DETAILED PROGRAMMING OF THE DIGI-COMP 1® EXPERIMENTS

INTRODUCTION

This booklet has been prepared because of the overwhelming acceptance of DIGI-COMP I and the many requests for details of the programming for each problem in its Manual. It can be helpful to everyone who desires to know more about programming and computer logical design.

An explanation of how each of the programs for the experiments in the DIGI-COMP I Manual were derived will be given. In doing so, it should become apparent how to derive programs for original problems.

The experiments given in the DIGI-COMP I Manual will be presented here in the order of the complexity of writing the program. Although this is not the order of the DIGI-COMP I Manual, the experiments will retain the same numbers for easy identification.

The experiments will illustrate various techniques used in programming and in computer logical design. The objectives of each experiment are outlined in the Table of Contents.

Chapter I

BOOLEAN ALGEBRA

Before we begin deriving the programs, a short course in Boolean Algebra is in order. Don't let that word scare you, for remember, the reason all digital computers use binary arithmetic is its simplicity, and Boolean Algebra is just the algebra of binary arithmetic.

On Page 13 of the DIGI-COMP I Instruction Manual, the terms used are defined as follows:

A = True = 1

$\overline{A}$ = False = 0 ($\overline{A}$ is read "Not A")

B = True = 1

$\overline{B}$ = False = 0 ($\overline{B}$ is read "Not B")

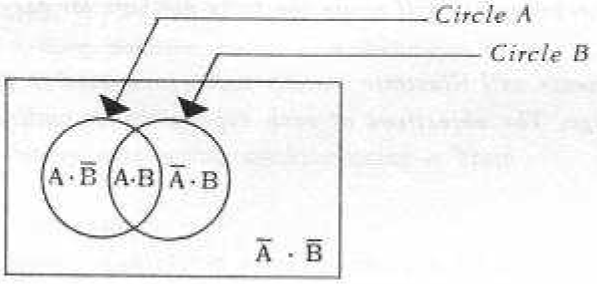
C = True = 1

$\overline{C}$ = False = 0 ($\overline{C}$ is read "Not C")

The operations or connections between the terms are:

The AND operation* is denoted by a dot, or multiplication sign** and the OR operation* is denoted by a plus or addition sign.** (The dot is omitted for brevity in some cases, but it should be understood that a dot is implied when there is no sign between terms.)

These operations may be illustrated graphically by what is known as a Venn Diagram.



Venn Diagram for Two Variables

FIGURE 1

**In formal logic the AND operation is called the INTERSECTION and the OR operation is called the UNION.*

***The dot and plus are used to denote the AND and OR operations simply because there are several rules of algebra which are similar to those of Boolean Algebra. However, you must remember that the dot is read AND and the plus is read OR. These do not, in any sense of the word, mean multiply and add.*

All points inside the ''A'' circle are represented by the variable A, and outside the ''A'' circle by the variable $\overline{A}$ (not A).

All points inside the ''B'' circle are represented by the variable B, and outside the ''B'' circle by the variable $\overline{B}$ (not B).

Where the two circles overlap, the points are in both the ''A'' circle AND the ''B'' circle. Thus the points in this area are designated $A \cdot B$ (A and B).

The points *outside* of both circles do not belong to either set, and are therefore designated $\overline{A} \cdot \overline{B}$ (not A and not B).

The points in A, but not in B are designated $A \cdot \overline{B}$ (A and not B) and similarly the points in B, but not in A are designated $\overline{A} \cdot B$ (not A and B).

Another way of looking at the Venn Diagram is illustrated in Figure 2.

The points in either ''A'' circle OR ''B'' circle represent the OR operation (or UNION) and would be written $A + B$ (A or B).

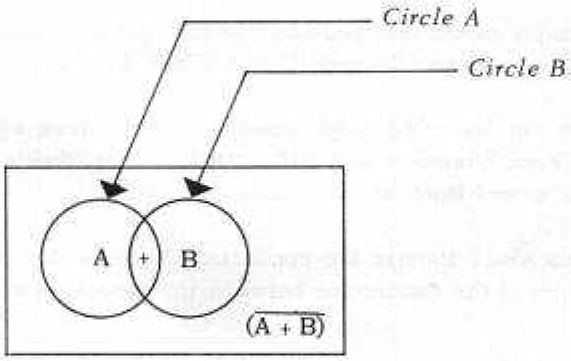


FIGURE 2

The area which is in neither circle is written $\overline{(A + B)}$ (read not A or B).

Now we are ready to develop the rules of Boolean Algebra using these Venn Diagrams.

In Figure 1 notice that all points within the area $A \cdot B$ are the same as those within $B \cdot A$, thus

$$A \cdot B = B \cdot A \qquad (1)$$

But also notice that points within the area $A \cdot \overline{B}$ do *not* equal the points within the area $\overline{A} \cdot B$.

In Figure 2, points within the area A + B are the same as points within the area B + A, thus

$$A + B = B + A \qquad (2)$$

It should be noted that the AND and OR functions are associative. That is:

$$(A \cdot B)\,C = A\,(B \cdot C) \qquad (3)$$

and

$$(A + B) + C = A + (B + C) \qquad (4)$$

Also, each function is distributive with respect to the other. That is:

$$A \cdot B + A \cdot C = A\,(B + C) \qquad (5)$$

and

$$(A + B)\,(A + C) = A + B\,C \qquad (6)$$

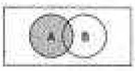
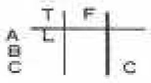
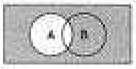
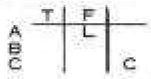
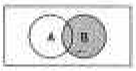
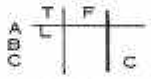
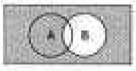
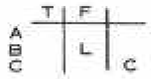
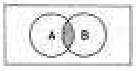
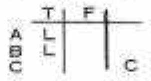
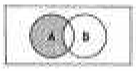
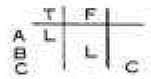
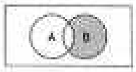
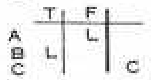
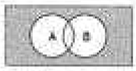
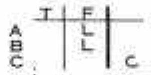
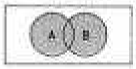
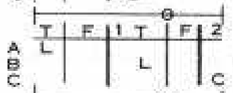
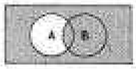
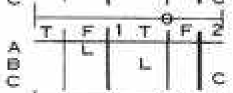
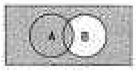
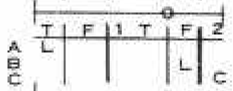
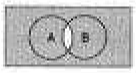
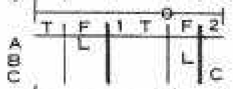
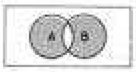
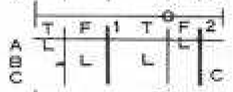
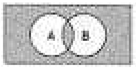
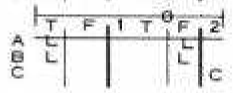
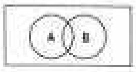
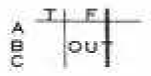
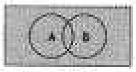
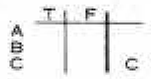
Equation (5) simply means that you can "factor" out a common term. Equation (6) defines the method for expanding a "binomial" in Boolean Algebra.

In Table 1, on the following page, some simple Boolean equations are presented with corresponding Venn Diagrams and DIGI-COMP coding sheets. Using the table we shall develop several useful Boolean relationships.

At this point we shall discuss the connection between the equations and the Venn Diagrams. In Chapter II the connection between the equations and the Coding Sheet will be discussed.

Table 1

Chart of Venn Diagrams and Digi-Comp Codes

BOOLEAN EQUATION	VENN DIAGRAM	DIGI-COMP CODING SHEET
(a) $C = A$		
(b) $C = \overline{A}$		
(c) $C = B$		
(d) $C = \overline{B}$		
(e) $C = A \cdot B$ $A \cap B$		
(f) $C = A \cdot \overline{B}$		
(g) $C = \overline{A} \cdot B$		
(h) $C = \overline{A} \cdot \overline{B}$		
(i) $C = A + B$ $A \cup B$		
(j) $C = \overline{A} + B$		
(k) $C = A + \overline{B}$		
(l) $C = \overline{A} + \overline{B}$		
(m) $C = (A \cdot \overline{B}) + (\overline{A} \cdot B)$		
(n) $C = (A \cdot B) + (\overline{A} + \overline{B})$		
(o) $C = 0$		
(p) $C = 1$		

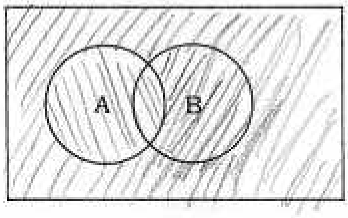
Note that a Venn Diagram for a function joined by the OR operation is just the super position (the union) of one diagram upon the other. (Any area shaded in either diagram is shaded in the joined diagram.)

A Venn Diagram for a function joined by the AND operation is the Intersection of the two diagrams. (The only area shaded is that which was shaded in BOTH separate diagrams.)

In Table 1, Equation (o) C = O, therefore, there are no points shaded in the Venn Diagram.

The Venn Diagram for Equation (p) has all points shaded. Thus a point anywhere in the Diagram satisfies the equation which is denoted by C = 1. Look at Equations (a) and (b) and shade in the Venn Diagram below for the Equation

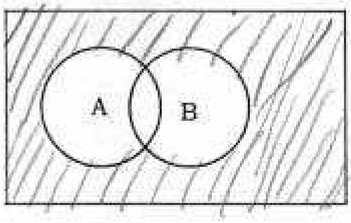
$$C = A + \overline{A}$$



If you did it properly the entire diagram should be shaded in. Thus

$$A + \overline{A} = 1 \qquad (8)$$

Now, look at Equation (i). In the Venn Diagram below, shade in the area $C = \overline{A + B}$. (The area not shaded in the Table.)



