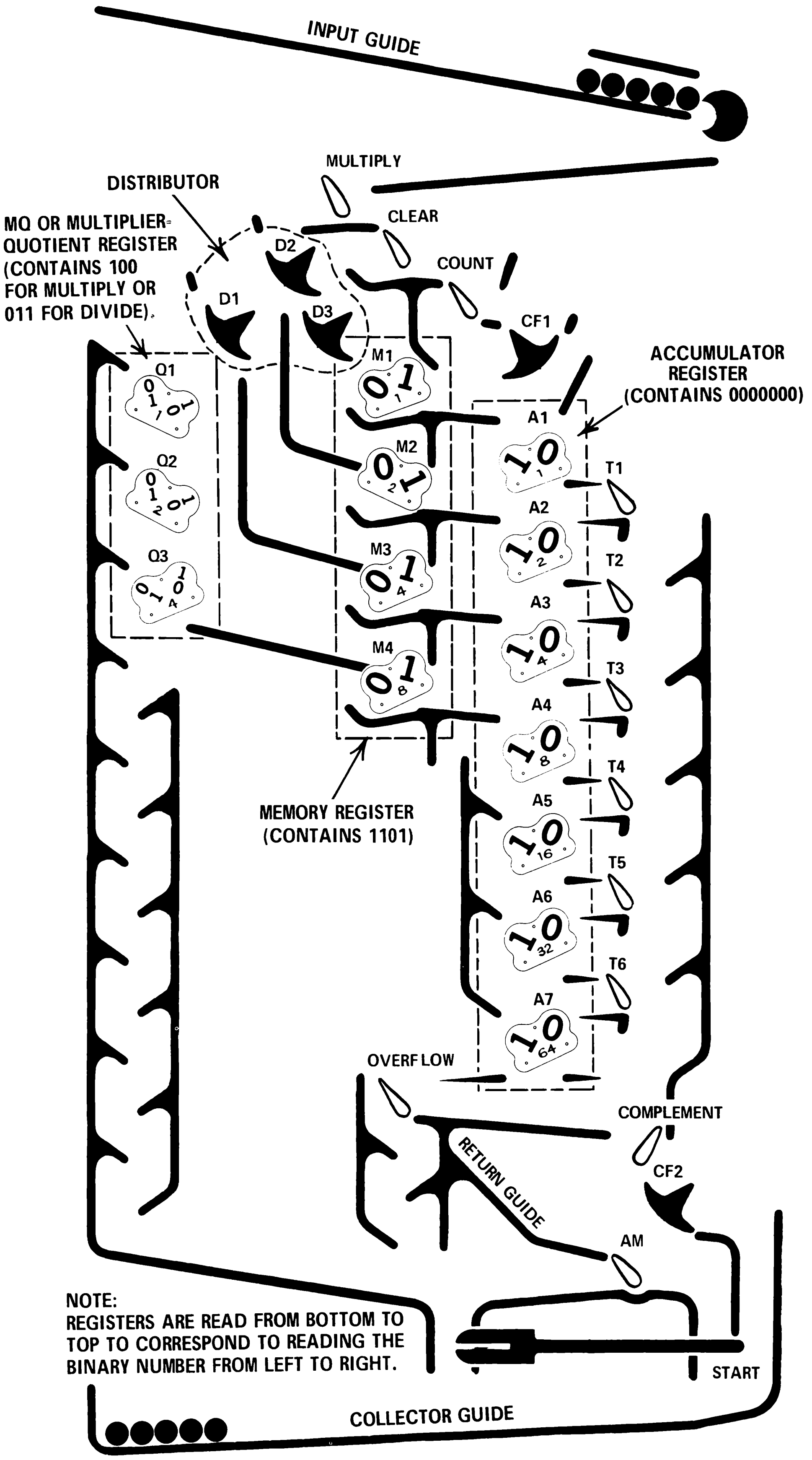
The operation of DIGI-COMP II can be entirely automatic or manual. With appropriate settings your DIGI-COMP II can be made to:

1. Count out a specified number of balls.
2. Count in binary arithmetic.
3. Add.
4. Multiply.
5. Multiply and add.
6. Obtain either the "1's" or "2's" complement.
7. Subtract.
8. Divide.
9. Zero or clear.
10. Overflow.
11. Halt.



Moreover, all of these operations may be performed either in a "MANUAL" or an "AUTOMATIC" mode. In the AUTOMATIC mode only a single pull of the START switch is required to complete any of the operations enumerated, but in the MANUAL mode the START switch must be pulled once for each machine cycle. Thus, the MANUAL mode permits a further slowing down of the machine for demonstration purposes if desired.

Your DIGI-COMP II operates in the Binary Number System. You will learn much about this important base two numbering system while using DIGI-COMP II. Indeed, the manner in which virtually all electronic computers work is intimately related to this binary or base two system.


INPUT GUIDE
MULTIPLY
DISTRIBUTOR
CLEAR
MQ OR MULTIPLIER-QUOTIENT REGISTER (CONTAINS 100 FOR MULTIPLY OR 011 FOR DIVIDE).
D2
COUNT
D1
D3
CF1
M1
ACCUMULATOR REGISTER (CONTAINS 0000000)
Q1
A1
M2
T1
Q2
A2
M3
T2
Q3
A3
M4
T3
A4
T4
MEMORY REGISTER (CONTAINS 1101)
A5
T5
A6
T6
A7
OVERFLOW
COMPLEMENT
RETURN GUIDE
CF2
AM
START
COLLECTOR GUIDE

NOTE:
REGISTERS ARE READ FROM BOTTOM TO TOP TO CORRESPOND TO READING THE BINARY NUMBER FROM LEFT TO RIGHT.

CHAPTER 2

CHECKING OUT YOUR DIGI-COMP II

Now that you have assembled DIGI-COMP II let us identify its basic parts and test them to be sure they operate properly.

Figure 1 shows the top of the computer and identifies the three binary number "registers" by name. They are the "ACCUMULATOR" REGISTER, the "M-Q" REGISTER and the "MEMORY" REGISTER.

These three registers are each comprised of a number of logic elements. One of the functions of these registers is to hold or "remember" binary numbers used in calculations. All of the arithmetic performed by the computer occurs in either the ACCUMULATOR or the M-Q REGISTER. The MEMORY REGISTER "remembers" a number but this number is not changed during the course of calculation.

Each of these registers is comprised of binary "logic elements". A register may contain a number having as many binary digits, or "bits", as the REGISTER has logic elements. Specifically, the ACCUMULATOR may contain any 7 bit number; the M-Q REGISTER any 3 bit number; and the MEMORY REGISTER any 4 bit number. Binary numbers will be explained in more detail later by actually operating DIGI-COMP II and noting what they represent.

Before describing the use of the three REGISTERS, let us first check them out to see if they are working properly.

As with an electronic computer, if the machine is to know what we want it to do, all of the switches must be set correctly. This is called "*initializing*" the computer. To help you remember how to do it, the following check list has been developed. Also, the *outline* of each "switch" or "flip-flop" in its correct normal position has been printed on the face of your DIGI-COMP II.

At the beginning of a new operation you will be instructed to, "INITIALIZE" the computer.

This means to follow the instructions given in the "New Operation Check List" below performing all the steps that need to be performed. If the elements to be reset are already in their correct position then no change need be made, of course.

NEW OPERATIONS CHECK LIST

1. Be sure that T1, T2, T3, T4, T5 and T6 are to the left.
2. Be sure that CF1 is to the left and that CF2 is to the right.
3. Set flip-flops D1, D2, D3, in the DISTRIBUTOR to the right as shown in Figure 1.
4. Set all the Control Switches to their "normal positions" according to the following table (Figure 1 also shows the control switches in their normal positions).

Normal Positions of Control Switches

Switch Name	Setting	Positions
MULTIPLY	OFF	left
CLEAR	OFF	left
COUNT	OFF	left
COMPLEMENT	OFF	right
OVERFLOW	RUN	left
A-M	AUTO	left

5. Place all of the balls (or a sufficient number) in the INPUT GUIDE at the top of the computer.

If the computer is "INITIALIZED" properly and if it is operated correctly and without error, it will never actually be required to change any settings in Step 1. In Step 4 it will be noticed that the "normal" position of all switches that have ON and OFF positions is OFF.

Notice also that the normal position of every switch is to the left except the COMPLEMENT SWITCH, which is to the right. Flip-flops D1, D2, D3 mentioned in Step 4 will have to be initialized after every operation in which the MULTIPLY SWITCH is ON. Otherwise, they will remain initialized correctly and need only be checked but not actually changed.

In each of the Experiments to follow, frequent requests will be made that the computer be "INITIALIZED" according to the New Operation Check List.

WHEN THE INITIALIZE COMMAND IS GIVEN YOU SHOULD SET ALL OF THE SWITCHES ACCORDING TO THIS CHECK LIST.

● TEST 1

First, "INITIALIZE" your DIGI-COMP II as described in the new operation Check List given on Page 4.

Next, turn all the flip-flops (A1, A2, A3, A4, A5, A6, A7) in the ACCUMULATOR REGISTER to the right. The vertical column of numbers is always read. So the accumulator is now read 1111111. (You may not know what this means yet, but it will become clear very shortly). Next, turn the "COUNT" switch to the "ON" (right) position and the "AM" switch to "MANUAL" (right) position. Pull the start switch and see what happens.

A ball should be triggered from the top, roll down past the "MULTIPLY" switch to the "CLEAR" switch and inside the "COUNT" switch. It should then go down through the ACCUMULATOR and change each "One" (1) to a "Zero" (0) before coming out the bottom and being caught in the COLLECTOR GUIDE.

If it doesn't work, it is probably because you didn't initialize DIGI-COMP II properly. Go back and try it again. If the marble jammed, look carefully at it and see if you can figure out why. Wiggle that flip-flop back and forth until it is nice and free, then set them all back to "one" (1) and try again.

Now, *before* you go on to the next test let us start learning how to "talk" to your computer in a language both of you can understand.

In communicating with computers programmers develop a special language or short-hand to save time. For DIGI-COMP II we can write the program for Test 1 using our own special language. This is called a *machine language*.

1. INITIALIZE
2. A = 1111111
3. COUNT = ON
4. AM = MANUAL
5. START

If you compare this program to the instructions for Test 1, you should begin to understand what each step means. To be sure you understand each command in the program, see your programmer's card for a complete list of possible commands and their definitions.

● ● TEST 2

In performing Test II, try setting all the switches correctly just by looking at the program below.

1. INITIALIZE
2. A = 1111111
3. CLEAR = ON
4. AUTO = MANUAL
5. START

If you did this correctly you should have first set the switches and flip-flops as directed in the New Operation Check List (Page 4). (The "COUNT" SWITCH should have been turned OFF.)

Next, you should have turned all the ACCUMULATOR flip-flops to the right (1) position. The third step was to turn the "CLEAR" switch to the right (on) position. Fourth, to turn the "AM" switch to "MANUAL".

Now, when you pull the START SWITCH the marble should go down the guide, to the right of the "MULTIPLY" switch, then go to the left of the "CLEAR" switch, drop to the lower level and change all the ACCUMULATOR flip-flops to "zero".

If it didn't work you should check the switch settings again. If only some of the flip-flops were changed to zero, wiggle the ones that did not flip. If they all changed to zero but the marble didn't come out the RETURN GUIDE, check the assembly instructions to be sure it is in correctly.

● ● TEST 3
●

1. INITIALIZE
2. A = 0000000
3. AM = MANUAL
4. START

If you followed the program correctly and your computer is assembled properly, the marble should have turned all of the ACCUMULATOR MODE T1, T2, T3, T4, T5, T6 FLIP-FLOPS to the right. If it didn't change them all wiggle the ones that didn't change.

Now leave everything as is and start another ball. It should change the ACCUMULATOR to all "ones" (1). Starting a third marble will cause all of the ACCUMULATOR MODE FLIP-FLOPS to turn back to the left. (Again if they don't all turn back, wiggle the ones that don't).

● ● TEST 4
● ●

1. INITIALIZE
2. AM = MANUAL
3. C = ON
4. Manually take a marble and place it in the guide just above the "COMPLEMENT" SWITCH. Let it roll down the guide into the COMPLEMENT SWITCH.

Notice that the ball first goes out of the COMPLEMENT SWITCH to the right. Next, it was directed out of CF2 to the left and simultaneously "flipped" CF2 to the opposite position. Then the ball was guided to enter the AM SWITCH which further directed it to the left.

Repeat Step 4 several times.

A peculiar result may now be noted. Although the first ball exited from CF2 to the left, the second ball exited to the right. Moreover, each successive ball alternated between the left and right exit out of CF2.

The element called CF2, is a simple "FLIP-FLOP" called the "second complement FLIP-FLOP" in DIGI-COMP II. It is used as part of the necessary circuitry to complement the ACCUMULATOR but this will be explained in more detail later. What should be noticed now is its FLIP-FLOP behavior.

Each successive ball entering CF2 can be imagined to be asking it the question, "Are you set to the right or to the left?" If the answer is "to the right", the ball flips it to the left and exits to the left. The next ball asks the same question but gets a different answer. CF2 responds that its setting is "to the left". In this case the ball will flip it to the right an d exit to the right. This FLIP-FLOP cycle will repeat over and over as long as it is questioned.

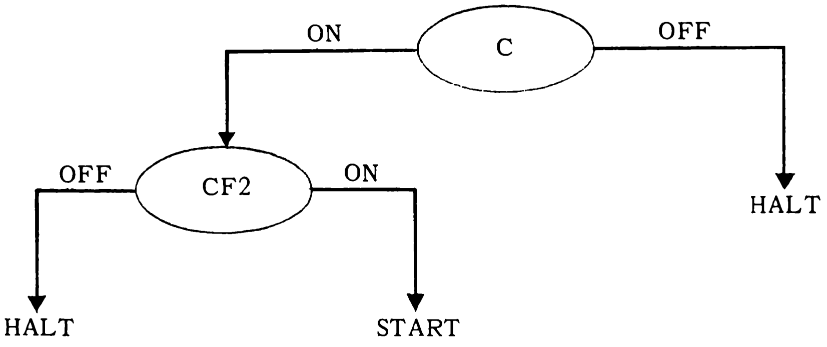
This questioning action of a ball in DIGI-COMP II, or an electrical impulse in a digital computer, is referred to as "INTERROGATING " the Logic Element for its binary "state". In DIGI-COMP II the "state" is the element's right or left setting, whereas in an electronic computer it may be one of two voltage levels, an "ON" or "OFF" condition or a magnetized or unmagnetized condition of a circuit element. It is called "BINARY" state because there are only two possible states. In an electronic computer millions of impulses interrogate and activate electronic FLIP-FLOPS in a second of operation!

●● TEST 5

1. INITIALIZE
2. AM = MANUAL
3. C = OFF
4. Repeat Step 4 of Test 4 again several times.

This time the ball will not be directed by the COMPLEMENT SWITCH to enter CF2. Instead the COMPLEMENT SWITCH will cause it to leave to the left and by-pass CF2 without interrogating and without flipping CF2. However, it should be clear that the ball is still questioning or interrogating the COMPLEMENT SWITCH this time, as indeed it was before.

The logical relationships between the COMPLEMENT SWITCH and FLIP-FLOP CF2 can be illustrated by what is known as a flow chart. The ball may be imagined to begin at the top and ask the questions indicated in each box. The answer that it obtains from the box determines which of the two paths it follows on its way out.



Thus the ball starting from the top asks the COMPLEMENT SWITCH a question and depending on the answer goes one of two ways, either to AM or to CF2. Similarly it asks the other Logic Elements questions and each time does one of two things depending on the answer, and then is directed to another logic element.

Flow charts of the type shown in Figure II are frequently used by computer people to display the logical relations between various elements and conditions.

In DIGI-COMP II switches such as the COMPLEMENT SWITCH and AM are only changed by the operator but FLIP-FLOPS like CF2 are changed by the machine process. Each of these binary logic elements may therefore be considered to store or memorize a "bit" or "binary digit" of information by virtue of its setting. Machines like DIGI-COMP II are frequently called "finite state" machines because there is a finite or limited number of states that the machine can take on.

Electronic computers work in a manner analogous to DIGI-COMP II. Instead of balls, electronic impulses in the form of voltage level changes are used but the net result is the same. One cannot see the voltage level in a transistorized circuit flip from one value to another or the magnetic field in a tiny iron doughnut called a "core" reverse itself, but these reversals do occur and in a manner entirely similar to the mechanical flipping of the FLIP-FLOP elements in DIGI-COMP II, but of course these changes occur millions of times faster.

CHAPTER 3

BINARY COUNTING

BINARY TO DECIMAL NUMBER CONVERSION

QUESTIONS:

1. How do computers count in binary?
2. How do "carries" take place in counting?
3. How much is each bit position of a binary number worth?
4. How does the Auto Manual Switch work?
5. How may a binary number be changed to its decimal value?
6. How may a decimal number be changed to its binary value?

EXPERIMENT I: BINARY COUNTING

(Consult Figure 1 to identify Logic Elements)

1. INITIALIZE
2. AM = MANUAL
3. A = 0000000
4. COUNT = ON
5. START

A ball will be triggered from the INPUT GUIDE to roll down the surface and add one into the ACCUMULATOR REGISTER. When the ball has reached the COLLECTOR GUIDE the ACCUMULATOR will read:

1
0
0
0
0
0
0

6. Repeat Step 5 another 6 times writing each successive accumulator reading down.

By recording each number obtained we get the following binary numbers:

	Number of times START switch was pulled							
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	0	1	0	1	0	1	0	1
Binary	0	0	1	1	0	0	1	1
number	0	0	0	0	1	1	1	1
obtained	0	0	0	0	0	0	0	0
in the	0	0	0	0	0	0	0	0
ACCUMULATOR	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

As a design convenience binary numbers are represented *vertically* in the ACCUMULATOR in DIGI-COMP II. However, it is customary to express binary numbers horizontally, rather than vertically, when writing them down. To write these numbers horizontally from left to right we write them to correspond with reading the ACCUMULATOR in DIGI-COMP II from bottom to top. Therefore, these same numbers would be more customarily expressed as,

1	0000001	or	1
2	0000010	or	10
3	0000011	or	11
4	0000100	or	100
5	0000101	or	101
6	0000110	or	110
7	0000111	or	111

(If you stand on the right side, which is the side the START SWITCH is on, you can see that you will read the ACCUMULATOR as an ordinary binary number.)

7. Pull the START switch once more and observe the "carries" that take place as one is added to 7. That is as,

0000111 + 0000001 becomes 0001000

Notice that A1 flips to "0" and the ball is carried to the entry of A2. Then A2 flips to "0" and the ball goes to A3. Then A3 flips to "0" and the ball goes to A4. Next, however, A4 is a "0" rather than a "1" and so no carry to A5 takes place. Instead, the ball leaves the ACCUMULATOR without affecting it further.

EXPERIMENT II:

8. AM = AUTO
9. START

The ninth ball will change the ACCUMULATOR TO 9 = 0001001 and will then be directed by the AM Switch to roll under the START SWITCH. This automatically triggers the next ball. (If the ball should ever happen to pass under the START SWITCH without triggering another one, just pull it manually.)

Now we can watch the automatic action of the computer as it counts. Every few seconds another ball is triggered from the supply at the top automatically and it adds one to the ACCUMULATOR in binary.

To stop the automatic counting it is only necessary to turn the AM Switch back to the Manual position. To re-start the automatic operation turn AM back again to Auto and Pull the START SWITCH.

BINARY TO DECIMAL NUMBER CONVERSION

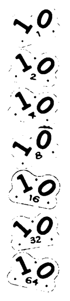
Let us stop for a minute and learn how to read out the numbers of the registers.

It is very simple to read out the decimal number held in each register in binary numbers.

1. Note that in the middle of each FLIP-FLOP read out there is a small number which you used during assembly to know where each Read-Out Indicator went.

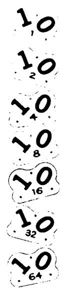
2. To read a register, simply add the values of these small numbers up for each place there is a 1, but *do not* add those numbers where there is a 0. For example, to read out the ACCUMULATOR register in the following case the numbers to be read are shown to the right of the column.

So the decimal number in the ACCUMULATOR is 86. Note that *only when a 1 is up* do we include the value of that FLIP-FLOP toward the total.



