THE LOGICAL DESIGN OF A SIMPLE MAGNETIC CORE DIGITAL COMPUTER

By

DAVID WAYNE FLEMING

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1958

Submitted to the faculty of the Graduate School of
the Oklahoma State University
in partial fulfillment of the requirements
for the degree of
MASTER OF SCIENCE
May, 1963

THE LOGICAL DESIGN OF A SIMPLE MAGNETIC CORE DIGITAL COMPUTER

Thesis Approved:

[illegible]
Thesis Adviser

C. M. Summers

[illegible]
Dean of the Graduate School

541928

PREFACE

The general purpose digital computer is a relatively new device. There has been much work done in the field of designing these computers. There have been many texts and papers written on the various aspects of design. Most writings refer to the individual units of a computer, without joining these units into a system. This thesis develops the logical design of the various units of a computer, and joins them into a workable unit.

The reader of this thesis should be familiar with Boolean algebra, simplification techniques, and the several types of logic gates and logic elements such as the bistable multivibrator.

The methods used in this analysis and design are not new, but the logical presentation of the problems of design, and the logical listing of the solutions to these problems should be an aid in the design of any digital device.

The author wishes to express his sincere appreciation to Professor Paul A. McCollum for his guidance and support given in the preparation of this thesis. He also wishes to thank Mr. William P. Cannon of the Federal Aviation Agency for his valuable technical advice. Also, the author is indebted to Mr. C. W. Mueller, and Mr. John E. Shaw of the Federal Aviation Agency Academy, Oklahoma City, Oklahoma for allowing him to use the Academy's Computer Laboratory to test the logic designed in this thesis. A special note of appreciation goes to my wife, Gwen, for her perseverence and support while writing this thesis, and for typing the manuscript.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The analysis and design of a digital computer presents many problems to the design engineer. He must consider the purpose of the machine. That is, is the computer to be a special purpose or a general purpose machine? If it is to be a general purpose machine, how versatile should it be? Or if it is to be a special purpose machine, have all of the requirements of the machine been met?

All of these problems indicate that a logical approach to the design of any digital computer must be made. The object of this thesis will be to synthesize a simple general purpose digital computer. The methods used for this synthesis will be applicable to the analysis and design of any logical device, whether it is a digital computer, or a control device for a radar system.

Design Methods

In many cases, the approach to the design problem may be intuitive, and in other cases logical methods using boolean algebra, Karnaugh maps, and logic simplification techniques will be used. It will be found in this thesis that a combination of both methods must be used. However, before any method can be used, a complete statement of the problem must be made.

The Simple Computer

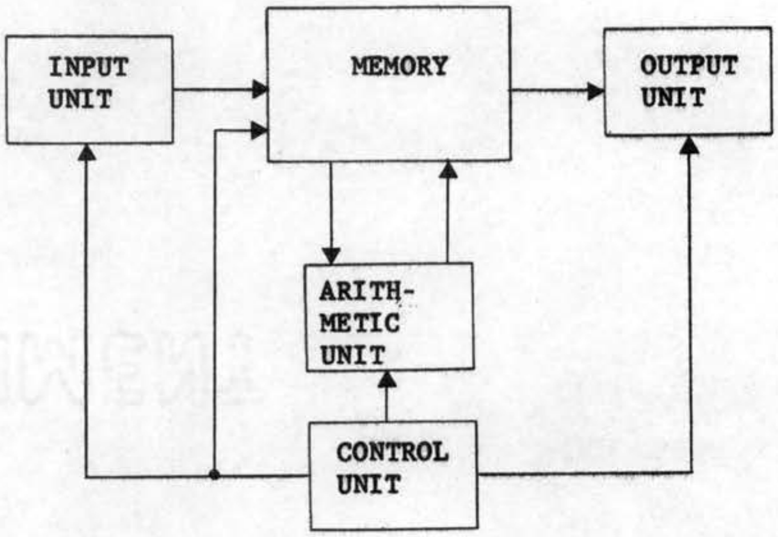


Figure 1. Block Diagram of a Digital Computer

The block diagram of a simple general purpose digital computer is shown in Figure 1. Each of the units must be taken into consideration in designing and analyzing the digital computer. This thesis will consider the logical design of the units shown in the block diagram. Particular attention will be paid to the logical design of the memory unit, the arithmetic unit, and the control unit. The control of the input and output units will be considered.

Specifications

A word length of 16 bits was selected. This was dictated by the source of parts and arbitrary design specifications. The computer was to be a binary machine, and all communications with the machine was to be in binary or in the octal numbering system. Each computer word was to be fixed length with a fixed binary or radix point. The computer will be designed using a coincident current ferrite core memory as the "brain" of the system.

Obviously, the main factor influencing the design of the computer will be the type of problems that are expected to be solved by the computer. This computer will be capable of solving problems involving addition, subtraction, multiplication, division, and any type of problem that may be reduced to some numerical form of analysis. Since this computer is a binary machine, its main application will be in the fields of science, and not so much in the business fields. In order to design a machine that is more versatile in the fields of business, a decimal machine should be designed. The methods outlined in this thesis can be applied to the design of either a binary or a decimal machine.

CHAPTER II

MAGNETIC CORE MEMORY

The theory of storing a bit of information in a single ferrite core is an important concept in considering the design of a simple magnetic core computer. The ferrite core material is selected with a characteristic hysterisis loop that is practically rectangular as shown in Figure 1.

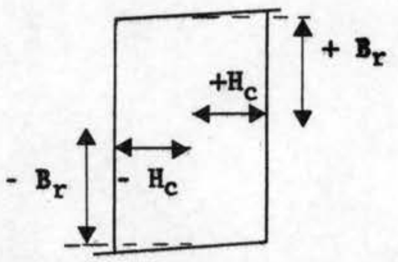


Figure 1. a. Typical Hysterisis Loop

Any core in memory may be storing either a logical one or a logical zero. Arbitrarily, let a positive residual flux density ($+B_r$) represent a stored one, and a negative residual flux density ($-B_r$) represent a stored zero. The individual cores may be switched from a one to a zero by applying a total magnetomotive force of $-H_c$ to the X and Y windings of a selected core. This is usually accomplished by applying $-H_c/2$ to the X winding and $-H_c/2$ to the Y winding. These two values add algebraically to a total of $-H_c$. Figure 1 illustrates the change in flux density when $-H_c$ is applied to a core storing a one. The core may be switched from a zero to a one by applying $+H_c/2$ to the X winding and $+H_c/2$ to the Y winding. In this case, the total magnetomotive force equals $+H_c$ and the core will be switched from a negative residual to a positive residual.

Each time the core is switched, the change in residual flux will induce a voltage in a sense winding. This information will be strobed or written into a buffer register, that serves as a buffer between core and the computer's control, arithmetic, and input-output logic.

A single core with the X, Y, inhibit, and sense windings is shown in Figure 2. A complete memory system may be built around this simple core, if proper care is taken in designing the system.

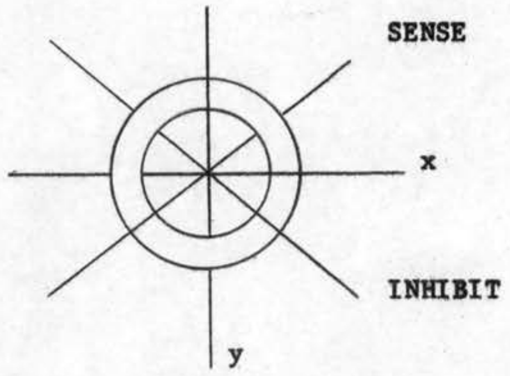


Figure 2. Ferrite Core Windings

In the specifications for this simple computer, the basic word length is 15 bits plus a sign bit, which totals 16 bits. 6 bits are to be used as the operation code, and 9 bits as the address portion of the computer word. Since there are 9 bits in each address, this means that there will be a total of 512 words stored in the magnetic core memory. This memory can be constructed as a 16x16x32 array. This may be described as 16 planes that are 16 cores by 32 cores. 16x32 = 512 words. One core plane with its 16 X windings and 32 Y windings and one sense winding is shown in Figure 3.

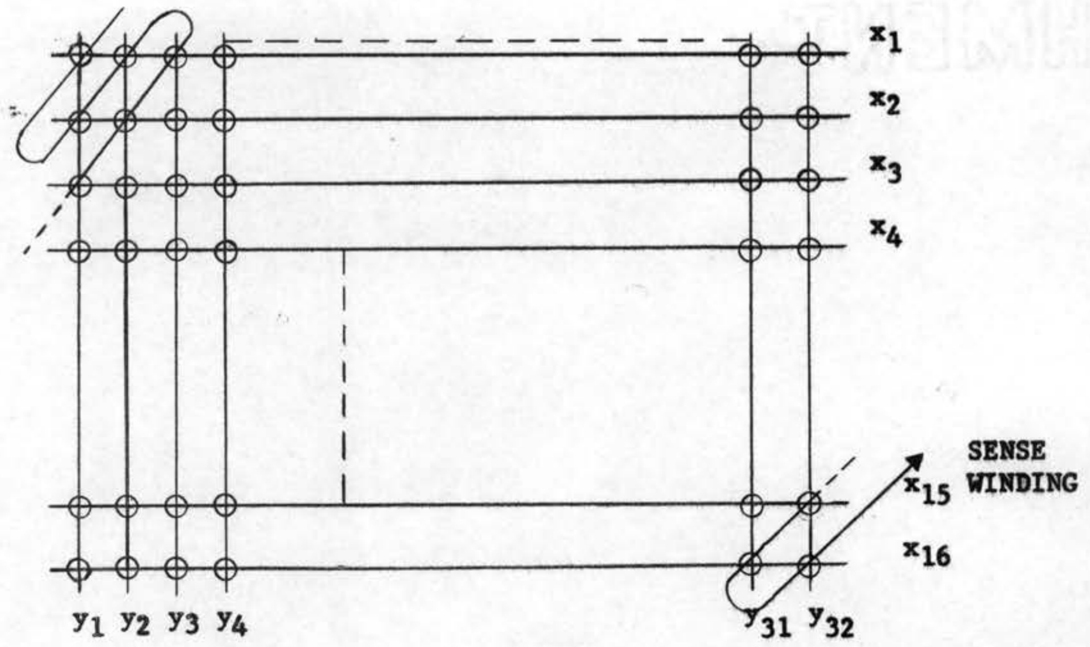


Figure 3. Windings of a Single Plane

Figure 4 is an illustration of the wiring techniques used in threading the cores of the several planes. The X row selector in one plane is in series with the corresponding X row selector of all planes of the core. The same is true of the Y column selectors. In each plane, there is one sense winding threading through all cores of that plane. There is also one inhibit winding threading through all cores of a single plane. Therefore, for this memory, there are 16 sense windings, 16 inhibit windings, 16 X windings, and 32 Y windings.

The following read-write cycle is one that is commonly used in conjunction with a coincident current ferrite core memory, and this cycle will be used with this computer.[1]

[1]Charles V. L. Smith, Electronic Digital Computers, New York, 1959, p. 295.