Comparing this Diagram to the one for Equation (h), shows that the points in $C = \overline{A} \cdot \overline{B}$ are the same as those for $C = \overline{A + B}$, therefore

$$\overline{A} \cdot \overline{B} = \overline{A + B} \qquad (9)$$

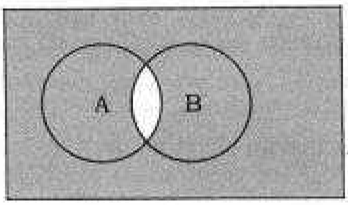
This is an illustration of DeMorgan's Theorem, which states that any binary expression is equal to the negation of the expression obtained by changing all unions to intersections and vice versa, and by replacing each variable with its negation. Another example would be

$$\overline{A \cdot B} = \overline{A} + \overline{B} \qquad (10)$$

This can be verified by looking at the Venn Diagram of the negation of Equation (e)

$$C = \overline{A \cdot B}$$

which is



and comparing it to the diagram for Equation (l).

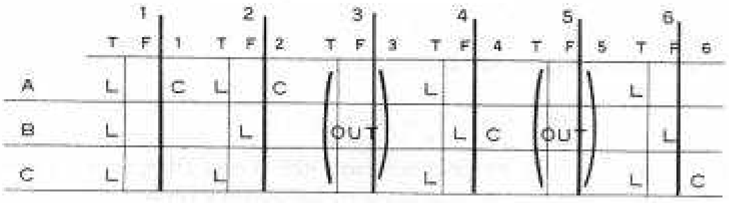
Many similar relationships can be derived from the Venn Diagrams. These are very convenient in the simplification of the programs to be written later.

EXPERIMENTS

EXPERIMENT 4 – BANK LOCK

Each morning the president of the MONEY bank has to open the bank vault. It is a combination lock using binary numbers. This morning the president was surprised to find he couldn't remember the combination. Can you open the vault for him?

Program DIGI-COMP as in the coding sheet below:



STEP 1 – Use Front Panel Card 1.

STEP 2 – You guess the binary combination and manually set the combination into the three flip-flops.

STEP 3 – Cycle the clock ONCE.

If you see $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ in the Read-Out, you have successfully opened the vault! If you do not see $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ try again! What is your chance of opening the vault on the first try?

Your chance of *guessing* the right combination to open the vault was only 1 out of 8, but you should have gotten it the first try – did you? This experiment was presented to be sure you understand how DIGI-COMP works.

You should have noticed that, in order to get $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ in the read-out, you had to move clock rods 2, 4 and 6 (the *set* rods). (Remember that to *SET* means to set a flip-flop to 1, and to *RESET*, means to reset a flip-flop to 0. To do this, the even numbered logic rods had to be free to move in when the clock was cycled.

Now, set up the Flip-Flops to Read-Out $\begin{smallmatrix}1\\0\\1\end{smallmatrix}$ and look at logic rods 2, 4 and 6. They are all free to move in when the clock is cycled. Try it!

To see how the Coding Sheet ties in with the Boolean Algebra equations, look at each logic rod individually.

Logic Rod #1 is connected to Clock Rod #1 which is a RESET Rod. If clock rod #1 is activated, it will move the A flip-flop to ZERO, because there is a clock tube on the A flip-flop in the #1 position. For Clock Rod #1 to be activated, Logic Rod #1 must be free to move in. For this logic rod to be free to move in, all three flip-flops must be in the "1" position. Now to put this in equation form:

Reset A when A = 1 and B = 1 and C = 1 (11)

Now we will replace each part of the sentence by its "short hand" algebraic counterpart as follows:

"Reset A" is replaced by "$\overline{A}$"

"When" is replaced by "="

"A = 1" is replaced by "A"

"AND" is replaced by "."

"B = 1" is replaced by "B"

"C = 1" is replaced by "C"

Therefore, the sentence (11) may be replaced by the equation:

$$\overline{A} = A \cdot B \cdot C$$

(notice how the logic rod is operating as an AND gate)

Now, to do the same thing for Clock Rod #2. If this rod is activated it will "SET" the A Flip-Flop to 1. And this will happen only if Logic Rod #2 moves in or if A = 1 AND B = 0 AND C = 1. Using the same relations as before:

$$A = A \cdot \overline{B} \cdot C$$

Clock Rods #4 and #6 operate just like #2, except they set Flip-Flops B and C. The equations for them are then:

$$B = A \cdot \overline{B} \cdot C$$

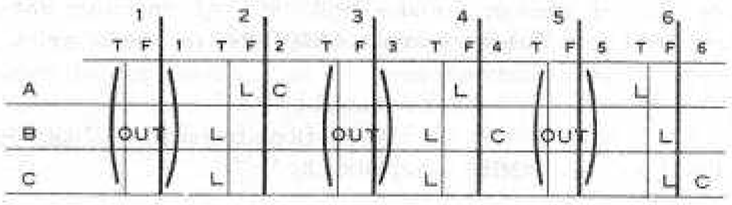
$$C = A \cdot \overline{B} \cdot C$$

Be sure you understand how these equations were derived from the Coding Sheet. This process, or the reverse of it, will be repeated in all Experiments to follow.

Now, turn back to Table 1 of Chapter I, and study the relation between the Boolean Equations and the Coding Sheets for the equations.

EXPERIMENT 5 – SEQUENTIAL BANK LOCK

To make it more difficult for anyone but the president of the bank to open the vault, a new type lock was installed. This lock requires *two* binary numbers in a *particular order* to open it. Program DIGI-COMP according to the coding sheet.



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card I.

STEP 2 – Make sure clock is all the way to the right.

STEP 3 – Manually move all flip-flops to the *right.* (See $\begin{smallmatrix}0\\0\\0\end{smallmatrix}$ in the Read-Out.)

STEP 4 – Guess the first binary number $\begin{smallmatrix}0&0&1\\0,&1,&0\end{smallmatrix}$ or $\begin{smallmatrix}1\\1\end{smallmatrix}$ and manually enter it in the A and B flip-flops.

STEP 5 – Cycle the clock ONCE. If the C flip-flop goes to "1" then the number you entered was correct and go to step 6. If it stays at "0" try Steps 4 and 5 again.

STEP 6 – Guess the second binary number $\begin{smallmatrix}0&0\\0,&1\end{smallmatrix}$ or $\begin{smallmatrix}1\\0\end{smallmatrix}$ not $\begin{smallmatrix}1\\1\end{smallmatrix}$ and manually enter it in flip-flops A and B.

STEP 7 – Cycle the clock ONCE. If you see $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ in the Read-Out, you have opened the vault. If not, try Steps 5 and 6 again.

This time what is your chance of opening the vault the first try?

This problem is twice as hard as Experiment 4 to *guess* the right combination; but it isn't any harder to figure out the solution if you understand how DIGI-COMP works.

To get the first number in the combination you should have noticed that if Logic Rod 6 were activated, a "1" would have appeared in the "C" Flip-Flop. Now set the "A" Flip-Flop to "1" and Logic Rod 6 is free to move in. Thus, the first combination you should have picked was $\begin{smallmatrix}1\\0\end{smallmatrix}$.

Now you want Flip-Flops A and B to be set to 1. So look at Positions 2 and 4. What combination will let these Logic Rods move in? That's right – $\begin{smallmatrix}0\\1\end{smallmatrix}$.

To do this problem in the more precise language of computers, you should write it in equation form:

Combination 1 – Set $C = A \cdot \overline{B} \cdot \overline{C}$

Combination 2 – Set $B = \overline{A} \cdot B \cdot C$
Set $A = \overline{A} \cdot B \cdot C$

Notice how the three equations compare with the Coding Sheet. "SET C" is programmed on Logic Rod 6; "SET B" on Logic Rod 4, and "SET A" on Logic Rod 2. See how, as in Experiment 4, the Logic Rod acts as an AND Gate. (A and B and C must be in the proper position for it to be activated.)

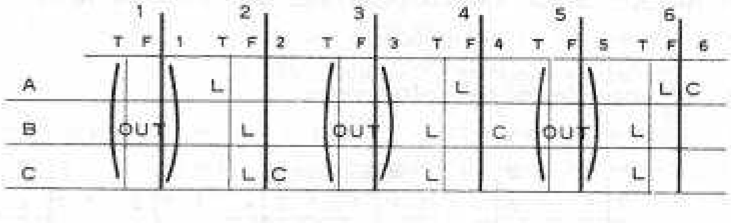
Logic Rods 1, 3 and 5 are for Resetting the flip-flops to zero, and Logic Rods 2, 4 and 6 are for Setting the flip-flops to ONE.

If a term has a BAR over it in the equation, (such as $\overline{B}$) put a Logic Tube on the "F" TAB of the B Flip-Flop.

If a term DOES NOT have a BAR over it (such as C) put a Logic Tube on the "T" TAB of the C Flip-Flop.

Since positions 2, 4 and 6 are identical except for the location of the Clock Tubes the selection of a position for a particular AND function is ARBITRARY. That is, positions 2, 4 and 6 could have been interchanged. (Since the EVEN numbered positions are for Setting to 1, and the ODD numbered positions are for Resetting to 0, THE EVEN AND ODD NUMBERED POSITIONS MAY NOT BE INTERCHANGED.)

For instance, if we interchanged positions 2 and 6 the Coding Sheet would be:



If you program DIGI-COMP according to this it will still operate in the same manner.

EXPERIMENT 7 – SPACE SHIP CHECK OUT
(A Job For An Astronaut)

This is the last moment before "blast off". It is time for you to make sure all the important parts ("systems") of your space ship are operating properly. You must now go through the final "check-out procedure".

If anything is wrong the launching will be canceled.

There are three systems to be checked:

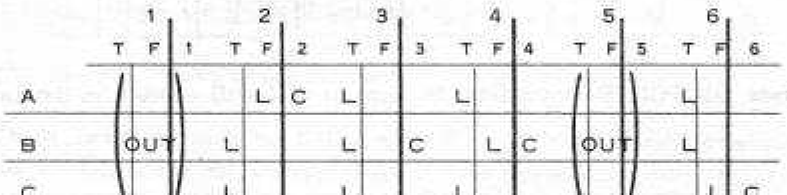
1. Oxygen system.
2. Space-Ship controls.
3. Radio.

To check these systems the following questions are asked. They must be answered correctly to avoid a cancellation of the launch.

1. How much oxygen is flowing?
2. Is the "control stick" in the correct position?
3. To what frequency is your radio turned?

All 3 answers have a numerical value. These values will be checked one at a time with the values stored in DIGI-COMP. If they check correctly the computer will indicate that the system is "A-O.K." If not, the computer will indicate a malfunction (not working properly).