=	0
=	2
=	4
=	0
=	16
=	0
=	+64
	86

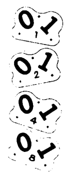
To be sure we understand, another example is,



=	1
=	2
=	0
=	0
=	0
=	32
=	+0
	35

So the number in this example is 35. Later when you learn about binary numbers you will be able to both read and set in binary numbers without the aid of the little decimal numbers on each FLIP-FLOP.

The MQ and MEMORY REGISTERS are read the same way. For example, for the MEMORY REGISTER –



=	1
=	0
=	4
=	+8
	13

and similarly for the MQ Register.

There are several methods of converting back and forth between binary and decimal numbers. The easiest type of conversion method to learn is a table. Take out your programmer's card and you will see such a Table.

Program DIGI-COMP II as follows and you will see how this table is made.

1. INITIALIZE
2. COUNT = ON
3. A = 0000000
4. START

Be sure to put *all* of the balls in the INPUT GUIDE. Each ball coming down through the ACCUMULATOR will add one to it.

As each additional ball comes down through the ACCUMULATOR, compare the readout with the next number in the Table. (If you get confused, the number of balls at the bottom should be the same as the Decimal number in the Table.)

With this Table it is possible to convert from binary to decimal or from decimal to binary simply by looking up the desired number and getting its equivalent.

From the Table you will notice that the *binary* numbers 1, 10, 100, 1000, 10000, 100000 and 1000000 correspond to the *base ten* numbers 1, 2, 4, 8, 16, 32, and 64. Each successive one of these numbers is *twice as large as its predecessor* and for reasons which will be explained later, they represent consecutive powers of two. In decimal or base ten the numbers 1, 10, 100, 1000, 10000, 100000 have an entirely different meaning than in binary, of course. Each of these numbers is *ten times its predecessor* in base ten.

You have learned that in decimal numbers, 683 actually means:

6 one hundreds + 8 tens + 3 ones.

Therefore, 683 can be written as:

$$
\begin{array}{llll}
 & 683 = 600 & + 80 & + 3 \\
\text{or} & 683 = (6 \times 100) & + (8 \times 10) & + (3 \times 1) \\
\text{or} & 683 = (6 \times 10 \times 10) & + (8 \times 10) & + (3 \times 1) \\
\text{or} & 683 = (6 \times 10^2) & + (8 \times 10^1) & + (3 \times 10^0)
\end{array}
$$

(Any number raised to the power to zero is equal to one, thus $10^0 = 1$.)

In the number 683 the digit 6 is in the "hundreds" position – the digit 8 is in the "tens" position – and the digit 3 is in the "ones" or "unit" position.

In other words we may say that each digit in any decimal number is a different power of ten depending on the position of the digit within the number.

What will each position in the binary number system mean?

In the binary system we have only two symbols (0 and 1). Therefore, EACH POSITION REPRESENTS A DIFFERENT POWER OF TWO (2).

That is, they are 2^0, 2^1, 2^2, 2^3, and so on.

An example of a binary number is: 110

This number is read: "ONE, ONE, ZERO."

The meaning of this binary number can be written as in the following example:

$$
\begin{array}{llll}
 & 110 = (1 \times 2^2) & + (1 \times 2^1) & + (0 \times 2^0) \\
\text{or} & 110 = (1 \times 2 \times 2) & + (1 \times 2) & + (0 \times 1) \\
\text{or} & 110 = (1 \times 4) & + (1 \times 2) & + (0 \times 1) \\
\text{or} & 110 = 4 & + 2 & + 0
\end{array}
$$

Given a binary number such as 0110101 we may express it as:

0100000 + 0010000 + 0000100 + 0000001 = 0110101

32 + 16 + 4 + 1 = 53

This gives us a convenient way to *convert any binary number to a decimal number* without even using the table. We merely write down the powers of two beginning from right to left above the binary number to be converted and sum those powers that correspond to "1" bits in the given number. Using the same example,

$$
\begin{array}{ccccccccccccc}
64 & & 32 & & 16 & & 8 & & 4 & & 2 & & 1 \\
\hline
(\,0 & & 1 & & 1 & & 0 & & 1 & & 0 & & 1\,)_2 \\
0 & + & 32 & + & 16 & + & 0 & + & 4 & + & 0 & + & 1 = 53
\end{array}
$$

The same technique can be used to read a binary number in the ACCUMULATOR REGISTER of DIGI-COMP II and mentally convert it to decimal to better understand the result. Beginning with the top bit of the ACCUMULATOR and proceeding down we consider that the bit positions are worth 1, 2, 4, 8, 16, 32, 64 (these are the small numbers printed on the readout indicators for easy reference. We suggest that you cover them with a small piece of tape to make you learn faster.) Wherever the bit is a "1" we sum this corresponding number but wherever it is a "0" we do not.

Suppose, on the other hand, we are given a decimal number, such as 74, and wish to express it in binary. We begin by putting down the numbers 1, 2, 4, 8, 16, 32, 64 in a string as before, but this time we start from the bottom of the list and ask:

> Can we subtract 64 from the number? If the answer is "yes", perform the subtraction obtaining a remainder and place a "1" in the last or bottom digit. If "no", place a "0" and consider the number to be the remainder.
>
> Next, can we subtract 32 from the remainder? If "yes", place "1" in the second from last digit and perform the subtraction. If "no", place a "0" and consider the present number to be the remainder.

Proceed in like fashion until the complete number is changed to binary. For example, to convert 74 to binary we begin with Step 1 at the bottom and proceed upward as follows:

Step 1 begin	74-64 = 10		Place 1 in 7th position
Step 2	10-32	cannot be subtracted	Place 0 in 6th position
Step 3	10-16	cannot be subtracted	Place 0 in 5th position
Step 4	10-8 = 2		Place 1 in 4th position
Step 5	2-4	cannot be subtracted	Place 0 in 3rd position
Step 6	2-2 = 0		Place 1 in 2nd position
Step 7	0-1	cannot be subtracted	Place 0 in 1st position

Therefore,

$$74 = \begin{matrix}0\\1\\0\\1\\0\\0\\1\end{matrix} = (\ 1001010\ )$$

This result can, of course, be checked by binary to decimal conversion or by checking the conversion Table on your programmer's card. That is,

Number	*Worth if "1"*	*Sum*
0	1	0
1	2	2
0	4	0
1	8	8
0	16	0
0	32	0
1	64	+ 64
		74

(which checks)

Summarizing, to convert a binary number to decimal we begin at the bottom of the register (left of the number) and subtract from the given decimal number. Each time we *can subtract* one of the numbers 64, 32, 16, 8, 4, 2, 1 we place a "1" bit in the corresponding position, and each time we *cannot subtract,* we place a "0" bit in the position. The remainder from the subtraction is carried forward each time the subtraction can be performed, but otherwise the last remainder obtained is carried forward.

QUESTIONS:

1. What is the binary equivalent of 63?
2. What is the binary equivalent of 112?
3. What is the decimal equivalent of 1101101?
4. What is the decimal equivalent of 1001011?

To check your answers consult the binary decimal conversion Table on your programmer's card.

The binary numbers you have obtained by counting on DIGI-COMP II are exactly the same as those used in most large scale electronic computers except they may have a different number of bits. Numbers of this nature are stored in MEMORY REGISTERS and used in Arithmetic Registers for calculation. Professional programmers who must work closely with the computer become very familiar with these numbers and are able to convert from binary to decimal and vice versa quite readily while they are in the process of checking out the program which they have written.

CHAPTER 4

ARITHMETIC OPERATIONS

The front panel of DIGI-COMP II has a chart printed on it which is meant to serve as a reminder for doing the basic arithmetic operations.

COUNT:

0001
0010
0011
0100
0101

In Chapter 3 we did some counting. To get more practice in counting and using the front panel chart, put 15 marbles in the INPUT GUIDE and Program DIGI-COMP II as follows:

1. INITIALIZE
2. A = 0000000
3. COUNT = ON

Now look at the front panel switch setting chart and compare each switch position for the COUNT OPERATION with those you programmed into the computer. They should be the same. Now, pull the START SWITCH and DIGI-COMP II will count until all the marbles have come down.
You should have (0001111) in the ACCUMULATOR now.

CLEAR:

0
0
0
0
0
0
0

Now program DIGI-COMP II according to the following:

1. INITIALIZE
2. A = 0001111
3. CLEAR = ON
4. AM = MANUAL
5. START

The ACCUMULATOR should have been cleared to 0000000 with one marble. Again, you should compare your switch settings with those called for on the front panel chart for the CLEAR operation.

ADDITION:

In addition of two numbers the method is to put (001) in the MQ REGISTER (*always* use the *upper set* of 01 numbers on the MQ FLIP-FLOPS except when reading out the answer to a problem in division), one number to be added in the MEMORY REGISTER and the other number in the ACCUMULATOR. The answer will appear in the ACCUMULATOR.
If we wish to add 13 to 71, for example, put the binary equivalent of 13 which is (1101) in the MEMORY REGISTER (M) and the binary equivalent of 71 which is (1000111) in the ACCUMULATOR REGISTER. In the language of DIGI-COMP II, the command M = 1101 means

"Enter $\begin{pmatrix}1\\0\\1\\1\end{pmatrix}$ in the MQ REGISTER". (Do not forget: unless told otherwise, always use the upper set of numbers on the Indicators of the MQ REGISTER.)

```
   0101
+  1011
  -----
  10000
```

With this as a background the program for this problem would be as follows:
1. INITIALIZE
2. MULTIPLY = ON
3. A = 1000111
4. M = 1101
5. MQ = 001
6. START

What should the ACCUMULATOR have when you finish?

If you set everything up properly the ACCUMULATOR should read 1010100.

Now try a problem yourself. The Program is the same; just change the numbers in the MEMORY and ACCUMULATOR REGISTERS.
Write the Program for adding 15 to 112. If you do it correctly the answer should be (1111111).

Make up some of your own addition problems. They can be any two numbers whose sum is less than 128, with one of the numbers less than 16.
When you do it check your answer by decimal to binary conversion to be sure it is correct.

MULTIPLICATION:

To multiply two numbers, put one of them in the MQ REGISTER and the other in the MEMORY REGISTER. The answer will appear in the ACCUMULATOR REGISTER.

```
   011
 X 110
 -----
  0110
 011
 -----
 10010
```

As an example let us multiply 4 times 13. To do this, set up DIGI-COMP II as follows:
1. INITIALIZE
2. MULTIPLY = ON
3. A = 0000000
4. M = 1101
5. MQ = 100
6. START

If you did it correctly the answer should have been (0110100).

Now try to multiply 3 times 15. The Program is the same as above except that M = 1111 and MQ = 011. If you do it correctly, the answer should be (0101101). Again you should make up some of your own problems and work them. The number in the MQ REGISTER must be 7 or less and the number in the MEMORY REGISTER must be 15 or less. Work out the answer in decimal arithmetic and convert it to binary to check your answer.

SUBTRACTION:

The general method used by DIGI-COMP II, and most electronic computers, is called complement and add. The technique is to put the smaller of the two numbers in the ACCUMULATOR REGISTER, perform the operation on it and add the larger number to it. The answer will appear in the ACCUMULATOR. In performing the COMPLEMENT operation DIGI-COMP II will take four marbles and shut off. Then you change the switches to the ADD mode and start again. To subtract 4 from 12 the program is as follows:

1. INITIALIZE
2. COMPLEMENT = ON
3. A = 0000100
4. M = 1100
5. MQ = 001
6. START
7. PAUSE*
8. MULTIPLY = ON
9. COMPLEMENT = OFF
10. START

$$\begin{array}{r} 110101 \\ -\ 10111 \\ \hline 011110 \end{array}$$

The number in the ACCUMULATOR should be (0001000).

Now subtract 14 from 15. The program is the same as above except that A = (0001110) and M = 1111. The answer should be (0000001). Try some problems yourself. Since the smaller number goes in the ACCUMULATOR it shouldn't be larger than 14 because the MEMORY REGISTER will not hold numbers larger than 15.

*The PAUSE command means that you should pause until the computer has completed its automatic cycle.

DIVISION:

Division is performed in DIGI-COMP II and in most electronic computers by repeated subtractions. Basically the technique used is to place the Dividend in the ACCUMULATOR and the Divisor in the MEMORY REGISTER. The Dividend is then complemented and added to the contents of the MEMORY REGISTER repeatedly. The answer will then appear in the LOWER set of numbers in the MQ REGISTER.

As an example, the program for dividing 25 by 5 is as follows:

1. INITIALIZE
2. MQ = 111
3. A = 0011001
4. M = 0101
5. COMPLEMENT = ON
6. START
7. COMPLEMENT = OFF
8. MULTIPLY = ON
9. OVERFLOW = HALT
10. START

Note that Steps 7, 8, 9 and 10 should be performed only after the complement operation is completed. The answer (quotient) should appear in the LOWER set of numbers in the MQ REGISTER. It should be 101.

Division gets more complicated if the divisor does not go into the dividend an even number of times. Because remainders and therefore binary fractions are involved these are discussed in Chapter 11.

In making up your own division problems you can follow the same program as above; just change the values of A and M.

The six basic operations illustrated in this chapter have programs which are not changed except by the numbers involved. The subtraction program is, for instance, the same except for the values of A and M for any subtraction problem. This situation is, as you might suspect, the same in electronic computers as it is in the case of DIGI-COMP II.

COMPILER LANGUAGE . . . DIGI-TRAN

To further reduce a programmers work, the manufacturers of electronic digital computers have worked out what are known as *"compiler"* or *"assembly languages"*. (Except for the INITIALIZE command the type of programming we have been doing so far is called *"machine language"*. This is a language that the computer can understand directly – no interpreter is necessary.) These *compiler languages* have been largely responsible for the fantastic growth rate of the computer industry. They have made it possible for a great number of people to write relatively difficult programs with very little professional training.

We can develop a *"compiler language"* called DIGI-TRAN which basically combines several commands in one.

For instance we can combine commands number 1, 2, 5, 6, 7, 8 and 9 for division into one large command which is simply called DIVIDE.
Then to write the program for dividing 25 by 5 is simply:

1. A = 0011001
2. M = 0101
3. DIVIDE

As a further convenience to programmers, electronic computers are equipped with automatic binary decimal converters.

So the program would then be written:

1. A = 25
2. M = 5
3. DIVIDE

DIGI-COMP II has no automatic converter, so we will stick with the binary numbers.

In the rest of this book the programs will be written in the compiler language. The six basic commands are listed below. The use of the one basic command implies each of the individual commands in the machine language. These are summarized in the chart on the front panel of DIGI-COMP II and are listed in detail below.

DIGI-TRAN COMMANDS

COUNT	1. INITIALIZE
	2. A = 0000000
	3. COUNT = ON
	4. START
CLEAR	1. INITIALIZE
	2. CLEAR = ON
	3. AM = MANUAL
	4. START
ADD	1. INITIALIZE
	2. MULTIPLY = ON
	3. MQ = 001
	4. START
MULTIPLY	1. INITIALIZE
	2. MULTIPLY = ON
	3. A = 0000000
	4. START
SUBTRACT	1. INITIALIZE
	2. COMPLEMENT = ON
	3. MQ = 001
	4. START
	5. PAUSE
	6. MULTIPLY = ON
	7. COMPLEMENT = OFF
	8. START
DIVIDE	1. INITIALIZE
	2. MQ = 111
	3. COMPLEMENT = ON
	4. START
	5. PAUSE
	6. COMPLEMENT = OFF
	7. MULTIPLY = ON
	8. OVERFLOW = HALT
	9. START

CHAPTER 5

THE ACCUMULATOR REGISTER

We have used the ACCUMULATOR REGISTER for counting and learning binary numbers in the previous sections. Now we will examine more of its properties.

QUESTIONS:

1. What is meant by ACCUMULATOR Overflow ?
2. How can we halt the Automatic Operation of the computer when an overflow occurs?
3. What is the meaning of the word "ACCUMULATOR"?
4. What is the octal system and how is it used?
5. What is a "machine cycle"?
6. How can we speed up counting on DIGI-COMP II?
7. How are the ACCUMULATOR FLIP-FLOPS constructed?

EXPERIMENT I:

Program DIGI-COMP II according to the following:

Step 1 – 1. INITIALIZE
2. AM = MANUAL
3. A = 1111100
4. COUNT = ON

Now pull the START switch 6 times, waiting each time until the ball reaches the bottom COLLECTOR GUIDE before pulling START again. Pay particular attention to what happens when ball number 4 is triggered from the INPUT GUIDE.

The ACCUMULATOR should read as follows after each ball is gated from the supply at the top.

Number of Balls Gated	After Gating Accumulator Reads
0	1111100
1	1111101
2	1111110
3	1111111
4	0000000
5	0000001
6	0000010

When ball number 4 is gated, an ACCUMULATOR OVERFLOW takes place. This means that the Register cannot hold the new number that results because it has more *binary* digits than the *Register* has bit positions.
Thus, when "1" is added to 1111111, the result should be 10000000. But the ACCUMULATOR cannot hold an 8 bit binary number and hence it records only the 7 low order bits giving 0000000 as the result. Moreover, an attempt is made by the ACCUMULATOR to carry a "1" into a non-existent eight bit position.

The overflow can be observed when the fourth ball is TRIGGERED. This ball exits from A7 on the left path and enters the OVERFLOW SWITCH.

Step II – Reprogram DIGI-COMP II as we did in Step I above, except set the AM switch to AUTO. Pull the START switch and count the balls as they are triggered and flip the AM Switch back to MANUAL after the sixth ball is triggered.

With the OVERFLOW SWITCH on RUN, the machine continues its automatic operation even though an overflow takes place on the fourth ball and it becomes necessary to stop the machine by turning the AM Switch to MANUAL.

Step III – Repeat Step II, but this time set the OVERFLOW SWITCH to its HALT position. Set the AM Switch to AUTO then Pull the START Switch.

This time the overflow on the fourth ball will halt the automatic operation of the machine, and the ACCUMULATCR will read 0000000. In many operations on a computer, it is necessary to detect an overflow and halt the operation or perform an alternate operation. On other operations, it is desired to ignore any overflow which might take place. The Overflow RUN or HALT switch allows us to exercise either of these options.

EXPERIMENT II:

SPEEDING UP DIGI-COMP II OPERATION

The ACCUMULATOR REGISTER gets its name from its ability to "accumulate" or "sum" numbers.

Step I – Set DIGI-COMP II to COUNT. (Check the front panel chart to be sure you did it correctly.)

A "machine cycle" is all of the actions that one ball completes in its journey from the INPUT GUIDE at the top of the computer to the COLLECTOR GUIDE at the bottom. Each ball having completed a machine cycle trips the START SWITCH automatically to initiate another machine cycle. You may wonder if more than one of these cycles can go on at the same time.

Step II – START

Step III – As the computer is automatically counting, visually follow a new ball when it is triggered. When it is about one third of the way down, pull the START SWITCH triggering another ball. When the first ball is about two thirds down the surface, pull the START SWITCH again.

There will now be three balls rolling down the surface at the same time. Moreover, as each ball reaches the START SWITCH and triggers another ball from above, it will replace itself keeping three balls rolling down the surface continuously. The machine will count correctly, even with multiple balls going down the surface at the same time and the counting operation is thus speeded up by a factor of three to one.

Virtually all electronic computers have an ACCUMULATOR REGISTER. Most of the large scale electronic computers use binary arithmetic in the ACCUMULATOR, just as in DIGI-COMP II. Therefore, learning how the ACCUMULATOR works in DIGI-COMP II should help in visualizing how the ACCUMULATOR works on an electronic computer.

Notice that the ACCUMULATOR is comprised of FLIP-FLOPS. If one of these elements is in the "zero" state an impulse flips it to the "one" state, but the impulse does not "carry" to the next FLIP-FLOP. These simple rules are sufficient to accomplish binary counting and addition.

