"DESIGN OF A DIGITAL LOGIC TRAINER"

by

James Dillino Allen

Thesis submitted to the Graduate Faculty of the

Virginia Polytechnic Institute

in candidacy for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Chairman, Dr. M. H. Hopkins, Jr.

Dr. W. W. Cannon

Dr. R. H. Miller

Prof. R. R. Wright

December 1965

Blacksburg, Virginia

TABLE OF CONTENTS

LISTING OF TABLES AND FIGURES

Numerical Listing of Figures

Numerical Listing of Tables

I. INTRODUCTION

This thesis shall be the design of a digital system for the express purpose of demonstrating the simplicity of digital logic systems (computers or control systems). As an example of logic design, it will show the sequential logic development of a particular function, the Full Adder, along with its final implementation. A similar sequential development procedure was used to develop each of the functions required in the Digital Logic Trainer (for example, each register shift pulse or steering command).

The design is made as general as possible in order to allow maximum flexibility with regard to type of basic power, logic levels (they must be negative), and the variation in circuit design of available logic components. Only the more basic logic elements were used. Some slight improvement in the design may be possible if some of the more exotic elements are available (i.e., Exclusive OR gates, Clock Drivers, Time Delays and Logic Expanders or four or more input NOR gates).

General comments are made concerning the relationships between the binary numbering system and the digital system. It is assumed that the reader thoroughly understands boolean algebra and the binary numbering system as it relates to the familiar decimal numbering system.

Also included, is a Digital Training Manual emphasizing the use of this Six-Bit Serial Binary Demonstration Computer.

II. THE RELATIONS BETWEEN BINARY LOGIC AND THE DIGITAL SYSTEM

General

Digital systems are usually constructed of two-state devices. This characteristic allows devices to be designed in a simple, but reliable way, so as to tolerate wide extremes in environment and application. The status of these system elements can be described in terms such as true-false, closed-open, up-down, voltage-no voltage, on-off, in-out, plus-minus, etc. Therefore, the binary system and binary arithmetic, in particular, is well adapted to the design and evaluation of digital systems.

The most common way of designating the binary states of an element is with the binary digits 1 and 0. In the following design, a "1" meaning true will be represented by a negative voltage (i.e., -6volts) and a "0" meaning false will be represented by a nominal zero volts. This relates the binary numbers directly into the system. Therefore, the only need is to discuss the system in terms of binary numbers.

System Logic Specification

One of the specifications that must be known before beginning the logical design of any digital control or computation mechanism is what basic type gate or logic element is to be used. From the standpoint of boolean algebra, the obvious logical elements would be "AND" and "OR" gates. Passive "AND" and "OR" gates are used to give multiple inputs to the logic elements, but because of the need for both isolation from input to output and power amplification, an active element is required. This active element is logically a transistor. The transistor used in the common emitter configuration also has a 180^o phase shift (inversion) property.

Therefore, the coupling of the passive "AND" and "OR" gates to the active source yields "Not AND" (NAND) and "Not OR" (NOR) gates. These are the two basic digital logic elements used today.

Although the NOR gate is typically designed using a negative power bus and the NAND gate is typically designed using a positive power bus, this is not a requirement. Generally, approximately zero (0) volts is defined as a logic "0" and voltage level (for example, -6 volts or +6 volts) is defined as a logic "1".

It is interesting to note that, by redefining the logic levels with relation to the voltage levels, a logic gate can be made to perform either the "NOR" or the "NAND" logic function. For example, assume the output of a particular logic gate is approximately zero volts at all times, except when all of its inputs are zero volts and at that time its output is a voltage level (for instance, -6 volts). This gate would perform NOR logic if logic "1" is defined as a voltage level and logic "0" is defined as zero volts. It would perform NAND logic if logic "0" is defined as a voltage level and logic "1" is defined as zero volts.

For the purposes of this study, the three-input NOR gate has been used as the basic logic element in the design. Also for this design, a voltage level is defined as logic "1" and approximately zero volts is defined as logic "0".

A digital system designer has at his command a large variety of other logic functions, most of which relate directly to the basic logic element. These include flip-flops, shift registers, full adders, pulse generators and many others. It would be possible to substitute combinations or NOR gates, or slight variations of NOR gates, for most of these auxiliary logic functions and, in fact, this is usually the basis for their design. In a system design however, it is preferable to use the auxiliary logic functions where applicable, in preference to the combination of basic logic elements, as this usually improves the understandability of the logic in the diagrams.

System Requirements

This system shall consist of three seven-bit shift registers, two separate counters, one full adder and associated logic. Its arithmetic capabilities shall include binary addition, binary subtraction and binary multiplication. Its logic capabilities shall include OR, AND and Exclusive OR functions.

It will be manually programmed by means of an Instruction Select Switch (ISS) on the front panel. The instruction selected will be performed at a rate determined by the position of the Execution Select Switch (ESS). The ESS has three positions: (1) Fast $\cong$ 20 kc., (2) Slow $\cong$ 2cps. and (3) Manual where the speed is determined by how fast the Manual pushbutton is depressed.

The Program counter (P register, Figure 9) determines what step or instruction of the program is being performed. The Step counter (S register, Figure 8) determines which step of the instruction is being performed.

For further explanation as to how the above are related, see Figure 7, Block Diagram and Instruction List.

III. DEFINITION OF SYMBOLS USED IN THIS DIGITAL LOGIC TRAINER

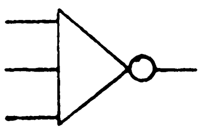


Figure 1a

Figure 1a shows the basic logic element used in this design. It is a three input NOR gate. As previously described, it is a three input passive OR gate, the output of which is connected th the base of a common emitter configuration PNP transistor. The output is taken at the collector of the transistor.

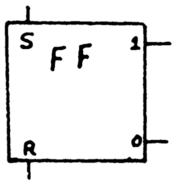


Figure 1b

Figure 1b is a Flip-Flop or basic memory storage device, sometime called a bistable multivibrator. It consists of two cross-connected NOR gates (see Figure 6).

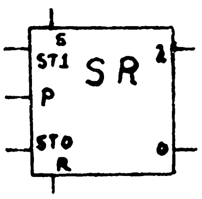


Figure 1c

Figure 1c is a Shift Register. It is a binary counter with the addition of steering terminals, such that, when a positive pulse is placed on P, the flip-flop assumes the state to which it was commanded by the steering terminals. That is, if ST1 = "1" and ST0 = "0", the flip-flop will set when a positive pulse is applied. Reverse steering causes the flip-flop to reset.

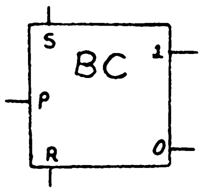


Figure 1d

Figure 1d is a <u>Binary-Counter.</u> It is a flip-flop with an added pulse input. When a positive going pulse of minimum amplitude and duration is placed on the pulse input, it causes the flip-flop to change state.

Figure 1e is a <u>Single Shot</u> or one- shot, sometimes called a monostable multivibrator. It essentially consists of two cross connected NOR gates, except that one of the cross connections is capacitive instead of resistive. This capacitive interconnection makes the multivibrator monostable, instead of bistable.

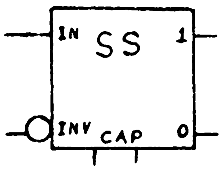


Figure 1e

When a negative pulse or condition is placed on IN or a positive pulse or condition on INV, the single shot is fired. This causes a true (logic "1") pulse of a specific length to appear at the "1" output. The length of the pulse can be drastically changed with the addition of a capacitor at the CAP terminal.

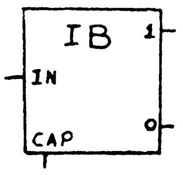


Figure 1f

Figure 1f is an <u>Input Buffer.</u> Its function is to accept a contact closure to the main unregulated dc power bus, usually called the light bus, and output a true and false logic level signal as a function of whether or not the contact is closed.

Figure 1g is a Trigger circuit. It is similar to a Schmitt trigger and its operation is similar to that of the input buffer above. The output of the input buffer usually has a lot of contact bounce and is, therefore, unsatisfactory for driving single shots or counters. Because of the much heavier filter on its input, the trigger's output has a good snap action making it an excellent device to drive single shots, and counters. Its filtering capabilities can be increased by the addition of a capacitor at the CAP terminals.

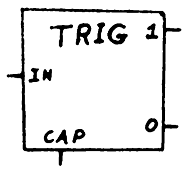


Figure 1g

Figure 1h is a typical 20 kc Pulse Generator, usually employing a unijuction transistor. As long as the INH input = "0", the output of PG is approximately six (6) microsecond pulses at a 20 kc. rate. The output is normally "1" but is pulsed to "0" for six (6) microseconds.

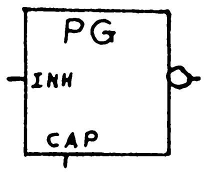


Figure 1h

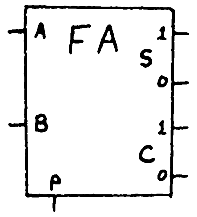


Figure 1j

Figure 1j is a Full Adder. It can be designed with only NOR gates but its function is much more easily explained and understood from the conventional full adder standpoint. The full adder sums three binary inputs, A, B and Carry (C). Its output is a logic level signal for both the true and false of the Sum and Carry for the preceding inputs.

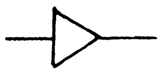


Figure 1k

Figure 1k is a Light Driver. It is a buffer and power amplifier able to carry the load required by light bulbs and to buffer that inductive load away from the logic lines.

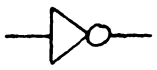


Figure 1m

Figure 1m is an Inverter. Its principal purpose is to invert the logic level signal at its input, although it usually has increased logic driving capabilities.

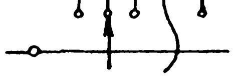


Figure 1n

Figure 1n is a common multipole Rotary Switch.

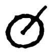


Figure 1o

Figure 1o is a Toggle Switch.

IV. DEVELOPMENT OF LOGIC FUNCTIONS FROM BASIC NOR GATES

Sequential Development of the Full Adder

The full adder will be used to demonstrate the sequential logic design of a typical logic function. A similiar procedure was used to develop each of the logic functions required in the development of the Logic Trainer (i.e., the A register shift pulse shown in Figure 10).

The full adder has three inputs and requires both the true and false functions of two outputs. It has a truth table as follows:

Table 1

Truth Table for Full Adder

Inputs			True Outputs	
A	B	C	SUM	CARRY
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

An Exclusive OR gate yields a true output when one, but not both, of its inputs is true. If an Exclusive OR gate were available, the obvious design from the truth table would be that shown in Figure 2.