Program DIGI-COMP according to the coding sheet:



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card I.

STEP 2 – Move all logic rods to INACTIVE (turn to the left) except Nos. 2 and 3.

STEP 3 – Guess how much oxygen is flowing and manually move the flip-flops to the numerical value (anything from 000 to 111).

STEP 4 – Cycle the clock ONCE. If the result is $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ in the Read-Out, then the system is operating "A-O.K.". If the result has a "0" ANYPLACE, then the system is NOT operating correctly. Move the flip-flops and cycle the clock until you get the "A-O.K." signal, $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$.

STEP 5 – Repeat Steps 3 and 4 for the next two questions. For the second question move logic rod number 4 to the ACTIVE POSITION (turn to the right). For the third question move logic rod number 6 to the right.

When all three systems are "A-O.K." you are ready for your interplanetary voyage.

Try to blast off the FIRST try.

It should be duck soup by now – if not, you'd better go back to Experiment 4 and start again.

To write the Boolean Equations for this program, proceed in the same manner as Experiment 4, only now you should be able to just look at the program on the Coding Sheet and write the equations. To check yourself, see Page 48 for the equations.

EXPERIMENT 3 – AUTOMATIC ELEVATOR

The automatic elevator is a common example of automation which uses MEMORY. A simple automatic elevator is the two floor elevator. The automatic system is required to REMEMBER at which of the two floors it has stopped.

In the elevator there are two buttons, one to tell the elevator to go to the 1st floor and the second to tell it to go to the 2nd floor. On each floor there is a button (CALL button) which you can push to tell the elevator to come to your floor.

Imagine the CALL button on the 1st floor is the same as the button in the elevator directing it to the 1st floor. Also, the 2nd floor CALL button is the same as the 2nd floor button in the elevator. Put FRONT PANEL CARD II into place. On this card there is marked "UP–DOWN". If the bottom flip-flop is at "1", the elevator is directed to the 2nd floor, and if it is at "0", to the 1st floor. (Look at the Read-Out next to UP–DOWN.) The floor at which the elevator is located is indicated by a "1" in the Read-Out by the floor.

DIGI-COMP can be used to show how this elevator works. **PROGRAM** DIGI-COMP according to the coding sheet.

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A			C			C												
B									C			C						
C		L		L			L				L							

To operate DIGI-COMP:

STEP 1 – Make sure logic and clock tubes are ONLY in the positions indicated on the coding sheet.

STEP 2 – Make sure all LOGIC RODS are in the ACTIVE POSITION (turned to the right) and the CLOCK is all the way out (to the right).

STEP 3 – Manually set the "**up–down**" flip-flop to "1" if you want to go up or to "0" if you want to go down.

STEP 4 – Cycle the clock.

This experiment is in the DIGI-COMP Manual to illustrate the concept of a MEMORY. DIGI-COMP does not really compute anything in this problem. It simply REMEMBERS which floor it is on, so that it knows which way to go when the button is pushed.

To see how it is done, program DIGI-COMP as in the Coding Sheet.

Now, set the Bottom "C" Flip-Flop to zero. Notice that the first and fourth Logic Rods are not blocked and can be pulled in. Then the first Clock Rod sets the "A" Flip-Flop to zero and the fourth Clock Rod sets the "B" Flip-Flop to 1.

The second and third Logic Rods operate in a similar manner to take the elevator to the second floor.

We said before that DIGI-COMP wasn't computing in this experiment – it was memorizing. It remembers which floor it is to go to, but not which floor it is at.

Set the Flip-Flops to read $\substack{1\\0\\1}$. This means that you want to go to the Second Floor – but you're already there! When you cycle the Clock the second and third Logic Rods are still free to move in; thus, the elevator motor would start needlessly.

Now, we want to correct the situation and make the elevator operate more efficiently. To do this put Logic Tubes on the "F" pegs at the second and third positions on the "A"$_0$ Flip-Flop. Now put the elevator on the first floor and push the second floor button $\substack{1\\1}$. The elevator moves to the second floor just as before.

Cycle the clock again and nothing happens. Now the elevator is operating more efficiently.

Set the "C" Flip-Flop to zero $\substack{1\\0\\0}$ and cycle the Clock – now you go back down.

What would you do with the first and fourth positions to keep them from actuating when you have $\substack{0\\1\\0}$?

Try it.

Now DIGI-COMP is COMPUTING – because it COMPARES where it wants to go with where it already is, and decides whether to go up, down, or stay still.

If you wish to become even more sophisticated, we can use DIGI-COMP to open the elevator door. To see how to do this, we must first determine when the door should be opened. That is, when you are where you want to be $\substack{0\\1\\0}$ or $\substack{1\\0\\1}$. So if we use Logic Rod 5 to open the door on the second floor, set DIGI-COMP to $\substack{1\\0\\1}$ and look at the T and F pegs in the number five position. Notice that if we program it by putting logic tubes as follows:

	5 T	5 F
A	L	
B		L
C	L	

The rod is free to move in and thus open the door.

This is a good illustration of how the logic rod operates as an AND gate.

A must = 1
AND
B must = 0
AND
C must = 1

For the door to be opened.

Now you should be able to figure out how to make the sixth Logic Rod (AND GATE) open the door on the first floor.

If you have trouble, turn to Page 48 for the complete program. Make sure you understand it before proceeding to the next Experiment.

EXPERIMENT 6 – HO-HUM (Logical Riddle)

In the Pacific Ocean there are two neighboring islands which few white men have seen. If you were to take a trip and visit these islands you would find the natives of island HO always tell the truth while the natives of island HUM always tell falsehoods. As an outsider you cannot tell to which island any of the natives belong. When you visit the islands you see a group of three natives standing around, and out of curiosity ask the first to which island he belongs.

FIRST NATIVE'S ANSWER: "BZYTPL."

Since he doesn't speak English, you cannot understand his answer. You ask his two friends: "What did he say?"

SECOND NATIVE ANSWERS: He say he belong "HUM".

THIRD NATIVE ANSWERS: "No, first man say he belong HO."

Riddle: To which island does the **third** native belong?

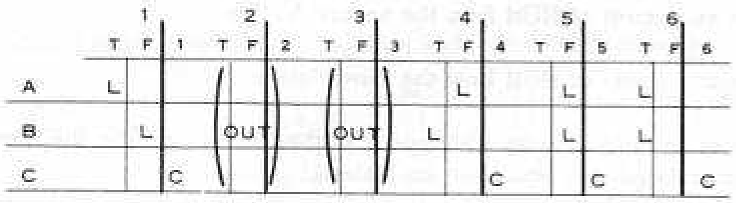
To operate DIGI-COMP:

STEP 1 – Put Front Panel Card III into the slots on the Front Panel.

STEP 2 – Make sure clock is all the way out (to the right).

STEP 3 – Program DIGI-COMP according to the coding sheet.

(You may find it helpful to move some of the logic rods to the INACTIVE POSITION (turn to the left) when putting on logic tubes.)



STEP 4 – Make sure logic rods 1, 4, 5 and 6 are in the ACTIVE POSITION (turn to the right).

STEP 5 – With the Manual Input Tabs enter the answers of the second and third natives into the flip-flops indicated by the Front Panel Card. If the answer was HO enter a "1", if HUM enter "0".

STEP 6 – Cycle the clock ONCE.

The answer in the Read-Out will be "1" if the third native is from HO and "0" if he is from HUM.

Try entering other answers for the 2nd and 3rd Natives. DIGI-COMP will always correctly tell you from where the third native comes.

To program a riddle of this sort, you must first fully understand the problem yourself. To do this you must break the problem down to its most simple form – this is called synthesizing a problem. There are four combinations of answers that you could get from the second and third Natives.

1. First, if the second Native answers HO and the third answers HUM, you know one of them is lying. But how do you determine which it is? To find the answer to this problem, you must remember the question you put to the natives. It was "which Island did he *say* he belonged to," not "which island did he *actually* belong to." And here lies the solution to the problem. If the first native was from HO, he would tell the truth and say he was from HO. If he was from HUM, he would lie and still say he was from HO. Therefore, you know that the Native who said he said he was from HUM was lying. Thus, in the first case, the third Native was from HUM.

2. If the second Native had answered HUM and the third answered HO, then the second was lying and the third was from HO.

3. If they both had answered HUM, then they both lied so that the third was from HUM.

4. If they both had answered HO, then they both told the truth so the third Native was from HO.

To program this, now that we understand the problem:

let A = an answer of HO from the second Native

B = an answer of HO from the third Native

then $\overline{A}$ = an answer of HUM from the second Native

$\overline{B}$ = an answer of HUM from the third Native

Since the answer to the riddle should appear in the "C" Flip-Flop the above four statements can be translated into equations as follows:

1. The first set of answers were:

   Second Native: HO (enter a 1 in Flip-Flop A)

   Third Native: HUM (enter a 0 in Flip-Flop B)

Since the third Native lied, we want the answer in C to be 0, when $A = 1$ *and* $B = 0$. So the first equation is:

$$\text{Reset } C = A\,\overline{B}$$

2. The second set of answers were:

Second Native: HUM (enter a 0 in Flip-Flop A)

Third Native: HO (enter a 1 in Flip-Flop B)

Since the third Native told the truth, we want the answer in C to be 1, when A = 0 *and* B = 1. So the second equation is:

$$\text{Set } C = \overline{A} \cdot B$$

3. The third set of answers were:

Second Native: HUM (enter a 0 in Flip-Flop A)

Third Native: HUM (enter a 0 in Flip-Flop B)

Since the third Native lied, we want the answer in C to be 0, when A = 0 *and* B = 0. So the third equation is:

$$\text{Reset } C = \overline{A} \cdot \overline{B}$$

alternate:
Clock rods: 1C, 2C
Logic rods: 1Bf, 2Bt
all the rest: (OUT)

4. The fourth set of answers were:

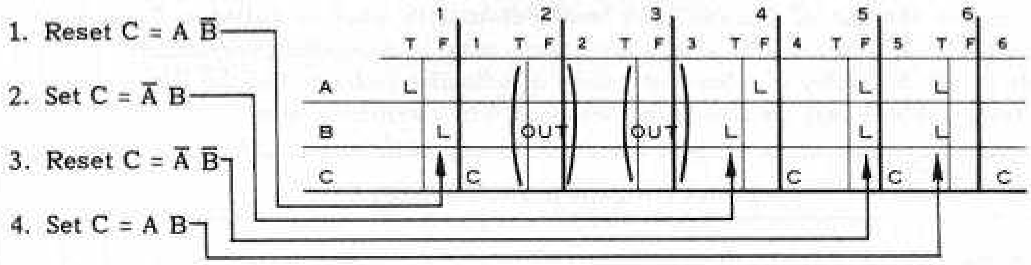
Second Native: HO (enter a 1 in Flip-Flop A)

Third Native: HO (enter a 1 in Flip-Flop B)

Since the Third Native told the truth, we want the answer in C to be 1, when A = 1 *and* B = 1. So the fourth equation is:

$$\text{Set } C = A \cdot B$$

Now, let's bring these four equations together and we're ready to program.