EXPERIMENT III: MORE ABOUT BINARY NUMBERS

So far we have shown you how to convert a binary number to a decimal number by making use of the fact that the right-most position (top-most in DIGI-COMP II) is worth "0" or "1", the second from right "0" or "2", etc., according to whether its binary digit is a "0" or a "1". What do these special numbers 1, 2, 4, 8, 16, 32, 64, etc., represent? In mathematics we learn that these numbers are "powers of two" meaning that they also may be represented by,

$$2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, \text{etc.}$$

where "2^k" means multiply "2" by itself "k" times. The superscript "k" is called the power of "2". For example, "2^5" means "2" to the 5th power or "2" multiplied by itself "5" times or "$2 \times 2 \times 2 \times 2 \times 2$". Notice that "$2^0$" is assumed to be "1" despite the fact that our definition does not otherwise tell us its value. It is consistent for reasons which will not be given here to assume that *any* number raised to the "0" power is always "1" with the exception of "0" which is undefined.

It is possible to base a numbering system on any positive whole number, not just two or ten. Let us consider a system based on eight, for example. Such a system is called a "base eight" or "octal" system.

THE OCTAL SYSTEM

The reason that our everyday numbering system is based on ten rather than eight is certainly because man has ten fingers on both hands rather than eight. If man had only a total of eight fingers he would undoubtedly have developed the octal system rather than our present decimal system. He would have counted as follows:

1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17,
20, 21, 22, 23, 24, 25, 26, 27, 30, 31, 32, 33, 34, 35, 36, 37
40.

Notice that man counting on his fingers would have called his eighth finger (10) instead of 8. Then starting over again on his 9th finger, or first finger on his other hand, he would have called it (11), etc. In octal there is no symbol "8" or "9" for eight or nine. Instead only the symbols 0, 1, 2, 3, 4, 5, 6, 7 are used.

A base eight number such as $(3745)_8$ may be interpreted as,

$$3 \times 8^3 + 7 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 \qquad \text{where,}$$

$$8^3 = 8 \times 8 \times 8,\ 8^2 = 8 \times 8,\ 8^1 = 8, \text{ and } 8^0 = 1$$

In other words,

$$(3745)_8 = 3 \times 512 + 7 \times 64 + 4 \times 8 + 5 = 1536 + 448 + 32 + 5$$

which is equal to 2021. All the numbers outside the parentheses are in base ten.

By similar reasoning a binary or base two number such as 10110101001 may be interpreted in decimal or base ten as,

$$1 \times 2^{10} + 0 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 +$$

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

or

$$1024 + 0 + 256 + 128 + 0 + 32 + 0 + 8 + 0 + 0 + 1 = 1449$$

All numbers in this equation are in base 10, not just the 1449.

Binary numbers are long and cumbersome to write down while octal numbers are shorter. A very simple relationship exists between binary and octal numbers and it is easy to convert from binary to octal and vice versa.

Consider the binary number, $(11110111100000010001)_2$. If the digits of this number are grouped three at a time starting from the right we obtain, (011 110 111 100 000 010 001), where a leading "0" was added to make the triplets come out even. As you should now know,

0 = (000)	2 = (010)	4 = (100)	6 = (110)
1 = (001)	3 = (011)	5 = (101)	7 = (111)

011	110	111	100	000	010	001
⇩	⇩	⇩	⇩	⇩	⇩	⇩
3	6	7	4	0	2	1

And thus each triplet can be replaced by one of the digits "0" through "7". Therefore, our binary number becomes $(3674021)_8$ in octal which is considerably shorter to write. To convert from octal to binary the reverse substitution is equally simple. A table of only 8 octal digits and their corresponding binary values need be known. This may be committed to memory very easily.

Octal numbers are frequently used by Computer Programmers as a shorthand notation for long binary numbers. The conversion from binary to octal and vice versa is extremely simple and can be done mentally after one has memorized the proper symbols "0" through "7" for each of the possible binary triplets (000) through (111).

BINARY TO DECIMAL CONVERSION USING OCTAL

A binary number such as (1101000110111010) can be converted to decimal more simply by first converting it to octal, then converting the octal number to decimal. This number is (001 101 000 110 111 010) or $(150672)_8$. We next convert to decimal as follows.

1. $1 \times 8 + 5 = 13$
2. $13 \times 8 + 0 = 104$
3. $104 \times 8 + 6 = 838$
4. $838 \times 8 + 7 = 6711$
5. $6711 \times 8 + 2 = 53688$

Therefore, our original binary number is equal to 53688 in decimal. If you have had algebra, the reason this works can be given as follows:

$(150672)_8 = 1 \times 8^5 + 5 \times 8^4 + 0 \times 8^3 + 6 \times 8^2 + 7 \times 8 + 2$ which can be expressed as,

$$((((1 \times 8 + 5) \times 8 + 0) \times 8 + 6) \times 8 + 7) \times 8 + 2$$

DECIMAL TO BINARY USING OCTAL

A decimal number such as 72589 can be converted to binary by using octal as follows. First we convert it to octal by repeated division by eight, keeping track of all of the remainders obtained.

	8	72589	
1st quotient	9076	1	1st remainder
2nd quotient	1134	4	2nd remainder
3rd quotient	141	6	3rd remainder
4th quotient	17	5	4th remainder
5th quotient	2	1	5th remainder
6th quotient	0	2	6th remainder

The remainders are then written in reverse order, from bottom to top, to obtain the octal number $(215641)_8$. Now conversion to binary is simple. We write, (010 001 101 110 100 001), or (10001101110100001). The explanation for why this works is similar to the reverse case already explained but it will not be given here.

CHAPTER 6

THE MQ (MULTIPLIER QUOTIENT REGISTER) AND THE DISTRIBUTOR

QUESTIONS:

1. Why are there a double set of numbers on the MQ FLIP-FLOP indicators?
2. How may the MQ REGISTER count up or down?
3. How high can we count in the MQ REGISTER?
4. What is an MQ OVERFLOW?
5. What is a DISTRIBUTOR CYCLE?
6. How does the MQ REGISTER count on a DISTRIBUTOR cycle?
7. Where does the MQ get its name?

The MQ REGISTER is the left-most *register* on the computer and it is comprised of 3 FLIP-FLOPS. The MQ REGISTER is intended to hold a binary number containing 3 binary digits or bits. Thus, the MQ can hold any binary number from 000 to 111, or, from "0" to "7" in decimal. As the name suggests, the MQ REGISTER is used for Multiplication and Division.

The DISTRIBUTOR like the MQ REGISTER, is also comprised of three FLIP-FLOPS, but its purpose is entirely different. Where the MQ is designed to store or hold a 3–bit binary number, the DISTRIBUTOR is not. Instead, it is the function of the DISTRIBUTOR to distribute the balls which consecutively arrive at its entry point four different ways.

EXPERIMENT I:

Program DIGI-COMP II as follows:

1. INITIALIZE
2. MULTIPLY = ON
3. AM = MANUAL
4. M = 0000
5. MQ = 111
6. A = 0000000

The indicators on the MQ REGISTER have a double set of numbers. One set is above the other set. The upper set is used for Multiplication, and the lower set is used for Division.

Pull the START SWITCH. The first ball will be delivered by the DISTRIBUTOR to the entry of the MQ REGISTER. Reading the upper set of numbers, the ball will subtract one from the MQ leaving (110) and roll over to the M4 bit of the MEMORY REGISTER. Since M4 is zero it will direct the marble to drop in the hole to its left and it will then return to the first surface via the RETURN GUIDE and finally be deposited in the COLLECTOR GUIDE at the bottom.

7. *Pull* START 3 more times, waiting and studying the path of each ball until it reaches the BALL COLLECTOR before triggering another ball.

The second ball will be directed by the DISTRIBUTOR to M2, the third ball to M3, and the fourth ball to M1. These first 4 balls have completed one DISTRIBUTOR CYCLE, meaning that 4 consecutive balls have been delivered to M4, M2, M3, and M1, respectively with the first ball doing the double duty of subtracting one from the MQ REGISTER first.

In other terminology the three DISTRIBUTOR FLIP-FLOPS D1, D2 and D3 are connected as a "Ring Counter" that counts to four in this case. A ring counter is one that returns to its starting point after a fixed number of counts.

1. AM = AUTO
2. START

The ACCUMULATOR REGISTER can also be considered as a Ring Counter.

This time the fifth ball will start a second DISTRIBUTOR Cycle by subtracting one from the MQ obtaining 101 and being delivered to M4 just as the first ball did. Again, balls 5, 6, 7, 8 will be delivered to M4, M2, M3, and M1 respectively just as in the first cycle of four balls. Let the computer run in this manner until it turns itself off.

By watching the upper set of numbers in the MQ, you will see that "1" is subtracted from the MQ by the first ball of every cycle of four balls. By watching the lower set of numbers, it will appear that "1" is added to the MQ by the first ball of every cycle of four balls. Thus, the counting in the MQ will be as follows:

Upper Set: 111, 110, 101, 100, 011, 010, 001, 000.
Lower Set: 000, 001, 010, 011, 100, 101, 110, 111.

Thus, the MQ REGISTER can count from 7 down to 0 using the upper set of numbers or from 0 up to 7 using the lower set of numbers. Any upper number of 3 bits can be deduced from a lower number 3 bits and vice versa by changing each "0" to "1" and each "1" to "0". A number obtained from a binary number by reversing the value of the bits is called the *One's Complement* of the number.

The machine will stop its automatic operation on an MQ REGISTER OVERFLOW. This will occur when the REGISTER reads 000 in the upper set or 111 in the lower set. The first ball on the next cycle of four will alter the MQ to 111 in the upper set or 000 in the lower set, but it will not be delivered to M4. Instead an MQ REGISTER OVERFLOW will direct the ball out of the left side of the lowest bit of the MQ REGISTER and it will bypass M4 and the TRIGGERING MECHANISM.

EXPERIMENT II:

Set 011 in the MQ REGISTER using the upper numbers. Set D1, D2 and D3 to their right positions. Pull the START SWITCH.

How many balls are triggered before the computer stops? How many times is a ball delivered to M4, to M3, to M2, to M1?

For each subtraction of "1" from the MQ REGISTER, 4 balls are required. As binary 011 represents the decimal number 3, we would expect 3 × 4 or 12 balls to be required before the REGISTER OVERFLOWED. Ball number 13 should then cause the overflow.

Each cycle of four balls delivers one ball each to M4, M2, M3, M1 in that order, assuming D1, D2, D3 are all initially set to the right. Therefore, if there are 3 cycles, exactly 3 balls will be delivered to each of the elements M4, M2, M3, M1.

The exact manner in which the MQ REGISTER is used in multiplication and division will be described in later experiments.

CHAPTER 7

THE MEMORY REGISTER AND ADDITION

QUESTIONS:

1. How large a number will the MEMORY REGISTER hold?
2. How do computers "read" the contents of the MEMORY REGISTER into the ACCUMULATOR?
3. How can we add two binary numbers?
4. Why is the order of the addition of MEMORY REGISTER bits important?
5. What is meant by destructive and non-destructive read out?
6. Why does the ACCUMULATOR REGISTER need to have more bits than the MEMORY REGISTER?

EXPERIMENT I:

TRANSFERRING THE CONTENTS OF THE MEMORY REGISTER TO THE ACCUMULATOR REGISTER

Program as follows:

1. INITIALIZE
2. CLEAR
3. M = 1101
4. MQ = 001
5. MULTIPLY = ON
6. START

The computer will trigger 5 consecutive balls then turn itself off.

A. The first ball will be delivered by the DISTRIBUTOR to the MQ REGISTER where it will subtract "1" from this REGISTER leaving 000 and then be delivered to the M4 bit of the MEMORY REGISTER. As M4 is a "1", the ball will be directed to F4 of the ACCUMULATOR REGISTER. This will have the effect of adding 1000 to the ACCUMULATOR obtaining 0001000 in that REGISTER. The ball will then roll past the START SWITCH triggering a second ball from above.

B. The second ball will be delivered by the DISTRIBUTOR to position M2 of the MEMORY REGISTER. As this position is a "0", it will drop through the left hole, bypassing the ACCUMULATOR and ultimately triggering another ball from above.

C. The third ball will be directed to M3. As this bit is "1", the number 100 will be added to the ACCUMULATOR obtaining 0001100 at this stage. A fourth ball will then be triggered.

D. The fourth ball will be directed to M1 and 1 will be added to the ACCUMULATOR obtaining 0001101. A fifth ball will then be triggered.

E. The fifth ball will follow the same route that the first ball took to the MQ REGISTER. However, in attempting to subtract one from this REGISTER, an overflow will occur leaving 111 in the MQ and the ball will leave the MQ to the left, bypassing the gating mechanism. The machine will thus shut itself off having transferred the number 1101 from the MEMORY REGISTER to the ACCUMULATOR. The ACCUMULATOR will read 0001101 (13 in decimal).

EXPERIMENT II:
ADDING THE CONTENTS OF THE MEMORY REGISTER TO THE ACCUMULATOR REGISTER

In the previous operation, the number 1101 in the MEMORY REGISTER was actually added to 0000000 in the ACCUMULATOR to produce 0001101 in the ACCUMULATOR. To prove that this sequence of addition is perfectly general, let us continue the following example:

Step I: Reset D1, D2, D3 to right and reset the MQ to 001 without changing the ACCUMULATOR or the MEMORY REGISTER.
Pull START.

The number 1101 in the MEMORY REGISTER will now again be added to 0001101 in the ACCUMULATOR and the answer in the ACCUMULATOR will be,

$$0011010 = 1101 + 0001101$$

Notice that the answer 0011010 has the same bit configuration as the number 1101 except that it is shifted left one place. Why should this be true? What is the number 0011010 in decimal? Starting from the right in evaluating this number, we have:

$$0 + 2 + 0 + 8 + 16 + 0 + 0 = 26 \text{ which is indeed equal to } 13 + 13$$

EXPERIMENT III:

Step I: Repeat Step I of Experiment II with the AM SWITCH on MANUAL. Pull START five times in succession attempting to predict before each new machine cycle what the next ball will do.

We should now have 0011010 + 1101 in the ACCUMULATOR. What is the answer in binary? Convert this number to decimal. Does the answer in the ACCUMULATOR require more bits than the MEMORY REGISTER? Why must the bit capacity of the accumulator exceed that of the MEMORY REGISTER?

EXPERIMENT IV:
ADD 14 TO 37

The Program is as follows:
1. ADD
2. A = 0100101
3. M = 1110

In general, we add two numbers to one another in DIGI-COMP II by setting the larger number in the ACCUMULATOR manually and setting the smaller number in the MEMORY REGISTER manually. The MQ is then set equal to 001 because the actual operation being performed is to add the MEMORY REGISTER to the ACCUMULATOR as many times as indicated by the number in the MQ REGISTER. Thus, if the MQ is set to 001, the number in the MEMORY REGISTER, as for this example 14, will be added to the number in the ACCUMULATOR, (37), exactly one time.

The smaller of the two numbers used in the addition cannot exceed the capacity of the MEMORY REGISTER which is (1111) or 15. The larger of the two numbers cannot exceed the capacity of the ACCUMULATOR which is (1111111) or 127. Actually, the second number must be smaller than 127, however. This is because the sum of the two numbers cannot exceed 127 without causing an ACCUMULATOR overflow.

EXPERIMENT V: ADDING CONSECUTIVE NUMBERS

A short cut can be used to add consecutive numbers such as 12 + 13 + 14 + 15. Since 4 numbers are to be added enter (100) in the MQ REGISTER.

Step 1: Now program DIGI-COMP II as follows:
1. ADD
2. CLEAR
3. M = 1111 (15)
4. MQ = 100 (4)
5. START

Step 2: When the fifth ball is first triggered, change the MEMORY REGISTER to 1110 (14) by flipping M1 from "1" to "0".

Step 3: When the ninth ball is first triggered, set the MEMORY REGISTER to 1100 (12).
When the 13th ball is first triggered, flip the MEMORY REGISTER to 1101 (13).
The Machine will shut itself off on the 17th ball with the required sum in the ACCUMULATOR.

TYPES OF READOUT

According to terminology common to electronic computers, the four balls which arrive at points M1, M2, M3, M4 "Read" the MEMORY REGISTER in the process of interrogating each of these SWITCHES M4, M2, M3, M1. They do not happen to destroy the information in the MEMORY REGISTER in the process, however. Therefore, we call this type of readout, "Non-Destructive" to distinguish it from "Destructive" readout which changes or destroys the information during the process. When multiplication is explained later it will be seen that the number in the MQ REGISTER is "destroyed" during the operation of "Reading" it.

Remember, in order to add two numbers, the computer is first Programmed to ADD then,

A. Set the larger number to be added in the ACCUMULATOR.
B. Next, set the smaller number, which must be less than 16, in the MEMORY REGISTER.
C. Next, set the MQ REGISTER to (001).
D. Next, pull START.

PROBLEMS:

Add a succession of different numbers of your own choosing to the ACCUMULATOR verifying the results after each addition.

1. ADD 9 and 67.
2. ADD 47 and 13.
3. Several numbers can be added together using a method similar to that of Experiment V, by arranging them such that only one MEMORY SWITCH need be changed for each successive addition. For example, if you wish to add 2 + 8 + 10 + 12 + 14 they could be added in the order (14 + 12 + 8 + 10 + 2). If these are written in binary form,

M_1	0	0	0	0	0
M_2	1	0	0	1	1
M_3	1	1	0	0	0
M_4	1	1	1	1	0
	14	12	8	10	2

you can see that only one bit changes at a time. (In programming this problem you would enter 0101 in the MEMORY REGISTER because there are 5 numbers to be added.)

4. ADD 9 to 125.

This will cause an ACCUMULATOR OVERFLOW, but since the OVERFLOW SWITCH is on RUN, the computer will complete the addition even though the result is wrong.

What answer do you get when you add 9 = (1001) to 125 = (1111101)? The correct answer is (10000110) but this is an 8–bit number which is too large for the ACCUMULATOR. We might expect that the answer would be (0000110) with the eighth (or high order) bit lost. Is this true?

You can still add numbers which cause an OVERFLOW if you realize it and add 2^7 or 128 to the ACCUMULATOR mentally.

RULES OF BINARY ADDITION

Throughout this chapter, you have been observing DIGI-COMP II add in binary arithmetic. Let us review the rules that you have been using.

It is very easy to add two binary numbers. THERE ARE ONLY FIVE RULES TO REMEMBER:

RULE 1	0 + 0 = 0
RULE 2	0 + 1 = 1
RULE 3	1 + 0 = 1
RULE 4	1 + 1 = 0 with a "1" carry (read: one, zero; 10)
RULE 5	1 + 1 + 1 = 1 with a "1" carry (read: one, one; 11)

Compare this with the decimal addition table! A few examples are:

1 + 0 = 1 2 + 3 = 5 3 + 6 = 9
8 + 4 = 2 with a "1" carry (12).
9 + 7 = 6 with a "1" carry (16).

In fact, there are *65* addition rules in the decimal system!

As you know when two decimal numbers are added, the "carry" from one column is added to the digits of the next column. For example:

$$\begin{array}{lr} \textit{carry} \longrightarrow & \textit{111} \\ & 976 \\ & +\ \underline{425} \\ & 1401 \end{array}$$

Exactly the same thing is done in the binary system. For example:

$$\begin{array}{lr} \textit{carry} \longrightarrow & \textit{1111} \\ & 11011 \\ & +\ \underline{10110} \\ & 110001 \end{array}$$

In the binary system the "CARRIES " FROM ONE COLUMN ARE ADDED TO THE "BITS" OF THE NEXT COLUMN.

NOTE that when *more* than three ones are added, there will be more than one carry. As an example:

$$\begin{array}{lccccc} \textit{carry} \longrightarrow & (\,1\,) & (\,1\,) & & & \\ \phantom{\textit{carry}} \searrow & (\,1\,) & (\,1\,) & & & \\ & 1 & 1 & & 3 \\ & 0 & 1 & \text{or} & 1 \\ & 1 & 1 & & 3 \\ & +\ 1 & 1 & & +\ 3 \\ \hline & (\,101\,)_2 & 0 & & (\,10\,)_{10} \end{array}$$

CHAPTER 8

MULTIPLICATION OF BINARY NUMBERS ON DIGI-COMP II

QUESTIONS:

1. How does DIGI-COMP II perform Multiplication?
2. For speed of Multiplication, which number should be placed in the MQ REGISTER and which one should be placed in the MEMORY REGISTER?
3. How does a number change in binary when it is doubled or quadrupled?
4. What is the function of the "Multiplier" vs. the "Multiplicand"?
5. Can an OVERFLOW occur during Multiplication?
6. Why does the 7–bit capacity of the ACCUMULATOR equal the 3--bit capacity of the MQ REGISTER plus the 4–bit capacity of the MEMORY REGISTER?
7. How may "MULTIPLY-ADD" be performed?
8. Can an OVERFLOW occur during MULTIPLY-ADD?
9. What is meant by "Modulo" arithmetic?
10. How does an electronic computer MULTIPLY using SHIFTING?