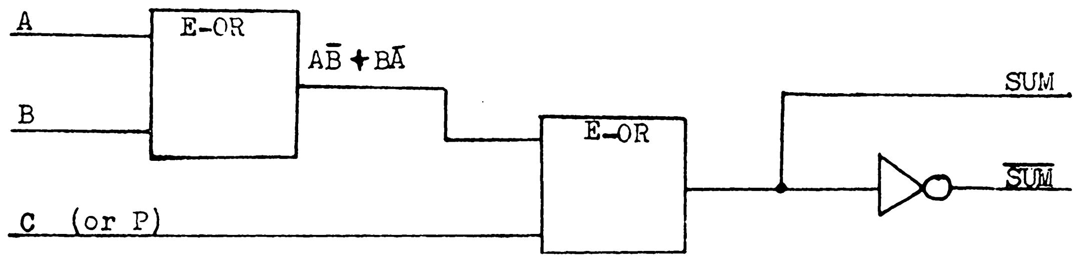
$$SUM = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

$$SUM = \bar{C}\ (A\bar{B} + B\bar{A}) + C\ (\bar{A}\bar{B} + AB)$$

$$SUM = \bar{C}\ (A\bar{B} + B\bar{A}) + C\ (\bar{A}\bar{B} + AB + A\bar{A} + B\bar{B})$$

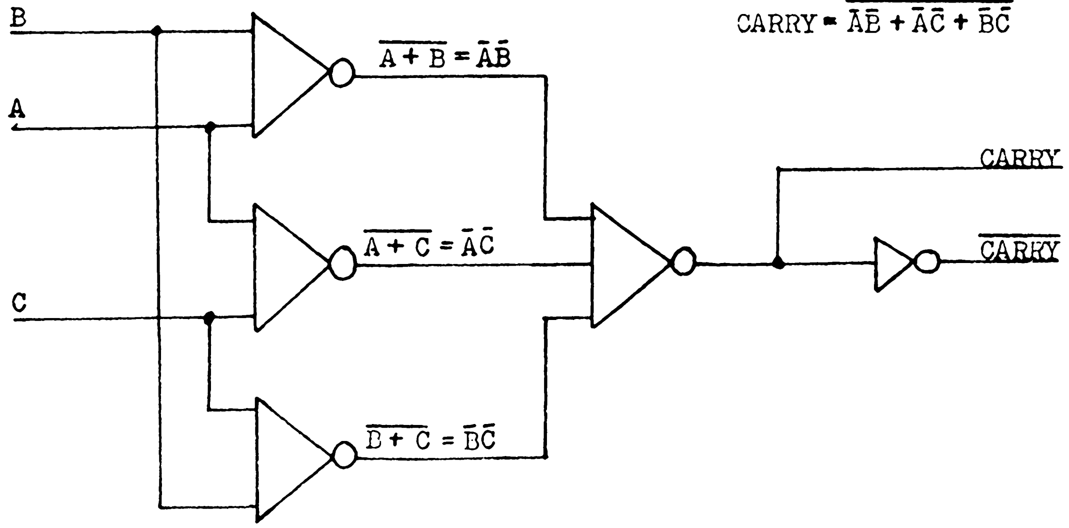
$$SUM = \bar{C}\ (A\bar{B} + B\bar{A}) + C\ (\bar{A} + B)\ (\bar{B} + A)$$

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



$$CARRY = AB + AC + BC$$

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


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

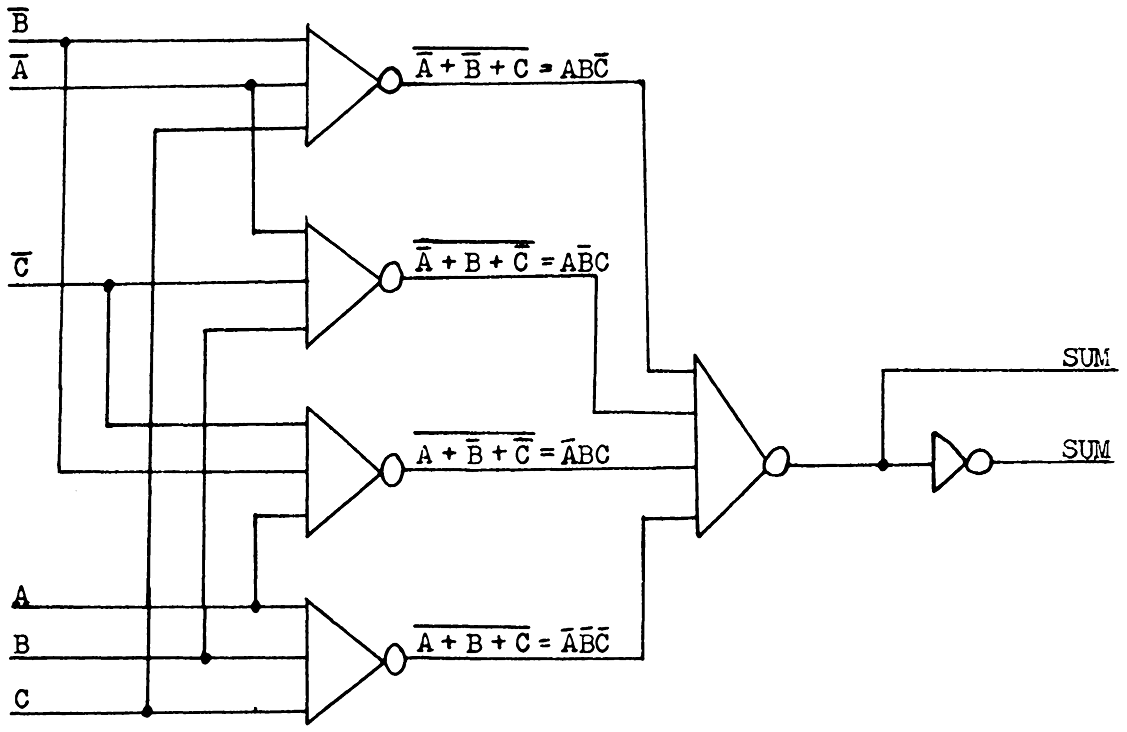
Initial Design of Full Adder

Figure 2

If the Exclusive OR gates are not available, the SUM could be developed as follows:

$$SUM = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

$$SUM = \overline{AB\bar{C} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}\bar{C}}$$



$\bar{B}$
$\bar{A}$
$\overline{\bar{A} + \bar{B} + C} = AB\bar{C}$
$\bar{C}$
$\overline{\bar{A} + B + \bar{C}} = A\bar{B}C$
SUM
$\overline{A + \bar{B} + \bar{C}} = \bar{A}BC$
$\underline{\overline{SUM}}$
A
B
$\overline{A + B + C} = \bar{A}\bar{B}\bar{C}$
C

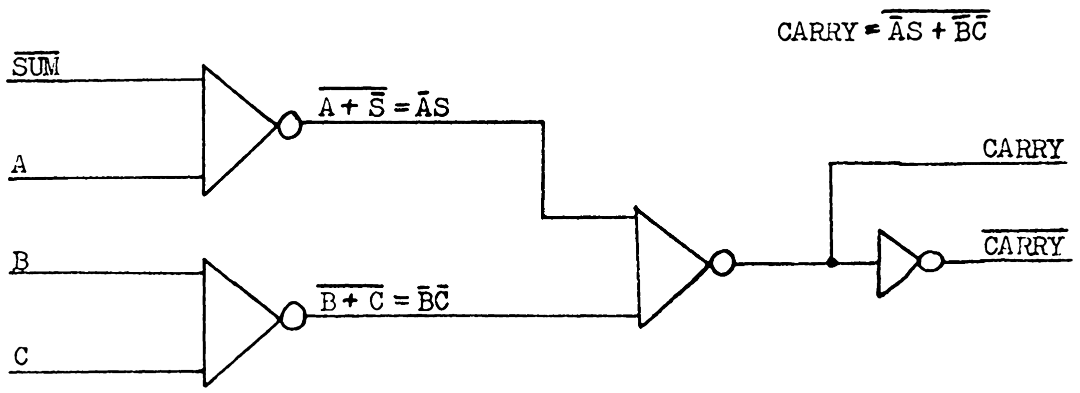
Sum Output of Full Adder

Figure 3a

If the $\overline{\text{SUM}}$ is available, the following Karnaugh Map shows a simplier way to develop CARRY.

		B			
A	1		1		
		1		0	SUM
	0		0		
		1		o	
			C		

Table 2



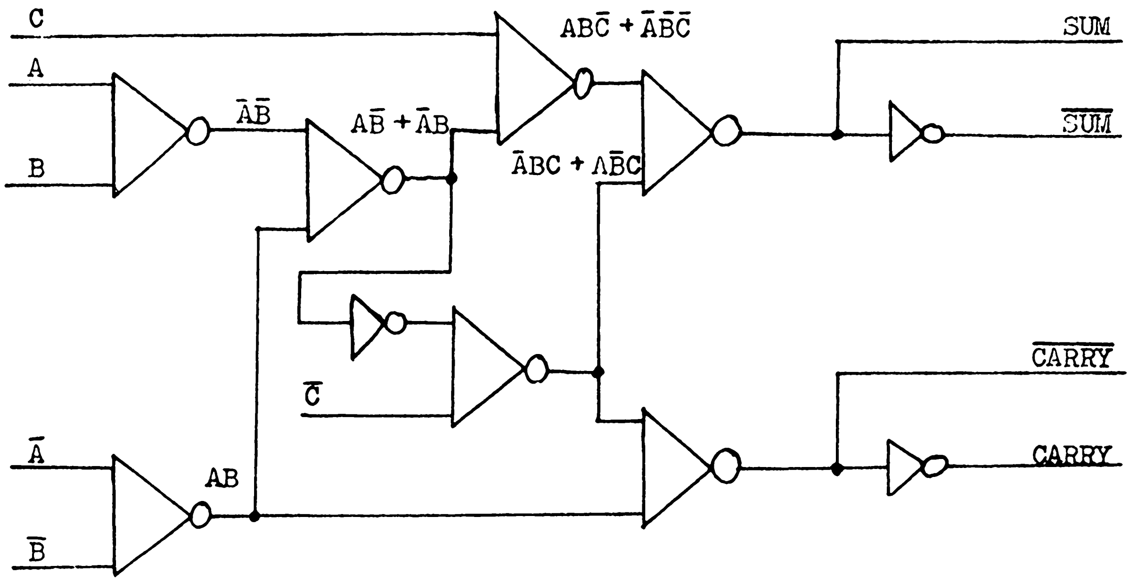
CARRY $= \overline{\bar{A}S + \bar{B}\bar{C}}$
$\overline{\text{SUM}}$
A
B
C
$\overline{A+\bar{S}} = \bar{A}S$
$\overline{B+C} = \bar{B}\bar{C}$
CARRY
$\overline{\text{CARRY}}$

CARRY Output of Full Adder

Figure 3b

This reduces the amount of elements required to generate CARRY from 5 to 4.

If a four input NOR gate is not available, a little manipulation is required to develop the SUM output. Using both Karnaugh Maps and boolean algebra, the following circuit was developed to perform the entire full adder function, without using a four input NOR gate.