1. Reset C = A $\overline{B}$
2. Set C = $\overline{A}$ B
3. Reset C = $\overline{A}$ $\overline{B}$
4. Set C = A B
1 2 3 4 5 6
T F 1 T F 2 T F 3 T F 4 T F 5 T F 6
A
B
C
L
OUT

Notice how the equations lead directly to the program, just as in the previous experiments.

In the first one, for instance, A means "put a logic tube on the 'T' peg of Flip-Flop A"; B means "put a logic tube on the 'F' peg of Flip-Flop B" in Position 1. EASY, ISN'T IT? The same follows for equations 2, 3 and 4.

You might wonder why we picked "and gates" 1, 4, 5 and 6. There is no special reason; you could program on any of the "and gates" as long as "and gates" 2, 4 and 6 are used for setting and 1, 3 and 5 are used for resetting.

EXPERIMENT 10 – BINARY COUNTER

Binary counters are used in nearly all digital computers and automatic devices. There are many uses for these counters.

They may count *events* or *things*. For example, a binary counter is used to count time so a computer will know when it has to do different operations. Binary counters can be used to automatically count candy bars being sent to the grocery store.

You have already counted *down* to "blast off" a space ship. Now see if you can count *up* from 0 to 7 in binary. Try writing the count from 000 to 111.

To check yourself, use DIGI-COMP. Program DIGI-COMP as in the coding sheet.

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A	L		C		L	C	L			L			L			L		
B							L		C		L	C	L			L		
C													L		C		L	C

To operate DIGI-COMP:

STEP 1 – Use Front Panel Card I.

STEP 2 – Make sure clock is out. Enter "0"'s into all 3 flip-flops.

STEP 3 – Cycle the clock slowly a number of times. Look at the Read-Out and watch DIGI-COMP count!

To program the Binary Counter, you would start with what is called a Truth Table. Truth Tables or other orderly arrays are always used by Computer Programmers. A Truth Table is just a display of rules that govern a particular problem. For the Binary Counter the Truth Table is just the Binary numbers from 0 to 7 written in order.

BINARY NUMBER TRUTH TABLE

Decimal No.	0	1	2	3	4	5	6	7	0
Flip-Flop A	0	1	0	1	0	1	0	1	0
Flip-Flop B	0	0	1	1	0	0	1	1	0
Flip-Flop C	0	0	0	0	1	1	1	1	0

Now look at the Truth Table and see when you want to change the setting of each of the Flip-Flops. For example:

Flip-Flop A should be changed from 0 to 1 when

A = 0 and B = 0 and C = 0
or
A = 0 and B = 1 and C = 0
or
A = 0 and B = 0 and C = 1
or
A = 0 and B = 1 and C = 1

These word statements can be shortened by writing them in the form of Boolean Algebra. To review the rules for this once more, remember that:

1. $\overline{A}$ stands for "the A flip-flop in the 0 position",

2. A stands for "the A flip-flop in the 1 position",

3. Replace the word "and" by a multiplication sign ($\cdot$)

4. Replace the word "or" by an addition sign (+)

Then the four above statements for changing (setting) flip-flop A from 0 to 1 become –

$$\text{Set } A = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C$$

See, it wasn't so hard!

Next, we'll write the statements for resetting Flip-Flop A from 1 to 0.

A = 1 and B = 0 and C = 0
or
A = 1 and B = 1 and C = 0
or
A = 1 and B = 0 and C = 1
or
A = 1 and B = 1 and C = 1

Rewriting these equations in Boolean form as we did before:

$$\text{Reset } A = A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot C$$

To reset B from 1 to 0

A = 1 and B = 1 and C = 0
or
A = 1 and B = 1 and C = 1

Writing these equations in Boolean form gives:

$$\text{Reset } B = A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

Now, we do the same thing for Setting B = 1 and get:

$$\text{Set } B = A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C$$

And similarly:

$$\text{Reset } C = A \cdot B \cdot C$$

$$\text{Set } C = A \cdot B \cdot \overline{C}$$

Now, if we bring these six equations for setting and resetting all of the flip-flops together, we get:

(1) $\text{Reset } A = A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot C$

(2) $\text{Set } A = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C$

(3) $\text{Reset } B = A \cdot B \cdot \overline{C} + A \cdot B \cdot C$

(4) $\text{Set } B = A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C$

(5) $\text{Reset } C = A \cdot B \cdot C$

(6) $\text{Set } C = A \cdot B \cdot \overline{C}$

These are the six equations that together determine how DIGI-COMP is to be programmed. In these six equations, however, there are fourteen "AND" functions (count them). But on DIGI-COMP there are only 6 "And Gates". Fortunately, however, we can reduce these equations by "factoring" out common terms in equation (1). (See Page 6.)

$$\begin{aligned}\text{Reset } A &= A\,(\overline{B} \cdot \overline{C} + B \cdot \overline{C} + \overline{B} \cdot C + B \cdot C)\\ &= A\,[(\overline{B} + B)\,\overline{C} + (\overline{B} + B)\,C]\\ &= A\,[1\,(\overline{C} + C)]\\ &= A\,(1)\\ &= A\end{aligned}$$

$$\text{Reset } A = A$$

Similarly, we find from equation 2, that:

$$\text{Set } A = \overline{A}\,(\overline{B} \cdot \overline{C} + B \cdot \overline{C} + \overline{B} \cdot C + B \cdot C)$$

Set A = $\overline{A}$ by the same reasoning.

Now, let's look at the B flip-flop – it's easier.

$$\text{Reset B} = A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

$$= A \cdot B\,(\overline{C} + C)$$

$$= A \cdot B\,(1)$$

$$= A \cdot B$$

$$\text{Set B} = A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C$$

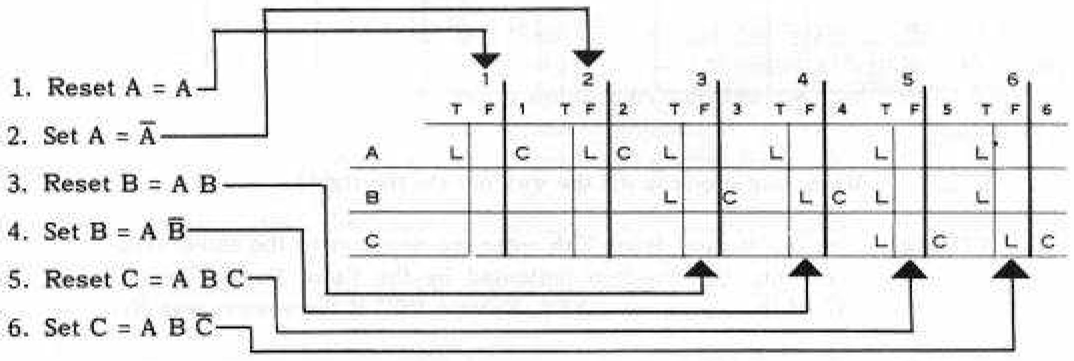
$$= A \cdot \overline{B}\,(\overline{C} + C)$$

$$= A \cdot \overline{B}\,(1)$$

$$= A \cdot \overline{B}$$

We're almost done!

Let's look at the six equations now. Compare them to the Coding Sheet and you'll see how it all fits together. If you're not absolutely sure how we did it, you'd better go back over the problem again. Remember, when there is no – over the letter, the logic tube goes on the "T" peg of the Flip-Flop, and when there is a – it goes on the "F" peg.



1. Reset A = A
2. Set A = $\overline{A}$
3. Reset B = A B
4. Set B = A $\overline{B}$
5. Reset C = A B C
6. Set C = A B $\overline{C}$
1 2 3 4 5 6
T F 1 T F 2 T F 3 T F 4 T F 5 T F 6
A
B
C
L
C

EXPERIMENT 8 – "GUESS THE NUMBER" (A Logical Riddle)

Here is a *logical riddle* to test your reasoning powers. Have a friend choose any number between 0 and 7, (0, 1, 2, 3, 4, 5, 6, 7). You are to find the number he chose from his answers to three questions. He will answer Yes or No to each question.

QUESTION 1: Is the number even?

QUESTION 2: Is the number 0, 1, 2 or 3?

QUESTION 3: Add 10 (ten) to the number and divide the result by 6; is the *remainder* 0, 1, 2 or 3? (For example if the number is 5, then 5 + 10 = 15 and 15 divided by 6 is 2 plus a remainder of 3. The answer to the statement would be *Yes.* If the number is 7, then 7 + 10 = 17 and 17 divided by 6 is 2 plus a remainder of 5. The answer would be *No.*)

Now try it:

STEP 1 – Put Front Panel Card IV in place.

STEP 2 – Program DIGI-COMP according to the coding sheet.

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A	L		C		L	C												
B							L		C		L	C	L				L	
C								L			L		L		C		L	C

STEP 3 – Make sure clock is all the way out (to the right).

STEP 4 – With the Manual Input Tab enter the answers to the three questions into the flip-flop indicated by the Front Panel. Enter a "1" if the answer was YES. Enter a "0" if the answer was NO.

STEP 5 – Cycle the clock ONCE.

The Read-Out will have the number your friend chose in the Binary System.

If you played Guess the Number you no doubt tried to beat DIGI-COMP. But unless you're pretty quick, you couldn't keep up. Here is where you will learn a great lesson about computers. Even though DIGI-COMP PROBABLY Guessed the Number much faster than you did, in a few minutes you'll be able to tell DIGI-COMP how to do it!!

Let us begin the solution by answering the three questions for each number from 0 to 7, and, again putting them in a Truth Table.