Multiplication is accomplished in DIGI-COMP II by adding the number in the MEMORY REGISTER to the ACCUMULATOR REGISTER as many times as the value of the number placed in the MQ REGISTER. *This is simply repeated Addition.*
For instance if you wish to multiply 9 × 2, 9 = (1001) is added to the ACCUMULATOR 2 = (010) times. Four balls are required each time the MEMORY REGISTER is added to the ACCUMULATOR.

The first ball on each cycle of 4 balls subtracts one from the MQ REGISTER and asks if the MQ were zero before subtraction. If the answer is "yes", the computer turns itself off without further action, as a result of any MQ REGISTER OVERFLOW. If the answer is "no", however, this ball then goes to M4 of the MEMORY REGISTER and asks if (1000), or 8, shall be added to the ACCUMULATOR. In this case as M4 = 1 the answer is "yes" and (1000) is added to the ACCUMULATOR. The ball then triggers the next ball from the INPUT GUIDE.

The second ball on each cycle of 4 is directed to M2. If M2 = 1 the number 2 = (0010) is added to the ACCUMULATOR. In this case M2 = 0 so the ball drops through the hole and thereby avoids the ACCUMULATOR, but it returns to the first surface via the RETURN GUIDE and triggers a third ball in the cycle of four.

The third ball is directed to M3 which in this case is also zero. Therefore, the number 4 = (0100) is not added to the ACCUMULATOR but a fourth ball is triggered from the INPUT GUIDE.

The fourth ball is directed to M1 which, in this case, is a one. The number one is therefore added to the ACCUMULATOR. Thus, for every four balls, if 9 = $(1001)_2$, is in the MEMORY REGISTER, an "8" and a "1" will be added to the ACCUMULATOR because the bit worth "8" and the bit worth "1" are both "1". However, neither "4" nor "2" are added because these bits are "0".

On each cycle of four balls, the first ball subtracts one from the MQ unless the MQ is zero. For 2 × 9, the MQ is originally $(010)_2$, and the first ball and the fifth ball subtract one from the MQ leaving $(001)_2$, and then $(000)_2$. The ninth ball begins the third cycle of four, but finding the MQ equal to all zeros it subtracts one leaving $(111)_2$ producing an MQ OVERFLOW which turns the computer off.

The multiplication of 2 = (010) times 9 = (1001) is illustrated in Figure 4 and Figure 5.

Notice that the answer to 2 × 9 or $(10)_2 \times (1001)_2$ is $(0010010)_2$ in the ACCUMULATOR. This bit configuration is the same as that of 9, but is shifted left one place.

EXPERIMENT I:
MULTIPLYING AND SHIFTING

Program DIGI-COMP II to multiply 4 × 9 as follows:

1. CLEAR
2. MULTIPLY
3. M = 1001
4. MQ = 100

Notice that 4 × 9 is (0100100) or 36 in the ACCUMULATOR which is again the same bit configuration as 9 = (1001), but shifted left two places.

In general, if a binary number is multiplied by a power of 2, such as 2, 4, 8, 16, 32, etc., the resulting number is shifted left 1, 2, 3, 4, 5, etc., places respectively. This is because these powers are represented by $(10)_2$, $(100)_2$, $(1000)_2$, $(10000)_2$, etc., in binary and just as in base ten multiplying by 10, 100, 1000, 10000, etc., shifts the number left 1, 2, 3, 4, 5, etc., places respectively with zeros trailing to the right.

EXPERIMENT II:
SPEEDING UP THE MULTIPLICATION OPERATION

In multiplying two numbers such as 5 × 7 the amount of work that the computer must do can vary depending upon how a problem is programmed.
For instance, program DIGI-COMP II as follows:

1. CLEAR
2. MULTIPLY
3. M = 0111
4. MQ = 101
5. START

Count the number of marbles used in the calculation.

Now, perform the same multiplication, but this time put the 7 in the MQ and the 5 in the MEMORY REGISTER. Again count the number of balls triggered. Were they the same? Why not?

When 5 is the multiplier in the MQ REGISTER, five additions of 7 are performed to accomplish the multiplication, but when 7 is the multiplier in the MQ, seven additions of 5 are performed. In the first case, 5 cycles of 4 balls are required plus one more ball to discover that the MQ is zero which makes a total of 21 balls. In the second case, 7 cycles of 4 balls plus one are required which makes a total of 29 balls. For speed of multiplication, then, the smaller number should be placed in the MQ and the larger number in the MEMORY REGISTER.

EXPERIMENT III:
THE LARGEST POSSIBLE NUMBERS THAT CAN BE MULTIPLIED

Program DIGI-COMP II to multiply and also CLEAR the ACCUMULATOR. Multiply the largest possible number :hat can be put in the MQ by the largest possible number than can be put in the MEMORY REGISTER. What is the result in the ACCUMULATOR? Is it correct? Did an OVERFLOW occur?

The largest number that can be placed in the MQ is $(111)_2 = 7$. The largest number than can be put in the MEMORY REGISTER is $(1111)_2 = 15$. Therefore, the answer should be 7 × 15 = 105. In binary, 105 is (1101001).

As 105 in binary does not require more than 7–bits, no ACCUMULATOR OVERFLOW results and the answer is correct.

In base ten, if any 3 digit number is multiplied by any 4 digit number, the answer will never require any more than 7 digits to represent. Similarly, in binary the product of any 3–bit number by any 4–bit number requires no more than 7–bits to represent. The reason that the ACCUMULATOR was designed to contain 7–bits is that its size must be equal to the size of the MQ, 3–bits, plus the size of the MEMORY REGISTER, 4–bits. As a result of the choice of this size, no ACCUMULATOR OVERFLOW can occur during multiplication if the ACCUMULATOR is initially set to zero before the operation is begun.

EXPERIMENT IV:

MULTIPLY-ADD OPERATION

Compute 4 × 10 + 9. The program for this is:

1. MULTIPLY
2. M = 1010
3. MQ = 100
4. A = 0001001
5. START

The computer will now multiply four times ten and add the result to nine. As previously indicated, DIGI-COMP II *performs multiplication by repeated addition.* Therefore, 10 = (1010) will be added to the ACCUMULATOR 4 = (100) times. But the ACCUMULATOR initially contains the number 9 = (1001) and therefore the answer will be (4 × 10 + 9) or 49 = (0110001).

The MULTIPLY-ADD function then gives us an easy means of having the computer convert a decimal number such as 49 to binary.

Let us try another example.

Convert 57 to binary using the computer to help perform the conversion.

EXPERIMENT V:

CONVERTING DECIMAL BINARY AUTOMATICALLY

To convert 57 to binary, you can think of it as being 5 × 10 + 7. Program DIGI-COMP II for this problem and the ACCUMULATOR will turn up the binary equivalent of 57.

The program is:

1. MULTIPLY
2. M = 1010
3. MQ = 101
4. A = 0000111
5. START

What is the largest decimal number that can be converted to binary by this method? Why?

As the MQ REGISTER can hold a maximum size number of 7 = $(111)_2$, the largest number which can be converted to binary by the method is 79. However, it is possible to convert still larger numbers by separating the work into two parts. The number 123 can be converted by expressing it as 64 plus 59, for example, and using the computer to convert the 59. In this case 59 is converted by computing (5 × 10 + 9) obtaining $(0111011)_2$. Then 64 = $(1000000)_2$ is mentally added to the result obtaining:

$$123 = (1111011)_2$$

EXPERIMENT VI:

PROBLEM: COMPUTE $(5 \times 13) + (4 \times 14)$

The technique is to multiply 4 × 14, then, without clearing the ACCUMULATOR, multiply 5 × 13. It will be automatically added.

The program is:

1. CLEAR
2. MULTIPLY
3. M = 1110
4. MQ = 100
5. START
6. PAUSE
7. MULTIPLY
8. M = 1101
9. MQ = 101
10. START

Check your answer by doing the program in decimal and converting to binary, or directly as follows:

$$4 \times 14 = (100)_2 \times (1110)_2 = (0111000)_2$$
$$5 \times 13 = (101)_2 \times (1101)_2 = (1000001)_2$$
$$5 \times 13 + 4 \times 14 = (1000001)_2 + (0111000)_2$$
$$5 \times 13 + 4 \times 14 = (1111001)_2$$

EXPERIMENT VII:

COMPUTE $(6 \times 13 + 5 \times 15)$

Program this problem the same as Experiment 6 except for the values of M and MQ. If the result is larger than the ACCUMULATOR can hold, this may be detected by performing the operation with the OVERFLOW SWITCH on HALT.

Let us summarize the simple rules of binary multiplication. Binary multiplication is carried out in a way similar to decimal multiplication. The binary multiplication table is:

$$1 \times 0 = 0 \qquad 0 \times 1 = 0$$
$$0 \times 0 = 0 \qquad 1 \times 1 = 1$$

This is even simpler than binary addition!
To multiply in the binary system you multiply by each bit of the multiplier and add. This is the same as in the decimal system.

For example:

```
        In binary:          In decimal:

           1 1 0 1                 1 3
     ×     1 1 0 1            ×    1 3
     -------------           ---------
           1 1 0 1                 3 9
         0 0 0 0
       1 1 0 1                   1 3
     1 1 0 1
     -------------           ---------
     1 0 1 0 1 0 0 1             1 6 9
```

CHAPTER 9

COMPLEMENT ARITHMETIC AND SUBTRACTION

QUESTIONS:

1. What is the "1's" COMPLEMENT of a binary number?
2. What is the "2's" COMPLEMENT or more simply, the complement of a binary number?
3. How does DIGI-COMP II obtain the COMPLEMENT of a binary number?
4. If the COMPLEMENT of a binary number is added to the original number, what is the result?
5. If the COMPLEMENT of a smaller number is added to a larger number, what does the result represent?
6. How may subtraction be performed by COMPLEMENT addition?
7. What is the COMPLEMENT of zero? Of 1, 2, 3, etc.?
8. What is the COMPLEMENT of the COMPLEMENT of any number?
9. How are negative numbers represented in COMPLEMENT arithmetic?
10. What should the number $(1000000)_2$ represent in COMPLEMENT arithmetic?
11. How do you make an automatic marble counter?
12. Could we have base ten COMPLEMENT arithmetic?
13. Why is COMPLEMENT arithmetic preferred to true subtraction and negative numbers?

If the number 25 were placed in the ACCUMULATOR, it would be represented by 0011001 as this register in DIGI-COMP II has 7–bits.

By definition, the "ONE'S COMPLEMENT" of a binary number in a register is that number which, if added to the REGISTER, would produce all one's in the register. The ONE'S COMPLEMENT of 25 in the ACCUMULATOR REGISTER would be 1100110 as this number plus 0011001 would equal 1111111.

The ONE'S COMPLEMENT of a binary number in a REGISTER may be obtained quite simply by the following rule. Change every bit in the register to its opposite value. Thus, if we change every bit of 0011001 to its opposite value we obtain 1100110 which is its ONE'S COMPLEMENT.

EXPERIMENT I:

THE ONE'S AND TWO'S COMPLEMENT

To illustrate using DIGI-COMP II program as follows:

1. INITIALIZE
2. COMPLEMENT = ON
3. AM = MANUAL
4. A = 0011001
5. START

A ball will be triggered to enter CF1, the FIRST COMPLEMENT FLIP-FLOP from left to right and deliver the ball to the hole at its right. The ball will then drop to the hole second surface and flip T1, T2, T3, T4, T5, T6, which are called the ACCUMULATOR MODE FLIP-FLOPS, from their normal left positions to their right positions. It will then return to the first surface via the RETURN GUIDE and be collected in the COLLECTOR GUIDE. The ACCUMULATOR is now said to be in the COMPLEMENT MODE rather than the ADD MODE because the MODE FLIP-FLOPS are to the right rather than to the left.

Pull START again.

The second ball will enter CF1 and flip it from the right position back to its original left position. Having flipped CF1, the ball will enter the ACCUMULATOR at A1 and flip every bit of the ACCUMULATOR to its opposite values. In other words, the ACCUMULATOR now contains 1100110 which is the ONE'S COMPLEMENT of 0011001. The second ball also flips CF2 to its opposite position.

Normally, if a ball flips an ACCUMULATOR FLIP-FLOP from "1" to "0", it carries a "1" to the next FLIP-FLOP, but if it flips an ACCUMULATOR FLIP-FLOP from "0" to "1" it does not. When the ACCUMULATOR is in the COMPLEMENT MODE, a ball is directed to the next FLIP-FLOP in both cases. This is because the corresponding MODE FLIP-FLOP directs any ball exiting to the right, back to the next lower FLIP-FLOP.

Pull START again.

The third ball will follow the same path as the first ball flipping CF1 to the right, but this time the MODE FLIP-FLOPS, T1 through T6, will be flipped back to the left returning the ACCUMULATOR to the ADD MODE.

Pull START again.

The fourth ball now flips CF1 back to its original (left) position, adds one to the ACCUMULATOR and flips CF2 back to its original (right) position. The ACCUMULATOR now contains (1100110) + (1) = (1100111). This number is called the TWO'S COMPLEMENT of the original number, (0011001).

By definition, the TWO'S COMPLEMENT of a given binary number in the ACCUMULATOR is that number which, if added to the given number, would produce all zeros with an ACCUMULATOR OVERFLOW. The OVERFLOW takes place as a result of an attempt to carry a one to the next higher, non-existent bit. The TWO'S COMPLEMENT of a binary number in a register may be simply obtained by first obtaining the ONE'S COMPLEMENT, then adding one to the result.

EXPERIMENT II:

If we add the number (1100111) to our original binary number (0011001), we obtain to 8–bits (10000000). But we have only 7–bits in the ACCUMULATOR and hence the result is (0000000) with an ACCUMULATOR OVERFLOW. Let us illustrate this using DIGI-COMP II.

Place all of the marbles in the top and program it as follows:

1. COMPLEMENT = OFF
2. COUNT = ON
3. OVERFLOW = HALT
4. AM = AUTO
5. START

The ACCUMULATOR will add one every few seconds until an ACCUMULATOR OVERFLOW occurs. As the OVERFLOW SWITCH is set to HALT, the computer will turn itself off when the overflow occurs.

Count the number of balls that are in the COLLECTOR GUIDE when the OVERFLOW occurs. How many are there?

There should be exactly 25 balls in the COLLECTOR GUIDE as 25 plus the COMPLEMENT of 25 should give zero with an ACCUMULATOR OVERFLOW.

EXPERIMENT III:

This technique may be used to make DIGI-COMP II count out a desired number of balls. Let us give another example to illustrate this point. Suppose it were desired to have the computer count out 9 balls.

1. INITIALIZE
2. A = 0001001
3. COMPLEMENT = ON
4. START
5. PAUSE
6. COMPLEMENT = OFF
7. OVERFLOW = HALT
8. COUNT = ON
9. START

Place the four balls in the COLLECTOR GUIDE back up in the INPUT GUIDE. The computer will now count until an OVERFLOW occurs. As the OVERFLOW SWITCH is in its HALT position, the computer will shut itself off when OVERFLOW occurs. Exactly 9 balls will be counted out as 9 plus the TWO'S COMPLEMENT of 9 are zero with an ACCUMULATOR OVERFLOW.

From now on, we will refer to the TWO'S COMPLEMENT of a number in a register as the "COMPLEMENT". This is because the TWO'S COMPLEMENT, or the COMPLEMENT, is much more important to the operation of the computer than the ONE'S COMPLEMENT.

The following is a list of the tasks DIGI-COMP II performs when it develops the COMPLEMENT of a binary number in the ACCUMULATOR.

A. The MODE FLIP-FLOPS are flipped from their normal ADD positions (to the left) to their COMPLEMENT positions (to the right) by the first ball.
B. The ONE'S COMPLEMENT of the ACCUMULATOR is next obtained by the second ball by flipping each bit in the ACCUMULATOR to its opposite value.
C. The MODE FLIP-FLOPS are next flipped back from their COMPLEMENT positions (to the right) to their ADD positions (to the left) by the third ball.
D. One is added to the result in the ACCUMULATOR by the fourth ball.

EXPERIMENT IV:
PROBLEM: FIND THE COMPLEMENT OF 12.

To do this:

1. COMPLEMENT
2. A = 0001100
3. START

The computer will automatically trigger four consecutive balls and develop the COMPLEMENT of 12 in the ACCUMULATOR in binary. This COMPLEMENT will be the ONE'S COMPLEMENT of 12, which is (1110100). Let us now add "12" to the ACCUMULATOR and see if the result is zero.

1. ADD
2. M = 1100
3. MQ = 001
4. START

Five balls will be triggered, all "0's" will be developed in the ACCUMULATOR, and an ACCUMULATOR OVERFLOW will occur. The computer will stop because the addition is complete, not because an ACCUMULATOR OVERFLOW occurred. This because the OVERFLOW SWITCH is on RUN, not on HALT.

EXPERIMENT V:
SUBTRACT 9 FROM 15

Now let's add 15 to the COMPLEMENT of 9. If 9 were added to the COMPLEMENT of 9 we would obtain zero. As 15 may be represented as 9 + 6, if 15 were added to the COMPLEMENT of 9 we would expect the result to be:

(COMPLEMENT of 9) + 9 + 6 = 0 + 6 = 6

In other words, 15 plus the COMPLEMENT of 9 would equal (15 - 9) or 6! Thus, with COMPLEMENT arithmetic, subtraction can be changed to addition. We shall illustrate this point with further experiment with DIGI-COMP II.

Now to add 15 to the COMPLEMENT of 9, the program is:

1. A = 0001001
2. COMPLEMENT
3. START
4. PAUSE
5. MQ = 1111
6. ADD
7. START

The ACCUMULATOR will develop 6 = (0000110) with an OVERFLOW occurring in the process. As the OVERFLOW SWITCH is on RUN, the computer will not halt when the OVERFLOW takes place.

Let us review subtraction briefly. To compute (15 - 9) we add the COMPLEMENT of 9 to 15. From now on if we place a bar above a number, as for example $\overline{9}$, this will mean the complement of that number. Using this notation, 15 plus the COMPLEMENT of 9 would be written:

$$15 + \overline{9}$$

It should be clear that:

$$15 - 9 = 15 + \overline{9} = 6$$

In other words, 15 minus 9 is the same as 15 plus the complement of 9.

EXPERIMENT VI:
WHAT IS THE COMPLEMENT OF 0, 1, 2, 3?

Program DIGI-COMP II for COMPLEMENT according to the chart on the front of the computer and enter each digit to find the answers.

If you followed the program correctly, the following answers were obtained:

$$\overline{0} = (0000000)_2$$

$$\overline{1} = (1111111)_2$$

$$\overline{2} = (1111110)_2$$

$$\overline{3} = (1111101)_2$$

Notice that the COMPLEMENT of "0", called "$\overline{0}$", is the same as the number itself, namely "0".

What is the COMPLEMENT of the COMPLEMENT of 23? of 47? or 52?

Program DIGI-COMP II and enter each number in turn. To check the answers, RE-COMPLEMENT them and see if you get the original number.

The answers to these questions are as follows:

$$23 = (0010111)_2, \quad \overline{23} = (1101001)_2, \quad 23 = (0010111)_2$$

$$47 = (0101111)_2, \quad \overline{47} = (1010000)_2, \quad 47 = (0101111)_2$$

$$52 = (0110100)_2, \quad \overline{52} = (1001011)_2, \quad 52 = (0110100)_2$$

It will be noticed in every case that the COMPLEMENT of the COMPLEMENT of a number is equal to the original number.

The COMPLEMENT of a number behaves in every respect, that we have considered thus far, as the negative of a number behaves.

A. The COMPLEMENT of "0" is "0". The negative of "0" is "0".
B. The COMPLEMENT of the COMPLEMENT of a number is the original number, just as the negative of the negative of a number is the original number.
C. A number plus its COMPLEMENT is zero just as a number plus its negative is zero.
D. A number plus the COMPLEMENT of another number is the difference of the two numbers just as a number plus the negative of another number is the difference of the two numbers.