Improved Design of Full Adder

Figure 4

So far it has been assumed that both the normal and inverse of each of the input functions are immediately available. If only the normal inputs are available, we must add one extra logic element. The circuit is shown in Figure 5.

$$\text{CARRY} = AB + BC + BC$$

$$\text{CARRY} = \overline{(A\bar{B}\bar{C} + \bar{A}B\bar{C}) + \bar{A}\bar{B}}$$

$$\text{SUM} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$\text{SUM} = \overline{AB\bar{C} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}\bar{C}}$$

$$\text{SUM} = \overline{(AB\bar{C} + \bar{A}\bar{B}\bar{C}) + (A\bar{B}C + \bar{A}BC)}$$

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

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

$$AB + \bar{A}\bar{B} = \overline{\bar{A}B + A\bar{B}}$$

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

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

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

Final Full Adder

$$A\bar{B}C + \bar{A}BC = \overline{(AB + \bar{A}\bar{B}) + (A\bar{B}\bar{C} + \bar{A}B\bar{C})}$$

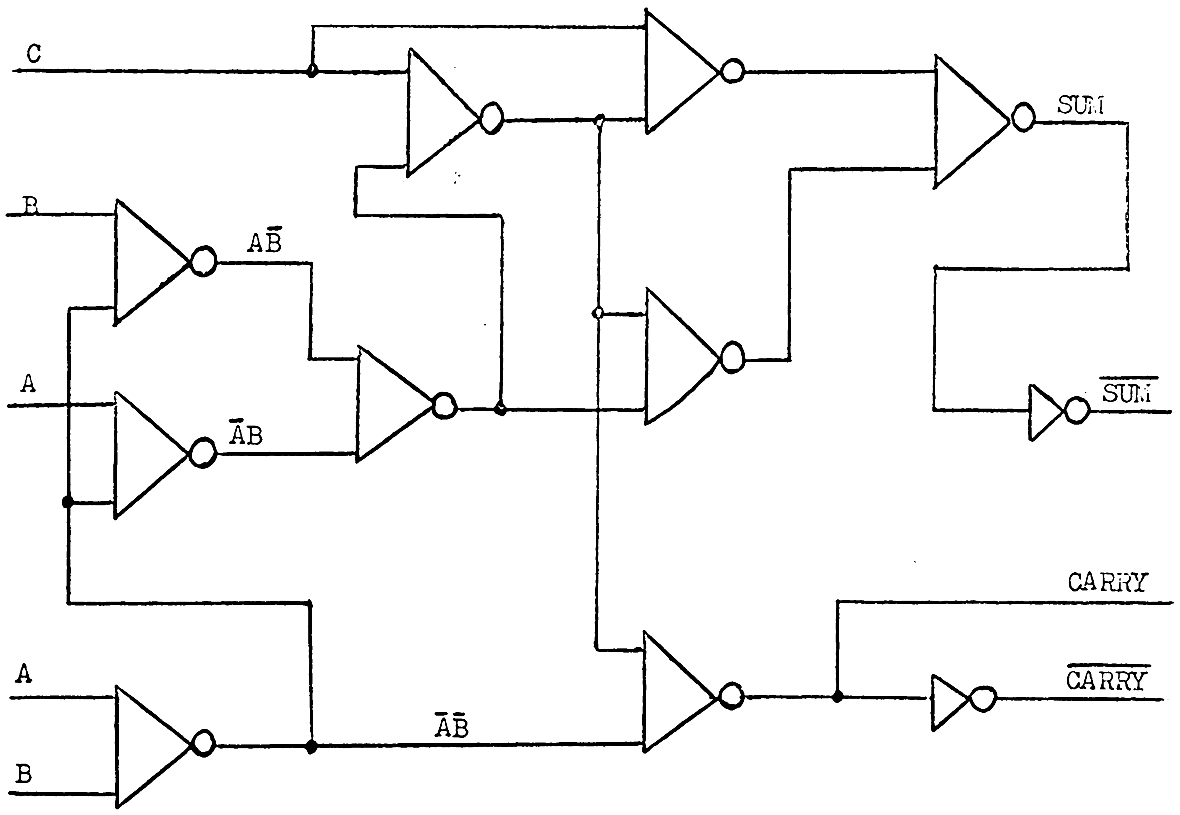
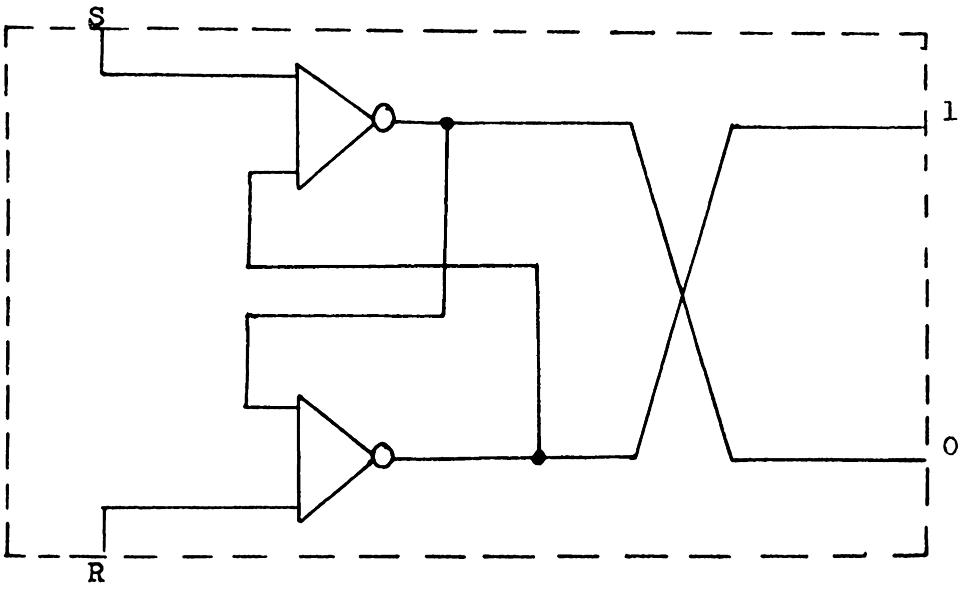


Figure 5

Development of the Flip-Flop from NOR gates.



Bistable Multivibrator

Figure 6

V. BINARY REPRESENTATION

Negative Numbers

In all discussions thus far, positive numbers have been implied. Can negative numbers be handled as easily, and if so, what form will they take?

In the binary system there are two principal arithmetic methods, which include procedures for representing negative numbers. The first is called one's complement arithmetic and the second is two's complement arithmetic. They are similar in that they each require the most significant bit (MSB) of the number to be used for polarity identification.

If the MSB = "1", it is a negative number. If the MSB = "0", it is a positive number.

They differ only in the value interpretation of their negative numbers. The value of a negative number in one's complement form is the sum of the binary value associated with each register location, in which there is a zero in the number. The value of a negative number in two's complement is the sum of the binary value associated with each register location, in which there is a zero in the number plus one.

The following are a few examples of interpeting negative numbers in complement arithmetic into the familiar decimal system.

Table 3

Binary to Decimal Conversion

Binary Representation	One's Complement	Two's Complement
1 1 1 0 0 1 0	-13	-14
1 1 0 1 1 1 0	-17	-18
1 1 1 1 1 0 1	- 2	- 3

Two's complement arithmetic was chosen for this design because it has only one zero value (zero = 0000000). One's complement arithmetic has two zero values (+ zero = 0000000 and - zero = 1111111). Extra implementation would be required to account for this.

The technique of complement arithmetic makes it unnecessary to provide a logical subtractor. In other words, subtraction can be performed using the addition procedures previously described, provided all negative numbers are in two's complement form.

Consider the following example of subtraction using the scheme of 10's complement arithmetic:

$$\begin{aligned} 14 - 7 &= 14 + 100 - 7 - 100 \\ &= 14 + 93 - 100 \\ &= 107 - 100 \\ &= 7. \end{aligned}$$

This process may seem complicated. However, if it is easy to take the tens complement of 7 (i.e., 100 - 7 = 93), and forget the most significant one in 107, subtraction effectively becomes addition.

The mechanization for taking the one's complement of (subtracting) a number in the binary system is simple. It requires only that the normal input be gated through an inverter before it is used in the Full Adder. To use two's complement arithmetic for subtraction requires also that the memory device used to store the Carry during shifts (CAR, Figure 10) be set initially. This adds one to the result obtained above, as required by two's complement arithmetic (see Tables 4a and 4b).

The two's complement of a binary number can be obtained by complementing the number and adding one. Complementing, simply means to change all the 1's to 0's and all the 0's to 1's. The following examples illustrate the fact that a number in two's complement form can be converted to normal form by simply taking the two's complement.

Table 4a

Complement Conversion Example A

32	16	8	4	2	1	BINARY
0	0	0	0	0	1	EXAMPLE NUMBER (DECIMAL ONE)
1	1	1	1	1	0	COMPLEMENT (DECIMAL 62 = -2)
0	0	0	0	0	1	ADD ONE
1	1	1	1	1	1	2'S COMPLEMENT (DECIMAL 63 = -1)
0	0	0	0	0	0	COMPLEMENT (DECIMAL 0)
0	0	0	0	0	1	ADD ONE
0	0	0	0	0	1	ORIGINAL EXAMPLE (DECIMAL ONE)

Table 4b

Complement Conversion B

32	16	8	4	2	1	BINARY
1	0	0	1	0	1	EXAMPLE NUMBER (DECIMAL 37 = -27)
0	1	1	0	1	0	COMPLEMENT (DECIMAL 26)
0	0	0	0	0	1	ADD ONE
0	1	1	0	1	1	2'S COMPLEMENT (DECIMAL 27)
1	0	0	1	0	0	COMPLEMENT (DECIMAL 36 = -28)
0	0	0	0	0	1	ADD ONE
1	0	0	1	0	1	ORIGINAL EXAMPLE (DECIMAL 37 = -27)

To solve the general problem of adding a positive number X to a negative number Y or (X+(-Y)) using the machine. X data with a "0" sign bit is placed in the A register. Y data in two's complement form with a "1" polarity bit is placed in the B register. During the arithmetic process, the machine applies the truth table shown in Table 1, digit by digit. The mechanization is shown on Figure 2. Initially, C is "0". For all remaining digits it takes the state of the previous Carry. The resulting answer will be X plus the two's complement of Y. It can take two forms, depending upon the numerical size of X and Y. If X is equal to or greater than Y, the sign bit will be "0" and the answer in normal form. If Y is greater than X, the sign bit will be "1", and the answer in two's complement form. Thus, the sign bit informs the operator of how to interpert the answer. If the sign bit is "0", the answer is positive and in normal form. If the sign bit is "1", the answer is negative and in two's complement form.

Summarizing, the Logic Trainer can add or subtract two numbers regardless of sign, provided that:

1. All positive numbers are in the normal form with a sign bit of "0".
2. All negative numbers are in two's complement form with a sign bit of "1".

The answer from an addition or subtraction instruction can have two forms:

1. If the sign bit is "1", the answer is negative and in two's complement form.
2. If the sign bit is "C", the answer is positive and in normal form.

How Multiplication Is Accomplished

Multiplication in the decimal system is normally accomplished with the aid of a multiplication table, which is usually memorized. The familiar procedure is illustrated below:

MULTIPLICAND			2	5	6
MULTIPLIER			1	2	3
			7	6	8
PARTIAL PRODUCT		5	1	2	
	2	5	6		
PRODUCTS	3	1	4	8	8

A number of partial products equal to the number of digits in the multiplier are developed. The position of the partial products is such that the least significant digit in the final product is determined by the first partial product, the next to least significant digit of the final product is determined by the first two partial products, and so on. This seemingly insignificant fact allows the final product to be developed, digit by digit, by adding the partial products successively and outputing the least significant digit. This procedure is illustrated as follows:

MULTIPLICAND	2	5	6		
MULTIPLIER	1	2	3		
INITIAL PARTIAL PRODUCT	0	0	0		
1ST PARTIAL PRODUCT	7	6	8		
1ST SHIFT RIGHT		7	6	8	
2ND PARTIAL PRODUCT	5	1	2	↓	
2ND SUM	5	8	8	8	
2ND SHIFT RIGHT		5	8	8	8
3RD PARTIAL PRODUCT	2	5	6	↓	↓
FINAL SUM	3	1	4	8	8

The procedure of adding and shifting may seem complicated, but analysis shows it to be essentially identical to that shown on page 21. This procedure is easier for a machine because it avoids the problems of having to calculate, temporarily store, and then add all the partial products in order to arrive at a final product. The sum of the partial products and finally the total answer in continuously stored in an accumulator register (A - C Register in this particular system). Summarizing, multiplication by adding and shifting is the most convenient procedure for a machine.