For instance, for a secret number of 0:

QUESTION 1: Yes, it is even.

QUESTION 2: Yes, it is less than 4.

QUESTION 3: No, the remainder is not less than 4.

Now we will fill out the Truth Table for all 8 secret numbers substituting 1 for Yes and 0 for No.

SECRET NUMBER TRUTH TABLE

Secret Number	0			1			2			3			4			5			6			7		
Flip-Flop	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
Answers to Questions	1	1	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	1	0	0	0	0	0
Secret Number in Binary Form	0	0	0	1	0	0	0	1	0	1	1	0	0	0	1	1	0	1	0	1	1	1	1	1

Now, we will do the same thing we did in the Binary Counter. We look at a particular Flip-Flop and see when it is necessary to change it. First, for Flip-Flop A, we can see from the Table that it is necessary to reset it from 1 to 0 whenever the secret number is 0, 2, 4 or 6. That is, when the answers to the questions are:

A = 1 and B = 1 and C = 0 (Secret No. = 0)
or
A = 1 and B = 1 and C = 1 (Secret No. = 2)
or
A = 1 and B = 0 and C = 1 (Secret No. = 4)
or
A = 1 and B = 0 and C = 0 (Secret No. = 6)

Now, changing this, the same as before, into Boolean Algebraic Equations, gives:

$$\text{Reset } A = A\,B\,\overline{C} + A\,B\,C + A\,\overline{B}\,C + A\,\overline{B}\,\overline{C}$$

Again we will factor the equation, pulling out the common factor A.

$$\text{Reset } A = A\,(B\,\overline{C} + B\,C + \overline{B}\,C + \overline{B}\,\overline{C})$$

And, just as in the binafy counter, since B and C can have any value,

$$\text{Reset } A = A$$

Next, looking at the Truth Table, we want to Set A = 1, when the secret number is 1, 3, 5 or 7; that is when:

A = 0 and B = 1 and C = 0 (Secret No. = 1)
or
A = 0 and B = 1 and C = 1 (Secret No. = 3)

or

A = 0 and B = 0 and C = 1 (Secret No. = 5)

or

A = 0 and B = 0 and C = 0 (Secret No. = 7)

Changing this to Boolean Algebra gives:

Set $A = \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + \overline{A}\,\overline{B}\,C + \overline{A}\,\overline{B}\,\overline{C}$

Set $A = \overline{A}\,(B\,\overline{C} + B\,C + \overline{B}\,C + \overline{B}\,\overline{C})$

Set $A = \overline{A}$

Now, we shall look back at the truth table and see when B is changed from 1 to 0.

Reset $B = A\,B\,\overline{C} + \overline{A}\,B\,\overline{C}$ (For 0 or 1)

Reset $B = B\,\overline{C}\,(A + \overline{A})$

Reset $B = B\,\overline{C}$

Similarly:

Set $B = A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$ (For 6 or 7)

Set $B = \overline{B}\,\overline{C}\,(A + \overline{A})$

Set $B = \overline{B}\,\overline{C}$

Next, to find when C changes:

Reset $C = A\,B\,C + \overline{A}\,B\,C$ (For 2 or 3)

Reset $C = B\,C\,(A + \overline{A})$

Reset $C = B\,C$

And for the last one –

Set $C = A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$ (For 6 or 7)

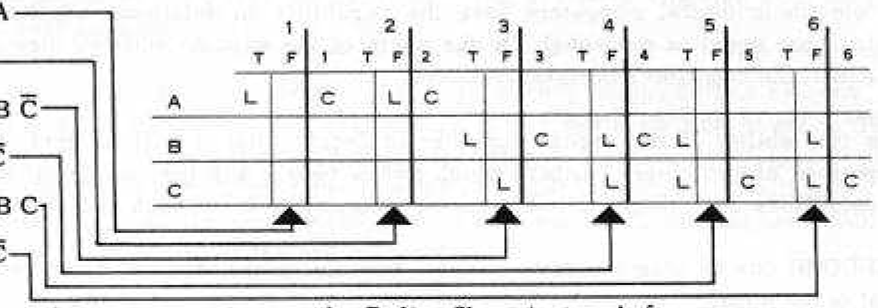
Set $C = \overline{B}\,\overline{C}\,(A + \overline{A})$

Set $C = \overline{B}\,\overline{C}$

To program DIGI-COMP, all we have to do is put logic plugs on the proper pegs, and we've built a machine that can outsmart almost anyone.

The six equations are:

1. Reset A = A
2. Set A = $\overline{A}$
3. Reset B = B $\overline{C}$
4. Set B = $\overline{B}$ $\overline{C}$
5. Reset C = B C
6. Set C = $\overline{B}$ $\overline{C}$



You can compare these six equations to the Coding Sheet just as before.

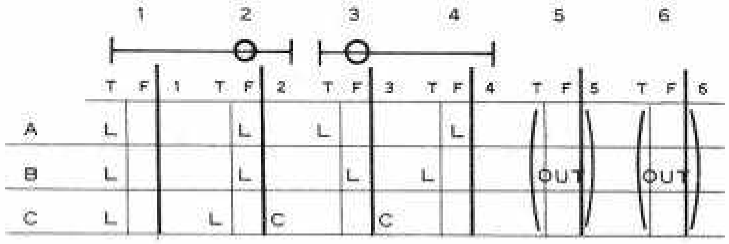
EXPERIMENT 12 – BINARY NUMBER COMPARATOR

All electronic digital computers have the capability to determine whether two binary numbers are equal or not equal. On the basis of the answer (whether they are equal or not equal) the computer can make *decisions.*

It is this ability of the digital computer to decide what it will do next, based on the comparison of two binary numbers which makes people say the computer "thinks". The computer does not "think" since someone programmed it for each choice it could make.

DIGI-COMP can be programmed to compare two binary numbers and determine if they are equal or not equal.

To perform this experiment, program DIGI-COMP as in the coding sheet.



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card VI.

STEP 2 – Make sure clock is out. Move the C flip-flop to "1".

STEP 3 – Enter the first bits of the two numbers into flip-flops A and B (see example below).

STEP 4 – Cycle the clock.

If the two binary bits are equal, C will have a "1" in its Read-Out. If they are not equal, C will have a "0" in its Read-Out. Stop when you see the first "0".

Repeat Steps 3 and 4 for the other bits of the two binary numbers. If at *any time* C is "0", then the two numbers are *not* equal.

FOR EXAMPLE: compare the two binary numbers

A – 1 0 1 1 0 ← enter these bits in A

and B – 1 1 1 1 0 ← enter these bits in B

enter these two bits 1st

enter these two bits 2nd

enter these two bits 3rd and so on.

You may think this is a very simple minded thing to do – "all *you* have to do is look at the numbers, and you will see they are different." But you have forgotten something – computers are only machines – *they can't look!*

To emphasize how powerful this operation is – because a computer can compare, it knows when it has the correct answer, even though it may not know how to get it. This is called an *Itterative Process* which is used very often in computers. To give you an idea how it works, if you give a computer a group of preset conditions, it will guess an answer and then test it on the conditions. Then it will guess another number which is more correct, and so on till it has the exact answer.

To program the Comparator, we must first decide when to *"set"* AND WHEN TO RESET the C Flip-Flop.

We want to SET C when A AND B are the same. We want to RESET C when A AND B are different. But also we want to keep C = 0 if ANY two bits differ.

O.K., let's write the equations.

We want to reset C if A and B are different, that is:

$$\begin{gathered}\text{when } A = 1 \text{ AND } B = 0\\ \text{OR}\\ A = 0 \text{ AND } B = 1\end{gathered}$$

then, in equation form, this would be:

$$\text{Reset } C = A\,\overline{B} + \overline{A}\,B$$

Also, we want to Set C if A and B are the same, that is

$$\begin{gathered}\text{when } A = 1 \text{ AND } B = 1\\ \text{OR}\\ A = 0 \text{ AND } B = 0\end{gathered}$$

Then in equation form, this would be:

$$\text{Set } C = A\,B + \overline{A}\,\overline{B}$$

This is fine for a start, but we've forgotten something.

If there is a difference between the two numbers being compared, we want to Reset C to "0" and LEAVE IT THERE. To satisfy this condition we shall put a logic tube on the T peg of the C Flip-Flop in both positions 1 and 2. Now if the C Flip-Flop is ever "0" it will remain there, for the duration of the number being compared, because the setting Logic Rods (1 & 2) will be blocked by these two Logic Tubes.

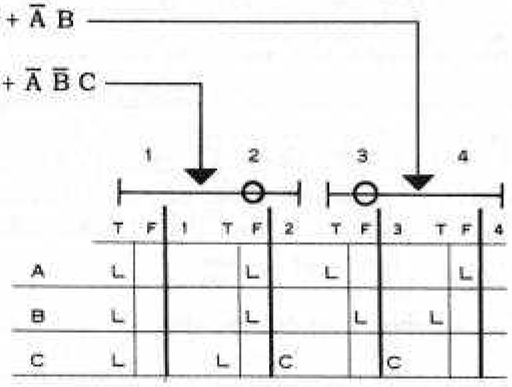
Thus the entire equation becomes:

$$\text{Set } C = A\,B\,C + \overline{A}\,\overline{B}\,C$$

And the Program follows directly from these two equations.

$$\text{Reset } C = A\,\overline{B} + \overline{A}\,B$$

$$\text{Set } C = A\,B\,C + \overline{A}\,\overline{B}\,C$$



Here, as usual, we arbitrarily chose to program Reset C on Logic Rods 3 & 4. The only thing that matters is that since this is a Reset function, the Clock Rod which is activated by the Double Slider (OR GATE) is ODD (Position 1, 3, or 5).

It should be clear now, if not before, where the OR GATE fits into the picture. In this case, the equations could not be simplified to remove the OR from the functions.

EXPERIMENT 13 – BINARY ADDER

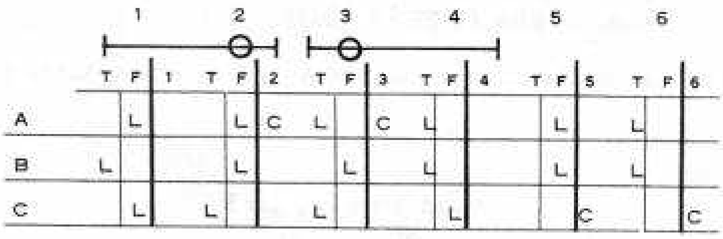
You have learned how to add binary numbers in the chapter on "The Language of Computers". Now you will *see* how a digital computer adds two binary numbers.