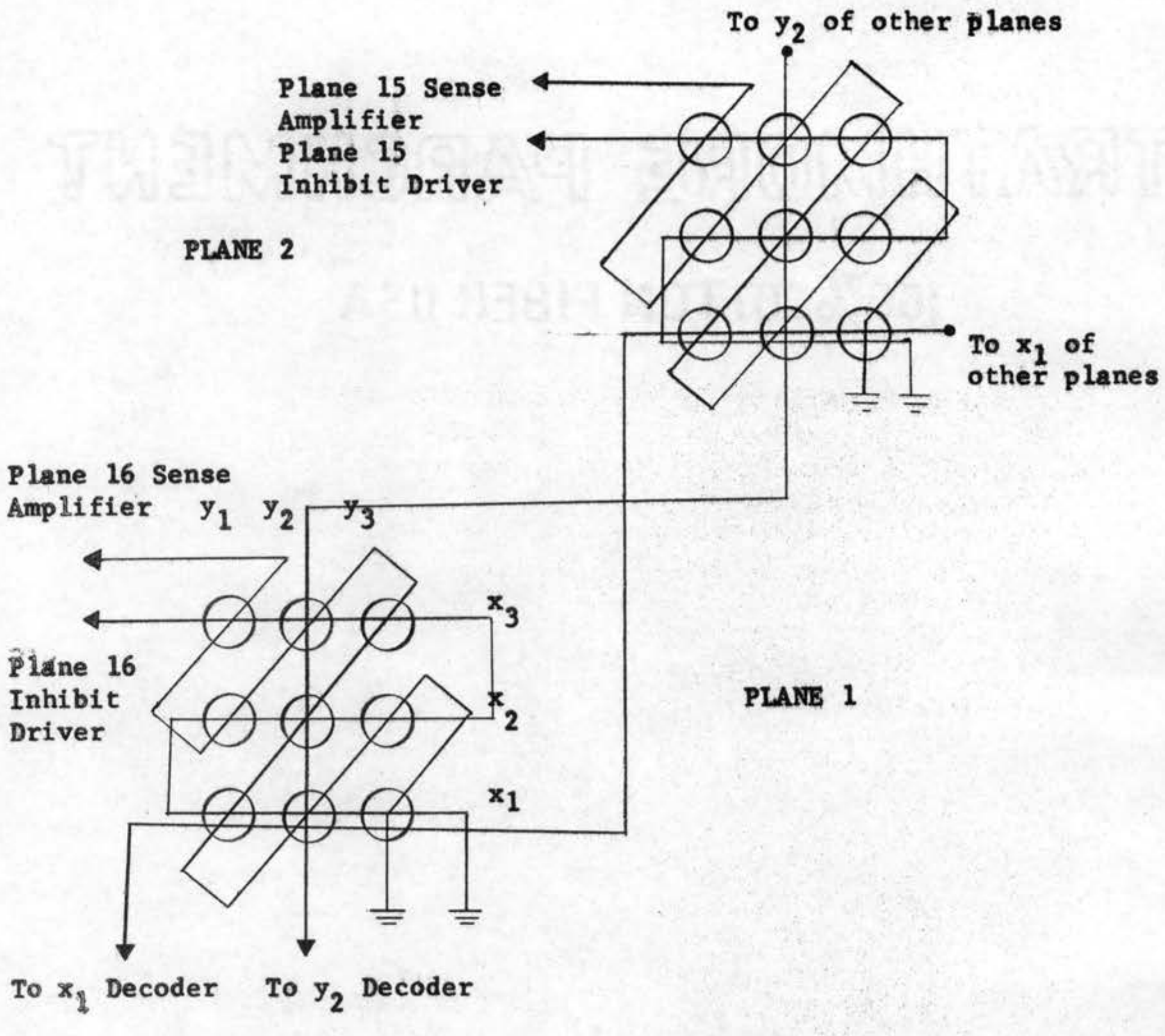


Figure 4. Multiple Plane Wiring

The Read-Write Cycle

1. Read all cores of a selected word to zero by applying a total of $-H_c$ magnetomotive force to the cores of the selected address.

2. (a.) Write ones into all cores of the selected address by applying a total of $+H_c$ magnetomotive force to the cores of that address. (b.) Simultaneously with step 2 a., inhibit the cores of the selected address that should be written zero.

The units necessary for the execution of this read-write cycle are

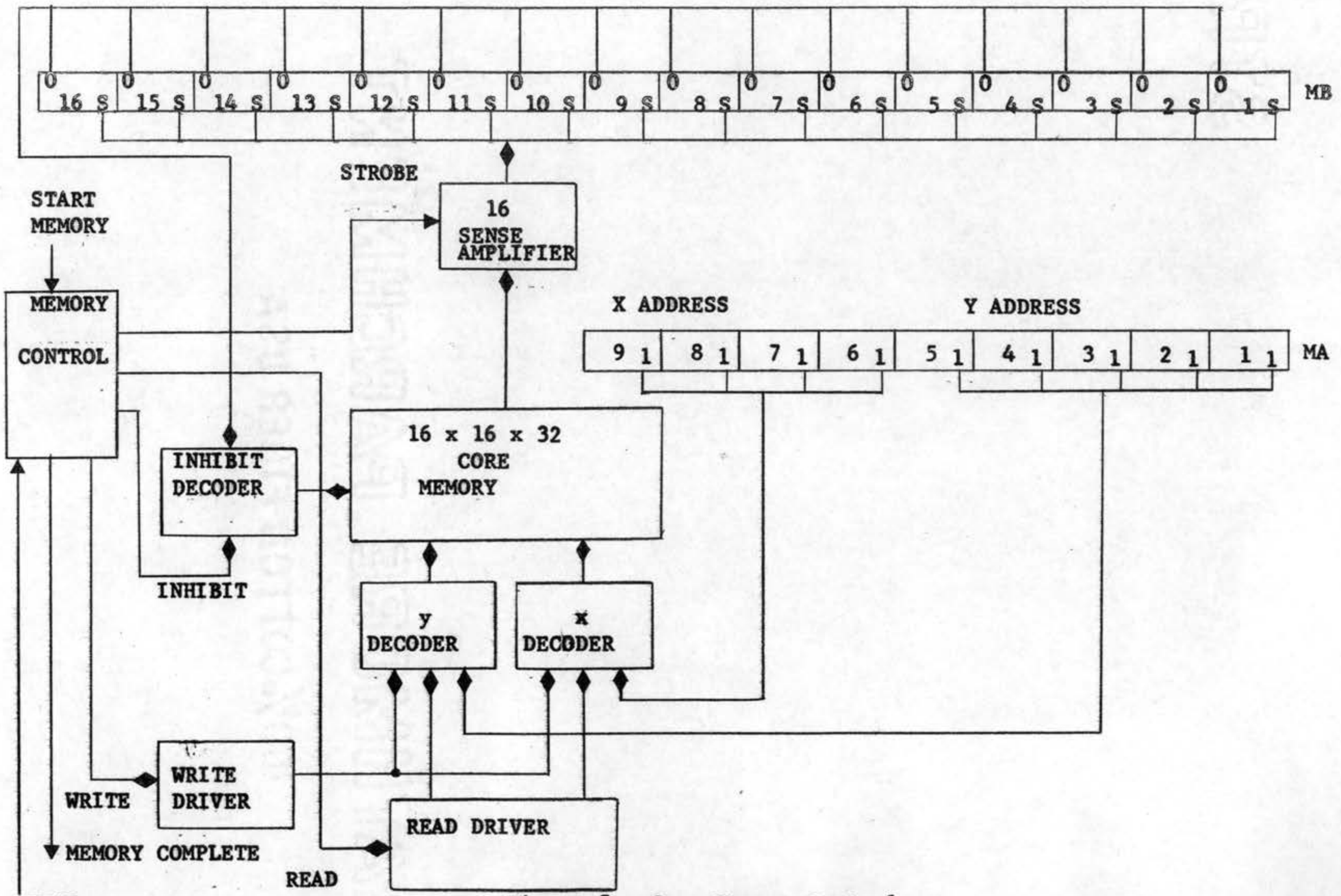


Figure 5. Core Memory Control

shown in Figure 5.

Execution of the Read-Write Cycle

Since the basic word address is made up of 9 bits, a 9 bit memory address register will be required to address the words stored in core. The basic computer word is 16 bits, therefore, a 16 bit buffer register will be needed to store the word read from memory, and to store the word to be written into memory. The memory address register is a parallel input, parallel output register consisting of 9 bistable multivibrators and is shown in Figure 6. The memory buffer register is parallel input, parallel output shift register that will be also capable of shifting right during arithmetic operations. The memory buffer register is shown in Figure 7.

The Memory Address Register

Some basic definitions must be made at this point in the design of the computer. Arbitrarily, negative logic has been chosen as the basic logic level for this design. A logic 0 will be defined as being equal to 0 volts, and logic 1 is equal to -3 volts. Since the logic 1 is more negative than the logic 0, this will be defined as negative logic. Conventional bistable multivibrators or binarys will be used in this computer. Definitions of the symbols used and the components used in all drawings are shown in the appendix.

The Memory Address register is shown in Figure 6 as a 9 bit parallel input register. This register may be cleared by the control logic by pulsing the Clear MA line. The set and reset inputs to these 9 binarys as well as all other binarys in the computer will be standard

-2.5 volt, 0.4 usec pulses. The parallel inputs to the Memory Address register are from the Memory Buffer register, and from the C register. The information from the memory buffer register is the address portion of the instruction word that has been read from memory, while the information from the C register is the address of the next sequential instruction as will be shown later in this thesis.

This register, as well as all registers in the computer, is conventional in design, and since innumerable registers of this type have been designed and are in use, it is felt that it is unnecessary to go into a sequential analysis of this logic element.

The Memory Buffer Register

The memory buffer register has been described as a parallel input, parallel output shift register that is also capable of shifting serially to the right during arithmetic operations. The design of this type of unit in a computer system is usually of a conventional nature. This unit may be designed intuitively from the specifications of the register.

Since there are 16 bits in the standard computer word, the memory buffer register must consist of 16 bistable multivibrators, or bistables. These bistables are capable of storing the two binary bits, 0 and 1. The logic that is shown in Figure 7 allows the computer and core control logic to read in bits of information to a cleared register by means of the read in pulses furnished by one of the two control units. Reading information into the memory buffer from the accumulator, is accomplished by clearing the memory buffer, and then pulsing the Read MB line. This will pulse the sampled one bits from the Accumulator into the Memory Buffer.