The binary multiplication table is very simple, as shown below:

Table 5

Binary Multiplication Table

		MULTIPLICAND	
		0	1
MULTIPLIER	0	0	0
	1	0	1

The following example (Table 6) applies the binary multiplication table to two three (3) digit numbers. Partial products are added successively and shifted one digit to the right in order to develop the final product.

Table 6

Positive Binary Multiplication

BINARY	4	2	1				
MULTIPLICAND (5)	1	0	1				
MULTIPLIER (3)	0	1	1				
INITIAL PARTIAL PRODUCT	0	0	0				
1ST PARTIAL PRODUCT	1	0	1				
1ST SUM	1	0	1				
1ST SHIFT RIGHT	0	1	0	1			
2ND PARTIAL PRODUCT	1	0	1				
2ND SUM	1	1	1	1			
2ND SHIFT RIGHT	0	1	1	1	1		
3RD PARTIAL PRODUCT	0	0	0				
3RD SUM	0	1	1	1	1		
3RD SHIFT RIGHT, FINAL PRODUCT (15)	0	0	1	1	1	1	
	32	16	8	4	2	1	BINARY

Notice that whenever the multiplier digit is zero (0), it is unnecessary to sum because the new partial product is simply the old partial product shifted right one digit. Summarizing, the rules for multiplication of positive numbers are as follows:

1. Starting from the least significant digit and working toward the most significant digit, investigate each multiplier digit.

2. If the multiplier digit is 0, shift the partial product one digit to the right.
3. If the multiplier digit is 1, add the multiplicand to the old partial product and shift the new partial product one digit to the right.

The previous rules are complete for positive numbers. In order to multiply negative numbers in two's complement form, additional procedures are required. In order to cover all cases, the additional procedures should allow for either positive or negative numbers (i.e., multiplier or multiplicand) and an initial partial product other than zero. This will allow the multiplication procedure to accomplish $(\pm X)(\pm Y)\pm Z$. The modifications required are as follows:

4. Retain the previous sign bit when shifting the new partial product right.
5. If the multiplier is minus, perform an additional step of subtracting the multiplicand from the final partial product.

The first four rules are obeyed by the machine during a multiplication routine. The fifth rule is programmed by the student. Initially, the multiplier is put into the C register, Figure 13. The multiplicand is put into the B register, Figure 11. The initial partial product is put into A register, Figure 10. When shifting right to develop a new partial product, the A register and the C register, <u>except</u> for 3C1, are shifted right one place. Register 3A1 retains its previous state following the right shift. Element 3C1 does not get involved in any shifting during the multiply operation. That is, the shift right of registers A and C takes place around 3C1. If the multiplier is minus, an additional instruction is required. This instruction subtracts the multiplicand B from the last partial product in A and puts the answer in A. The A and C registers contain the final answer. Registers 1C1, 1C2 and 1C4 contain the least significant octal digit. 2A1, 2A2 and 2A4 contain the most significant octal digit. Unit 3A1 contains the sign of the answer. Unit 3C1 is not involved in the answer. The following examples illustrate the multiplication procedures with positive and negative numbers:

Table 7a

Multiplication Example A

BINARY	SIGN	4	2	1				
MULTIPLICAND X = -3	1	1	0	1				
MULTIPLIER Y = +2	0	0	1	0				
INITIAL PARTIAL PRODUCT (0)	0	0	0	0				
1ST SHIFT RIGHT	0	0	0	0	0			
2ND PARTIAL PRODUCT	1	1	0	1				
2ND SUM	1	1	0	1	0			
2ND SHIFT RIGHT	1	1	1	0	1	0		
3RD SHIFT RIGHT FINAL PRODUCT (-6)	1	1	1	1	0	1	0	
		32	16	8	4	2	1	BINARY

Table 7b

Multiplication Example B

BINARY	SIGN	4	2	1				
MULTIPLICAND (-3)	1	1	0	1				
MULTIPLIER (-5)	1	0	1	1				
INITIAL PARTIAL PRODUCT	1	1	0	1				
1ST SHIFT RIGHT	1	1	1	0	1			
2ND PARTIAL PRODUCT	1	1	0	1	↓			
1ST SUM	1	0	1	1	1			
2ND SHIFT RIGHT	1	1	0	1	1	1		
3RD PARTIAL PRODUCT	0	0	0	0	↓	↓		
2ND SUM	1	1	0	1	1	1		
3RD SHIFT RIGHT	1	1	1	0	1	1	1	
ADDITIONAL INSTRUCTION ADD TWO'S COMPLEMENT OF MULTIPLICAND	0	0	1	1	↓	↓	↓	
FINAL PRODUCT = 15	0	0	0	1	1	1	1	
		32	16	8	4	2	1	BINARY

VI. SIGNAL FLOW AND COORDINATION

In general, the logic diagram (figure 7 through figure 13) is arranged from front to back by function. Where possible, complete functions are shown on a single sheet. Frequently more than one sheet will be required. In this case adjacent sheets are used.

On a particular sheet signal flow is generally arranged from left to right and from top to bottom just as a book is read.

Wherever signals interconnect between two pages, they are cross referenced. Where a signal leaves a sheet, the destination will be shown by sheet number.

Where a signal arrives on a sheet, the source will be shown by sheet number and signal name. The signal name, with its mnemonic significance (see Nomenclature, page 29), will allow the function on the new sheet to be understood more easily without an excessive amount of thumbing through the diagram.

VII. NOMENCLATURE

Where appropriate, each logic element in this system will be given a name or mnemonic description consistent with its functional description in this application. This mnemonic description usually consists of the initial letters of the significant words in the function description (for example, CSP = C register Shift Pulse) or an abbreviation of this function (for example, EXCT = Execute instruction). These mnemonics can be a significant aid in understanding and troubleshooting. Only a little familarity with a system is required before the mnemonics can be recalled. On single output elements, the mnemonic is prefixed by a 1 or a 0 to denote whether the output is the normal or the inverse of the function described by the mnemonic. This 1 or 0 is called the polarity identification. If the output signal is true for the function described, the polarity bit is 1. Conversely, if the output signal is false for the function described, the polarity bit is a 0. On dual output elements, both polarities of signal are available. In this case, the polarity bit is omitted from the mnemonic tag line, but is still associated with the appropriate output signal, and will be used as signal identification when that signal is an input to another element. Various other identifying marks may be found in several symbols to clarify the operation of certain input signals. Examples are S = SET, R = RESET, P = PULSE, ST = STEERING, and CAP = CAPACITOR terminals.

VIII. EXPERIMENTAL VERIFICATION

The design was experimentally verified on a Programmable Digital Logic Console (PDLC) at North American Aviation, Space and Information Systems Division, Downey, California. The PDLC consists of interconnected racks or consoles with sections manufactured by Control Logic, Inc. and other sections produced by Epsco, Inc. The particular configuration of logic elements most convenient for the design contained the following:

1. 40 NOR gates
2. 40 OR gates
3. 80 Inverters
4. 40 AND gates
5. 60 NAND gates
6. 5 4-Bit Shift Registers
7. 30 Flip-Flops
8. 20 Single Shots
9. 8 Power Amplifiers
10. 10 STROBE gates
11. 30 Output Lights.

The flip-flops were specially designed such that, when the set and reset inputs were tied together and used as a pulse input, the flip-flop became a counter.

Although larger complements of digital logic were available, the above was more than sufficient. The only restriction on the design was that not more than four-bit shift registers were available. This required a minor modification of the design from a seven-bit word length to a four-bit word length. This minor modification necessitated the following wiring changes:

1. Delete line from 0S7 to 1S7, Figure 9.
2. Install line from 0S4 to 1S7, Figure 9.
3. Delete line from 0S6 to Sheet 6, Figure 9.
4. Install line from 0S3 to Sheet 6, as deleted in 3 above.
5. Delete lines from 02S1, 02S2, and 02S4, Figure 9.
6. Install line from 02S4 to 1RM7, Figure 9.
7. Delete Registers 2A1, 2A2 and 2A4, and connect directly from 3A1 to 1A4, Figure 10.
8. Delete Registers 2B1, 2B2 and 2B4 and connect directly from 3B1 to 1B4, Figure 11.
9. Delete Registers 2C1, 2C2 and 2C4. Establish inputs to 1C4 identical to those originally for 2C4, Figure 13.

This procedure was essentially designed to check the validity of the logic; therefore, the design need only be checked at one speed (one position on the Execution Select Switch (ESS), Figure 9). The ESS and the pulse generator were deleted and the manual step switch buffered into the single shot, CP (see Figure 9).

The Instruction Select Switch (ISS) was deleted and the input buffers for the instructions were replaced with flip-flops (see Figure 9). The flip-flops were set each time that instruction was to be performed.

The data switches and data input buffers (see Figure 6) were replaced by wires to the appropriate logic bus, depending on what the input data should be at any time. The Power Available Reset (1PAR, see Figure 8) was deleted.

The full adder was designed from logic elements as shown in Figure 5.

The PDLC was connected to simulate the Digital Trainer (Figures 7 through 13), after incorporating the variations discussed above. Each instruction was correctly executed in the manual mode, as discussed above.

The conclusion was drawn, that the design, as given in Figures 7 through 13, was correct.

The major problems encountered in verifying the design were equipment problems. They included loading and very minor variations between the logic elements in the PDLC and those used in the Digital Logic Trainer. No other problems were encountered.

IX. DIGITAL TRAINING MANUAL

General

The following is a list of rather specific specifications and requirements. In actuality many of these specifications and requirements can have tremendous flexibility with only a slight variation of the basic design.

System Start Stop

The machine is ready for digital operations when the following check list is complete:

1. The machine is plugged into a basic AC power outlet of the type required by the power supplies (i.e., 120 volts).
2. The AC power switch is on.
3. The AC power lamp is on. This indicates that the two fuses are good and AC power is available (see Figure 13).

Operating Specifications

The operating characteristics of the machine are summarized in the following list:

1. Register Size - Six Binary Bits, plus a polarity bit.
2. Data - (a) Positive numbers normal form with a 0 sign bit.
   (b) Negative numbers two's complement form with a true sign bit.

3. Step Rate

(a) Fast - greater than 10,000 CPS.

(b) Slow - from 1 to 10 CPS.

(c) Manual - Operator control.

4. Number Size - six binary bits

(a) Input Numbers are -64 to +63 decimal

(b) Answers - Maximum answer for addition of subtraction of two numbers is -64 to + 63 decimal. Maximum answer for multiplication of two numbers is -4096 to +4095 decimal or twelve binary bits (see page 26).

5. Program and Step Counters

(a) Counting code is binary.

6. Operator Devices

(a) Instruction Selector Switch (ISS) and lamp.

(b) Execute pushbutton and lamp.

(c) Execution Select Switch (ESS) or rate selector.