Remember, in addition, there can be a "carry" from the previous column.

```
FOR EXAMPLE:    carry    11
                          110
                       +  011
                       ------
                         1001
```

The *adder* is a basic unit in all digital computers. Your DIGI-COMP can be programmed to operate as an adder which will work exactly as an adder in large electronic digital computers works.

Program DIGI-COMP as in the coding sheet.



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card VII.

STEP 2 – Make sure clock is out. Manually enter the first two binary bits to be added into the A and B flip-flops (see example). Make sure the C flip-flop is "0".

STEP 3 – Cycle the clock ONCE.

A will remember the *sum* and C will remember the *carry*.

STEP 4 – Copy the Read-Out of A onto a sheet of paper. This is the first bit of your answer.

STEP 5 – Enter the second bit of the two numbers onto A and B. DO NOT TOUCH C (C remembers the carry from the previous step).

STEP 6 – Cycle the clock ONCE.

Repeat Steps 4, 5 and 6 until you have added the two numbers. Remember to copy the carry after the last operation.

To program the Binary Adder: First make the usual Truth Table with all possible values for A, B and C (the carry). Add them up and see what should go into the A and C Flip-Flops when DIGI-COMP is clocked.

A plus B plus C = Ans. plus	*Carry*
0 plus 0 plus 0 = 0 plus	0
0 plus 0 plus 1 = 1 plus	0
0 plus 1 plus 0 = 1 plus	0
1 plus 0 plus 0 = 1 plus	0
1 plus 0 plus 1 = 0 plus	1
1 plus 1 plus 0 = 0 plus	1
0 plus 1 plus 1 = 0 plus	1
1 plus 1 plus 1 = 1 plus	1

Now, look at the answer column and see when the A plane should be Set to 1.

You want to Set A when the answer is 1, or when:

A = 0 and B = 0 and C = 1
or
A = 0 and B = 1 and C = 0
or
A = 1 and B = 0 and C = 0
or
A = 1 and B = 1 and C = 1

or putting this into Boolean Algebra:

$$\text{Set A} = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$$

To reduce this equation we shall use a new "trick" which is simple but not obvious. If you look at the last two terms in the equation ($A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$) the value of "A" is 1, therefore since the A Flip-Flop must be in the 1 position for these to be activated there is no use in setting it to 1, and the equation reduces to:

Set A = $\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}$ (which is programmed on the first two And Gates.)

Similarly, if you do the work yourself, and you should be able to, you will find that:

$$\text{Reset A} = A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$
$$\text{Set C} = A \cdot B$$
$$\text{Reset C} = \overline{A} \cdot \overline{B}$$

If you have trouble, see Page 48 for the complete solution. Thus, the four equations are:

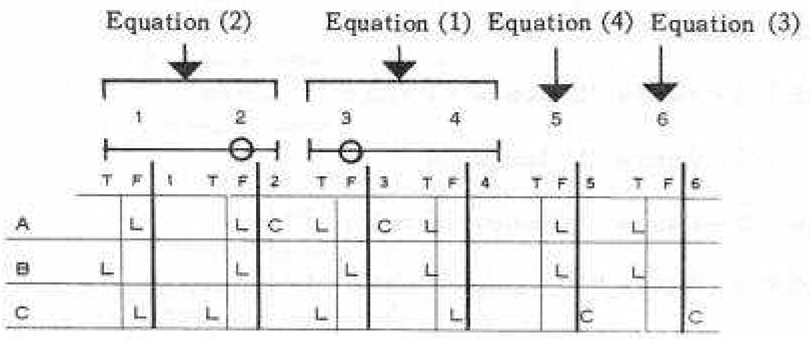
(1) Set $A = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C}$

(2) Reset $A = A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$

(3) Set $C = A \cdot B$

(4) Reset $C = \overline{A} \cdot \overline{B}$

The Coding Sheet follows directly from these equations.



EXPERIMENT 14 – BINARY SUBTRACTOR

Binary subtraction is as simple as binary addition.

The rules for binary subtraction are:

RULE 1: 1 – 1 = 0

RULE 2: 1 – 0 = 1

RULE 3: 0 – 1 = 1 with a "1" borrow.

RULE 4: 0 – 0 = 0

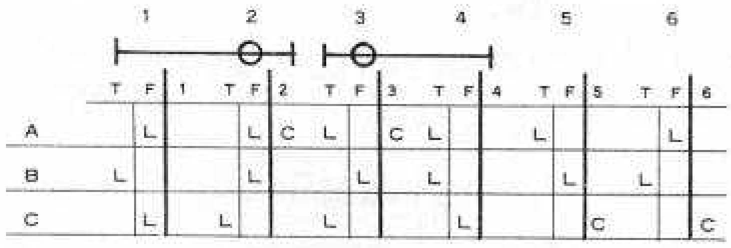
RULE 5: 1 – 1 and a "1" borrow = 1 with a "1" borrow.

RULE 6: 1 – 0 and a "1" borrow = 0

RULE 7: 0 – 1 and a "1" borrow = 0 with a "1" borrow.

RULE 8: 0 – 0 and a "1" borrow = 1 with a "1" borrow.

The program for a binary subtractor is given in the coding sheet. Compare this coding sheet with the one for the binary adder. Notice the only difference is at Position 5 and 6.



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card VIII.

STEP 2 – Make sure clock is out. Make sure C is "0".

STEP 3 – Enter the first bit of the *subtrabend* (1st No.) in the top flip-flop and the first bit of the *subtractor* (2nd No.) in the middle flip-flop.

STEP 4 – Cycle the clock ONCE.

STEP 5 – Copy the RESULT (in top flip-flop) on a piece of paper.

STEP 6 – Enter the second bits of the two numbers. DO NOT TOUCH C as it contains the "borrow" from the previous step.

STEP 7 – Cycle the clock ONCE. Repeat Steps 5, 6 and 7 until the two numbers are subtracted.

REMEMBER, the number subtracted must be less than, or equal to the number from which it is being subtracted.

To program a Binary Subtractor we use the same technique as in the Binary Adder. First, we will translate the rules into a Truth Table.

A minus B minus C = Answer	*C Borrow*
1 minus 1 minus 0 = 0	0
1 minus 0 minus 0 = 1	0
0 minus 1 minus 0 = 1	1
0 minus 0 minus 0 = 0	0
1 minus 1 minus 1 = 1	1
1 minus 0 minus 1 = 0	0
0 minus 1 minus 1 = 0	1
0 minus 0 minus 1 = 1	1

Now look at the answer column and see when the A plane should be Set to 1.

You want to Set A when the answer is 1, or when:

A = 1 and B = 0 and C = 0
or
A = 0 and B = 1 and C = 0
or
A = 1 and B = 1 and C = 1
or
A = 0 and B = 0 and C = 1

Now, putting this into Boolean Algebra –

$$\text{Set } A = A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,C + \overline{A}\,\overline{B}\,C$$

Again, since A is already Set to 1 the first and third terms can be neglected.

$$\text{Set } A = \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$$

Try to work out the rest of the equations on your own. They are:

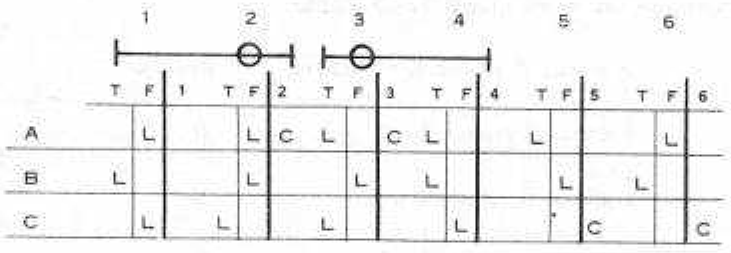
$$(1)\ \text{Set } A = \overline{A}\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$$

$$(2)\ \text{Reset } A = A\,\overline{B}\,C + A\,B\,\overline{C}$$

(3) Reset C = A $\overline{B}$

(4) Set C = $\overline{A}$ B

If you have trouble, turn to Page 49 for the complete solution. Again, the equations correspond directly to the Coding Sheet.



In most electronic computers, Binary Subtraction is not performed in the direct manner which is used by DIGI-COMP.

If a larger number were to be subtracted from a smaller number the result would be negative. Since most computers are not equipped to handle negative numbers, subtraction is performed by what is called a Complement and Add operation.

The Complement of a number, often more particularly called the 2's Complement, may be obtained from the given number by first changing every "0" bit to a "1" and vice-versa, (this is called the 1's Complement) then adding one to the resulting number. Thus, the 1's Complement of (0010110) is (1101001) and the 2's complement is (1101001) + 1 = (1101010).

To explain by example how the entire subtraction process goes, suppose we want to subtract 22 = (0010110) from 39 = (0100111). If DIGI-COMP were to do the problem with its direct subtraction technique, it would subtract one bit at a time as follows:

```
Subtrahend     0100111    39
Subtractor    -0010110   -22
              --------   ---
Difference       10001    17
```

Now, if an electronic computer were to do the same problem it would first find the 2's complement of 39 which is (1011001), add it to 22, and take the 2's complement of the sum.

```
2's Complement of Subtrahend    1011001
                  Subtractor    0010110
                               --------
             SUM                1101111

1's Complement of Sum     0010000
2's Complement of Sum       10001 = 17
```

As you can see, this procedure is quite laborious by hand, but with high speed computers it proves to be worthwhile.

EXPERIMENT 15 – BINARY MULTIPLIER

Binary multiplication is carried out in a way similar to decimal multiplication. The 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.