It is, therefore, possible to represent the negative of a number by its COMPLEMENT and accomplish subtraction by the same circuitry that is used for addition. However, some further care needs to be exercised. The complement of 23 called $\overline{23}$ was found to be (1101001). If this number were read as a positive number, we would call it:

$$\begin{array}{ccccccccc} & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \\ \hline (& 1 & 1 & 0 & 1 & 0 & 0 & 1 &)_2 \end{array}$$

$$64 + 32 + 0 + 8 + 0 + 0 + 1 = 105$$

Yet, of course, the negative of 23 is not 105.

We avoid this inconsistency as follows:

If we wish the ACCUMULATOR *to represent both positive and negative numbers, we consider that any number with a "1" in the high order position is a negative number, and any number with a "0" in the high order position is positive.* Thus, the number 1001110 is a negative number in COMPLEMENT form because its high order, or left-most bit, is a "1".

EXPERIMENT VII: WHAT NEGATIVE NUMBER DOES 1001110 REPRESENT?

While the computer is still set up for COMPLEMENTING, place this number in the ACCUMULATOR and START.

DIGI-COMP II will develop the COMPLEMENT of 1001110 which is 0110010. This number may now be evaluated as a positive number because the high order bit is "0".

$$32 + 16 + 0 + 0 + 2 + 0 = 50$$

Therefore, 1001110 must be the COMPLEMENT representation for the negative of 50, or for -50.

EXPERIMENT VIII:

What is the highest positive number that can be represented if both positive and negative numbers are to be considered in a 7–bit ACCUMULATOR.

To answer this, we must ask – what is the highest number that can be represented by a 7–bit number with a "0" in the high order position?

The answer is 0111111 which is:

$$\begin{array}{ccccccccc} & & 32+ & 16+ & 8+ & 4+ & 2+ & 1 & \\ \hline (& 0 & 1 & 1 & 1 & 1 & 1 & 1 &)_2 \end{array}$$

$$32 + 16 + 8 + 4 + 2 + 1 = 63$$

EXPERIMENT IX: WHAT IS $\overline{63}$?

Or, in other words, what is the representation for -63 to 7-bits in COMPLEMENT form? Use DIGI-COMP II to obtain this result by placing 0111111 in the ACCUMULATOR and PROGRAMMING to COMPLEMENT.

The answer is (1000000) + 1 or (1000001)

It is important to realize that we can either consider that the ACCUMULATOR on DIGI-COMP II represents *positive* numbers only, going from "0" to "127", or we can consider that the ACCUMULATOR represents *positive* and *negative* numbers going from -63 to +63. In the first case the limits 0 to 127 are represented by (0000000) to (1111111) in binary. In the second case the limits of -63 to +63 are represented by (1000001) to (0111111 $)_2$. Thus, if we add "1" to "-63", for example, we obtain (1000010 $)_2$ which is 62. Therefore, (1000010 $)_2$ represents a "-62" as we would expect.

When considering that the ACCUMULATOR represents both positive and negative numbers, a positive number may be evaluated directly to determine its decimal equivalent. A negative number must first be complemented, however, before we may evaluate it. If a number in the ACCUMULATOR has a "1" bit in its high order position (bottom bit on DIGI-COMP II), we infer that it is a negative number and we must COMPLEMENT the number to determine its value.

If the ACCUMULATOR is considered to represent both positive and negative numbers, what are the decimal equivalents of:

$(1000101)_2$? $(1100000)_2$? $(1010101)_2$? and $(1001001)_2$?

To find the answer, place each of these numbers in the ACCUMULATOR and then COMPLEMENT them.

The COMPLEMENTS of the numbers are:

$-(0111011)_2$, $-(0100000)_2$, $-(0101011)_2$, $-(0110111)_2$, or

	32	16	8	4	2	1	
-(0	1	1	1	0	1	1)	= -61
-(0	1	0	0	0	0	0)	= -32
-(0	1	0	1	0	1	1)	= -43
-(0	1	1	0	1	1	1)	= -55

EXPERIMENT X:
FIND THE COMPLEMENT OF (1000000)

Strangely enough, the COMPLEMENT of 1000000 = 1000000 or -1000000 = 1000000.

In algebra if a number "X" satisfies the equation -X = X, then it is necessary that X = 0. Could 1000000 be another way to represent zero? What would be the answer if we added 1000000 to 1? to 2? to 3? We would obtain 1000001, 1000010, and (1000011). As these numbers have a "1" in their high order bit, we must COMPLEMENT them to determine their value. COMPLEMENTING we obtain, 63, 62, 61. Therefore, 1000000 acts like -64 when we add it to 1, 2, 3. Suppose we were to add -1 to -63 by the rules of binary arithmetic. What would we obtain?

1 = 0000001 therefore, by COMPLEMENTING (-1) = 1111111, 63 = 0111111 therefore, by COMPLEMENTING, (-63) = 1000001 adding we obtain 1000000 plus an OVERFLOW.

For the reasons given it is perhaps most logical to interpret 1000000 as -64. There is no +64 using positive and negative numbers with a 7–bit ACCUMULATOR, however. Moreover, the COMPLEMENT of (-64) gives the erroneous answer of (-64). For our purposes, we will call this number (-64) if we encounter it recognizing, however, that inconsistencies can arise by using it.

We have shown how to compute (A - B) when "A" is less than or equal to the largest number, 15, which can be represented in the MEMORY REGISTER. The answer is computed as (A + $\bar{B}$), or A plus the COMPLEMENT of B, where $\bar{B}$ means the COMPLEMENT of B.

Suppose "A" is larger than 15, however. Then, although it is possible to place "B" in the ACCUMULATOR and COMPLEMENT "B", "A" may not next be added as "A" exceeds the size of the MEMORY REGISTER.

There is an alternate way in which (A - B) can be computed in this case to avoid this problem if "B", generally the smaller of the two numbers, is less than or equal to 15. To compute (A - B) in this case we compute (B + $\bar{A}$) then take the COMPLEMENT of the result. This is equivalent to recognizing that (A - B) is the negative of (B - A). Thus, we may write:

$$(A - B) = -(B - A)$$

or

$$(A - B) = \overline{(B + \bar{A})}$$

which means that we may compute (A - B) by adding "B" to the COMPLEMENT of "A", then taking the COMPLEMENT of the result.

EXPERIMENT XI: COMPUTE (35-13)

1. COMPLEMENT
2. A = 0100011 (35)
3. START
4. PAUSE
5. ADD
6. M = 1101
7. START
8. PAUSE
   (This produces the negative of the desired answer)
9. COMPLEMENT
10. START
   -(13 - 35) or (35 - 13) will now appear in the ACCUMULATOR.

The two methods of performing subtraction may be summarized as follows:

A. Compute (A - B) when "A" is less than or equal to 15, the maximum number the MEMORY REGISTER will hold.

   – Place "B" in the ACCUMULATOR.
   – COMPLEMENT the ACCUMULATOR.
   – Place "A" in the MEMORY REGISTER.
   – ADD the MEMORY REGISTER to the ACCUMULATOR.

B. Compute (A - B) when "B" is less than or equal to 15.
   – Place "A" in the ACCUMULATOR.
   – COMPLEMENT the ACCUMULATOR.
   – Place "B" in the MEMORY REGISTER.
   – ADD the MEMORY REGISTER to the ACCUMULATOR.
   – COMPLEMENT the ACCUMULATOR.

The question naturally arises, "could we have base ten complement arithmetic?" The answer to this question is "yes". In base ten arithmetic the 9's COMPLEMENT is analogous to the "1's" COMPLEMENT in binary. Let us assume we have a 5 digit base ten register, then the 9's COMPLEMENT of 342 would produce all 9's in the register. The 9's COMPLEMENT of 342 (or 00342) for a 5 digit register is 99657.

The ten's COMPLEMENT of 342 is that number which would produce all 0's in the register if it were added to 342 with an OVERFLOW attempt. It may be verified that in base ten the ten's COMPLEMENT of a number is its 9's COMPLEMENT plus 1. Thus, the ten's COMPLEMENT of 00342 is 99657 + 1 = 99658. Notice that:

```
    99658
 +  00342
 --------
   100000  =  00000 plus an OVERFLOW.
```

In base ten (8123 - 7645) may be computed by first obtaining the 9's COMPLEMENT of 07645 which is 92354, then adding 1 to obtain the COMPLEMENT 92355, then adding (8123 + 92355).

```
    08123
 +  92355
 --------
   100478  =  00478
```

COMPLEMENT arithmetic is used in computers instead of extra circuitry which would otherwise be required for true negative arithmetic. In the case of base ten, the high order "digit" or left-most digit must be reserved to indicate the algebraic sign of the number just as in binary. In binary if the high order bit were a "1", the number was considered to be negative. In base ten, if the high order digit is a "9", the number is considered to be negative. In base ten just as in binary, a negative number must be COMPLEMENTED to interpret its value.

DIVISION
OF BINARY NUMBERS ON DIGI-COMP II

QUESTIONS:

1. How is division performed on DIGI-COMP II?
2. What is the largest dividend and divisor that can be used?
3. What is the largest quotient that can be developed?
4. How can the DIVIDE command be further automated?
5. What happens in division if the divisor does not divide the dividend a whole number of times?
6. How can both the quotient and the remainder be calculated if the division results in a non-zero remainder?

Division is performed on DIGI-COMP II by repeated subtraction. To see how many times the divisor will divide (or go into) the dividend, the dividend is placed in the ACCUMULATOR, the divisor is placed in the MEMORY REGISTER, and the computer is made to count in the MQ REGISTER the number of times that the divisor can be subtracted from the dividend.

In the MQ or MULTIPLIER QUOTIENT REGISTER, the *lower set of numerals* is used for division. It will be recalled that the upper set of numerals was used for multiplication.

As subtraction is actually performed by COMPLEMENT ADDITION on DIGI-COMP II, a somewhat more accurate description of division may be given as follows. The dividend is first placed in the ACCUMULATOR REGISTER and is then COMPLEMENTED. This gives the negative of the dividend in the ACCUMULATOR. The divisor is then placed in the MEMORY REGISTER and the MQ REGISTER is set to zero. The OVERFLOW SWITCH is then set to HALT and the DIVIDE operation is begun by pulling START.

On each cycle of 4 balls the MEMORY REGISTER, containing the divisor, is added to the ACCUMULATOR, containing the negative of the dividend, and the count of how many such additions have thus far been performed is increased by one in the MQ REGISTER. The operation stops when the ACCUMULATOR OVERFLOWS (goes from minus to zero).

EXPERIMENT I:
DIVIDE 45 BY 9

To do this, the program is:
1. A = 0101101
2. M = 1001
3. DIVIDE

The machine will halt on an ACCUMULATOR OVERFLOW with the answer $5 = (101)_2$ appearing in the MQ REGISTER on the lower set of numerals. Notice that on each cycle of 4 balls the MQ is increased by one by the first ball of the group if the lower set of numerals are read.

EXPERIMENT II: DIVIDE 52 BY 13

1. A = 0110100
2. M = 1101
3. DIVIDE

DIGI-COMP II performs division by repeated addition of the divisor to the COMPLEMENT, or negative, of the dividend until an ACCUMULATOR OVERFLOW stops the machine. The quotient is obtained by counting the number of additions required. Remember that when both positive and negative numbers are considered to be represented in the ACCUMULATOR, the limits of numbers that may be represented are from −63 to +63. Therefore, it would be expected that the dividend could not be greater than +63 and have the division be correct. The division does come out correct following the same procedure, however, even for dividends greater than +63.

EXPERIMENT III: DIVIDE 78 BY 13

1. A = 1001110
2. M = 1101
3. DIVIDE

The computer should add 13 to the COMPLEMENT of 78 exactly 6 times before the machine halts on an ACCUMULATOR OVERFLOW. The quotient of 6 will appear in the MQ REGISTER.

Strictly speaking, DIGI-COMP II performs division automatically only in the case that the divisor goes into the dividend a whole number of times. With additional manual assistance, division may be also performed in the case where both a quotient and a remainder need be obtained.

EXPERIMENT IV: HOW TO HANDLE REMAINDERS

To illustrate how division may be performed to obtain a quotient and a remainder, let us divide 50 by 11. The computer will be made to add 11 to the COMPLEMENT, or negative, of 50 as many times as required to obtain an ACCUMULATOR OVERFLOW. This will be 5 times as 55 = 5 × 11 is larger than 50; but 44 = 4 × 11 is smaller. The machine having recognized that it requires 5 additions of 11, will stop on an OVERFLOW part way through the fifth addition of 11. It will be necessary to complete this fifth cycle of four balls manually.

Having added 11 to the ACCUMULATOR 5 times, the result in this Register will be 55 - 50 = 5. The correct remainder, however, is 11 minus this number or 11 plus the COMPLEMENT of this number. Therefore, it will next be necessary to COMPLEMENT the ACCUMULATOR and to add 11 to obtain the true remainder. The correct quotient is one less than the number appearing in the MQ REGISTER.

To illustrate this problem, program DIGI-COMP II as follows:

Step 1:
1. A = 0110010
2. M = 1011
3. DIVIDE

Step 2: When the computer stops on an ACCUMULATOR OVERFLOW, set the AM SWITCH to MANUAL, and pull START as many times as required to return the DISTRIBUTOR elements D1, D2, D3 to their correct initial positions, all to the right. This completes the fifth cycle.

Step 3: Turn the MULTIPLY SWITCH to its OFF position, turn the OVERFLOW SWITCH, back on RUN then COMPLEMENT the ACCUMULATOR.

We are now ready to add 11 to the COMPLEMENT of the ACCUMULATOR. As the quotient in the MQ REGISTER is one too large, we may also subtract one from the MQ at the same time.

Step 4: Manually flip every element in the MQ REGISTER to its opposite position. Now the 5 = (101) which appeared on the lower numerals will appear on the upper numerals. Also, the first ball on the next cycle of four will subtract one from the MQ if we consider the top set of numerals.

Step 5: Turn the MULTIPLY SWITCH to its ON position and pull START four consecutive times. This will add the 11 to the COMPLEMENT of the ACCUMULATOR.

Step 6: We may now read the quotient in the MQ using the top numerals and the remainder in the ACCUMULATOR.

Although the steps necessary to obtain both the quotient and the remainder in division are numerous and complicated, they can easily be remembered if the principles involved are clearly understood. These principles are summarized as follows:

A. The divisor is placed in the MEMORY REGISTER and it is added to the COMPLEMENT, or negative, of the dividend as many times as required to make the computer halt on an ACCUMULATOR OVERFLOW. The MQ REGISTER, which is initially set to zero, counts the addition performed.

B. The last cycle must be completed manually by pressing START a sufficient number of times to return D1, D2, D3, in the DISTRIBUTOR to their correct initial positions, all to the right.

C. The MQ REGISTER must be reduced by one to obtain the quotient in the process of adding the divisor back into the COMPLEMENT of the ACCUMULATOR. The ACCUMULATOR will then contain the remainder.

Complicated as this procedure sounds, an electronic computer follows a similar but even more complicated sequence procedure to perform division. However, the division is completely automatic and the user need not be concerned with the process. Just as in the procedure followed in DIGI-COMP II, repeated subtraction is performed by COMPLEMENT ADDITION but the ACCUMULATOR is shifted in the process to shorten the number of cycles and the time required. Again, as with DIGI-COMP II, overdraws occur which are detected by OVERFLOWS and they are added back in to develop the answer.

PROBLEMS:

1. DIVIDE 63 by 21
2. DIVIDE 84 by 21 (Will you get the correct answer even though the dividend is larger than 64?)
3. DIVIDE 47 by 14 (What is the remainder?)
4. DIVIDE 63 by 11 (What is the remainder?)

CHAPTER 11

BINARY FRACTIONS

The following discussion, while largely mathematical, illustrates a very important part of binary mathematics and its application to digital computers.

In base ten the number .3708 is called a *decimal fraction* and the period in front of the number is called the *decimal point.* This same decimal fraction can be expressed in several different forms, as for example: $3 \times (1/10) + 7 \times (1/100) + 0 \times (1/1000) + 8 \times (1/10000)$ or by obtaining a common denominator as, $\frac{3708}{10000}$

By similar reasoning, the binary number (.1101) is called a *binary fraction* and the period is called the *binary point.* Just as in decimal this binary fraction may be expressed as,

$$1 \times (1/10)_2 + 1 \times (1/100)_2 + 0 \times (1/1000)_2 + 1 \times (1/10000)_2$$

where it must be remembered that,

$$(1/10)_2 = 1/2,\ (1/100)_2 = 1/4,\ (1/1000)_2 = 1/8,\ (1/10000)_2 = 1/16$$

in base ten. We may express this binary fraction more simply as,

$$(.1101)_2 = 1/2 + 1/4 + 1/16$$

if we wish. Also, just as in the base ten case we may obtain a common denominator and express the fraction as,

$$\frac{(1101)_2}{(10000)_2} = \frac{13}{16}$$

Where the denominator, as in decimal fractions, is always one place larger than the number of places to the *right* of the binary point of the binary fraction.

This provides an easy way of converting a binary fraction to a decimal fraction.

Suppose we are given the binary fraction $(.11011)_2$, for example, and asked to convert it to decimal. Beginning at the left, the binary portions are worth 1/2, 1/4, 1/8, 1/16, 1/32. Therefore, we may express the fraction as,

$$\frac{(11011)_2}{32}$$

Now by converting the binary integer in the numerator to decimal we obtain,

$$(\overset{16}{1}\ \overset{8}{1}\ \overset{4}{0}\ \overset{2}{1}\ \overset{1}{1})_2 = 27$$

and therefore the answer is, $\frac{27}{32}$

PROBLEMS:

Convert the following binary fractions to base ten fractions,

A. .1001
B. .0111
C. .11111
D. .010101

ANSWERS:

A. $.1001 = \frac{(1001)_2}{16} = \frac{9}{16}$

B. $.0111 = \frac{(1.11)_2}{16} = \frac{7}{16}$

C. $.11111 = \frac{(11111)_2}{32} = \frac{31}{32}$

D. $.010101 = \frac{(10101)_2}{64} = \frac{21}{64}$

Of course, these fractions could be further converted to decimal fractions by performing the division. The fraction (9/16), for example, is .5625 and therefore $(.1001)_2 = .5625$ in decimal.

Every binary fraction (of a limited length) if converted to a decimal fraction results in a decimal fraction of a limited length. However, the opposite is not true. Suppose we are given a decimal fraction such as .7 and asked to convert it to a binary fraction. This simple decimal fraction results in an unending binary fraction.

One of the simplest methods of converting a decimal fraction, such as .7, to binary is to begin by representing it as the original decimal fraction in the numerator of a fraction whose denominator is one. For example, $.7 = \frac{.7}{1}$

Next, we multiply the numerator and denominator of this fraction by two which of course does not change the value of the fraction. Thus,

$$.7 = \frac{1.4}{2}$$

But $\left(\frac{1.4}{2}\right)$ can be further expressed as,

$$\frac{1.4}{2} = \frac{1 + .4}{2} = \frac{1}{2} + \frac{.4}{2}$$

Continuing in like manner with $\left(\frac{.4}{2}\right)$ we obtain,

$$.7 = \frac{.7}{1} = \frac{1.4}{2} = \frac{1}{2} + \frac{.4}{2} = \frac{1}{2} + \frac{.8}{4} = \frac{1}{2} + \frac{1.6}{8} = \frac{1}{2} + \frac{1}{8} + \frac{.6}{8} = \frac{1}{2} + \frac{1}{8} + \frac{1.2}{16}$$

$$= \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{.2}{16} \quad \text{etc.}$$

Thus, .7 can be represented as,

$$.7 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} + \frac{.2}{16}$$

or as,

$$.7 = (.1011)_2 + \frac{.2}{16}$$

By continued multiplication,

$$.7 = (.1\overbrace{0110}^{A}0110011001100110 \ldots \text{Repeats A})_2$$

This rule may be followed mechanically to convert a decimal, such as .3, as follows:

$2 \times .3 = .6$	0
$2 \times .6 = 1.2$	1
$2 \times .2 = .4$	0
$2 \times .4 = .8$	0
$2 \times .8 = 1.6$	1
$2 \times .6 = 1.2$	1
$2 \times .2 = .4$	0
$2 \times .4 = .8$	0
etc.	