7. Arithmetic Operations

(a) Add

(b) Subtract

(c) Multiply

(d) Complement

8. Logic Operations

(a) AND

(b) OR

(c) Exclusive OR.

Sub-System Operation

Power Application Reset

All bi-stable elements and logic seal type circuits are subject to a race when the power supply is energized. This race is to establish which of the two stable states will be true. Factors which influence this race are element loading, component tolerance and other factors.

In order to have the equipment energize in a predictable manner, the power application reset function is used to initialize the bi-stable elements. A signal is generated by application of power, which is used to clear or set all bi-stable elements. The circuit to accomplish this function is shown on Figure 8 as 1PAR. When power is applied, the output if 1PAR will be true for a short time. This definite time that 1PAR is true, after power is applied, is used to initialize the bi-stable elements.

Overflow

Whenever the overflow lamp is on, the calculation result is incomplete because it is too large for the machine to handle. Any addition or subtraction result that is more negative than 64 or more positive than 63 will turn the overflow lamp on. Any multiplication result that is more positive than 4095 or more negative than 4096 will turn the overflow lamp on.

Instruction List

The machine is capable of carrying out any of fifteen distinct operations. Each of these operations is termed an instruction. The following list summarizes the fifteen instructions and the operations they perform. An individual explanation of each instruction follows the instruction list.

		DESCRIPTION	OPERATION
0	CLR	Clear All Registers	$0 \rightarrow A$, $0 \rightarrow B$, $0 \rightarrow C$, $0 \rightarrow P$, $0 \rightarrow S$
1	CLA	Clear A Register	$0 \rightarrow A$
2	INP	Input Data in Parallel	Data $\rightarrow B$
3	INS	Input Data Serially	Data $\rightarrow B$
4	ADD	Add $A + B$	$A + B \rightarrow A$, $B \rightarrow B$
5	SUB	Subtract $A - B$	$A - B \rightarrow A$, $B \rightarrow B$
6	XAC	Exchange A and C	$A \rightarrow C$, $C \rightarrow A$
7	MPY	Multiply	B Multiplied By C and answer placed in AC
8	CPM	Complement A	Inverse $A \rightarrow A$
9	BTS	Conditional Branch	If 3C1 1, Add 1 to Program Counter
10	BRU	Unconditional Branch	Data $\rightarrow$ Program Counter
11	LSR	Long Right Shift	Shift A and C Together One Place
12	ANAB	"AND" A with B	A "AND" $B \rightarrow A$, $B \rightarrow B$
13	ORAB	"OR" A with B	A "OR" $B \rightarrow A$, $B \rightarrow B$
14	ERAB	"Exclusive OR" A & B	A "Exclusive OR" $B \rightarrow A$, $B \rightarrow B$

Instruction Explanation

CLR This instruction requires one step. Executing this instruction clears (resets all bits in) the A register, B register, C register, Step counter and Program counter. It usually is the first instruction in a program. In this case, its function is to initialize the registers.

CLA This instruction requires one step. Executing this instruction clears the A register.

INP This instruction requires one step. Executing this instruction transfers data from the data switches (see Figure 8) into the B register (see Figure 11) in parallel fashion. Any old data is automatically cleared before new data is transferred into the B register.

INS This instruction requires seven steps. Executing this instruction transfers the data from the data switches to the B register in a serial fashion. Any old data is automatically shifted out as the new data is shifted into the B register. This instruction is also used to put new data into the machine.

ADD This instruction requires seven steps to execute. It serially adds the number in the A register to the number in the B register bit by bit through the full adder and shifts the answer into the A register. During this operation, the B register shifts in a ring. Therefore, the number in B at the beginning of the instruction will return to B after the instruction.

SUB This instruction requires seven steps. Similar to the ADD instruction, it adds the number in the A register to the two's complement of the number in the B register and shifts the answer into the A register. The answer resulting from this operation will be A - B. A circulating right shift is performed in the B register.

XAC This instruction requires seven steps. It exchanges A and C register data. It is usually applied during the initial programing of multiplication in order to get the multiplier into the C register.

MPY This instruction requires eight steps for each 1 in the data bits of the C register, plus one step for each 0 in the C register. If 1C1 is true, it performs an add operation and then shifts A and C together one place to the right. The shift takes place around 3C1 (data shifts from 1A1 to 2C4, see Figure 13). 3C1 retains its previous state through out the operation. If 1C1 is false, only the shift right is performed. As explained in the section on the multiplication of binary numbers, the computer performs one of above operations a total of six times depending on whether 1C1 is true or false each time. For a more detailed explanation, please see pages 21 through 27.

CPM This instruction requires seven steps to execute. It inverts (one's complement) each digit in the A register.

BTS This instruction requires one step to execute. If 3C1 is false, the program counter will be advanced one count just as it normally is each time an instruction is executed. If 3C1 is true, The program counter will be advanced two counts. This instruction is required immediately after a multiplication procedure in order to program the additional subtraction required when the multiplier is minus; that is, when 3C1 is true.

BRU This instruction requires one step to execute. It changes the program counter to the number set on the data input switches. It allows the programmer to make a random change in the program when required. This instruction can be used after a multiplication procedure in order to bypass the subtraction step if the sign of the multiplier is positive; that is, if 3C1 is false.

LSR This instruction requires one step to execute. It shifts the A and C register one bit to the right. The operation, in detail, is as follows: A zero is shifted into 3A1, 3A1 is shifted into 2A4, 2A4 is shifted into 2A2 and so on. 1A1 is shifted into 3C1, 3C1 is shifted into 2C4, 2C4 is shifted into 2C2 and so on. 1C1 is shifted out of the machine.

ANA This logic instruction requires seven steps to execute. Digit by digit A and B register data is logically AND'ed. The result is put into the A register. A circulating right shift or ring shift is performed in the B register. The logic function accomplished by this operation is as follows: If 1A1 and 1B1 were initially true, 1A1 will be true after the operation, and so on for all seven digits.

ORA This logic instruction requires seven steps to execute. Digit by digit A and B register data is logically OR'ed. The result is put into the A register. A circulating right shift or ring shift is performed in the B register. The logic function accomplished by this operation is as follows: If 1A1 or 1B1 were initially true, 1A1 will be true after the completed operation, if 1A2 or 1B2 was originally true, 1A2 will be true after the operation, and so on for all seven digits.

ERA This logic instruction requires seven steps to execute. Digit by digit A and B register data is Exclusive OR'ed. The result is put into the A register. A circulating right shift or ring shift is performed in the B register. The logic function accomplished by this operation is as follows: If 1A1 or 1B1, but not both, were initially true, 1A2 will be true after the operation, and so on for all seven digits.

Computer Experiments

The instructions described previously can be put together in building block fashion to perform an infinite variety of arithmetic or logic tasks. This is termed programming and the person who instructs the machine is a programmer.

Following are a number of simple experiments that can be performed by the programmer. Performing these experiments will illustrate what each of the instructions actually do and how they can be combined to perform simple digital tasks. Before attempting the experiments, the programmer should familiarize himself with all the available instructions.

Table 8a

Clearing and Loading The Machine

PROG CTR.	INSTRUC-TIONS	SIGN	DATA 4	2	1	4	2	1	ACTION
XX	CLR								Machine Initalized to 0's
00	INP	1	1	1	1	1	1	1	Data in B
01	ADD								Data in A and B
02	XAC								Data in B and C
03	ADD								Data in A, B and C
04	CLA								A to 0's
05	ADD								Data in A, B and C
06	CLA								A to 0's
07	INS	1	0	1	0	1	0	1	Data in B
10	INS	0	1	0	1	0	1	0	Data in B
11	INP	1	0	1	0	1	0	1	Data in B

NOTES: The operator should perform the program with various step rates, (Fast - Slow - Manual). This experiment was designed to illustrate the following:

(a) Step XX illustrates the clear all instruction.

(b) Step 4 illustrates the clear A instruction.

(c) Performing Steps 7, 10 and 11 illustrates data loading. Notice that the B register accepts exactly the new data (Step 10 and 11) while destroying the old (Step 7 or 10).

Table 8b

Solve the Equation $X = a + b - c$

PROG	INSTRUC-				DATA				
CTR.	TION	SIGN	4	2	1	4	2	1	ACTION
XX	CLR								Machine Initalized to 0's
00	INP	$\longleftarrow$			a			$\longrightarrow$	$a \rightarrow B$
01	ADD								$a \rightarrow A$
02	INP	$\longleftarrow$			b			$\longrightarrow$	$b \rightarrow B$
03	ADD								$a + b \rightarrow A$
04	INP	$\longleftarrow$			c			$\longrightarrow$	$c \rightarrow B$
05	SUB								$a + b - c \rightarrow A$
06	Answer in A								

NOTES: The operator should perform the program with various step rates (Fast - Slow - Manual). He should run the program a number of times with various numerical sizes and signs for the numbers a, b, and c.

Table 8c

Solve the Equation $X = c(b) + a$

PROG	INSTRUC-				DATA				
CTR.	TION	SIGN	4	2	1	4	2	1	ACTION
XX	CLR								Machine Initalized to 0's
00	INP	←			c			→	Multiplier → B
01	ADD								Multiplier → A
02	XAC								Multiplier → C
03	INP	←			a			→	Initial Partial Product → B
04	ADD								Initial Partial Product → A
05	INP	←			b			→	Multiplicand → B
06	MPY								Multiply
07	BTS								Test Multiplier
10	BRU	0	0	0	1	1	0	0	Branch to Answer if Multiplier is
11	SUB								Subtract if Multiplier is Negative
12	Answer in C and A								

NOTES: The operator should perform the program in Table 8c, with various step rates (Fast - Slow - Manual). He should run the program a number of times with various numerical sizes and signs for the number a, b and c.

Table 8d

Determine if a = b

PROG	INSTRUC-	DATA							
CTR	TION	SIGN	4	2	1	4	2	1	ACTION
XX	CLR								Machine Initalized to 0's
01	IMP	←			a			→	$a \rightarrow B$
02	ADD								$a \rightarrow A$
03	CPM								$a \rightarrow A$
04	IMP	←			b			→	$b \rightarrow B$
05	ERAB								A "Exclusive OR" $b \rightarrow A$

If each digit of the A register is true, the numbers were equal. This can be checked using the machine as follows:

06	LSR								First Shift
07	BTS								Test First Digit
10	Bru	0	0	1	1	0	1	0	$a \neq b$, Branch to Final instruction
11	LSR								Second Shift
12	BTS								Test Second Digit
13	BRU	0	0	1	1	0	1	0	$a \neq b$, Branch to Final Instruction
14	LSR								Third Shift
15	BTS								Test Third Digit

Table 8d

(Continued)

PROG CTR.	INSTRUCTION	SIGN	DATA 4	2	1	4	2	1	ACTION
16	BRU	0	0	1	1	0	1	0	a$\neq$b, Branch to Final Instruction
17	LSR								Fourth Shift
20	BTS								Test Fourth Digit
21	BRU	0	0	1	1	0	1	0	a$\neq$b, Branch to Final Instruction
22	LSR								Fifth Shift
23	BTS								Test Fifth Digit
24	BRU	0	0	1	1	0	1	0	a$\neq$b, Branch to Final Instruction
25	LSR								Sixth Shift
26	BTS								Test Sixth Digit
27	BRU	0	0	1	1	0	1	0	a$\neq$b, Branch to Final Instruction
30	LSR								Seventh Shift
31	BTS								Test Polarity
32	The Numbers are Not Identical.								
33	The Numbers are Identical.								