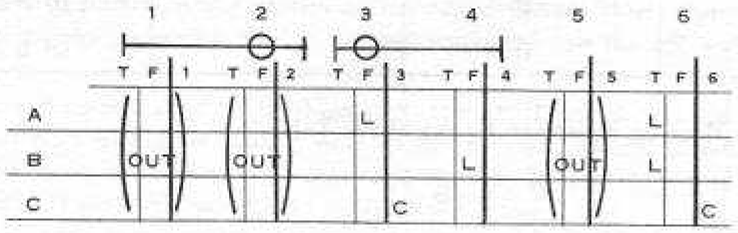
For example:

```
              In binary:        In decimal:

                 1 1 0 1              13
               x 1 1 0 1            x 13
               ---------            ----
                 1 1 0 1              39
               0 0 0 0
             1 1 0 1                  13
           1 1 0 1
           -------------            ----
           1 0 1 0 1 0 0 1           169
```

DIGI-COMP can be used to multiply two binary numbers according to the binary multiplication table.

Program DIGI-COMP as in the coding sheet.



To operate DIGI-COMP:

STEP 1 – Use Front Panel Card IX.

STEP 2 – Enter the first multiplicand bit into the top and the first multiplier bit into the middle flip-flop

STEP 3 – Cycle the clock ONCE. C will hold the result – copy it on paper.

STEP 4 – Enter additional bits of the multiplicand and cycle the clock, copying the answer each time (as in the example).

STEP 5 – Enter the second bit of the multiplier and repeat Step 4 for all bits of the multiplicand. Copy the answer as in Step 4 but shifted to the left one bit (see example).

STEP 6 – Repeat Step 5 for all bits of the multiplier.

You may use DIGI-COMP to add the partial products to get the final result (use the Binary Adder Experiment).

After deriving the Coding Sheets for the adder and subtractor, this problem should seem much easier. To begin, as usual, we shall write the binary equations for all possible combinations of the product of A and B.

$$A \textit{ times } B = C$$

$$1 \text{ times } 0 = 0$$

$$0 \text{ times } 0 = 0$$

$$0 \text{ times } 1 = 0$$

$$1 \text{ times } 1 = 1$$

These equations may now be put in the form of a Truth Table as follows:

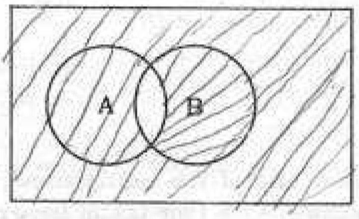
Multiplicand	Multiplier	Result	Equation
A	$\overline{B}$	$\overline{C}$	$(A \cdot \overline{B} = \overline{C})$
$\overline{A}$	$\overline{B}$	$\overline{C}$	$(\overline{A} \cdot \overline{B} = \overline{C})$
$\overline{A}$	B	$\overline{C}$	$(\overline{A} \cdot B = \overline{C})$
A	B	C	$(A \cdot B = C)$

Now to write the Boolean equations for the above Table.

$$C = A \cdot B$$

$$\overline{C} = A \cdot \overline{B} + \overline{A} \cdot \overline{B} + \overline{A} \cdot B$$

To reduce the equation for $\overline{C}$, look at the Venn diagrams for equations f, g and h in Table 1, Page 7. In the Venn Diagram below, shade in the Union of these three functions.



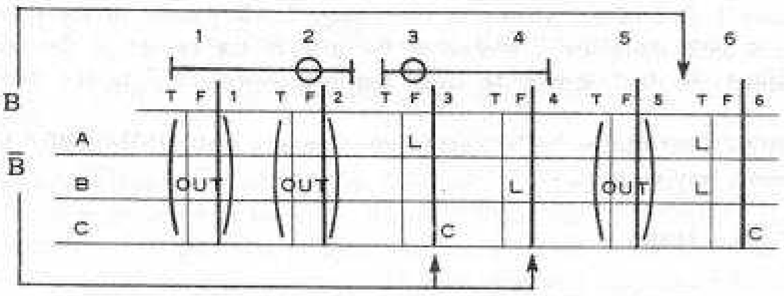
Now, compare your Diagram with the others in Table 1. If you shaded it in properly, it should be the same as the Diagram for Equation (i). Therefore:

$$A \cdot \overline{B} + \overline{A} \cdot \overline{B} + \overline{A} \cdot B = \overline{A} + \overline{B}$$

The Boolean Equations are then:

$C = A \cdot B$

$\overline{C} = \overline{A} + \overline{B}$



In the multiplication of two decimal numbers, if you recall the true meaning, the operation was defined as a multiple addition problem. For instance, to multiply 23 x 45, actually means 45 + 45 + 45 + 10 (45 + 45). It could be written as follows:

```
   45
 x 23
 ----
   45 ┐
   45 ├──── Add 45 - 3 times
   45 ┘
  450 ┐
  450 ┘──── Shift 1 place and add 45 - 2 times
 ----
 1035
```

If you do a multiplication problem with this technique you don't have to remember your multiplication table, all you have to know how to do is add.

Electronic computers usually use this approach to multiplication problems. By doing so, they don't have to memorize any multiplication tables either. Thus, most electronic computers do not multiply directly, as DIGI-COMP does, but use the add and shift approach instead.

Since the only numerals used in the binary system are 1 and 0, the add and shift technique is much more expedient than it was for the decimal system. If there is a 1 in the multiplier, the multiplicand is added once and shifted. If there is a 0 in the multiplier, the multiplicand is just shifted with no addition necessary.

As an example of the Add and Shift operation, suppose we multiply (10010) times (1101).

```
     1 0 0 1 0
       1 1 0 1
     ---------
     1 0 0 1 0
 1 0 0 1 0
1 0 0 1 0
---------------
1 1 1 0 1 0 1 0
```

As in the binary subtractor, electronic computers could do the operation directly, but by using the Complement and Add for subtraction and Shift and Add for multiplication, the computer needs only to add to accomplish all three operations.

EXPERIMENT 11 – GAME OF NIM

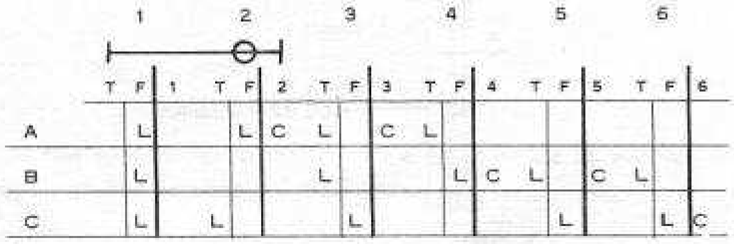
NIM, an ancient game over 2,000 years old, was played by the wise men of the east. Two wise men sat at a table, each one having a pile of stones by his side. The first wise man places either one or two stones in the center of the table. The second wise man *adds* 1 or 2 of his stones to the stones in the center of the table. Each player takes a turn until there are 7 stones in the pile in the center of the table. The wise man who added the last stones to make the pile *equal* to 7 is the smartest of all wise men.

DIGI-COMP claims to be very wise. Can *you* beat DIGI-COMP? Either you, or the computer, may go first.

To play NIM:

STEP 1 – Place Front Panel Card I in position.

STEP 2 – Program DIGI-COMP according to the coding sheet.



	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A		L			L	C	L		C	L								
B		L					L				L	C	L		C	L		
C		L		L				L						L			L	C

STEP 3 – Make sure clock is all the way out.

STEP 4 – Make sure all logic rods are turned to the right.

STEP 5 – Make sure all flip-flops are all the way out (to the right).

YOU OR DIGI-COMP MAY GO FIRST.

If you go first:

STEP 6 – Add either 1 or 2 to the number in the Read Out $\begin{smallmatrix}0\\0\\0\end{smallmatrix}$.
If you want to add 1, move the flip-flops with the Manual Input Tabs until you see $\begin{smallmatrix}1\\0\\0\end{smallmatrix}$ which is 1, in the Read Out. If you want to add 2, move the flip-flops until you see $\begin{smallmatrix}0\\1\\0\end{smallmatrix}$, which is 2, in the Read Out.

STEP 7 –
It is now DIGI-COMP'S turn. Cycle the clock ONCE.

If the computer goes first:

STEP 6 –
Cycle the clock ONCE.

STEP 7 –
It is your turn. Use the translation chart to help you translate the numbers in the Read Out to decimal. Add 1 or 2 to this number and translate back to binary. Enter the binary number by moving the flip-flops with the Manual Input Tabs until you *see* the number in the Read Out.

YOU MUST ADD EITHER *1 OR 2* TO THE NUMBER IN THE READ OUT ON YOUR TURN.

STEP 8 – Repeat Steps 6 and 7 (taking turns with the computer) until you see $\begin{smallmatrix}1\\1\\1\end{smallmatrix}$ in the Read Out. This is 7.

THE FIRST TO MAKE THE PILE *EQUAL* TO 7 WINS!

Read the problem *carefully*, program DIGI-COMP and play the game a few times to familiarize yourself with it.

Now, as always, we have to analyze the conditions of the problem before we can program it. Each new turn brings a new decision to be made. We must figure out what would happen under all circumstances and then decide the "best" routine for DIGI-COMP to follow. We say "best" because, obviously many programs would work, but only one would make DIGI-COMP win most of the time.

Our tool for analyzing NIM is called a "FLOW CHART". These are used daily by programmers to analyze the problems in giant Electronic Computers. We will make the Flow Chart for the case that DIGI-COMP goes first.