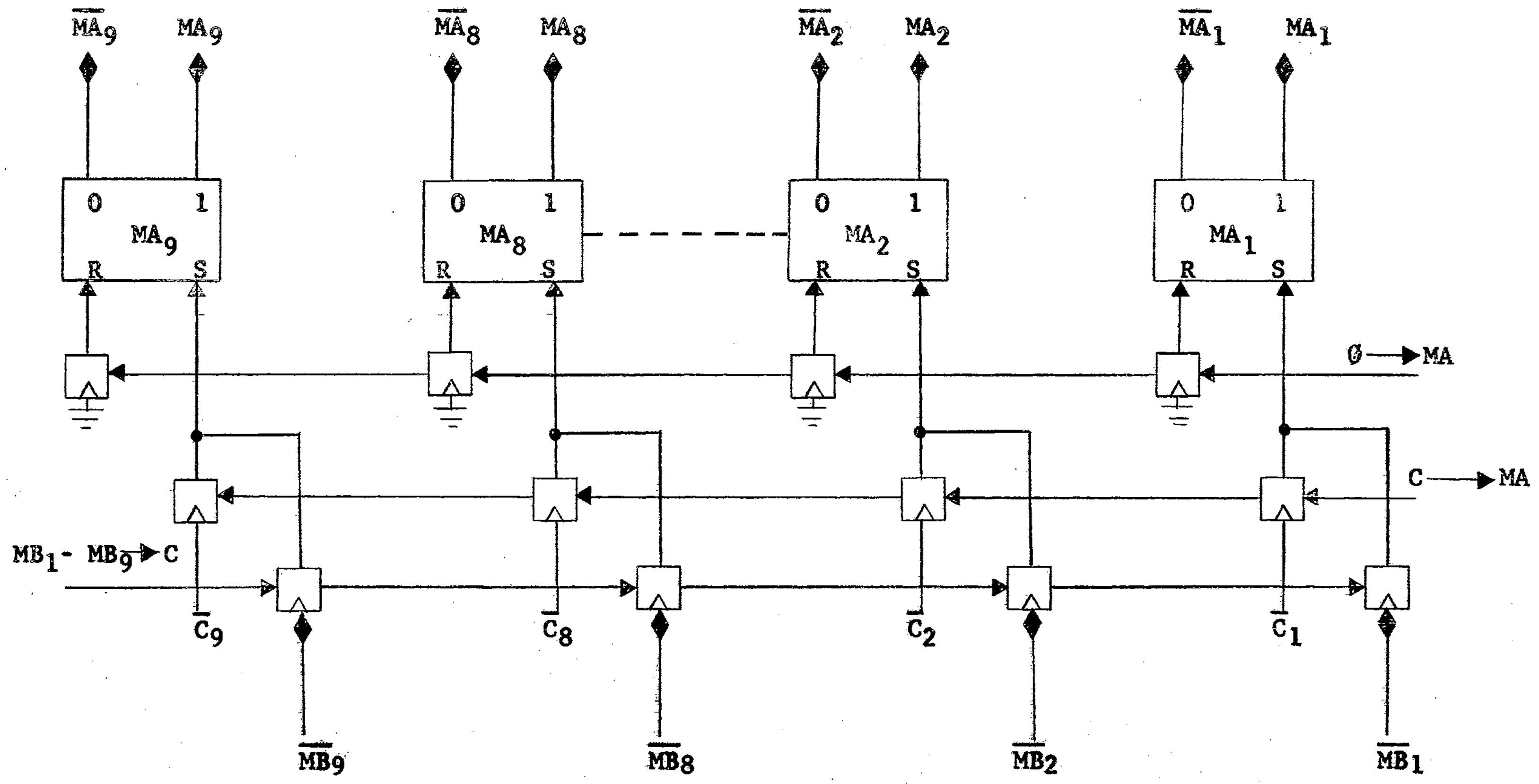


Figure 6. MA Register

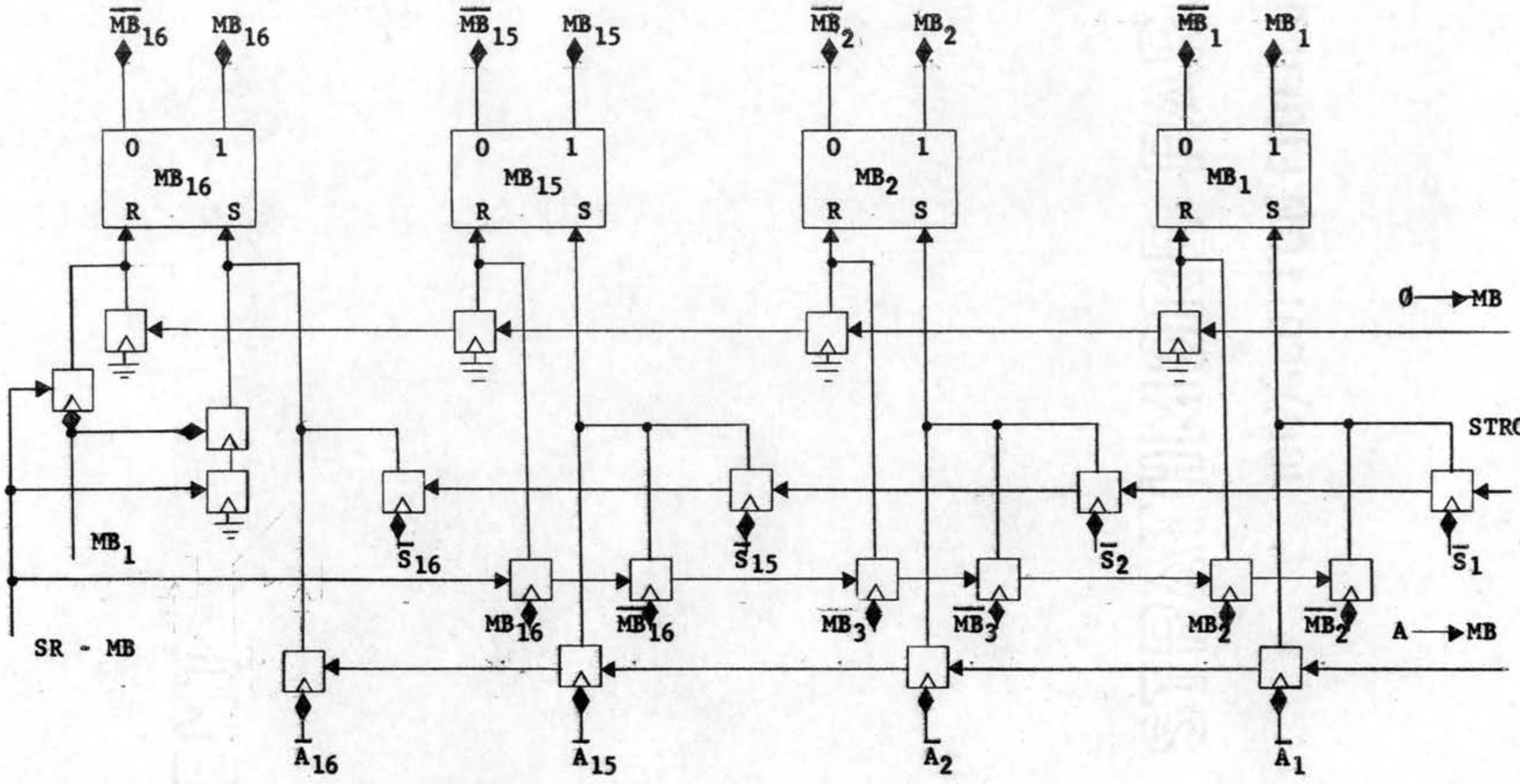


Figure 7. MB Register

Information is read from core into the Memory Buffer in a similar fashion. This may be accomplished by first clearing the Memory Buffer, and the pulsing the Strobe input line. This is part of the memory cycle, and the Strobe pulse of the memory cycle will strobe all ones sensed by the sense amplifiers into the bistables of the memory buffer. The zero outputs of the Memory Buffer are fed to the inhibit decoder logic of the memory control unit. This results in inhibiting the writing of ones into those cores during the write cycle. This Memory Buffer register is also capable of shifting the data stored in it to the right serially. This is accomplished by enabling or inhibiting the inverters connected to the set and reset inputs of the bistables.

The Address Decoders

The X and Y decoders decode a particular address stored in the MA register. For example, suppose the address 133_8 were stored in the MA register. In binary this is equal to 001011011. The 4 most significant bits would be decoded by the X decoder as X_2 and the 5 least significant bits would be decoded as Y_{27}. The X and Y decoders would direct the read and write currents of the read-write cycles to the X_6 and Y_{27} coordinates of the core memory. Figure 8 shows this method of addressing for the X coordinate.

The Control of the Memory Cycle

It requires approximately 2 micro-seconds (usec) to read a core to 0 or to write in a 1. The core memory read-write cycle outlined by Figure 9 allows approximately 2 usec for these operations.

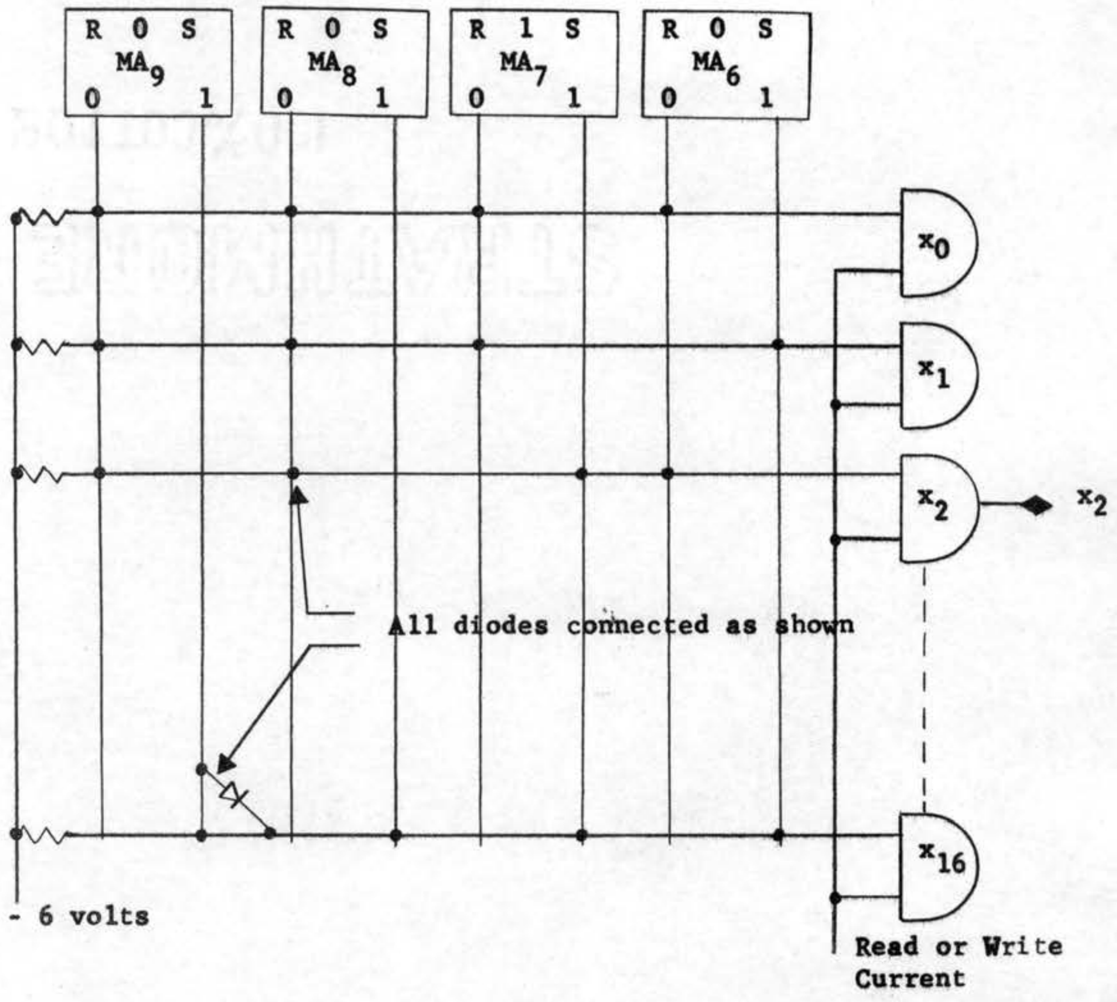


Figure 8. X Decoder

The start memory pulse must be furnished by the main computer control logic.. If a read cycle is to be initiated, a strobe enable level must be furnished to the memory control unit so that a strobe pulse may be gated to strobe information read from core into the memory buffer. The read time is started at approximately 2 μsec and will be completed at approximately 4 usec.[2] The information read from core must be strobed during the latter half of the read cycle to reduce the possibility that the noise induced in the sense windings by the half select cycles will have died down enough to have no effect upon the information read into the memory buffer. It has been found that this noise generated by the half select currents occurs only during the first few tenths of a usec of the read cycle. Therefore, if the sense amplifiers are strobed during the latter half of the read cycle, only the desired information will be read from core into the memory buffer. The effect of this noise generated by the half select cycles was observed experimentally by varying the strobe delay, and observing the errors occuring in the information being read from core.