Then, $.3 = (.01001100110 \ldots \text{ Repeats A })_2$

Notice that to 6 binary digits,

$$.3 \approxeq (.010011)_2$$

where $\approxeq$ means "approximately equal to". Let us check the reasonableness of this by expressing $(.010011)_2$ as a decimal fraction.

$$(.010011)_2 = \frac{(10011)_2}{64} = \frac{19}{64} = .296875$$

$$\text{as, } 64 \overline{\smash{)}\,19.000000}\quad .296875$$

It is interesting to note that a binary fraction containing N bits after binary point, with the last bit a "1", requires N decimal places to exactly express the binary fraction as a decimal.

Convert the following decimals to binary.

A. .625
B. .35
C. .6
D. .425
E. .13

Answers,

A. $.625 = .101$
B. $.35 \;\; = .01\overbrace{0110}^{A}011001100110 \ldots$
C. $.6 \;\;\; = .10\overbrace{0110}011001100110 \ldots$
D. $.425 = .011\overbrace{0110}011001100110 \ldots$
E. $.13 \;\; = .\overbrace{00100001010001111010 1} \ldots$

Notice that in each of the numbers .35, .6, .425, .3, .7 the binary digits (0110) repeat themselves. However, in the number .13 this cycle of 4 digits does not occur. This suggests that an explanation of this peculiar result might be interesting to pursue.

EXPERIMENT I:

It is customary to divide the English measure of distance called the "inch" into halves, fourths, eights, sixteenths, etc. These divisions behave very well in binary point but not so well in decimal point. For instance suppose we wished to add the measurements:

1 1/2" + 7/8" + 1 3/4" + 1 5/8" + 1 3/8". We could represent these fractions in *binary* as 1.1 + .111 + 1.11 + 1.101 + 1.011

Now, if we consider that the MEMORY REGISTER on DIGI-COMP II represents a binary fraction of the form X.XXX we can add these numbers as follows, (You can assume there is a binary point anywhere, in any *register*. *Just remember where it is!*)

$$S = 1.100 + 0.111 + 1.110 + 1.101 + 1.011$$

Now: Compute the sum of the given measurements on DIGI-COMP II.

Try to write the program for this problem. Then check below to be sure it is correct before programming DIGI-COMP II.

1. A = 0001.100
2. M = 0.111
3. ADD
4. PAUSE
5. M = 1.110
6. ADD
7. PAUSE
8. M = 1.101
9. ADD
10. PAUSE
11. M = 1.011
12. ADD

ADD the measurements manually and compare them with the binary answer in the ACCUMULATOR.

CHAPTER 12

SUMMARY OF INFORMATION NECESSARY TO OPERATE DIGI-COMP II

1. *Naming the important components.* (Consult Figure 1)
   - 1.1 *There are 6 Control Switches:*

	Name	*Position* *Left*	*Right*
1.1.1	MULTIPLY	OFF	ON
1.1.2	CLEAR	OFF	ON
1.1.3	COUNT	OFF	ON
1.1.4	COMPLEMENT	ON	OFF
1.1.5	OVERFLOW	RUN	HALT
1.1.6	AM	AUTO	MANUAL

   - 1.2 There are 3 REGISTERS which hold binary numbers:
     - 1.2.1 The ACCUMULATOR REGISTER.
       - 1.2.1.1 Has a capacity of 7–bits.
       - 1.2.1.2 Is located at right side of computer.
       - 1.2.1.3 Performs most of the arithmetic.
     - 1.2.2 The MEMORY REGISTER.
       - 1.2.2.1 Has a capacity of 4–bits.
       - 1.2.2.2 Is located in the middle of the computer.
       - 1.2.2.3 Does not perform arithmetic but holds numbers for use by the ACCUMULATOR.
     - 1.2.3 The MQ REGISTER
       - 1.2.3.1 Has a capacity of 3–bits.
       - 1.2.3.2 Is located at the left side of computer.
       - 1.2.3.3 Performs arithmetic for MULTIPLY and DIVIDE.
       - 1.2.3.4 Has two sets of numerals on the indicators. The top set of numerals is used for multiplication. The bottom set of numerals is used for division.
   - 1.3 There is a START SWITCH at the lower right which doubles as a Gating Mechanism when the AM SWITCH is on AUTO. Balls will be directed to roll past the switch thus automatically gating a successor ball.
   - 1.4 There is "other circuitry" which is used by the computer.
     - 1.4.1 The DISTRIBUTOR is comprised of 3 FLIP-FLOPS, D1, D2, D3, and is used to distribute consecutive balls along 4 different paths. *It is used in addition, subtraction, multiplication and division.*
     - 1.4.2 The ACCUMULATOR MODE FLIP-FLOPS are comprised of 6 FLIP-FLOPS, T1, T2, T3, T4, T5, T6. They are normally all to the left, in which case the ACCUMULATOR is in the "ADD MODE". During the COMPLEMENT operation they will be set to the right, in which case the ACCUMULATOR is in the "COMPLEMENT MODE", after which they will then be returned to the left.
     - 1.4.3 The first and second COMPLEMENT FLIP-FLOPS are called CF1 and CF2. CF1 is initially set to the left and CF2 to the right.
   - 1.5 There are two important guides which hold the balls while operations are performed.
     - 1.5.1 The INPUT GUIDE at the top of the computer.
     - 1.5.2 The COLLECTOR GUIDE at the bottom of the computer.
2. *Reading the Register.*
   - 2.1 All three REGISTERS hold binary or base two numbers only. These numbers are comprised of strings of "0's" and "1's".
   - 2.2 To convert a binary number such as $(0110101)_2$ to decimal it is necessary to associate each digit position with one of the powers of two in the following manner:

$$\begin{array}{ccccccc} \mathit{64} & \mathit{32} & \mathit{16} & \mathit{8} & \mathit{4} & \mathit{2} & \mathit{1} \\ (\,0 & 1 & 1 & 0 & 1 & 0 & 1\,)_2 \end{array}$$

Whenever a "1" appears the power of two above it is added into a sum, but whenever a "0" appears nothing is added. Thus, in the example we obtain,

$$0 + 32 + 16 + 0 + 4 + 0 + 1 = 53$$

and $(0110101)_2 = 53$

2.3 To convert a decimal number, such as 105, to binary it is necessary to subtract the highest power of two that can be subtracted from the number and place a "1" in the corresponding position. Then it is necessary to subtract from the remainder to form a new remainder and place a "1" in that position, etc. For example, to convert 105 to binary,

105–64	= 41	place a 1 under 64
41–32	= 9	place a 1 under 32
9–16	= cannot be subtracted	place a 0 under 16
9–8	= 1	place a 1 under 8
1–4	= cannot be subtracted	place a 0 under 4
1–2	= cannot be subtracted	place a 0 under 2
1–1	= 0	place a 1 under 1

$$\begin{array}{lccccccc} & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 105 = (& 1 & 1 & 0 & 1 & 0 & 0 & 1\,)_2 \end{array}$$

2.4 All registers are read from bottom to top to correspond with reading a binary number from left to right.

3. *The "Normal" positions of the Control Switches.*
All the CONTROL SWITCHES having ON-OFF positions are normally set OFF. The OVERFLOW SWITCH, is normally set on RUN and the AM SWITCH, is normally set on AUTO. These settings are summarized as follows:

	Name	*Normal* *Setting*	*Position*
3.1	MULTIPLY	OFF	LEFT
3.2	CLEAR	OFF	LEFT
3.3	COUNT	OFF	LEFT
3.4	COMPLEMENT	OFF	RIGHT
3.5	OVERFLOW	RUN	LEFT
3.6	AM	AUTO	LEFT

Notice that the normal position of all SWITCHES is to the left except for the COMPLEMENT SWITCH which is to the right.

4. *Initializing the computer before an operation.*
4.1 Set the ACCUMULATOR MODE FLIP-FLOPS T1 thru T6 all to the left.
4.2 Set CF1 to the left and CF2 to the right. These logic elements need to be set in this manner only once when the computer is first set up.
4.3 Set D1, D2, D3, of the DISTRIBUTOR all to the right.
4.4 Take all the balls, or a sufficient number, from the COLLECTOR GUIDE and place them in the INPUT GUIDE.
4.5 Set the CONTROL SWITCHES to their "normal" positions (all to the left except the COMPLEMENT SWITCH, which is set to the right) as described in 3.

5. *Clearing the Accumulator to zero.*
5.1 INITIALIZE the computer as described in 4.
5.2 Turn the CLEAR SWITCH, ON (right).
5.3 Turn AM, to MANUAL (right).
5.4 Pull START.

6. *Automatic Counting.*
6.1 CLEAR the ACCUMULATOR to zero as described in 5.
6.2 INITIALIZE the computer as described in 4.
6.3 Turn the COUNT SWITCH, ON (right).
6.4 Pull START.

7. *Manual Counting.*
7.1 Follow steps 6.1, 6.2, 6.3.
7.2 Turn the AM SWITCH, to MANUAL (right).
7.3 Pull START for each time one is to be added.

8. *Multiply-Add Operation (A×B+C).*
8.1 INITIALIZE the computer as described in 4.
8.2 Place any 3–bit multiplier, "A", which is less than or equal to 7, in the

MQ REGISTER.

8.3 Place any 4–bit multiplicand, "B", which is less than or equal to 15, in the MEMORY REGISTER.

8.4 Place any 7–bit number, "C", in the ACCUMULATOR ($A \times B + C$) should not exceed 127.

8.5 Turn the MULTIPLY SWITCH, to its ON position (right).

8.6 Pull START

9. *Multiply (A×B).*

9.1 Perform the same as 8 except *clear* the ACCUMULATOR to zero first and omit 8.4.

10. *Add (B+C) .*

10.1 Perform the same as 8 except set "A" equal to one, $(001)_2$ in 8.2.

11. *Complement the Accumulator.*

11.1 INITIALIZE the computer as described in 4.

11.2 Turn the COMPLEMENT SWITCH, ON (left).

11.3 Pull START.

12. *Subtract* (B - C). Subtraction may be performed in either of two ways, but in either method the answer will appear in the ACCUMULATOR.

12.1 Case 1. The number "B" is *less* than 16. Subtraction is performed by adding B to the COMPLEMENT of C.

12.1.1 Place C in the ACCUMULATOR and COMPLEMENT the ACCUMULATOR as described in 11. Turn the COMPLEMENT SWITCH OFF (right).

12.1.2 Place B in the MEMORY REGISTER.

12.1.3 Place (001) in the MQ REGISTER and turn the MULTIPLY SWITCH to its ON (right) position.

12.1.4 Pull START.

12.2 Case 2. The number "B" is *greater* than 15, but "C" is *less* than 16. Subtraction is performed by adding "C" to the COMPLEMENT of B, then COMPLEMENTING the result.

12.2.1 Reverse the numbers "B" and "C" and follow 12.1.1 thru 12.1.4.

12.2.2 Turn the MULTIPLY SWITCH OFF (left) and follow steps 11.2 thru 11.3.

13. *Count Out "A" balls from the Input Guide to the Collector Guide.*

13.1 Place the number "A" in the ACCUMULATOR in binary.

13.2 COMPLEMENT the ACCUMULATOR as described in 11.

13.3 INITIALIZE the computer as described in 4.

13.4 Turn the OVERFLOW SWITCH to HALT.

13.5 Turn the COUNT SWITCH to ON.

13.6 Press Start. Exactly "A" balls will be triggered and the computer will stop on an ACCUMULATOR OVERFLOW.

14. *Divide A/B.* (The divisor "B" must go into the dividend "A" a whole number of times).

14.1 Place "A" in the ACCUMULATOR, "B" in the MEMORY REGISTER, and set the MQ REGISTER to (000) using the lower set of numerals.

14.2 COMPLEMENT the ACCUMULATOR as described in 11.

14.3 INITIALIZE the computer as described in 4.

14.4 Turn the OVERFLOW SWITCH to HALT.

14.5 Turn the MULTIPLY SWITCH to its On (right) position.

14.6 Pull START.
The answer may be read in the MQ REGISTER using the lower numerals.

15. *Divide A/B to obtain a quotient and a remainder.*

15.1 Follow the procedure for DIVIDE given in 4.

15.2 Turn the AM SWITCH to MANUAL.

15.3 Pull START as many times as required (0, 1, 2, or 3) to return the DISTRIBUTOR elements D1, D2, D3 to their initial right positions.

15.4 COMPLEMENT the ACCUMULATOR as described in 11, then turn the COMPLEMENT SWITCH OFF.

15.5 Flip each element in the MQ REGISTER to its opposite position. Turn the AM SWITCH to MANUAL.

15.6 Pull START four times.
The quotient may be read from the MQ REGISTER using the upper set of the numerals and the remainder may be read in the ACCUMULATOR.

CHAPTER 13

SPECIAL PROBLEMS AND APPLICATIONS FOR DIGI-COMP II

This chapter discusses several practical areas that illustrate how a DIGITAL COMPUTER is used for solving problems. Each area has illustrative problems, some very simple and others more advanced.

Try to do each problem yourself and then check your program against those given at the end of the chapter. *Remember,* however, that the sequence of your program steps need not necessarily be the same as those given in the answers. The important point is that you have not omitted any steps, made errors, or written too long a program.

You will undoubtedly be able to create many, many more problems to program than we have shown you here. By now you should be able to "talk" to your DIGI-COMP II and program it to solve your own problems! Good Luck and we hope you have fun.

I. COMPUTING AREAS

Area is a measure of the surface of an enclosed figure. It is measured in square units, as for example square feet. If it were desired to compute the area of a rectangle which is 3 feet wide by 5 feet long, we would multiply 3 by 5 to obtain 15 square feet of surface inside the rectangle. This, of course, means that 15 squares each of which were 1 foot long by 1 foot wide would exactly cover the surface of the rectangle if they were placed beside one another inside the rectangle.

In general if a rectangle is "L" units long and "W" units wide the area inside the rectangle is given by the formula,

$$A = L \times W$$

or "A" is equal to the length times the width.

Use DIGI-COMP II to solve the following problems.

PROBLEM 1. Calculate the Area of a rectangle 5 feet wide and 13 feet long. See program page 60 to check your own program.

PROBLEM 2. A rectangular box is 4 feet by 9 feet by 14 feet. Calculate the sum of the areas of its two smallest sides.

PROBLEM 3. It is desired to spray a chemical on a large outdoor area by using an airplane. The area is 5 miles wide and 9 miles long. With one loading of chemicals the airplane can spray 15 square miles. How many loadings of chemicals will be required?

To solve this problem we wish to first calculate the number of square miles to be sprayed by multiplying 5 times 9, then we wish to divide this result by 15 to determine the number of loadings required.

II. CALCULATING THE WEIGHT OF AIR

At sea level the atmospheric pressure is 14.7 lbs. per square inch. This means that a column of air one inch long by one inch wide extending upward to the sky as high as the air extends would weigh 14.7 lbs. For purposes of simplicity, let us call atmospheric pressure 15 lbs. per square inch.

PROBLEM 4. Calculate the weight of air at sea level on a rectangular surface 2 inches wide by 3 inches long by the formula Weight = $2 \times 3 \times 15$ pounds.

III. THE PRINCIPLE OF ARCHIMEDES

According to Archimedes principle, an object under water is buoyed upward by a force which is equal to the weight of the water which it displaces. For example, if a tank is filled with water up to an overflow spout and a large stone weighing 100 lbs. in air is placed in the water causing a certain amount of water to overflow, and if this water weighed 60 lbs., then the stone would be buoyed upward in the water by 60 lbs. of force. This means that the stone would weigh (100 - 60) or 40 lbs. under water.

PROBLEM 5. An object that weighs 28 lbs. in air displaces 13 lbs. of water. Calculate its weight under water using DIGI-COMP II.

IV. ITERATION

Computers are frequently programmed to repeat a given calculation over and over again a great number of times until a sought-after result is obtained. The square root of a number is normally computed in this "iterative" manner instead of following the procedure that is customarily taught.

Without using the computer try to compute the square root of 13 by the following set of instructions:

A. Guess a number, such as 4, which is an approximation to the square root of 13.
B. MULTIPLY this approximation by itself and ADD the result to 13.
C. DIVIDE the number you obtain by twice your approximation and call this new result your next approximation. Write it down.
D. Using this next approximation to the square root of 13, repeat steps B, C and D rounding all the results to two decimal places.
E. Upon each repetition of steps B and C the approximation will get closer and closer to the square root of 13. Stop the iteration when the approximation to the square root of 13 is no longer changing to within two decimal places.

To illustrate:	1st Result	2nd Result	3rd Result
Step			
B	29	26.18	26.03
C	3.63	3.61	3.61

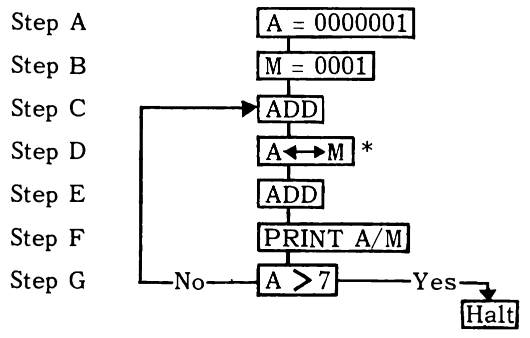
The square root of 13 is 3.606 to three decimal places.

PROBLEM 6. Calculate the Square Root of 2. The basic technique is as follows:

A. Set the ACCUMULATOR equal to one.
B. Set the MEMORY REGISTER equal to one.
C. ADD the MEMORY REGISTER to the ACCUMULATOR.
D. EXCHANGE the number now in the ACCUMULATOR with the number now in the MEMORY REGISTER.
E. ADD the MEMORY REGISTER to the ACCUMULATOR.
F. Write down a fraction with the number now in the ACCUMULATOR as the numerator and the number in the MEMORY REGISTER as the denominator.
G. Go back to C and repeat steps C, D, E and F until the number in the ACCUMULATOR is too large to perform step D.

Now Convert each of the fractions obtained in E to a decimal by dividing the numerator by the denominator using a piece of paper and a pencil.
These fractions should approach closer and closer to the square root of 2, which is 1.414, on each iteration.

This program could be shown in a "Flow Diagram" similar to the one discussed on page 8. This is a technique used by programmers to visually illustrate the sequence of events and therefore clarify the procedure of the program.



*The symbol ↔ indicates an exchange of the values of REGISTERS. In this case it means to put the contents of MEMORY into the ACCUMULATOR and the contents of the ACCUMULATOR into the MEMORY REGISTER.

Compare the flow chart with steps A thru G. This should help make it clear what each box is saying. An oval generally indicates that a question is being asked. In this case, it is asking if A is greater than 7.

Just to check a few of your steps the answers you obtained should have been,

	Start	1st Iteration	2nd Iteration
ACCUMULATOR	1	3	7
MEMORY REGISTER	1	2	5

Notice that the fractions

$$\frac{1}{1} = 1.00,\ \frac{3}{2} = 1.5,\ \frac{7}{5} = 1.4,$$

approach 1.414, closer and closer with each iteration. Repeat problem 6 with statement "A" changed to "Set the ACCUMULATOR equal to 2." Do the fractions still approach the square root of two?

V. SUMMING INFINITE SERIES

Suppose we dropped a ball from a height of 14 feet and observed that it bounced up to a height of 7 feet after hitting the ground. As this is half its original height we would naturally expect that the same ball would bounce up to a height of 3 1/2 feet if it were next dropped from 7 feet. But is this right?

If we were simply not to catch the ball after its first drop from 14 feet we would expect it to bounce up to 7 feet on the first bounce, 3 1/2 feet on the second, 1 3/4 feet on the third, etc., etc., thus making an infinite number of bounces. But this seems absurd. We know from experience that the ball does stop bouncing. Therefore, we cannot believe that the ball makes an infinite number of bounces before it stops a few seconds later, so, we are tempted to conclude that this model is in some way wrong. Perhaps as the bounces get smaller the ball actually bounces to less than half its original height, or maybe it suddenly stops bouncing after a certain number of bounces.

If we were to experiment further, however, we would find that, fraction of one-half is indeed maintained on every bounce, to as far as we can easily measure. How can we account for the fact that the ball does stop bouncing?