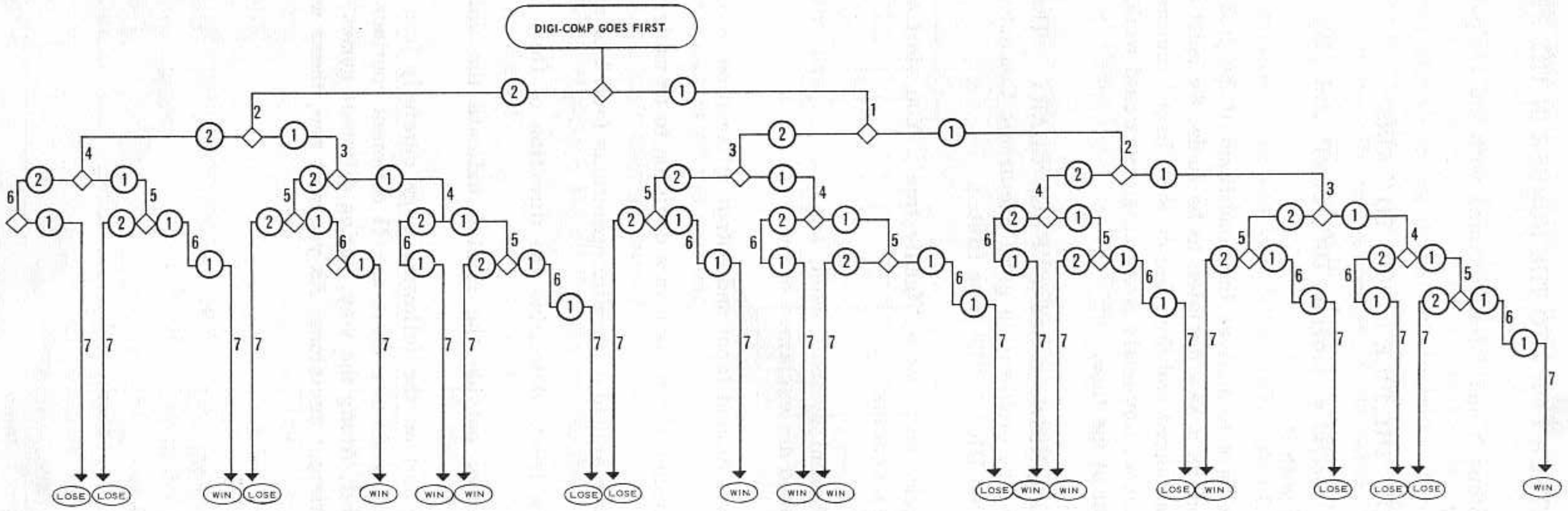
The Flow Chart works just like a "family tree". You start at the top and follow down each of the lines to a conclusion.

There are many different symbols used, and they depend only on personal preference. The symbols most used are explained below.

1. Put an oval around input and output information (start and conclusion).
2. Put a diamond where there is a decision to be made.
3. Put a circle around arithmetic operations (add 1 or add 2).
4. The arrow heads determine the direction of flow of the information.
5. The numbers outside the circles indicate the values at those points.

Go thru the Flow Chart on the following page carefully from start to the various conclusions. As you will soon see there are 21 different routines that could be followed if DIGI-COMP goes first. Along the way, during different games, DIGI-COMP must be able to make some 40 different decisions. As you will see, these will be based on the probabilities of winning.

FLOW CHART



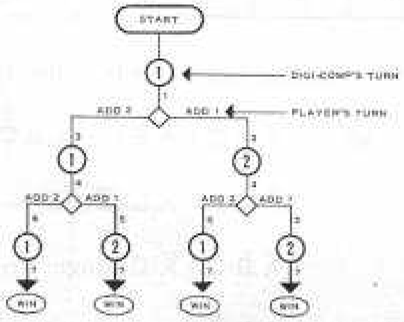
Now that we've finished this momentus task we must decide the "best" routine for DIGI-COMP to follow. There are 21 possible sequences, so we will look at each decision separately.

To make these decisions we shall look at the probability of winning for each choice. To start with, if DIGI-COMP goes first, it can either add one or two. If it adds two, it will win 4 out of 8 times (or 50% of the time). If it adds one, it will win 7 out of 13 times (or about 54% of the time). So it's better to add 1 the first time.

Next, it's the Player's turn and he can add either 1 or 2. If the Player adds 1, DIGI-COMP would win 2 out of 5 times if it added 1, and 2 out of 3 times if it added 2. So the next move for the computer to make is to add 2. Now there are 4 rocks on the pile. So if the Player adds 2 then DIGI-COMP will add 1. And if the Player adds 1 DIGI-COMP will add 2. In either case DIGI-COMP wins.

Now let's go back and see what should happen if the Player added 2 rocks to the pile on his first turn. If DIGI-COMP added 2 it would win half the time, if it added 1 it would win 2 out of 3 times. So DIGI-COMP should add 1 rock again making the pile 4 and putting the Player in the same predicament he was in before.

Now, let's make the Flow Diagram over again, just showing the proper decisions that DIGI-COMP should make.



You can see that by calculating the probabilities (playing the odds) that we have made it impossible for a Player to win if DIGI-COMP starts first.

The next step is to make a Truth Table:

If the Read-Out is:	0	1	2	3	4	5	6	7
Then DIGI-COMP should be set to:	1	-	4	4	-	7	7	-

To complete the Truth Table, we have to go through the same reasoning for the Player going first.

You should be able to do that, just retrace the same steps we took before and calculate the probabilities of DIGI-COMP winning.

The Truth Table will look like this:

If the Read-Out is:	0	1	2	3	4	5	6	7
Then DIGI-COMP should be set to:	-	3	4	4	?	7	7	-

There is a question mark under 4 because there is no clear-cut path to take. If DIGI-COMP gets in this predicament it will LOSE 2 out of 3 times. The next best thing to do is to add one rock and hope the Player doesn't notice. Well, that isn't so terrible. It wouldn't be any fun if DIGI-COMP always won!

Now, we have the hard part all done; all we have left to do is program DIGI-COMP. To start we will combine the two Truth Tables and write them in binary form as in the other programs.

If Read-Out	A	0	1	0	1	0	1	0	1
is:	B	0	0	1	1	0	0	1	1
	C	0	0	0	0	1	1	1	1
Then it	A	1	1	0	0	1	1	1	1
should be	B	0	1	0	0	0	1	1	1
set to:	C	0	0	1	1	1	1	1	1

Just as before, we will write the logical equations from the Truth Table.

$$\text{Set A} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C$$

$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,C\,(\overline{B} + B)$$

$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,C \text{ (Logic Rods 1 and 2)}$$

$$\text{Reset A} = A\,B\,\overline{C} \text{ (Logic Rod 3)}$$

$$\text{Set B} = A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C$$

$$= A\,\overline{B}\,(\overline{C} + C)$$

$$= A\,\overline{B} \text{ (Logic Rod 4)}$$

$$\text{Reset B} = \overline{A}\,B\,\overline{C} + A\,B\,\overline{C}$$

$$= B\,\overline{C}\,(\overline{A} + A)$$

$$= B\,\overline{C} \text{ (Logic Rod 5)}$$

$$\text{Set C} = \overline{A}\,B\,\overline{C} + A\,B\,\overline{C}$$

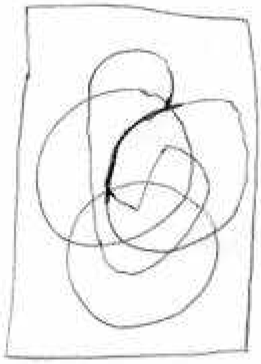
$$= B\,\overline{C}\,(\overline{A} + A)$$

$$= B\,\overline{C} \text{ (Logic Rod 6)}$$

As usual, the Coding Sheet follows directly from the equations. It took a while to organize the problem, but if you do it step by step, it really isn't so difficult. Let us emphasize again, that these are exactly the steps you would take if you were programming the world's largest electronic digital computer. First, analyze the problem and put into some kind of a form that will lead you directly to the key information. Next, translate the information into mathematical form and code it in such a way that your particular computer will understand it.

Now you should be prepared to invent your own problems and programs to solve them. E.S.R., Inc., will be making more publications from time to time and if your program is printed, and you were the first to submit it, we will send you a check for $10.00 and print your name in our publication.

a venn diagram w/ 4 circles



ANSWERS TO QUESTIONS

EXPERIMENT 3

The complete program would be:

		1			2			3			4			5			6	
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A			C		L	C		L					L				L	
B		L							C		L	C		L		L		
C		L		L			L				L		L				L	

EXPERIMENT 7

The Boolean Equations are:

(Position 2) $A = \overline{A}\ B\ C$

(Position 3) $\overline{B} = A\ B\ C$

(Position 4) $B = A\ \overline{B}\ C$

(Position 6) $C = A\ B\ \overline{C}$

EXPERIMENT 13

We want to Reset A when the answer is 0, that is:

A = 0 and B = 0 and C = 0
or
A = 1 and B = 0 and C = 1
or
A = 1 and B = 1 and C = 1
or
A = 0 and B = 1 and C = 1

Putting these in equation form would give:

$$\text{Reset } A = \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C$$

By the same "trick" as before the first and fourth terms drop out, leaving:

$$\text{Reset } A = A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

The rest is easy: to find the equation for Set C, look at the Truth Table to see when the carry is 1 and you will see that:

$$\text{Set } C = A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot C$$

Again, since C is already 1 in the first and third terms the equation becomes:

$$\text{Set } C = A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

Factoring common terms gives:

$$\text{Set } C = A \cdot B\,(\overline{C} + C)$$

$$\text{Set } C = A \cdot B$$

Now to find the expression for Reset C. Since the process is repetitious we shall just write the equations.

$$\text{Reset } C = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C}$$

$$= \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C$$

$$= \overline{A} \cdot \overline{B}\,(\overline{C} + C)$$

$$\text{Reset } C = \overline{A} \cdot \overline{B}$$

EXPERIMENT 14

The procedure is exactly the same for this experiment as for the previous one.

To find the equation for Reset A, look at the Truth Table and see when the answer is 0.

That is when:

A = 1 and B = 1 and C = 0
or
A = 0 and B = 0 and C = 0
or
A = 1 and B = 0 and C = 1
or
A = 0 and B = 1 and C = 1

or in algebraic form:

$$\text{Reset } A = A \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C$$

The second and fourth terms drop out, as before, since the A Flip-Flop must be 0 for the functions to be true.

This leaves:

$$\text{Reset } A = A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C$$

Next, we reset the C Flip-Flop when there is no borrow necessary. That is:

$$\text{Reset } C = A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C$$

Now the first and third terms drop out as before, leaving

$$\text{Reset } C = A \cdot \overline{B}\,(\overline{C} + C)$$

$$\text{or} \quad \text{Reset } C = A \cdot \overline{B}$$

And lastly, we want to Set C when a borrow is required.

$$\begin{aligned}\text{Set } C &= \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot C + \overline{A} \cdot B \cdot C + \overline{A} \cdot \overline{B} \cdot C \\ &= \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C \\ &= \overline{A} \cdot B\,(\overline{C} + C) \\ \text{Set } C &= \overline{A} \cdot B\end{aligned}$$

BLANK CODING

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A																		
B																		
C																		

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A																		
B																		
C																		

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A																		
B																		
C																		

	1			2			3			4			5			6		
	T	F	1	T	F	2	T	F	3	T	F	4	T	F	5	T	F	6
A																		
B																		
C																		