Table 8d

(Continued)

The first five steps could also be performed as follows:

PROG	INSTRUC-				DATA				
CTR.	TION	SIGN	4	2	1	4	2	1	ACTION
XX	CLR								Machine Initialized to 0's
01	INP	←			a			→	$a \rightarrow B$
02	ADD								$a \rightarrow A$
03	INP	←			b			→	$b \rightarrow B$
04	ANAB								a "AND" b $\rightarrow$ a

NOTES: The operator should perform the program at various step rates (Fast - Slow - Manual). He should perform it with $a=b$ and $a \neq b$.

There are, of course, an infinite variety of programs which can be performed using the trainer. The above were only some of the more simple examples and we shall leave to the ingenuity of the student whatever other experiments he deems necessary.

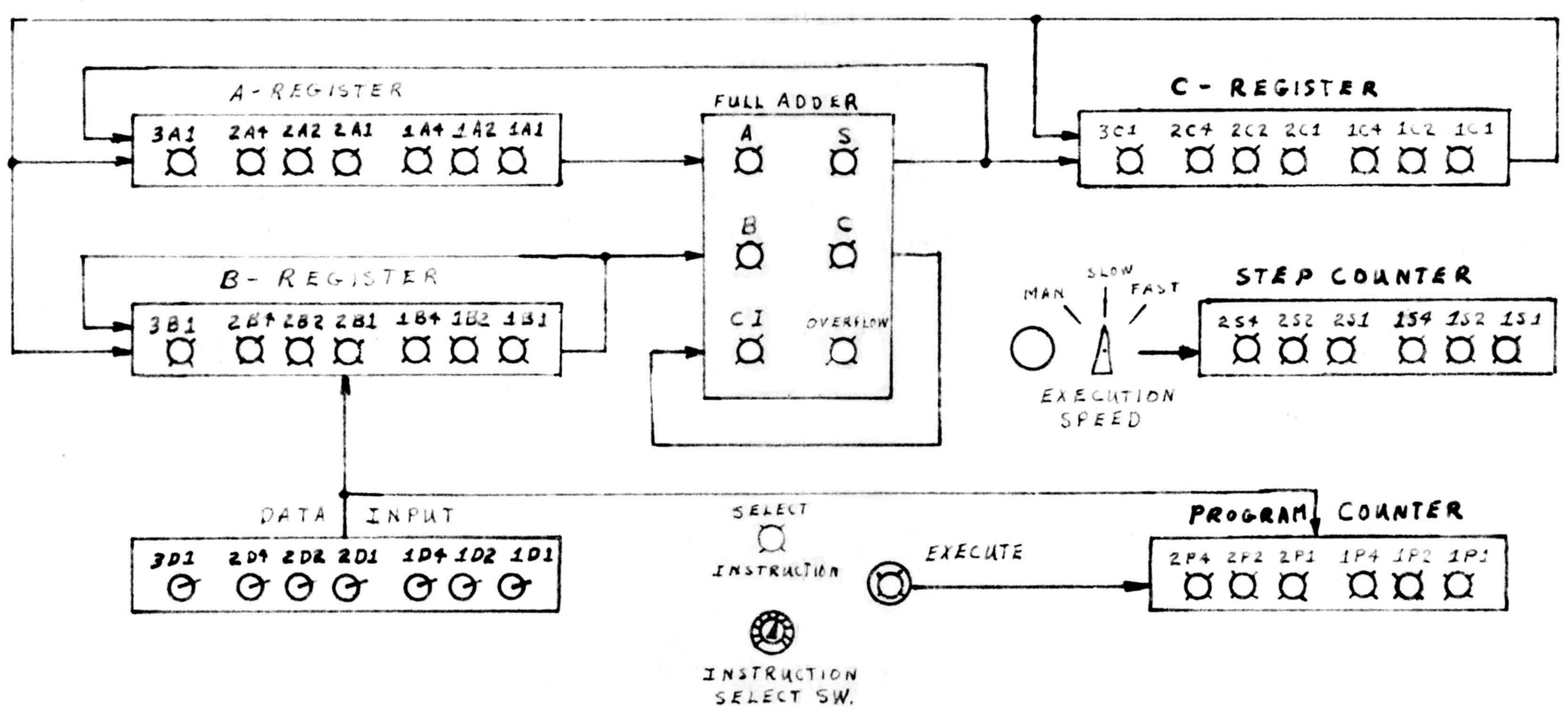
X. SYSTEM LOGIC DESIGN

Alphabetic Mnemonic Listings of Logic Elements

A	A register
ADD	ADD
AIN	A Input to Full Adder
ANAB	"AND" A with B logically
AOM	Add on Multiply
B	B register
BIN	B Input to Full Adder
BRU	Branch Unconditionally
BTS	Branch on Test (3C1) Set
C	C register
CAR	Carry Register
CLA	Clear the A register
CLR	Clear all registers
CP	Clock Pulse
CPE	Clock Pulse Extended
CPM	Complement the A register
CSP	C register Shift Pulse
D	Data input
DA	Data Auxiliary

ERAB	"Exclusive OR" A with B logically
EXCT	Execute
EXCTA	Execute Auxiliary
INP	Input Data in Parallel
INS	Input Data Serially
LAC	AC Power Light
LSR	Long Shift Right
MIB	Manual Input Buffer
MPY	Multiply
MSTG	Manual Step Trigger
MXCA	Multiply Execution Complete Auxiliary
MXCE	Multiply Execution Complete
NLB	Negative Light Bus (dc)
NLPWR	Negative Logic Power (dc)
OF	Overflow
ORAB	"OR" A with B logically
P	Program counter
PAR	Power Available Reset
PG	Pulse Generator
RM7	Register Multiply Seven
RST	Reset system

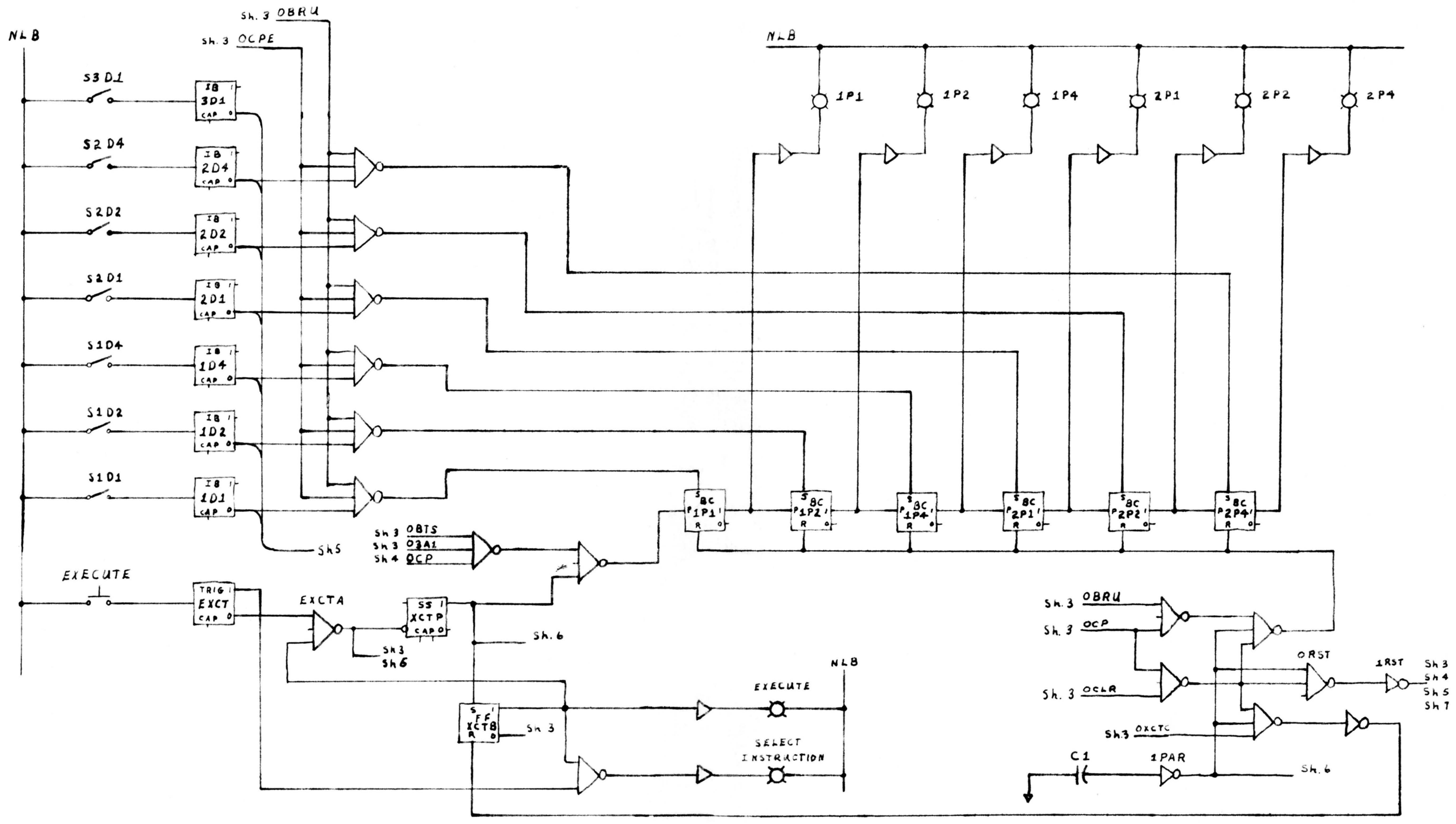
S	Step Counter
S7	Step seven
SAB	Shift the A & B registers
SAC	Shift the A & C registers
SAD	Serial Adder
1SE	One Step Execution
SR7	Set to Register Seven Steps
SUB	Subtract
XAC	Exchange A and C registers
XCTB	Execute Buffer (Flip-Flop)
XCTC	Execution Complete
XCTP	Execute Pulse