The write cycle is always performed in two steps. First, all planes that are to be written 0 are inhibited by the information stored in the memory buffer through the inhibit decoders. These inhibit decoders apply $-H_c/2$ to all cores of the word to be written 0. This information is obtained from the memory buffer. All other cores of the addressed word will be written 1 through the action of the X, Y decoders and the write driver. The inhibit current starts at approximately 8.5 usec. This overlaps the write current which starts at approximately 6 usec and continues until approximately 8 usec. This overlap is required to insure that no cores to be written 0 will be written 1. The memory control logic will furnish a memory complete

[2]Charles V. L. Smith, <u>Electronic Digital Computers</u>, New York, 1959, p. 295.

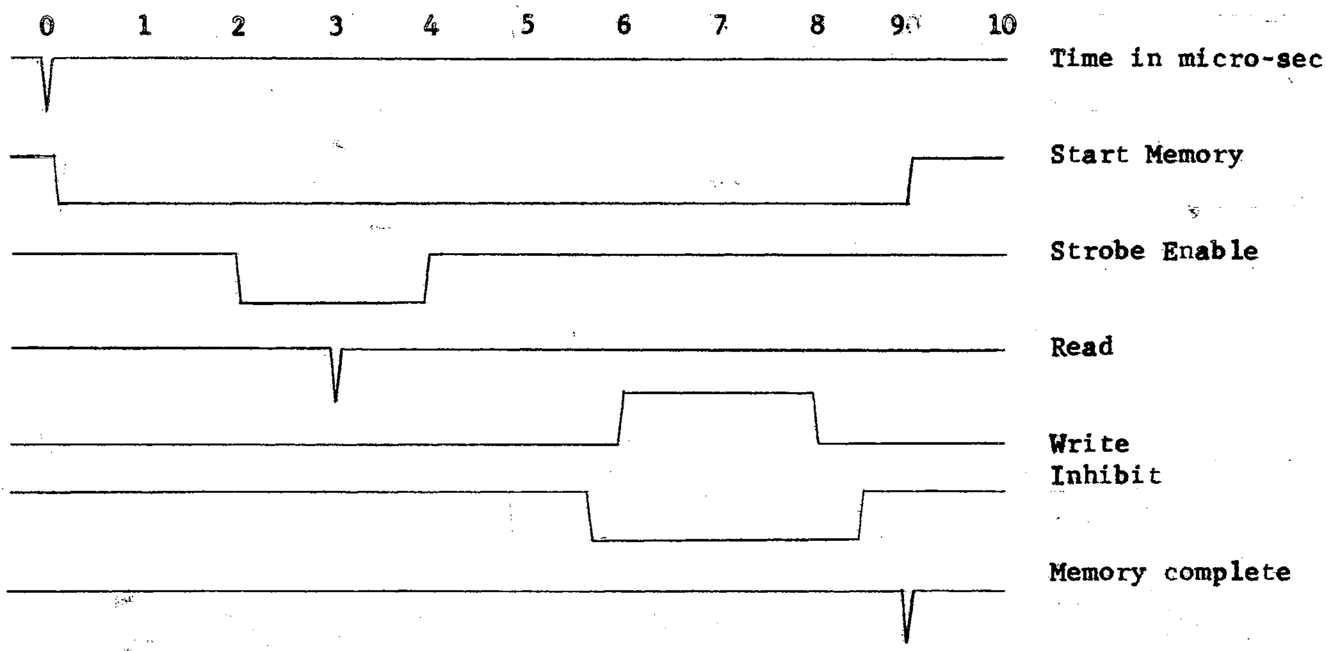


Figure 9. Typical Core Memory Read-Write Cycle

pulse to the main computer control at 9 usec after the initiation of the memory cycle.

All of the units that are required to control the timing of the read-write cycle are shown in Figure 10. The delay units may be constructed from one-shot multivibrators. The logic elements that produce the read, write, and inhibit control levels may also be constructed from one-shot multivibrators. These control levels are furnished to the read-write drivers and to the inhibit decoders of the memory control logic.

This complete system will be capable of storing information in the core memory array, and the information can be written into or read from the memory buffer depending upon the instructions from the main computer control. These instructions will be outlined in the following chapters of this thesis.

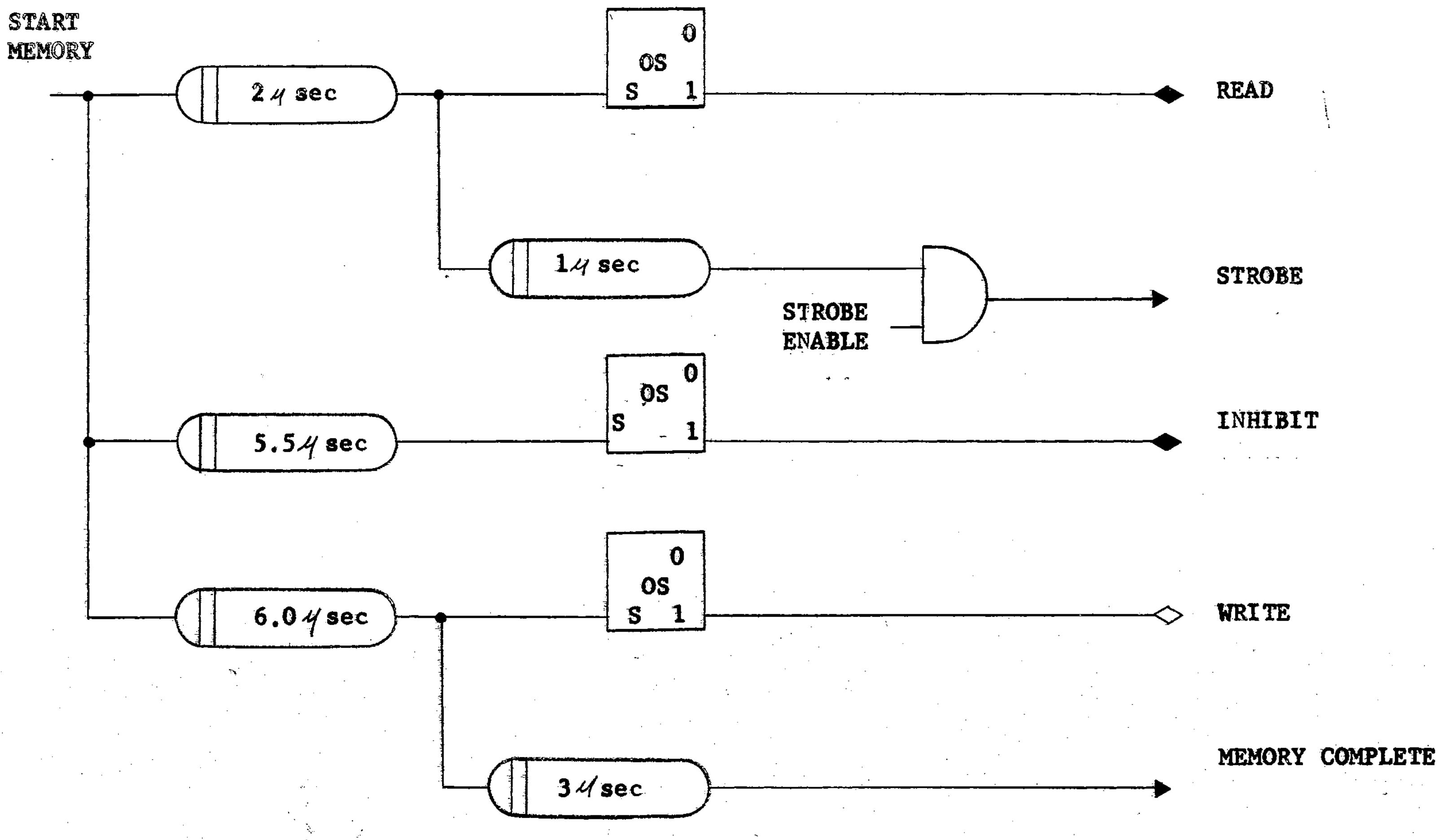


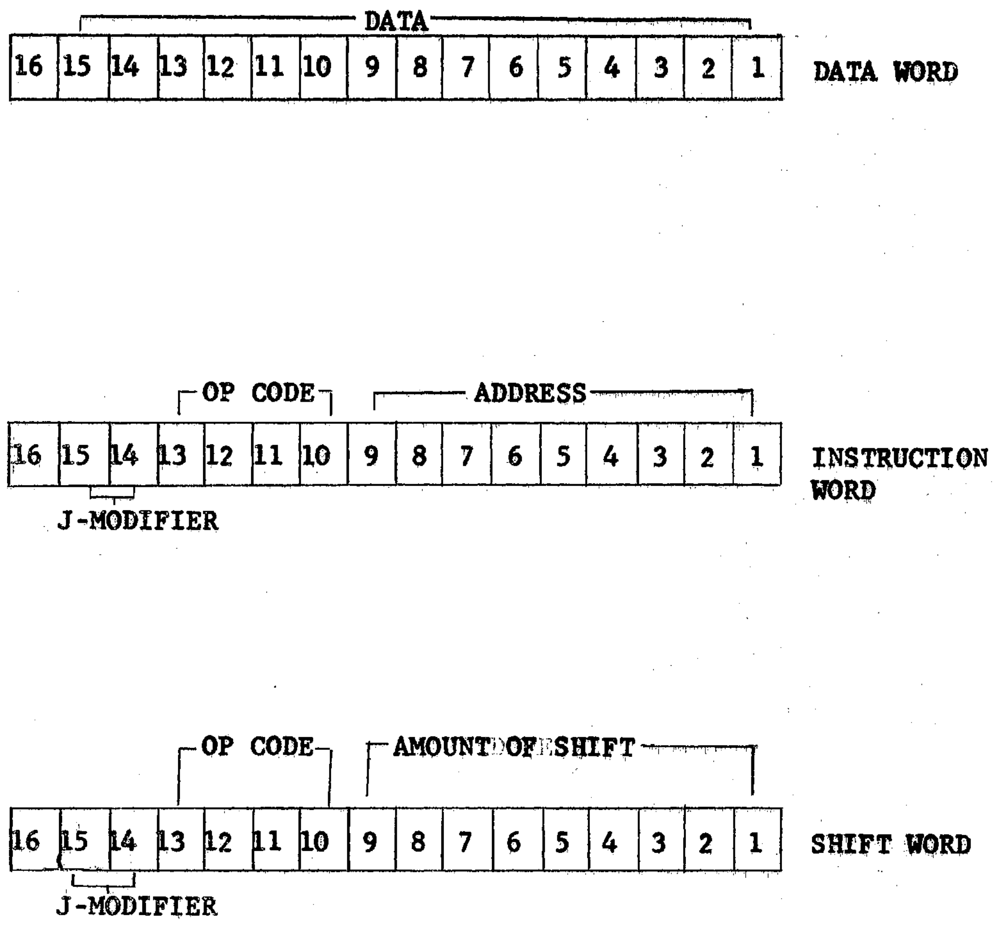
Figure 10. Memory Control Unit

CHAPTER III

MECHANIZATION OF THE COMPUTER

The basic capabilities of this computer were outlined in Chapter I. Since computer time will not be considered as a prime factor in this design, a serial binary arithmetic unit will be designed. Laboratory modules available for testing this design are capable of operating at a design frequency of 500 kilocycles. Therefore the computer clock will operate at 500 kilocycles. The standard pulses used with this design are also dictated by the test modules used. These standard pulses will be -2.5 volt, 0.4 usec pulses.