Let us assume that this strange behavior of the ball does in fact occur, that is that the ball bounces to half its prior height on each bounce, and it bounces an infinite number of times before it stops. Let us attempt to calculate the sum of all the distances that the ball falls. If this is the case, off hand, we would expect this sum to be infinite. We add.

S = 14 + 7 + 3 1/2 + 1 3/4 + 7/8 + . . . but let us add this sum in binary.

PROBLEM 7. Compute "S" (the sum) by the expression,

S = (1110.) + (111.) + (11.1) + (1.11) + (.111) + . . .

This problem is solved by adding and shifting. The program will be written out in long hand here because of its complexity. It is also given in DIGI-TRAN on page 60.

Program: (In the set of instructions that follows the location of the binary point must be remembered in interpreting the numbers).

1) Place (1110.) in the ACCUMULATOR as (0001110.).
2) Place (0111.) in the MEMORY REGISTER.
3) ADD the MEMORY REGISTER to the ACCUMULATOR obtaining (0010101.).
4) Manually shift the ACCUMULATOR left one place by moving each bit down one place obtaining the number (010101.0) in the ACCUMULATOR.
5) Place (011.1) in the MEMORY REGISTER.
6) ADD the MEMORY REGISTER to the ACCUMULATOR.
7) Shift the ACCUMULATOR left one place again. The binary point is now understood to be in the position shown; (XXXXX.XX).
8) Place (01.11) in the MEMORY REGISTER.
9) Add the MEMORY REGISTER to the ACCUMULATOR.

As no more ACCUMULATOR SHIFTS can take place without losing information we can no longer add further binary fractions without losing precision.

What is the number we have so far in the ACCUMULATOR as a decimal? To figure this out we must remember where the binary point is and evaluate the number as follows remembering that 1/2 = .5 and 1/4 = .25 in decimals:

16	8	4	2	1	.5	.25	
(1	1	0	1	0.	0	1)	= 26.25

Although we cannot shift the ACCUMULATOR further to the left to permit the addition of (.111), (.0111), (.00111), (.000111), etc. we round*
each of the numbers to two binary places and add in an attempt to gain further precision in the answer. Thus we would round,

		Truncated Number	+	Rounding Correction	=	Final Number
(.111)	as	(00.11)	+	(00.01)	=	(01.00)
(.0111)	as	(00.01)	+	(00.01)	=	(00.10)
(.00111)	as	(00.00)	+	(00.01)	=	(00.01)
(.000111)	as	(00.00)	+	(00.001)	=	(00.00)

Therefore, it is only necessary to add (01.00) + (00.10) + (00.01) = (01.11) to the ACCUMULATOR to gain further precision.

10) Place (01.11) in the MEMORY REGISTER.
11) ADD it to the ACCUMULATOR. What is the answer in decimal?

If we were able to retain complete precision in our addition and if we were to continue to add the number (1110.) + (111.) + (11.1) + (.111) + (.0111) + etc., would you expect the sum to grow without limit or always to be less than some limiting number? The strange fact is that the sum of this infinite number of terms does not grow infinitely large but approaches a limiting number! In fact in this particular case, the limiting number is 28. (If you operated DIGI-COMP II correctly and added the sum (01.11) three numbers (.111), (.0111), and (.00111) rounded upward as instructed in Step 10 you should have obtained exactly 28). Thus, the sum of an infinite number of numbers can be a finite (Not infinite) number. In this case even if the ball makes an infinite number of bounces, it travels only a total of 28 feet downward.

How Long will the Ball Bounce?
Perhaps more interesting than the distance traveled would be the total time it would take to bounce an infinite number of times. Is it possible that we could add all the times taken between the bounces and obtain a finite number even assuming that the ball bounced an infinite number of times? The answer is "yes"! In fact, if a ball were dropped from 14 feet and bounced up to half its prior height on each bounce, the total time taken for the infinite number of bounces would only be about 5.5 seconds.

Although we will not compute this answer we can mentally compute the time taken in a slightly altered case. Suppose the ball were dropped from 16 ft. and bounced up to 1/4 of its prior height on each bounce. Then an accepted formula for motion (from Physics) tells us that it would take one second to hit the ground after it was first released. Moreover it would take 1/2 a second to bounce up to 4 ft. after its first bounce and 1/2 second more to fall to the ground again, or a total of one second more between the first and second bounces.

Further calculations would show that 1/2 second would be taken between the 2nd and 3rd bounces, 1/4 second between the 3rd and 4th bounce, 1/8 second between the 4th and 5th bounce, etc.

Therefore, the total time in seconds to bounce an infinite number of times would be,

$$T = 1 + 1 + 1/2 + 1/4 + 1/8 + 1/16 \text{ etc.}$$

*In rounding a binary number, it is conventional to round *up* if there is a one in the first position lost by the rounding. It may seem a great inconvenience to have to do this rounding off, but the truth of the matter is that even in the world's largest electronic computer, the programmer is still faced with the problem.

Fortunately the programmer does not have to do this manually, it is accomplished simply by inserting the ROUND command in the program at the proper time.

or in binary,

T = (1.) + (1.) + (.1) + (.01) + (.001) + (.0001) + etc.

This means the total time would be,

T = (10.111111111111111 . . .)

in binary. But the unending fraction (.11111 . . .) in binary is equal to (1.) just as the unending decimal (.9999 . . .) is equal to 1. Therefore,

T = (10.) + (1.) = (11.) = 3 seconds. In other words, the ball would bounce an infinite number of times in 3 seconds!

If this bothers you add the following fraction directly instead of assuming that (.11111 . . .) is equal to one.

S = 1/2 + 1/4 + 1/8 + 1/16 + 1/32 + etc.

Note that,

1/2 + 1/4 = 3/4
1/2 + 1/4 + 1/8 = 7/8
1/2 + 1/4 + 1/8 + 1/16 = 15/16
1/2 + 1/4 + 1/8 + 1/16 + 1/32 = 31/32

And that we can never reach "1" no matter how many terms we added but we can come as close to "1" as we like. Perhaps the theoretical model which assumes that the ball bounces an infinite number of times is not so bad after all!

VI. THE POPULATION EXPLOSION

Let us assume that in a certain country an average number of three children are born to a family. Let us further assume that only 2 1/2 of the three children can expect on the average to grow to adulthood, marry and have a similar average-sized family. As 2 parents produce on the average 2 1/2 children who will in turn be parents, we can think of this as each of the parents producing 1 1/4 children who will in turn reproduce themselves in like manner.

Let us assume still further that it takes on the average 35 years for a person in this hypothetical country to reproduce the average number of 1 1/4 children. This 35 years then will be called a "generation". For purposes of simplicity let us further assume that as soon as a person reproduces 1 1/4 children he is no longer counted in the population. Now if one person produces 1 1/4 persons as a first generation, these 1 1/4 persons will produce 1 1/4 × 1 1/4 persons as a second generation, and these people will in turn produce 1 1/4 × 1 1/4 × 1 1/4 people as a third generation, etc.

PROBLEM 8. How many people will be produced by one person in 3 generations? About how many years will be required for the population to double?

Again, because of the complexity of the program, it is written out in long hand below. It is written in DIGI-TRAN on page 61.

Program:

1) Place (1.01) = 1 1/4 in the MQ REGISTER.
2) Place (01.01) in the MEMORY REGISTER.
3) CLEAR the ACCUMULATOR first, then multiply the MQ REGISTER by the MEMORY REGISTER. The product thus far will be in the ACCUMULATOR in the form (XXX.XXXX).
4) Round this number to four significant binary digits. In this case (1.01) × (01.01) should give (001.1001) in the ACCUMULATOR which should be rounded to (1.101).
5) Place 1.101 in the MEMORY REGISTER.
6) Again place (1.01) in the MQ REGISTER.
7) CLEAR the ACCUMULATOR.
8) MULTIPLY the MQ REGISTER by the MEMORY REGISTER. The answer should be of the form (XX.XXXXX).
9) Round the answer to the form (XX.XX) and express as a decimal. The ACCUMULATOR now should read 00010.001 which is 2, so the population has doubled in 3 generations.

VII. AREA OF CIRCLE

In geometry we learn that the distance around a circle, called the circumference, is about 3 times the distance across a circle, called the diameter. Actually, the circumference is said to be not 3 times the diameter but "π" (called pi) times the diameter, where π can be mathematically computed to be 3.14159265 . . ., a never ending decimal fraction.

Having defined the number π in this manner it can be proved that the area "A" of a circle is given by the formula,

$$A = \pi r^2$$

where "r" is the radius. This formula means that the area of a circle whose radius is "7", for example, may be calculated to be approximately,

$$A = 3.14 \times 7 \times 7$$

PROBLEM 9. Using DIGI-COMP II calculate the area of a circle given that the radius is 7 inches.

a) To do this the number 3.14 must first be expressed as a binary fraction.
b) As 3.14 must be placed in the MEMORY REGISTER to perform the multiplication we must express it as a number having no more than 4 binary bits.
c) The 4–bit binary fraction (11.00) represents the decimal number 3.00. The 4–bit binary fraction (11.01) is 3 1/4 or 3.25 in decimal. As 3.25 is closer to 3.14 than 3.00 we will choose to represent π to 4–bits of accuracy as (11.01). Program to compute the Area of a Circle of radius 7.

Read this thru and then try writing your DIGI-TRAN program. See page 61 for correct program.

1) Place π = (11.01) in the MEMORY REGISTER remembering that the binary point is between the 2nd and 3rd bit positions.
2) CLEAR the ACCUMULATOR to zero.
3) Place 7 = (111) in the MQ REGISTER.
4) MULTIPLY 7 × π . The product in the ACCUMULATOR should be (10110.11). We must be mentally aware that the answer has two binary fraction places as the sum of binary fraction places of the two numbers is 2 places.

   We now wish to multiply the result thus far (10110.11) by 7 = (111) again.
5) Therefore, we must round (10110.11) to 4 binary bits in order that we can place it in the MEMORY REGISTER. Rounding we obtain (10110.), where only the left most 4–bits are significant.
6) The left 4–bits of the number (10110.) are now placed in the MEMORY REGISTER as (1011) and we mentally remember that there is actually another "0" bit in the number to the right before the binary point.
7) Place the radius (111) in the MQ.
8) CLEAR the ACCUMULATOR.
9) MULTIPLY (111) × (1011). It must be remembered that the 7–bit answer in the ACCUMULATOR has an understand "0" bit to the right before the binary point. In other words, the number (1001101) in the ACCUMULATOR should be read as (10011010). Converting this number to decimal we obtain,

128	64	32	16	8	4	2	1	
(1	0	0	1	1	0	1	0.)	= 154.

   If the same problem were carried out in decimal we would obtain,

   $A = 3.14 \times 7 \times 7 = 153.86$ square inches.

PROBLEM 10. Calculate the Area of a Circle of radius 6 using this method. See if you can write the program from memory and then check it against the answer on page 61.

The answer calculated in decimal is 113.04. How close are you?

VIII. RANDOM NUMBER GENERATION

There are many problems programmed for computers which require the computer to produce a long list of random numbers. As there is no wheel of fortune inside the computer a means must be found using its normal ability to ADD, SUBTRACT, MULTIPLY and DIVIDE to produce the desired list.

The method used on the computer must be simple. This is because most problems demand a great quantity of random numbers to obtain meaningful results. Even though an electronic computer is very fast, time on the machine can be very costly, as much as several hundred dollars per hour on a large computer. Therefore, it is desirable to reduce the required computer time as much as possible.

One of the simplest methods used to generate random numbers requires one multiplication, one addition and one division for each number generated. On a binary computer the divisor is ordinarily chosen to be a perfect power of two so that the actual division need not really be performed. All that is necessary, for example, to divide a binary number such as (1101101.) by 2^4 or 16, is to move the binary point from its right-most position 4 places to the left. Thus, (1101101.) or 109 divided by 10000. or 16 is (110.1101) or 6 with a remainder of 13.

Strictly speaking, the computer does not generate a set of truly random numbers. If it did there would be no rule which would relate one number of the set to the next number of the set as is actually the case. For this reason computer generated numbers using a simple rule are called "pseudo random".

DIGI-COMP II may be used to generate pseudo random numbers between "0" and "15" by the following method.

A. Start with any number between 0 and 15. Call this number "R". (Let us start with "0", for example).
B. Compute the number 5 times "R" plus 13. DIVIDE this number by 16 and write down the remainder obtained after the division. (In other words write down the low order 4–bits after the MULTIPLY-ADD operation).
C. Call this remainder by the name "R" and go back to instruction 2.
D. Repeat instructions 2 and 3 as many times as required to generate a list of pseudo random numbers which eventually repeat themselves.

If we start with R = 0 then the remainder of ($5 \times R+13$) /16 is 13. Now we set R = 13 and calculate the next remainder of ($5 \times R+13$) /16 which is 14. Proceeding in this manner we should generate the following list of numbers.

Step Number	*Pseudo Random Number*
1	13
2	14
3	3
4	12
5	9
6	10
etc.	.
	.
	.

PROBLEM 11. Generate a set of random numbers using the formula,

$$S = \text{Remainder of } \frac{5 \times R + 9}{16}$$

where "R" is a given random number and "S" is the next random number. Then set "R" equal to "S" and use the formula again to generate the next "S", etc. Stop when the list eventually repeats itself. Suppose we start generating the list by setting R = 6.

As a check on your work the first few numbers obtained should be,

Cycle	Random Binary Number	Equivalent Decimal Number
1	(0110)	6
2	(0111)	7
3	(1100)	12
4	(0101)	5
.	.	.
.	.	.

Each of the number 0 thru 15 are generated once and only once until the list repeats itself. This implies that the list is exactly 16 numbers long.

IX. SIMULATION

Simulating a Baseball Game.

Computers are frequently used to simulate a complicated process when information is known (or estimated) for only isolated events. For example, if we know the probabilities that each baseball player or a team will hit a single, or a double, or a triple, or a home run, or walk, or get out, we can simulate nine innings at bat and ask how many runs would be expected in a nine inning game. It would be necessary to simulate the game a number of times to get an average number of expected runs.

Using pseudo random numbers discussed in Problem 11, let us try to simulate several innings at bat of a team called the "Bob Cats".

First, the Bob Cats have 9 men whose names are A, B, C, D, E, F, G, H, I. Some of them are better hitters than others. The following table gives the probability that each man might get a single, a double, a triple, a home run, a walk or an out if he is at bat. The table gives the number of chances out of 16 that any of these things will happen.

TABLE 1

	Single	Double	Triple	Home Run	Walk	Out
A	4	1	1	0	1	9
B	3	1	1	1	0	10
C	2	2	1	2	1	8
D	1	1	2	3	2	7
E	2	2	1	1	0	10
F	2	1	1	1	0	11
G	3	1	0	1	1	10
H	3	2	0	1	1	9
I	1	1	1	1	0	12

Reading the table we see that man "C", for example, has 2 chances out of 16 to hit a single, 2 chances out of 16 to hit a double, 1 out of 16 for a triple, 2 out of 16 for a home run, 1 out of 16 for a walk, and 8 out of 16 to get out. For any given man, read across the table to determine his chances of a single, double, a triple, etc. The numbers must add to 16 because it is assumed that one of these 16 possibilities must occur.

Now let us form another table which will reflect these probabilities but use the pseudo random numbers from 0 to 15 to determine which of these events occur for a given time at bat.

TABLE 2

	Single	Double	Triple	Home Run	Walk	Out
A	(0-3)	4	5	-	6	(7-15)
B	(0-2)	3	4	5	-	(6-15)
C	(0-1)	(2-3)	4	(5-6)	7	(8-15)
D	0	1	(2-3)	(4-6)	(7-8)	(9-15)
E	(0-1)	(2-3)	4	5	-	(6-15)
F	(0-1)	2	3	4	-	(5-15)
G	(0-2)	3	-	4	5	(6-15)
H	(0-2)	(3-4)	-	5	6	(7-15)
I	0	1	2	3	-	(4-5)

This table says that if either the random number "0" or "1" is obtained when "C" is at bat we will assume that he has hit a single. This corresponds to 2 chances out of 16. If "2" or "3" is obtained we will assume that "C" has hit a double, if 4 a triple, if 5 or 6 a home run, if 7 a walk and if 8 thru 15 an out. These correspond to 2 chances for a double, 1 for a triple, 2 for a home run, 1 for a walk and 8 for an out.

Let us further assume that a single will drive any player on base 2 bases and a double (or better) will drive all players home.

PROBLEM 12. Use the table of random numbers generated in Problem 10. When you have used all 16 of the numbers repeat the cycle. Draw a baseball diamond on a piece of paper and get some coins to simulate the players on the base paths.

Now take each random number in turn, and each batter in turn and look up the outcome in Table 2. You can play as many innings as you wish using the regular rules of baseball. You could change your batting order around at the beginning of each game if you wish. You could also develop your own team, by changing the hitter's ability around. To do this you have to be sure that each player has 16 possibilities as it is in Table 2.

To make it fair, you should also be sure that you and your opponent have equal strength ball teams. That is in Table 2 the Bob Cats will get 21 singles out of 9 × 16 or 144 times at bat. So it doesn't matter how you change your team around so long as they can get a total of 21 singles out of 144 times at bat. The same holds true for doubles, triples, homers, and outs.

X. AN ACCOUNTING PROBLEM

One should not have the impression that computers are used only for scientific calculations. Actually, computers are used more for all other applications than they are for scientific work. In particular financial data processing accounts for a very substantial use of computers.

Typically, in financial data processing a huge "file" of information is stored magnetically either on magnetic tape or on a magnetic disc which looks similar to a record player. This "master file" of information contains great numbers of "records" of information which are all pretty much alike except each record applies to a different situation. During the course of a certain period of time, let's say a month for example, a number of transactions occur which must be used to "update" the information in many of the records in the master file.

As an example, let us assume that company ABC sells items on credit to many people including a man by the name of Jones. Any time a credit sale takes place to Jones or any time that Jones pays something on his account a transaction occurs which must either add or subtract from the amount that Jones owes company ABC. Therefore, these transactions are batched or saved and once a month all of Jones transactions are used to update his record on the master file. Of course, similar transactions apply to many other records in the file.

PROBLEM 13. Mr. Jones buys the following items on the following dates during the month.

Date	*Quantity*	*Amount Each*
March 2	3	$5
March 15	2	$13
March 13	4	$7

He owed the ABC company $20 at the beginning of the month. Mr. Jones made the following payments on his account during the month.

Date	*Amount*
March 20	$14
March 25	$13

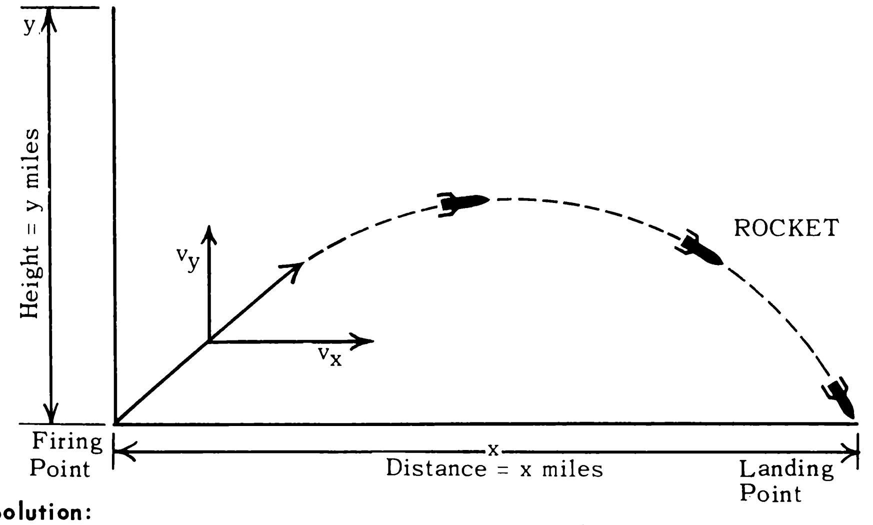
Questions:

1) How much does Mr. Jones owe as of March 31?
2) What was the maximum amount that he owed during the month?

The proper solution of this problem consists of calculating the outstanding balance in Mr. Jones' account chronologically. These will have to be recorded on paper. In an electronic computer, the balance would be automatically printed each time the PRINT COMMAND is given.

XI. BALLISTIC MISSILE CALCULATIONS

PROBLEM 14. A ballistic missile is fired down range with a velocity V of 2 miles per second. If the rocket is fired at an angle θ of 45^o with the Earth, how far down range will it land?