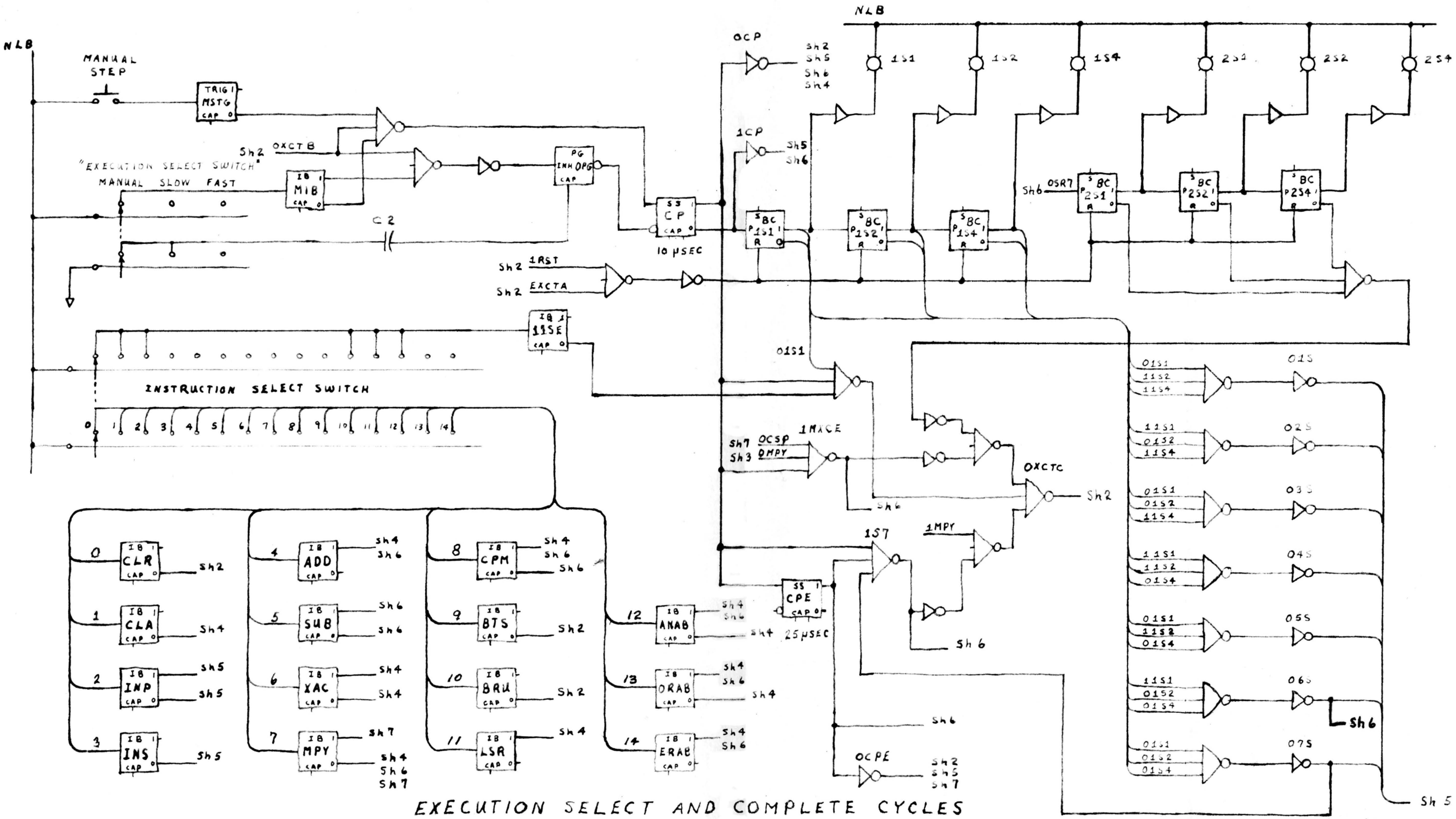
INSTRUCTION LIST

NUMBER	ABBREVIATION	DESCRIPTION	OPERATION
0	CLR	CLEAR ALL REGISTERS	$0 \rightarrow A, 0 \rightarrow B, 0 \rightarrow C, 0 \rightarrow P, 0 \rightarrow S$
1	CLA	CLEAR A REGISTER	$0 \rightarrow A$
2	INP	INPUT DATA IN PARALLEL	DATA $\rightarrow$ B
3	INS	INPUT DATA SERIALLY	DATA $\rightarrow$ B
4	ADD	ADD A + B	$A + B \rightarrow A$, $B \rightarrow B$
5	SUB	SUBTRACT	$A - B \rightarrow A$, $B \rightarrow B$
6	XAC	EXCHANGE A & C	$A \rightarrow C$, $C \rightarrow A$
7	MPY	MULTIPLY	B MULTIPLIED BY C & ANSWER PLACED IN AC
8	CPM	COMPLEMENT A	INVERSE A $\rightarrow$ A
9	BTS	CONDITIONAL BRANCH	IF 3C1 = 1, ADD 1 to PROGRAM COUNTER
10	BRU	UNCONDITIONAL BRANCH	DATA $\rightarrow$ PROGRAM COUNTER
11	LSR	LONG SHIFT RIGHT	SHIFT A AND C TOGETHER ONE PLACE
12	ANAB	"AND" A & B	A "AND" B $\rightarrow$ A, B $\rightarrow$ B
13	ORAB	"OR" A & B	A "OR" B $\rightarrow$ A, B $\rightarrow$ B
14	ERAB	"EXCLUSIVE OR" A & B	A "EXCLUSIVE OR" B $\rightarrow$ A, B $\rightarrow$ B