The standard computer word has been determined to be 16 bits long. A diagram of the computer word is shown in Figure 11. The data word consists of 15 bits plus a sign bit, while the instruction word consists of a 2 bit modifier, an operations code (OP CODE), and the address of the information to be operated on. The basic shift instruction word is essentially the same as any instruction word, except the address portion does not refer to an address, but to the amount of shift.



Bit 16 = Sign Bit

0 = + 1 = -

Figure 11. Computer Word Format

Selected Instructions

This computer will be capable of performing the following list of instructions. These instructions are described fully in the listing, and no description of them will be needed here. The instructions are coded by the 6 most significant bits of the instruction word. Bits 15 and 14 list the J-Modifier of each instruction, while bits 13, 12, 11, and 10 define the particular operation in a code grouping. For example, the subtract instruction has the OP CODE 21. This may be expressed in binary as 010001. The 2 most significant bits 01 represent the J-Modifier 1 and the remaining bits represent the Code 01. These will be decoded by the computer logic, and will control the operation that the computer will perform. In this example, the decoded command will tell the computer to subtract.

Computer Instructions

INSTRUCTION		J-MODIFIER	CODE	EXPLANATION
JAN	00	0	00	Test the sign bit of the accumulator, if it is negative, jump to the address indicated, if it is positive, proceed to the next sequential instruction.
JOF	20	1	00	Test the overflow indicator, if it is on, clear the overflow indicator and jump to the address indicated, if it is off, proceed sequentially.
JMP	40	2	00	Unconditionally jump to the address indicated.
HLT	60	3	00	Halt the computer. The MA register contains the address of the next instruction.
ADD	01	0	01	Add the contents of the memory address indicated to the accumulator. The algebraic sum will be stored in the accumulator.

INSTRUCTION		J-MODIFIER	CODE	EXPLANATION
SUB	21	1	01	Subtract the contents of the memory address indicated from the accumulator. The algebraic difference will be stored in the accumulator.
CLA	41	2	01	Clear the accumulator and add the contents of the address indicated to the accumulator. The sum will be stored in the accumulator.
CLR	61	3	01	Clear the accumulator to zero.
INC	42	2	02	Add one to the contents of the address indicated. The sum will be stored in the accumulator and in the indicated address.
DEC	62	3	02	Subtract one from the contents of the address indicated. The difference will be stored in the accumulator and in the indicated address.
ARS	03	0	03	Shift the contents of the accumulator to the right. The amount of the shift is indicated by sixteen minus the contents of the address portion of the instruction. For example: 0.03002 would be interpreted as a shift right instruction and 16-2 indicates a shift of 14 bits.
LRS	23	1	03	Shift the contents of the accumulator and B registers to the right. The two registers act as one 31 bit register. The amount of shift is indicated by sixteen minus the contents of the address portion of the instruction.
ALS	43	2	03	Shift the contents of the accumulator to the left. The amount of the shift is indicated by sixteen minus the contents of the address portion of the instruction.
LLS	63	3	03	Shift the contents of the accumulator and B registers to the left. The two registers act as one 31 bit register. The amount of shift is indicated by sixteen minus the contents of the address portion of the instruction.

INSTRUCTION		J-MODIFIER	CODE	EXPLANATION
MUL	04	0	04	The multiply instruction will multiply positive magnitude numbers. The multiplier must first be loaded into the B register by the programmer. The multiplicand may be only 14 bits in length. The product of B times the contents of the memory address indicated is formed in the accumulator and the B register. The most significant part of the product will appear in the accumulator, and the least significant part will appear in the B register. The programmer must handle sign control.
DIV	24	1	04	The divide instruction will divide positive magnitude numbers. The dividend must first be loaded into the B register by the programmer. The divisor is located in the address indicated by the divide instruction. The quotient will be formed in the B register, and the remainder will appear in the accumulator. The programmer must handle sign control, and division by zero is prohibited.
CP1	44	2	04	The ones complement of the accumulator is formed and stored in the accumulator.
CP2	64	3	04	The twos complement of the accumulator is formed and stored in the accumulator.
PNT	05	0	05	The contents of the memory address indicated will be printed.
STO	25	1	05	Store the contents of the accumulator in the address indicated.
CMP	06	0	06	Compares the contents of the memory address indicated with the accumulator. If A is greater than M, the HI indicator is turned on. If A is equal to M, the EQ indicator is turned on.
JHI	26	1	06	Test the HI indicator, if it is on, jump to the address indicated. If it is off, proceed sequentially.

INSTRUCTION		J-MODIFIER	CODE	EXPLANATION
JLO	46	2	06	Test the LO indicator, if it is on, jump to the address indicated. if it is off, proceed sequentially.
JEQ	66	3	06	Test the EQ indicator, if it is on, jump to the address indicated. If it is off, proceed sequentially.

Basic Instruction Cycle

The basic instruction cycle can be divided into 3 parts. These are selection, decoding, and execution of the instruction. A complete listing of all these functions must be made for each of the instructions, before the logical design of the computer may proceed. Since the addition or subtraction of 16 bit numbers in a serial arithmetic unit will require 16 timing pulses, the execution cycle for these arithmetic instructions will require 16 timing pulses. The selection and decoding must occur before the execution cycle and, it was discovered that these cycles could both be performed easily for all instructions in a total of 16 timing pulses. The arithmetic operations of multiply and divide also require a control cycle. For purposes of simplicity in design, this third cycle also consists of 16 timing pulses. Therefore, the complete instruction cycle will consist of 3 phases of 16 timing or clock pulses in each phase. A complete analysis of how the steps in each phase were determined is given in the following paragraphs.

The first part of the instruction cycle consists of selection of the instruction. This will be accomplished by manually setting the MA register to the address of the first instruction of the program. Then as will be explained later in the section on the control unit, the computer will be started on its cycle of operation. The tables for each phase and timing pulse will be explained in detail.

TABLE I

PHASE 1 MECHANIZATION

TIME	OPERATIONS
T_0	$\emptyset \rightarrow$ MB, C, D
T_1	(START MEMORY) (STROBE ENABLE), MA $\rightarrow$ C
T_2	$\emptyset \rightarrow$ MA, COUNT C
T_3	$M_1 - M_9 \rightarrow$ MA, $M_{10} - M_{15} \rightarrow$ D
T_4	JUMP = $00(J_0A_{15} + J_1OF + J_2 + J_3) + 06(J_1HI + J_3EQ + J_2LO)$,
	$\emptyset \rightarrow$ OF = $00J_1OF$
T_5	$\emptyset \rightarrow$ (COUNT)(CARRY)(CO), $\emptyset \rightarrow$ HILO = $06J_0$
T_6	$\emptyset \rightarrow$ (TIME)(PHASE)(JUMP) = JUMP, $00J_3$ = ORDER, $00J_3$ = STOP
T_7	$\emptyset \rightarrow$ MB, CO = $03 + 04(J_0 + J_1) + 06$
T_8	(START MEMORY)(STROBE ENABLE) = $01 + 02 + 04 + 05J_0 + 06J_0$
T_9	$\emptyset \rightarrow$ A = $(01 + 02)(J_2 + J_3)$, A $\rightarrow$ MB = $05J_1$
	COMP A = $04(J_2 + J_3)$
T_{10}	PRINT = $05J_0$, START MEMORY = $05J_1$, MB $\rightarrow$ A = 02
T_{11}	JUMP = $01J_3 + 04J_2 + 05 + 00 + 06\bar{J}_0$
T_{12}	$\emptyset \rightarrow$ MA = JUMP
T_{13}	C $\rightarrow$ MA = JUMP
T_{14}	PRESET COUNT = 03, $\emptyset \rightarrow$ OF = $01 + 02 + 04 + 06J_0$, JUMP
T_{15}	$\emptyset \rightarrow$ (PHASE) = $01J_3 + 04J_2 + 05 + 00 + 06\bar{J}_0$, $\emptyset \rightarrow$ JUMP = JUMP

At time T_0 in TABLE I, the three registers MB, C, and D are cleared to 0. This is done in preparation for reading the instruction word out of memory into the MB register. C is cleared in preparation for storing the contents of the MA register. D is cleared so that the

OP CODE of the instruction word can be transferred and stored there. At T_1, the start memory pulse should be furnished to the memory control unit. At the same time, a strobe enable level should be furnished to the memory control unit insuring that this will be a read cycle. This will read the instruction word out of core into the memory buffer. Also at this time the contents of the MA register will be transferred into the C register. At T_2 the MA register is cleared in preparation for the address portion of the instruction word. The C register is now holding the address of the instruction being executed, and this register will now be increased in magnitude by 1 which will be the address of the next sequential instruction. At T_3 the OP CODE will be transferred to the D register where it may be decoded and the address portion of the word will be transferred to the MA register where it is decoded. This completes the selection and decoding cycles of all instructions, and the remainder of the operations will deal with the execution cycle. At this time, the D register holds the OP CODE, the MA register holds the address or quantity that the instruction will work with, and the C register holds the address of the next instruction.

At time T_4 the execution of the Jump instructions is initiated. A bistable that is described as the JUMP flip-flop will be set if any of the jump instructions have been decoded, and if the proper conditions have been met. For example, the JUMP flip-flop will be set if there is a JAN instruction and the accumulator is negative, or if there is a JOF instruction and the overflow indicator is on, or if there is an unconditional jump (JMP) or a halt (HLT) instruction, or if there is a compare test instruction such as $06J_1$HI, $06J_2$LO, or $06J_3$EQ. The overflow indicator will be cleared if this is a JOF instruction and the overflow indicator is on. At T_5 the Operations Control Counter, the carry indicator, and CO indicators will be cleared in preparation for arithmetic operations. The HILO indicator will clear if this is a compare instruction. These indicators will usually be bistable multivibrators. At T_6, it is possible that the execution of some of the instructions may be complete. If the JUMP flip-flop was set at T_4, the timing counter, the phase counter and the Jump flip-flop will be cleared and the instruction cycle will be complete. The timing cycle will start over again at T_0, and the memory address register is holding the address of the next instruction to be executed. If this instruction were a halt instruction, the computer will halt and the MA register will be holding the address of the next instruction if the operator wishes to start the computer again.

If the computer did not jump out of phase, the execution cycle will continue for the remainder of the instructions. At T_7 the MB register will be cleared in preparation for data from memory if this is an arithmetic instruction, or in preparation for data from the A register if this is a store instruction. Also at this time, the CO or Shift control flip-flop will be set if this is a shift multiply, divide, or compare instruction. At T_8, there will be a read cycle if this is an arithmetic, output, or a compare instruction. This brings the data to be operated on out of core. At T_9, the A register will be cleared if this is an increment or decrement memory instruction, or a clear or a clear and add instruction. The A register is transferred into the MB register if this is a store instruction. Also, the A register will be complemented if this is a complement instruction. At T_{10}, the output printer will be signaled to print the contents of the memory buffer if this is a print instruction, or a write cycle will be initiated if this is a store instruction. Also, at T_{10}, the MB register will be transferred into the A register if this is an increment or decrement instruction. At T_{11}, a number of the instructions could be completed. The instruction has been executed and the JUMP flip-flop should be set if the instruction were a Jump instruction and the computer did not jump out of phase previously, or if this were a CLA instruction, a CP1 instruction, a PNT or a STO instruction. If the computer is to jump out of phase at this time, the address of the next sequential instruction that is stored in the C register must be transferred to the MA register. At T_{12}, the MA register will be cleared if the JUMP flip-flop has been set. At T_{13}, the contents of the C register will be read into the MA register if the JUMP flip-flop is set. At T_{14}, the shift control counter will be preset if this is a shift instruction, or the Overflow flip-flop will be cleared if this is an arithmetic operation. Also at T_{14}, the JUMP flip-flop will be set. At T_{15} the phase counter will be cleared for the conditions listed and the JUMP flip-flop will be cleared. At this time, all of the instructions other than the arithmetic instructions, the SHF instruction and CP2 have been completed. Therefore, for these completed instructions the computer will jump phase and start at T_0 of the next instruction indicated by the contents of the MA register. For the uncompleted instructions, the computer will continue its instruction cycle on into P_2.