Solution:
Assume the acceleration, g, due to gravity* to be $\frac{1}{170}$ miles per second, or $\frac{1}{170}$ mi/sec^2

The basic equations of a ballistic missile** are (ignoring air resistance since the missile spends most of its time above the atmosphere).

1) $X = v_x \cdot t$, *where:* v_x is the component of the missile's velocity in the downrange or X direction and *t* is the time of missile's flight,
2) and $t = \frac{v_y}{g}$, where v_y is the component of the missile's velocity upward, and g is the gravitational acceleration for the Earth. Since for the Earth $g = \frac{1}{170}$ mi/sec2, we can rewrite equation (2) as

*The acceleration of a body due to gravity is a result of the gravitational attraction between the mass of the Earth and the body (in this case, the missile). Since the Earth's shape and density are not constant, the acceleration will vary somewhat from place to place on the face of the Earth. Of course a large electronic computer can handle many more decimal places than DIGI-COMP II and therefore these slight gravitational variations could be put into the computer.

Question –
Would the acceleration due to gravity be different on other planets?

Answer –
Yes, it would vary very much. On Mercury, it would be smaller than on the Earth and on Jupiter, much more. In fact an earthman would probably not be able to stand up on Jupiter since he would weigh so much more.

**An elementary physics book or certain encyclopedias will show where these equations come from.

3) or $t = 170\ v_y = \frac{v_y}{g} = \frac{v_y}{1/170} = 170\ v_y$.

4) Let us substitute this value of t given in equation (3) in equation (1) giving $x = v_x \cdot t = v_x \cdot 170\ v_y = 170\ v_x \cdot v_y$.

5) Now if the missile is fired at a 45^o angle, θ with a velocity V then $v_x = v_y$ and $V^2 = 2v_x^2$ since $V^2 = v_x^2 + v_y^2$
Therefore equation (4) becomes

6) $X = 170\ v_x \cdot v_y = 170\ v_x \cdot v_x = 170\ v_x^2 = \frac{170}{2} V^2 = 85\ V^2$.

In this case since $V = 2$ mi/sec^2, the multiplication of $2 \times 2 \times 85$ will not only cause an ACCUMULATOR OVERFLOW but the number 85 is too large to be put in the MEMORY REGISTER. So we will have to develop a special technique for multiplying. To accomplish this we shall express 85 as follows:

7) $85 = 8 \times 10 + 5$
The problem can now be programmed on DIGI-COMP II by multiplying $2 \times 2 \times 8$, converting the ACCUMULATOR to decimal, mentally multiplying that by 10 and writing the answer down. Next multiply $2 \times 2 \times 5$ on DIGI-COMP II, convert this answer to decimal and add it to the previous one. Now you will have the total number of miles down range that the missile travels.

PROBLEM 15.

If the missile is fired at a different angle θ than 45^o then the components of the velocity V in the x and y directions are given by trigonometry as

$$v_x = V \cos \theta$$
and
$$V_y = V \sin \theta$$

Therefore Equation (6) becomes

8) $X = 170\ v_x v_y = 170\ V \cos\theta\ V \sin\theta$

9) or $X = 170\ V^2 \sin\theta \cos\theta$.

Older electronic missile computers used to carry along a "table" of values of sin θ and cos θ stored in an electronic memory. Today, however, the size of the electronic components is so small and their speed so great that often the computer designer simply adds additional components, and then programs the computer to compute the value of the sin and cos of θ in a few millionths of a second.

Now, if we assume the value of sin θ and cos θ has been computed it becomes a simple matter to program your DIGI-COMP II to accomplish the final computation for X. For example, let us assume $\theta = 30^o$.
Then $\sin\theta = 0.50$ and $\cos\theta = .85$
(These can be obtained from any standard table given in trigonometr y books and many other math books and encyclopedias.)
so that $\sin\theta \cos\theta = .425$.

Then Equation (9) becomes for this case

10) $X = 170 \cdot V^2 \cdot .425$

Multiplying .425 times 170 we find that

11) $X = 72.25V^2$

If we still wish to use the initial velocity of $V = 2$ mi/sec^2 the multiplication will obviously cause an OVERFLOW in the ACCUMULATOR. This time you may do the multiplication using the same technique developed in Problem 14. The number 72.25 can be broken up as follows:

12) $72.25 = 7 \times 10 + 2.25$

Try other values of the angle θ and other values of V to see what the pattern of impact or landing points for our missile might be.
Note: Further interesting exercises would be to solve Equations (1) and (2) for t and v_y. Then the height the missile attains would be

$$y = v_y \cdot t$$

You should try setting up these equations on your DIGI-COMP II as Special Exercises.

PROGRAMS FOR PROBLEMS

PROBLEM 1.
1. M = 1101
2. MQ = 0101
3. MULTIPLY

PROBLEM 2.
The smallest two faces of the box are 4 by 9 ft. The problem could be solved by multiplying 4 × 9 and then doing a MULTIPLY–ADD operation. The program is:

1. CLEAR
2. MQ = 100
3. M = 1001
4. MULTIPLY
5. PAUSE
6. MQ = 100
7. MULTIPLY–ADD*

PROBLEM 3.
1. CLEAR
2. MQ = 101
3. M = 1001
4. MULTIPLY
5. PAUSE
6. M = 1111
7. DIVIDE

PROBLEM 4.
In this problem we will use some new commands. These are callêd transfer commands. For instance, "M = A" means "put the contents of the ACCUMULATOR into the MEMORY REGISTER. " (That is, if A is 0001101 then make M = 1101). Similarly MQ = M means "put the contents of the MEMORY REGISTER into the MQ REGISTER," Etc.

1. CLEAR
2. MQ = 010
3. M = 0011
4. MULTIPLY
5. PAUSE
6. MQ = A
7. M = 1111
8. MULTIPLY

PROBLEM 5.
1. CLEAR
2. A = 0011100
3. M = 1101
4. SUBTRACT
5. COMPLEMENT

PROBLEM 6.
1. A = 0000001
2. M = 0001
3. ADD
4. A⟷M
5. ADD
6. PRINT A/M
7. If (A > 7); 8,3
8. HALT

Several new commands have been used here. If you are not sure of their definitions refer to your Programmers Card.

PROBLEM 7.
1. A = 0001110
2. M = 0111.
3. ADD

*The MULTIPLY–ADD command is the same as MULTIPLY except that you don't clear the ACCUMULATOR.

4. SHIFT LEFT*
5. M = 011.1
6. ADD
7. SHIFT LEFT
8. M = 01.11
9. ADD
10. M = 0.111
11. ADD

PROBLEM 8.

1. MQ = 1.01
2. M = 01.01
3. MULTIPLY
4. ROUND (X.XXX)**
5. M = A
6. MQ = 1.01
7. CLEAR
8. MULTIPLY
9. ROUND (XX.XX)

PROBLEM 9.

1. M = 11.01
2. CLEAR
3. MQ = 111
4. MULTIPLY
5. ROUND (XXXX)
6. M = A
7. MQ = 111
8. CLEAR
9. MULTIPLY

PROBLEM 10.

1. M = 11.01
2. CLEAR
3. MQ = 110
4. MULTIPLY
5. ROUND (XXXX)
6. M = A
7. MQ = 110
8. CLEAR
9. MULTIPLY

PROBLEM 11.

1. A = 0001001
2. M = 0110
3. MQ = 101
4. MULTIPLY -ADD
5. PAUSE
6. ROUND A (XXXX)
7. M = A
8. A = 0010000
9. DIVIDE
10. PAUSE
11. M = A
12. A = 0010000
13. SUBTRACT
14. PAUSE
15. COMPLEMENT
16. PRINT
17. M = A
18. If A = 6; 18, 1***
19. HALT

*SHIFT LEFT means to manually shift the ACCUMULATOR left by one place, by moving each bit down one place, obtaining the number.

** This command says "IF A = 6, go to step 8, if A does not equal 6, go back to Step 1. The first number after the semi-colon always corresponds to a "yes" answer and the second to a "no" answer.

***The round command means to round the number to the form shown in the parentheses. (Remember that if the highest order bit which is truncated is a 1, that 1 should be added to the lowest order bit which is preserved).

PROBLEM 12.

The game should be played on paper. You use the numbers your DIGI-COMP II generated in Problem 11.

PROBLEM 13.

Program	Computer Output Would Be Date	Balance
1. A = 0010100	March 1	$20.00
2. MQ = 011		
3. M = 0101		
4. MULTIPLY–ADD		
5. PAUSE		
6. PRINT	March 2	$35.00
7. MQ = 010		
8. M = 1101		
9. MULTIPLY–ADD		
10. PAUSE		
11. PRINT	March 15	$61.00
12. M = 1110		
13. SUBTRACT		
14. PAUSE		
15. COMPLEMENT		
16. PAUSE		
17. PRINT	March 20	$47.00
18. MQ = 100		
19. M = 0111		
20. MULTIPLY–ADD		
21. PAUSE		
22. PRINT	March 23	$75.00
23. M = 1101		
24. SUBTRACT		
25. PAUSE		
26. COMPLEMENT		
27. PAUSE		
28. PRINT	March 25	$62.00

PROBLEM 14.

1. MQ = 010
2. M = 0010
3. MULTIPLY
4. PAUSE
5. MQ = A
6. M = 1000
7. MULTIPLY
8. PAUSE
9. PRINT*
10. MQ = 100
11. M = 0101
12. MULTIPLY
13. PAUSE
14. PRINT

PROBLEM 15.

1. MQ = 010
2. M = 0010
3. MULTIPLY
4. PAUSE
5. MQ = A
6. M = 0111
7. MULTIPLY
8. PAUSE
9. PRINT*
10. MQ = 100
11. M = 10.01
12. MULTIPLY
13. PAUSE
14. PRINT

*In Step 9 of Problems 14 and 15, multiply the decimal value of the ACCUMULATOR by 10 before writing it down.

CHAPTER 14

HOW TO PROGRAM YOUR DIGI-COMP II PROBLEMS ON A COMMERCIAL ELECTRONIC DIGITAL COMPUTER

There are many different compiler languages available for use in computers. All of the languages are similar in concept but vary slightly in application as indeed DIGI-TRAN itself does. Since FORTRAN has been for years one of the more popular computer languages it has been chosen to illustrate how the DIGI-COMP II language that you have learned can be translated into another typical computer language. The problems that will be written in FORTRAN in this chapter can then be applied to numerous existing commercial digital computers such as the –

IBM 1620
IBM 1401
IBM 1410
IBM 7070
IBM 7074
IBM 704
IBM 709
IBM 7090
CDC 1604
Honeywell 800

DIFFERENCES BETWEEN "DIGI-TRAN AND FORTRAN"

There are three basic differences between the DIGI-TRAN and the FORTRAN languages. FIRST, and most important, is that in the use of DIGI-COMP II, most commands actually refer to things you must do, while in a large electronic computer, the commands specifically tell the machine what to do. Once the program is punched in to the cards the programmer does nothing until the problem is completed.

SECONDLY, a computer has the ability to read mathematical signs which are coded on the cards. That is, an ADD command would be written as a plus (+) sign rather than the word ADD. This is simply a convenience to the programmer. FORTRAN could just as easily have been designed to accept ADD instead of +.

The THIRD difference results in a further convenience to the programmer. The computer is equipped with an automatic binary-decimal converter. So the programmer puts in his data in decimal form and reads it out in decimal form. Of course the computer operates in the binary system just as your DIGI-COMP II does.

Actually in your studies with DIGI-COMP II you have learned more of the fundamentals of computer operation than many programmers do working with electronic computers. This is so because the input-output conversion work has been done for them by the electronic computers, while with DIGI-COMP II you had to do the work of converting. In fact, if you have learned everything in this DIGI-COMP II Manual and you have mastered the material in our DIGI-COMP I Manuals* you will have most of the basic theoretical working knowledge to understand the organization and operation of ELECTRONIC DIGITAL computers!

* There are two DIGI-COMP I Manuals – the Basic and the Advanced.

Perhaps the easiest way to introduce FORTRAN is to write some of the programs of Chapter 13 in FORTRAN and compare them to the original DIGI-TRAN Programs.

PROBLEM 1:

If A equals the area of the rectangle then the program would simply be written:

```
A = 5 * 13
```

Where the asterisk means MULTIPLY.

PROBLEM 2:

The MULTIPLY and MULTIPLY-ADD operations of this problem can be summarized in one command.

```
A = 4 * 9 + 4 * 9
```

PROBLEM 3:

The division of two numbers is indicated by a diagonal slash (/). So the program for multiplying 5 times 9 and dividing that by 15 is:

```
A = 5 * 9 / 15
```

PROBLEM 4:

The Transfer Command used in this problem has the same symbology and meaning it would have in FORTRAN, however it is unnecessary since the electronic computer will accept several sub-commands within one statement. If the weight of the air is W then the program would be:

```
W = 2 * 3 * 15
```

PROBLEM 5:

In an electronic computer negative numbers are handled automatically. There is no need to recomplement the answer. The weight (W) of the object in water would be simply given by:

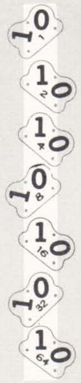
```
W = 28 - 13
```

The FORTRAN LANGUAGE contains many other commands which make it possible to do such things as the Iteration in Problem 6 and the other programs which follow. The explanation of these commands however involve concepts which are beyond the scope of this Instruction Manual.

In spite of this, it should be clear that the type of programming you have been doing for DIGI-COMP II is not too different from that for an electronic computer, and that we have dispelled some of the mysteries that surround computers.

THIS REPLACES UPPER HALF OF PAGE 12. IT BETTER DEMONSTRATES THE PROPER METHOD FOR READING DIGI-COMP II REGISTERS.

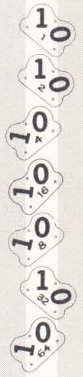
So the decimal number in the ACCUMULATOR is 86. Note that *only when a 1 is up* do we include the value of that FLIP-FLOP toward the total.



= 0
= 2
= 4
= 0
= 16
= 0
= +64

86

To be sure we understand, another example is,

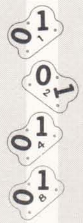


= 1
= 2
= 0
= 0
= 0
= 32
= +0

35

So the number in this example is 35. Later when you learn about binary numbers you will be able to both read and set in binary numbers without the aid of the little decimal numbers on each FLIP-FLOP.

The MQ and MEMORY REGISTERS are read the same way. For example, for the MEMORY REGISTER –



= 1
= 0
= 4
= +8

13

and similarly for the MQ Register.

Your DIGI-COMP II has been carefully inspected. If you should damage or lose any part, send the name and number of the part (see parts list in assembly instructions) together with 50¢ for handling & mailing costs.

to

E.S.R., Inc.
34 LABEL STREET
MONTCLAIR, N.J. 07042

MORE ABOUT YOUR DIGI-COMP II

As you can well imagine, a great deal of effort has gone into the writing of this Manual. However we expect to develop an advanced Manual for DIGI-COMP II at a later date, just as we wrote the Advanced Manual for DIGI-COMP I. If you wish to be kept posted on these further developments, send in the coupon below and we will put you on our mailing list.

DIGI-COMP II®
Programmer's Card

MACHINE LANGUAGE COMMANDS

Command	Definition
MULTIPLY=ON	Turn MULTIPLY SWITCH to the RIGHT
MULTIPLY=OFF	Turn MULTIPLY SWITCH to the LEFT
CLEAR=ON	Turn CLEAR SWITCH to the RIGHT
CLEAR=OFF	Turn CLEAR SWITCH to the LEFT
COUNT=ON	Turn COUNT SWITCH to the RIGHT
COUNT=OFF	Turn COUNT SWITCH to the LEFT
COMPLEMENT=ON	Turn COMPLEMENT SWITCH to the LEFT
COMPLEMENT=OFF	Turn COMPLEMENT SWITCH to the RIGHT
OVERFLOW=RUN	Turn OVERFLOW SWITCH to the LEFT
OVERFLOW=HALT	Turn OVERFLOW SWITCH to the RIGHT
A-M=AUTO	Turn A-M SWITCH to the LEFT
A-M=MANUAL	Turn A-M SWITCH to the RIGHT
START	Pull the START SWITCH
PAUSE	Pause until the computer has completed its automatic cycle before continuing with program.
A=XXX.XXXX	Means that there is an imaginary Binary Point between the bits in A4 and A5. It must be remembered by the programmer.
M=A	Put the contents of the MEMORY REGISTER into the ACCUMULATOR REGISTER.
$A \leftrightarrow M$	Interchange the contents of the ACCUMULATOR and MEMORY REGISTERS.
PRINT X	Write down the value of X.
If (A > X); 8, 3	If A is greater than X go to Step 8, if A is less than or equal to X go back to Step 3.
SHIFT LEFT	Manually shift the ACCUMULATOR left by one place by moving each bit down one place.
ROUND (XX.XX)	Round the number to the form shown in the parentheses. (Remember that if the highest order bit which is truncated is A 1, that 1 should be added to the lowest order bit which is preserved.)

DIGI-TRAN COMPILER LANGUAGE COMMANDS

Digi-Tran Command	Machine Language Program
COUNT	1. INITIALIZE 2. A =0000000 3. COUNT=ON 4. START
CLEAR	1. INITIALIZE 2. CLEAR=ON 3. AM=MANUAL 4. START
ADD	1. INITIALIZE 2. MULTIPLY=ON 3. MQ=001 4. START
MULTIPLY	1. INITIALIZE 2. MULTIPLY=ON 3. A=0000000 4. START
SUBTRACT	1. INITIALIZE 2. COMPLEMENT=ON 3. MQ=001 4. START
SUBTRACT	5. PAUSE 6. MULTIPLY=ON 7. COMPLEMENT=OFF 8. START
DIVIDE	1. INITIALIZE 2. MQ=111 3. COMPLEMENT=ON 4. START 5. PAUSE 6. COMPLEMENT=OFF 7. MULTIPLY=ON 8. OVERFLOW=HALT 9. START
COMPLEMENT	1. INITIALIZE 2. COMPLEMENT=ON 3. START
MULTIPLY-ADD (A×B+C)	1. INITIALIZE 2. MULTIPLY 3. A=C 4. START

BINARY-DECIMAL CONVERSION TABLE

Decimal No.	Binary No.	Decimal No.	Binary No.	Decimal No.	Binary No.
0	0000000	43	0101011	86	1010110
1	0000001	44	0101100	87	1010111
2	0000010	45	0101101	88	1011000
3	0000011	46	0101110	89	1011001
4	0000100	47	0101111	90	1011010
5	0000101	48	0110000	91	1011011
6	0000110	49	0110001	92	1011100
7	0000111	50	0110010	93	1011101
8	0001000	51	0110011	94	1011110
9	0001001	52	0110100	95	1011111
10	0001010	53	0110101	96	1100000
11	0001011	54	0110110	97	1100001
12	0001100	55	0110111	98	1100010
13	0001101	56	0111000	99	1100011
14	0001110	57	0111001	100	1100100
15	0001111	58	0111010	101	1100101
16	0010000	59	0111011	102	1100110
17	0010001	60	0111100	103	1100111
18	0010010	61	0111101	104	1101000
19	0010011	62	0111110	105	1101001
20	0010100	63	0111111	106	1101010
21	0010101	64	1000000	107	1101011
22	0010110	65	1000001	108	1101100
23	0010111	66	1000010	109	1101101
24	0011000	67	1000011	110	1101110
25	0011001	68	1000100	111	1101111
26	0011010	69	1000101	112	1110000
27	0011011	70	1000110	113	1110001
28	0011100	71	1000111	114	1110010
29	0011101	72	1001000	115	1110011
30	0011110	73	1001001	116	1110100
31	0011111	74	1001010	117	1110101
32	0100000	75	1001011	118	1110110
33	0100001	76	1001100	119	1110111
34	0100010	77	1001101	120	1111000
35	0100011	78	1001110	121	1111001
36	0100100	79	1001111	122	1111010
37	0100101	80	1010000	123	1111011
38	0100110	81	1010001	124	1111100
39	0100111	82	1010010	125	1111101
40	0101000	83	1010011	126	1111110
41	0101001	84	1010100	127	1111111
42	0101010	85	1010101	128	10000000

etc.