BLOCK DIAGRAM AND INSTRUCTION LIST
SHEET 1
FIGURE 7



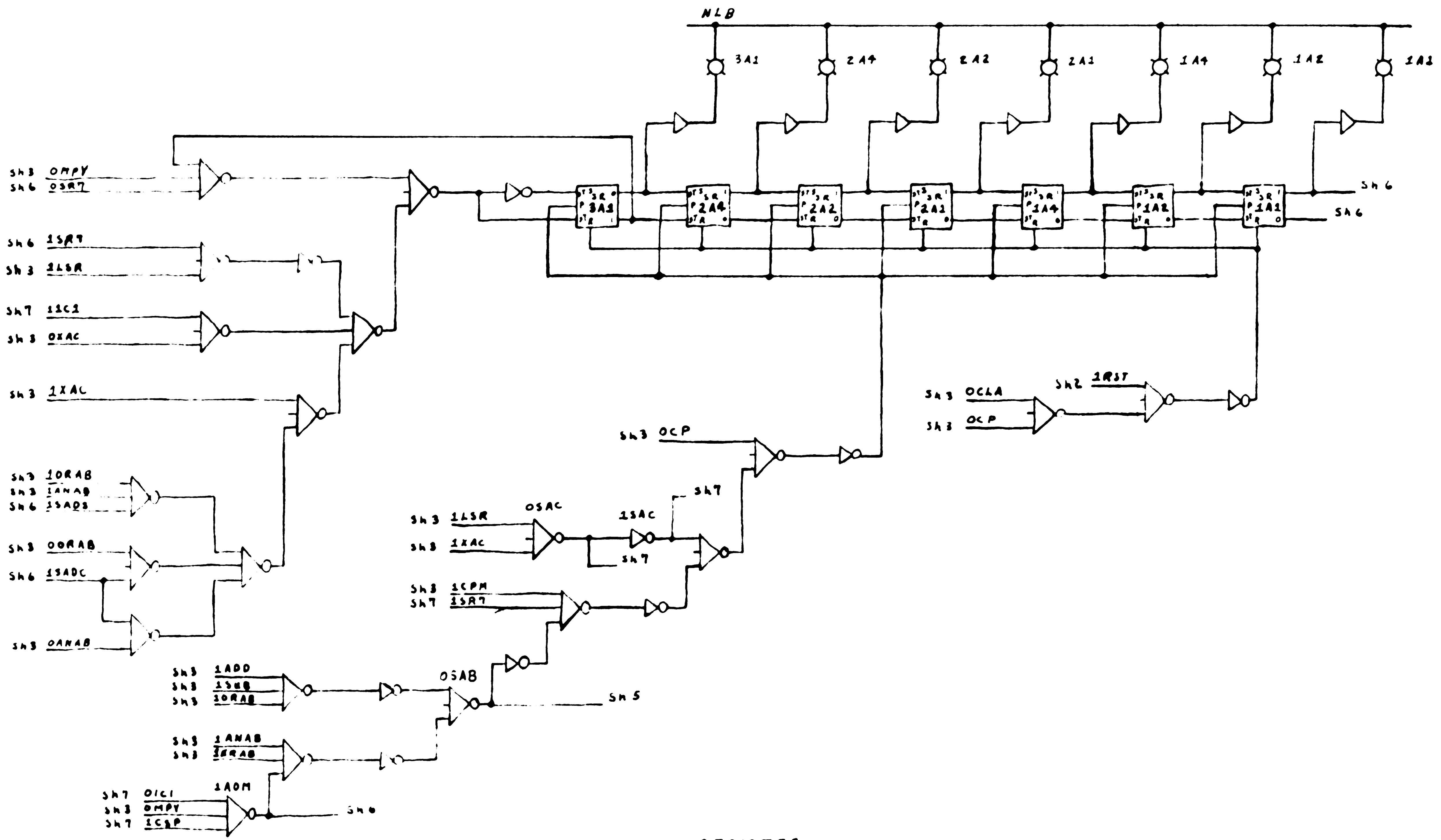
EXECUTE CYCLE AND PROGRAM COUNTER (SHEET 2)
FIGURE 8



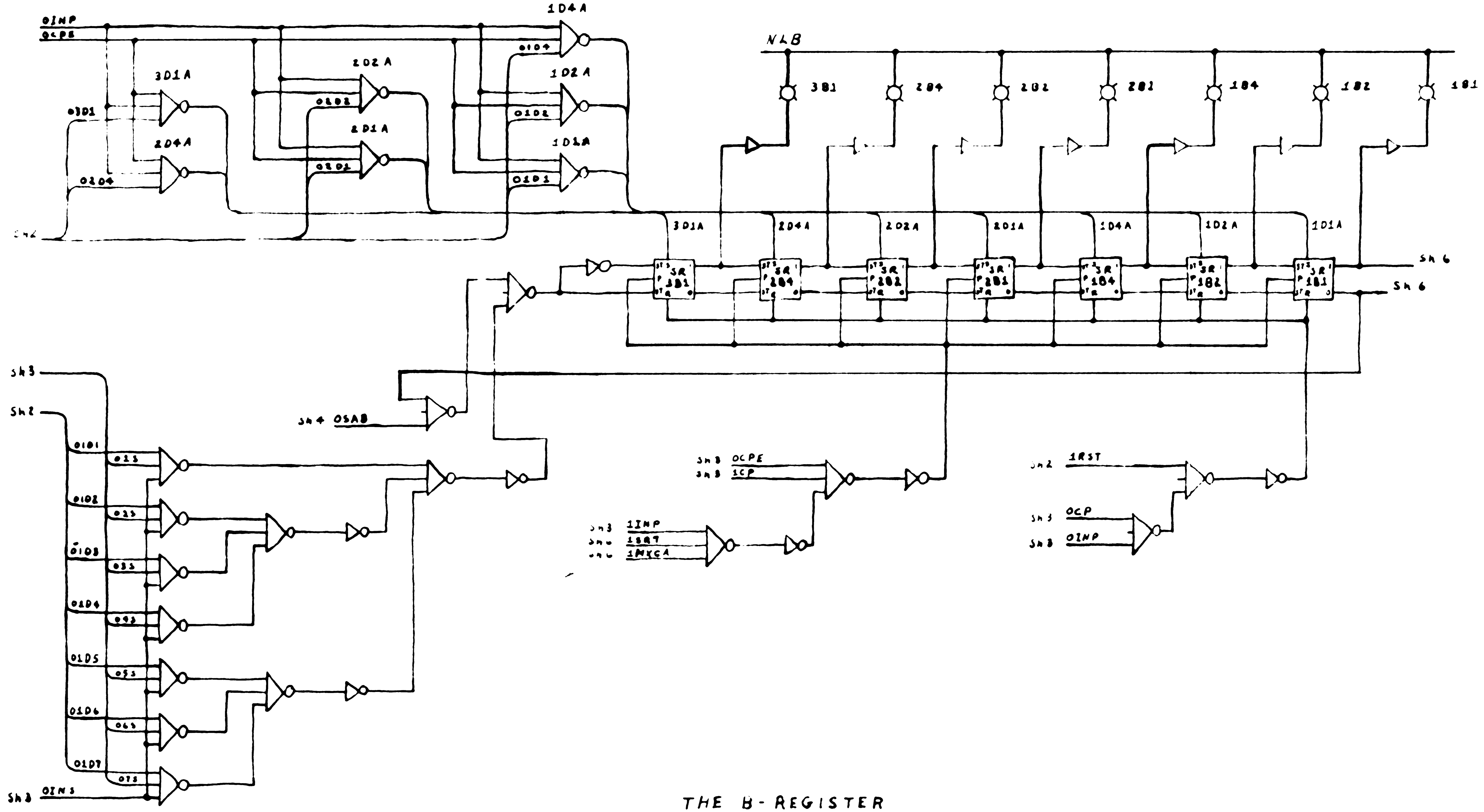
EXECUTION SELECT AND COMPLETE CYCLES

SHEET 3

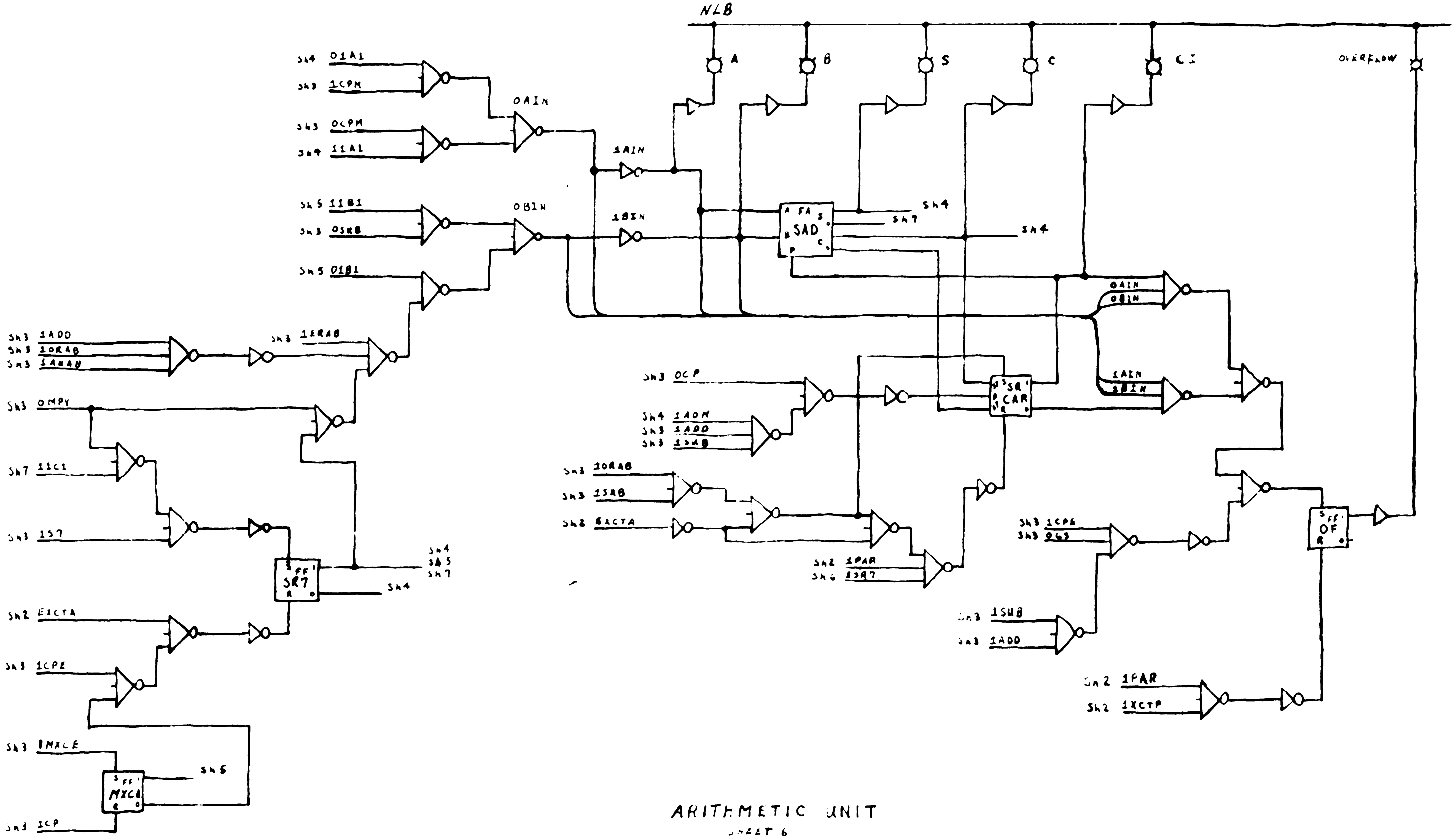
FIGURE 9



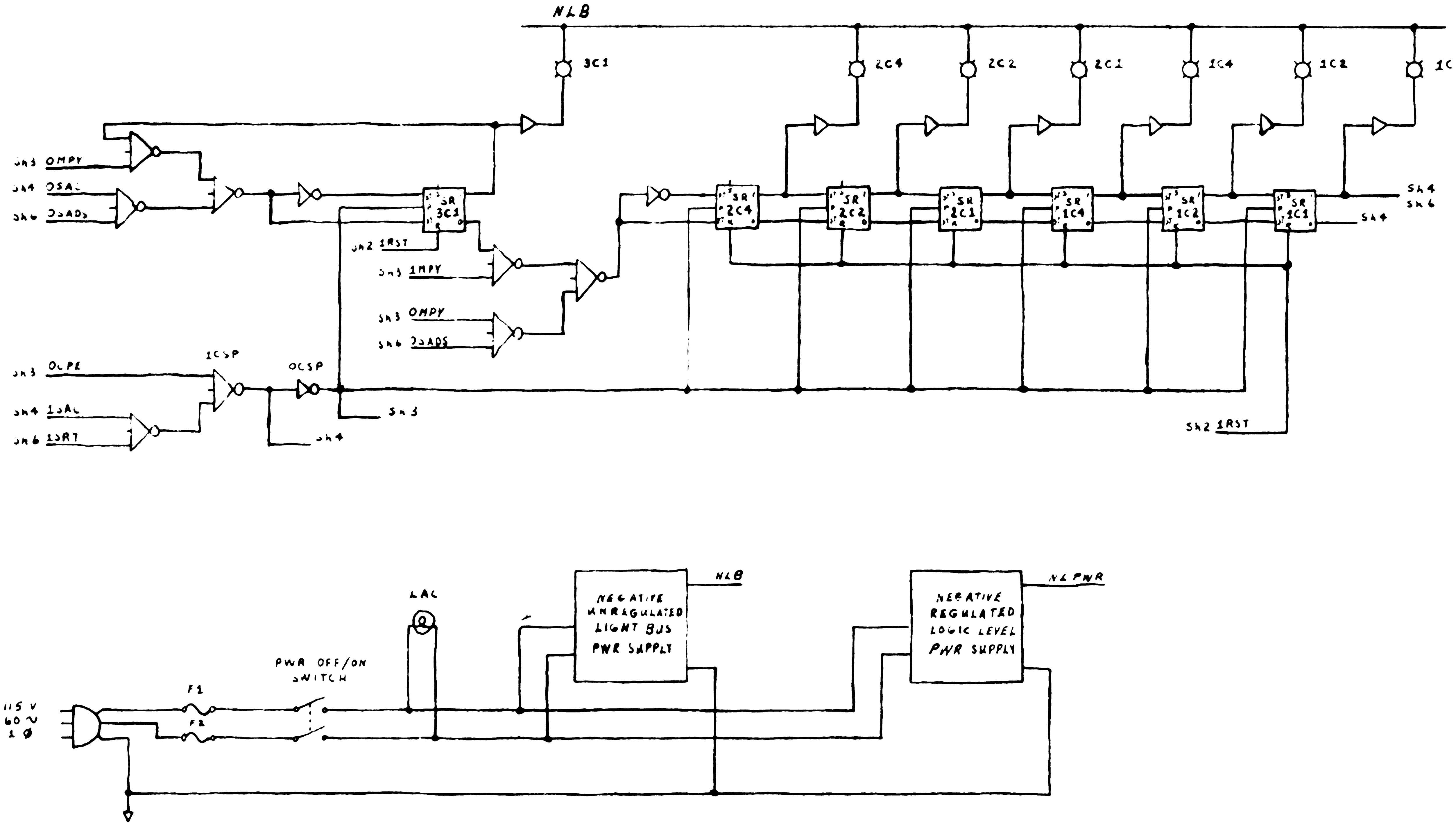
THE A-REGISTER
SHEET 4
FIGURE 10



THE B-REGISTER
SHEET 5
FIGURE 11



ARITHMETIC UNIT
SHEET 6
FIGURE 12



C - REGISTER AND POWER SUPPLIES
SHEET 7
FIGURE 13

XI. SUMMARY

This thesis was developed with the hope that it would take some of the awe from the everyday term, digital computer. The term digital means that the function is only defined at discrete values, locations or positions. The opposite of digital is analog, which has a continuous spectrum of values. The term digital usually implies binary digital and is, therefore, restricted to a two-state device.

In this thesis a binary digital system, capable of performing arithmetic operations, was developed from this simple two-state device. This binary digital system or digital computer could be programmed to add, subtract or multiply and perform many other logic functions. The above operations were each performed serially on a group of binary digits stored in seven bit registers. The system was manually programmed using an Instruction Select Switch.

During each phase of the development of the system, an example of the procedure, used to reach a conclusion during that phase, was explained at length. This conclusion, in many cases, was designer's prerogative. Many other conclusions could have been selected, yielding a functionally identical system using different logic configurations.

It is hoped that following and continually reapplying the procedures outlined in this thesis, particularily in the development of the Full Adder, would allow for the design of any moderately complex digital system.

XII. BIBLIOGRAPHY

Literature Examined

Flores, I. Computer Logic, Prentice-Hall, Inc., New Jersey, 1960

Karnaugh, M., "The Map Method for Synthesis of Combinational Logic Circuits", Transactions A.I.E.E., Part 1: Communications and Electronics, Volume 72, Pages 593 through 599.

Millman, J. and Taub, H., Pulse and Digital Circuits, McGraw-Hill, 1956

Stibitz, G. and Larrivee, J., Mathematic and Computers, McGraw-Hill, 1957

Stauss, L., Wave Generation and Shaping, McGraw-Hill, 1960

XIII. ACKNOWLEDGEMENTS

An expression of appreciation is extended to the following individuals whose efforts have contributed both to the idea and the practical realization of the thoughts from which this Thesis has grown:

Professor B. L. Dennison, Major Professor

, Critic

, Typist

Dr. M. H. Hopkins

Dr. W. W. Cannon

Dr. R. H. Miller

Professor R. R. Wright.

ABSTRACT

In this thesis a binary digital system, capable of performing arithmetic operations, was developed from a simple two-state (binary digital) device. This binary digital system or digital computer could be programmed to add, subtract or multiply and perform many other logic functions. The above operations are performed serially in seven bit shift registers. The system is manually programmed using an Instruction Select Switch.

During each phase of the development of the system, an example of the procedure used to reach a conclusion during that phase, was explained at length. It is hoped that following and continually reapplying the basic procedures outlined in this thesis, particularily in the development of the Full Adder, would allow for the design of any moderately complex digital system.