P_2 is concerned with arithmetic operations through the computers arithmetic unit, and with shift operations. P_2 is also made up of 16 time pulses. These are required to complete the serial addition or subtraction of the 16 bit binary words, or to shift the 16 bit registers left or right. The operations in P_2 will be analyzed as were those in P_1.

TABLE II

PHASE 2 MECHANIZATION

TIME	OPERATIONS
T_0	$M\text{-}A \longrightarrow A = 06CC$, $MB \longrightarrow MB$,
	$A \pm MB \longrightarrow A = 01\bar{J}_3 + 04(J_0 + J_1) + 06\overline{CD}$
	$A \pm 1 \longrightarrow A = 02 + 04J_3$, BEGIN SHIFT = 03
T_1	
T_2	
T_3	Shift Stops When Complete
T_4	
T_5	
T_6	
T_7	
T_8	
T_9	
T_{10}	
T_{11}	
T_{12}	
T_{13}	
T_{14}	JUMP
T_{15}	$\emptyset \longrightarrow$ JUMP, OF = (OVERFLOW)(01 + 02 + 04 + 06)

Although the outline of operations that must be performed in P_2 is rather simple, there are a number of very important features of the logical design listed. The Arithmetic Unit must be capable of performing the various modifications of addition and subtraction listed. Since this computer will have the capabilities of addition, subtraction, multiplication, and division built into it, the logic must take care of these different possibilities, as well as the features of complementing, comparison, and incrementing and decrementing memory. All of these features are simple variations of addition and subtraction. The Shift command will also be executed in P_2. The steps for the execution of this instruction are listed in the P_2 mechanization chart.

The design of the logic for each of the steps listed will be discussed in Chapter IV. Therefore, this discussion will be limited only to what should happen at a particular time but not why.

At time T_0, all of the simple arithmetic operations begin. The accumulator will be added to the memory buffer register in serial form, and the sum will be stored in the accumulator. This operation should occur for the instructions ADD, CLA, MUL, and in some cases, DIV, or CMP. For the instructions SUB, and sometimes DIV and CMP, the contents of the memory buffer register will be subtracted from the accumulator, and the difference will be stored in the accumulator. There are cases during the execution of the CMP instruction that the accumulator will be subtracted from the memory buffer register and the difference will be stored in the accumulator. If the instruction being executed is an INC or a CP2 instruction, 1 will be added to the accumulator, and the sum will be stored in the accumulator for the CP2 instruction, and the sum will be stored both in the accumulator and memory for the INC instruction. If the instruction being executed is the DEC instruction, 1 will be subtracted from the accumulator, and the difference will be stored both in the accumulator and in memory. All of the operations just described require 16 clock pulses to complete, and will be completed at the end of P_2. The SHF instruction also begins at T_0 of P_2.

The count preset in the Operations Control Counter in P_1, will control the completion of the shift operation. The design of this feature will be discussed in Chapter IV. The JUMP flip-flop is always set at T_{14} regardless of the Phase of operation. This step is necessary in the control of the timing of the computer as will be described in Chapter IV. The MUL and DIV instructions require the computer to step through P_2 16 times. There are 16 bits in the multiplier, and each one must be tested to form a partial product. This will require 16 partial products to be formed, requiring 16 trips through P_2. The DIV instruction requires division by 16 bit binary numbers. Therefore, there will be 16 trial divisions, or 16 trips through P_2 by the computer logic in order to determine the proper quotient. These trips through P_2 will be accounted for by advancing the count stored in the Operations Control Counter. This increase in the count occurs at time T_{14} in P_2. The CMP instruction may require as many as three trips through P_2, and these trips will be accounted for by the Operations Control Counter. These trips will be accounted for by increasing the count stored in the Operations Control Counter at T_{14} in P_2. The JUMP flip-flop must be cleared at T_{15} of P_2. Also at this time, the Overflow control flip-flop will be set if there is an arithmetic overflow during any of the arithmetic operations in P_2. This completes the mechanization of all instructions in P_2.

During P_3, the computer logic must decide whether or not to repeat P_2. If the execution of the instruction is not complete, the control logic will cause a repeat trip through P_2. If the execution of the instruction is complete, the control logic must set up the address of the next instruction and proceed with the program being executed.

TABLE III

PHASE 3 MECHANIZATION

TIME	OPERATIONS
T_0	$\emptyset \longrightarrow MB = 02$, Set $B_1 = 04J_1\bar{A}_{16}$
	$LO = \tau_0 A_{16} 06$, $HI = \tau_2 A_{16} 06$
T_1	$LRS_1 = 04J_0\overline{COUNT}_{15}$, $LLS_1 = 04J_1\overline{COUNT}_{15}$, $A \longrightarrow MB = 02$
	$JUMP = 04(J_0 + J_1)(\overline{COUNT}_{15}) + 06\tau_0\bar{A}_{16} + 06\tau_1$
T_2	Phase $2 = 04(J_0 + J_1)(\overline{COUNT}_{15}) + 06\tau_0\bar{A}_{16} + 06\tau_1$
	$\emptyset \longrightarrow JUMP = JUMP$, ADVANCE COUNT $= 04(J_0 + J_1) + 06$
T_3	START MEMORY = 02
T_4	$\emptyset \longrightarrow MA$
T_5	$C \longrightarrow MA$
T_6	
T_7	
T_8	
T_9	
T_{10}	
T_{11}	
T_{12}	
T_{13}	
T_{14}	JUMP
T_{15}	$\emptyset \longrightarrow$ (PHASE)(JUMP)

At time T_0, the word from memory that has been incremented or decremented by 1 will be transferred to the memory buffer register, so that this new word can be stored back in memory. If this is a DIV instruction, the least significant bit of the B register will be set 1 if the divisor "goes" into the dividend. If this is a CMP instruction, and the count in the Operations Control Counter is equal to 0000, and the sign bit of the accumulator is negative, the LO indicator will be set. If this is a CMP instruction and the count in the Operations Control Counter is equal to 0010, and the sign bit of the accumulator is negative, the HI indicator will be set. At T_1, the accumulator and B registers will be shifted 1 bit to the right if this is a MUL instruction and 16 partial products have not been formed. At T_1, the accumulator and B registers will be shifted 1 bit to the left if this is a DIV instruction and 16 trial divisions have not been made. The JUMP flip-flop will also be set if MUL or DIV is not complete, or if the execution of CMP is not complete. At T_2, the control logic will jump back to P_2 if the execution of the MUL, DIV, or CMP instructions is not complete. The JUMP flip-flop will be cleared if it has been set. At T_3, the incremented or decremented word will be written into memory if the instruction being executed is an INC or DEC instruction. At T_4, the MA register must be cleared in preparation to receive the address of the next sequential instruction that has been stored in the C register since P_1. At T_5, the contents of the C register will be transferred in parallel into the MA register. If the master clock reaches this point in P_3, the execution of the instruction is complete, and the control logic will mark time until the end of this Phase. At T_{14}, the JUMP flip-flop will be set. At T_{15}, the control logic will clear the Phase Counter, and the computer will start out at P_1T_0 of the next instruction of the program being executed. This completes the mechanization of all instructions of this computer. The logical design of all the timing, control, and arithmetic operations will be discussed in Chapter IV.

CHAPTER IV

LOGICAL DESIGN OF THE COMPUTER

The logical design of the remainder of the computer, will be divided into three major groups. These groups are the Timing Unit, the Control Unit, and the Arithmetic Unit. The guide in designing these units will be the mechanization charts of the three phases. In most cases, the units will be designed intuitively from the mechanization charts, and in some cases, classical boolean algebra methods with Karnaugh map simplifications will be used.

The Timing Unit

The logic used in this design will be conventional And/Nand, and Or/Nor logic. The specific gating structure used will be described in the Appendix. The inverter symbols used will also be described in the Appendix. The bistable multivibrators used require standard - 2.5 volt, 0.4 usec pulses to set, reset, and to complement the flip-flops. Each flip-flop also has a negative standard pulse output if the flip-flop is triggered at the complement input. This pulse will be capable of driving the base of only one transistor. The 1 and 0 level outputs of these flip-flops will be capable of driving 14 base loads. The And/Nand and Or/Nor modules will be capable of driving 8 units of base load. Each inverter module will be capable of driving 8 units of base load. The master clock, which is an astable multivibrator, is capable of driving 8 base loads. These loading restrictions must be followed.

The master clock will be the source of all pulses used in the logic of this computer. The control of the master clock is shown in Figure 12. There are four basic modes of operation for this computer. These modes are the START, ORDER, CYCLE, and PULSE modes. The START mode is used to start the computer on a normal cycle of operation. If this mode is used, pressing the start switch will cause the computer logic to start on the instruction cycle of the program to be executed. The computer will continue on this cycle of operation until a programmed halt, or until an invalid operation occurs. If the ORDER mode is used, the computer logic will cause the execution of an instruction one basic phase at a time, which will be explained later in this chapter. If the CYCLE mode is used, the same instruction will be repeated until the halt switch is operated. If the PULSE mode is used, the computer logic will step through the instruction to be executed one pulse at a time. Using this information and the mechanization charts, the design of the Master Timing Control, Figure 12, may be explained.

The rectangular blocks in Figure 12 labeled P/S, O/S, S/S, C/C, Clear, and Clear OF are pulse generators that are operated by mechanical switches. These switches will be the operator's controls for this computer. A conventional Schmitt circuit provides a standard - 2.5 volt, 0.4 usec pulse each time the mechanical switch is depressed.

Only one of the four modes of operation will be used at any one time. The START, ORDER, and CYCLE modes use the master clock as a source of operating pulses. Therefore, in each of these modes, the

START control will be set. The PULSE mode, however, uses a pulse generator, as a source of operating pulses. Therefore, during the PULSE mode of operation, the master clock must be turned off. During the memory cycles listed at times P_1T_1, P_1T_8, P_1T_{10}, and P_3T_3 the master clock must be turned off until the completion of the memory cycle. This explains the $\overline{PB}$ level controlling the clock pulses. This $\overline{PB}$ level is furnished by the memory control logic, and will be explained in the section on Memory Control. This explains the need for the $\overline{PULSE}$, START, and $\overline{PB}$ control levels, and the P/S input to the clock control furnishes the single pulses for single pulse operation. The master clock pulses are labeled C_p and the Timing Counter will be controlled by delayed clock pulses labeled $\triangle C_p$. The variable delay $\triangle A$ may be adjusted so that the proper time relationship between the clock pulses and the timing levels generated by the Timing Counter will be attained. This relationship is shown in Figure 13.

The Master Timing Control logic may now be easily explained. All controls of the computer should be cleared initially before the start of the execution of any instruction. If the Clear pulse generator is energized, a clear pulse will be gated to all of the controls that should be cleared at this time. In Figure 12, these controls are START, PULSE, ORDER, and CYCLE. Other controls such as the PB control and others will also be cleared at this time, their logic will be analyzed in conjunction with an explanation of the design of these controls. If S/S, C/C, P/S, or O/S is energized, the START control will be set. Since PULSE and PB were cleared, clock pulses and delayed clock pulses will be generated. This will start the computer on an instruction cycle and the instructions will be executed as outlined in the mechanization charts. It will be best

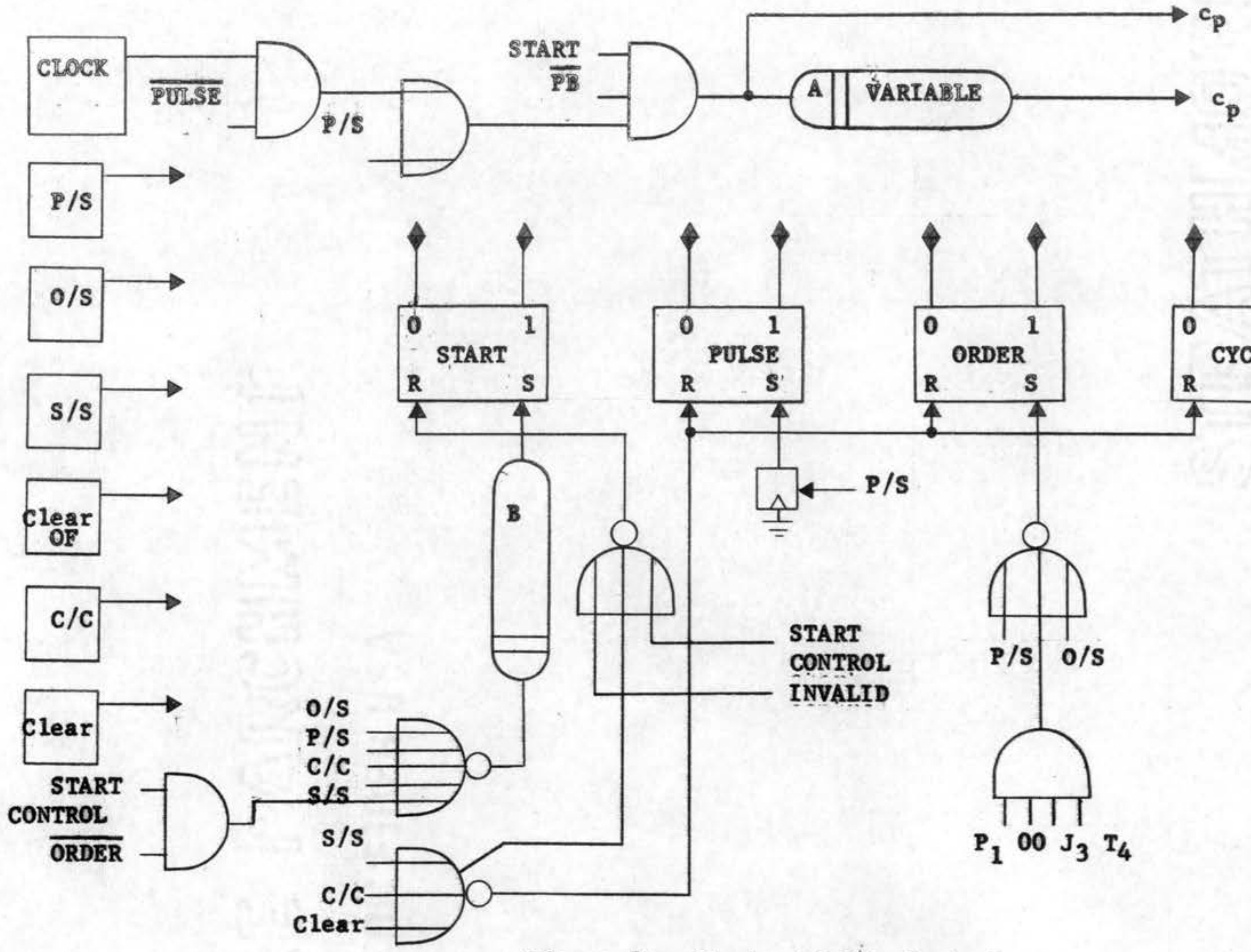
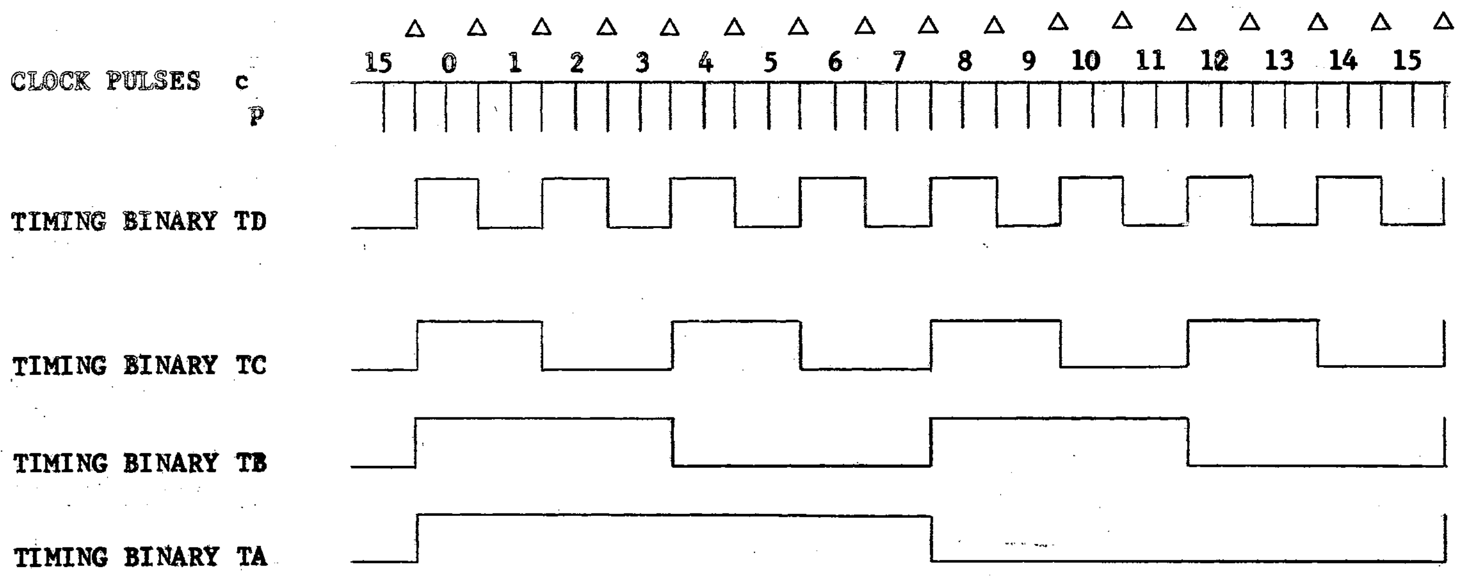


Figure 12. Master Timing Control

at this time to assume that the computer has been started by the S/S control in the start mode. All other modes are variations of the start mode, and will be explained in the section on Variable Operations. S/S will clear the four controls in Figure 12 and after a delay of 6 usec, the START control will be set allowing clock pulses and delayed clock pulse to be generated. An explanation of stopping the computer and an instruction cycle will be given after an analysis of the Timing Counter, Timing Decoder, and the JUMP controls have been made.

The Timing Counter

The Timing Counter is of conventional design, and uses gated flip-flops requiring standard positive pulses to set, reset, and to complement the flip-flops. This counter is driven by the delayed clock pulses so that the time relationship shown in Figure 13 may be attained. Since there are 16 clock pulses in each major phase of operation, the Timing Counter must be capable of counting 16 clock pulses. Thus four flip-flops are required. Some provision must be made to advance the phase from P_1 to P_2 or from P_2 to P_3 if required. After 16 clock pulses have been counted in a particular phase, an Advance Phase pulse will be generated and fed to the Phase Counter shown in Figure 16. The phase will be advanced if another phase of operation is required. This Timing Counter will clear when the Clear pulse generator is energized on an initial start of the computer. According to the mechanization chart, the Timing Counter will be cleared at P_1T_6 if the JUMP control has been set. Also, the mechanization chart indicates that the Timing Counter should be cleared if Phase 2 exists at P_3T_2. The Timing Counter always clears at T_{15}.



6 USEC DELAY BETWEEN PHASES

Figure 13. Typical 1 Phase Timing Chart

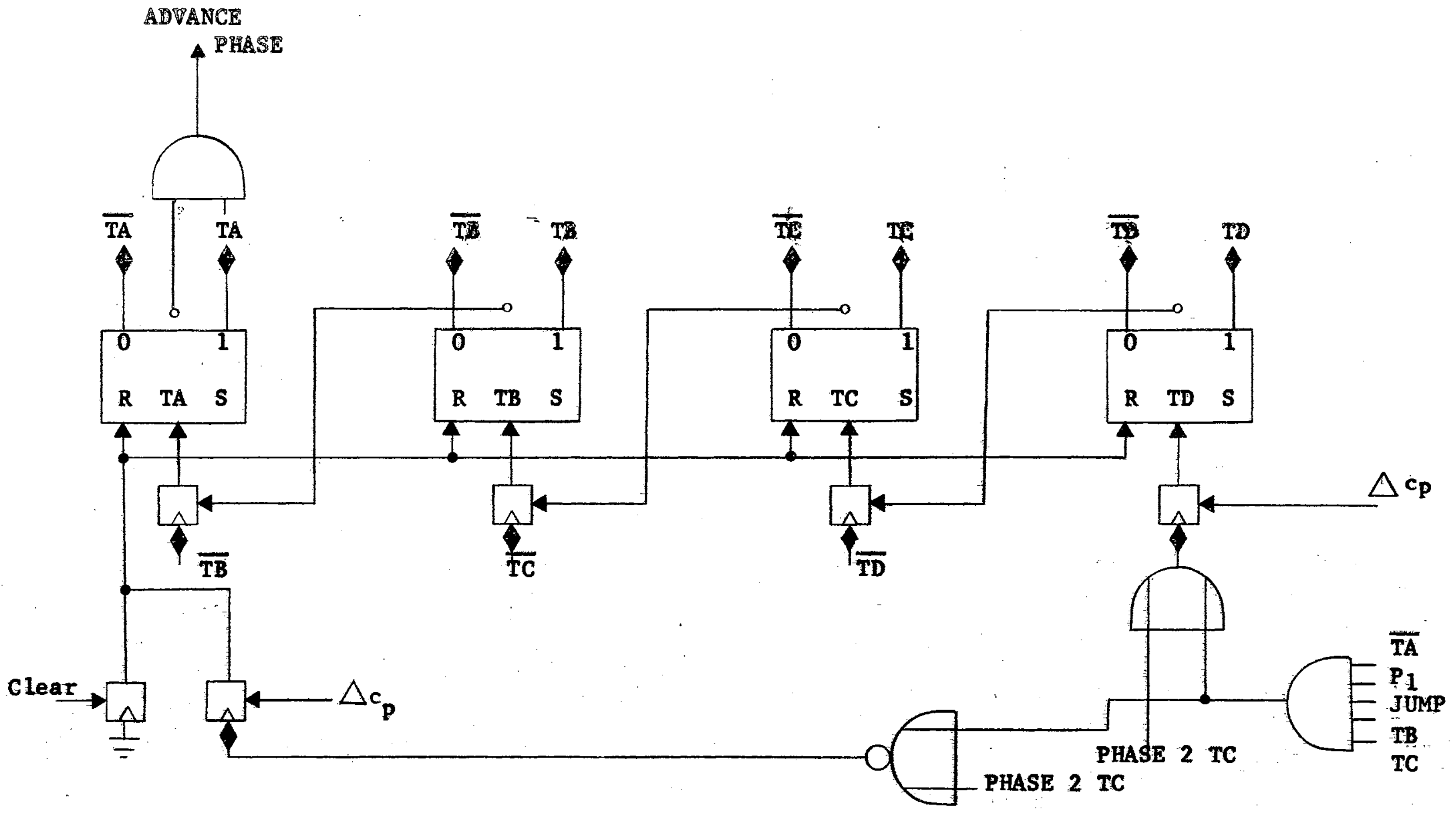


Figure 14. The Timing Counter

The Timing Decoder

The Timing Decoder is of conventional design, using And Gate gating. A decoding matrix could be used, but since unlimited And Gate modules were available for test purposes, the design used in Figure 15 was satisfactory. In any phase, the first clock pulse will be T_0. Since the Timing Counter will start out cleared, the $\overline{TA}$, $\overline{TB}$, $\overline{TC}$, and $\overline{TD}$ levels will all be logic 1. These levels will be used to gate T_0. $\triangle T_0$ will advance the count in the Timing Counter to 0001 and $\overline{TA}$, $\overline{TB}$, $\overline{TC}$, and TD will be at a logic 1 level gating T_1. This process continues in sequence until all clock pulses of a phase are gated. Then the process starts over again.

Phase Control

Since there are three phases of operation, there must be some method for keeping track of the phase. The simplest way to do this is with a Phase Counter. The three phases dictate that there must be two flip-flops in the Phase Counter. The mechanization charts outline the control of the Phase Counter. Initially, the Phase Counter will be cleared by a Clear pulse from the Clear pulse generator. In P_1, the mechanization chart requires that the Phase Counter be reset at P_1T_6 if the JUMP flip-flop has been set. Since the Phase Counter is already reset at this time, no logic will be required to accomplish this. At P_1T_{15}, the mechanization chart requires that the Phase Counter not be advanced if the instruction being executed is a CLR, CP1, PNT, STO, JAN, JOF, or a CMP instruction. Therefore, the Phase Counter is not allowed to advance at P_1T_{15}, if these conditions are

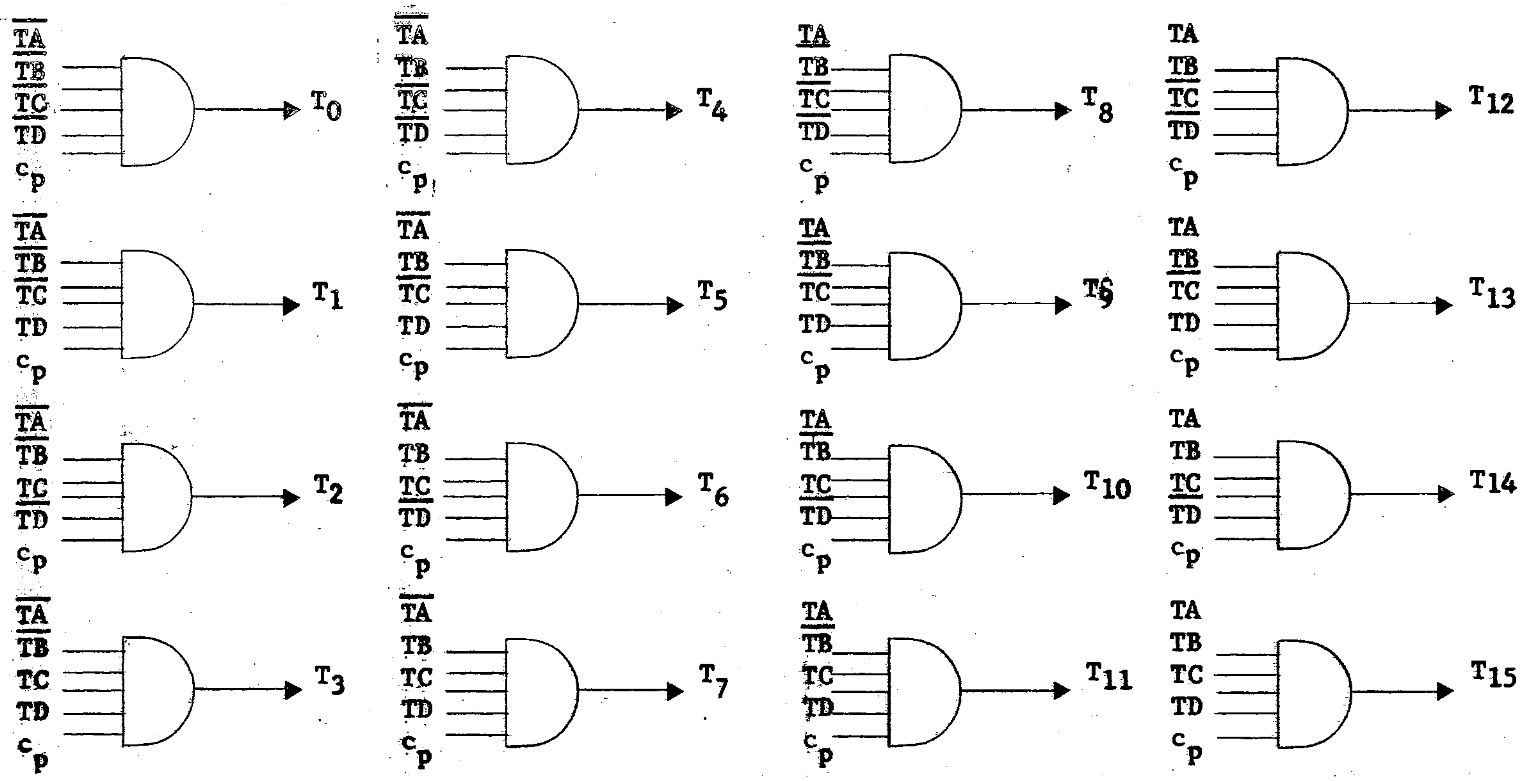


Figure 15. The Timing Decoder

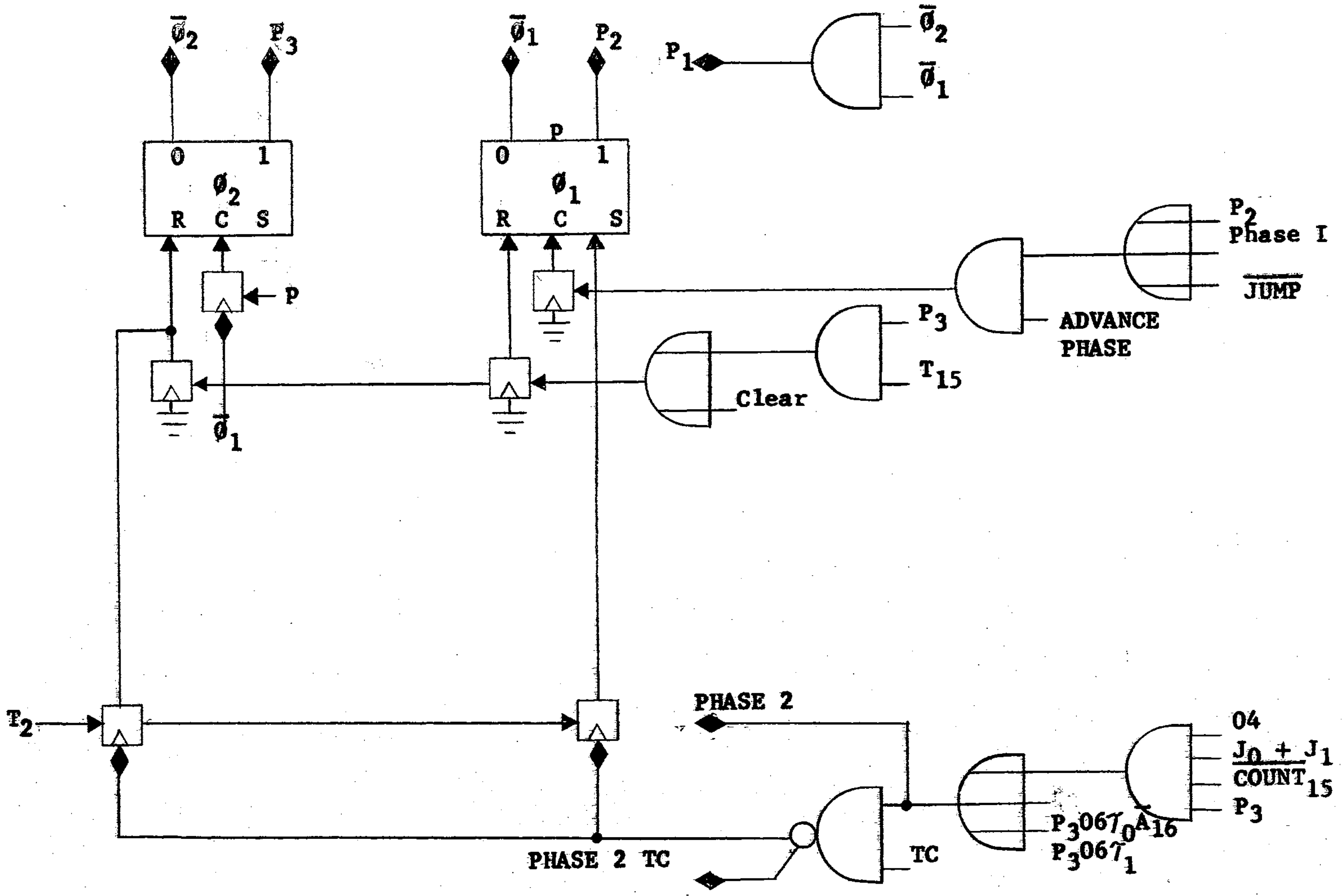


Figure 16. Phase Control

met. The mechanization chart also requires that the phase always be advanced at the end of P_2. The phase will never be advanced if the JUMP control is set earlier than T_{14}.

The mechanization chart requires that the phase be switched to P_2 if at P_3T_2 MUL, DIV, or CMP have not completed their execution cycles. The Timing Counter, as explained earlier, must clear to zero at this time if the execution of these instructions is not complete. The logic for determining the incompleteness of these execution cycles will be explained in the sections on multiply, divide, and compare. The Phase Counter will always clear to zero at P_3T_{15}.

Jump Control

The JUMP control is used to control the switching from one phase to another, and is also used to tell the various control units of the computer if the execution of an instruction is complete. The mechanization chart may be used to design the different controls for the JUMP control. The JUMP control must be set at P_1T_4 if the instruction being executed is a JMP, HLT, JAN and the accumulator is negative, JOF and the overflow indicator is on, JHI and the HI indicator is on, JLO and the LO indicator is on, JEQ and the EQ indicator is on. It will also be set at P_3T_1 if the execution of the MUL, DIV, or CMP instructions is not complete. The JUMP control is always set at T_{14}, regardless of the phase. The JUMP control must be set at P_1T_{11} if the instruction being executed is JAN, STO, JOF, PNT, CLR, or CPl. The JUMP control clears initially if the Clear pulse generator is energized. If the JUMP control is set, it will always reset at T_2, T_6, or T_{15}, and a Start Control pulse will be generated and fed back to the Master Timing Control Unit. If a Start Control pulse is generated, the START control

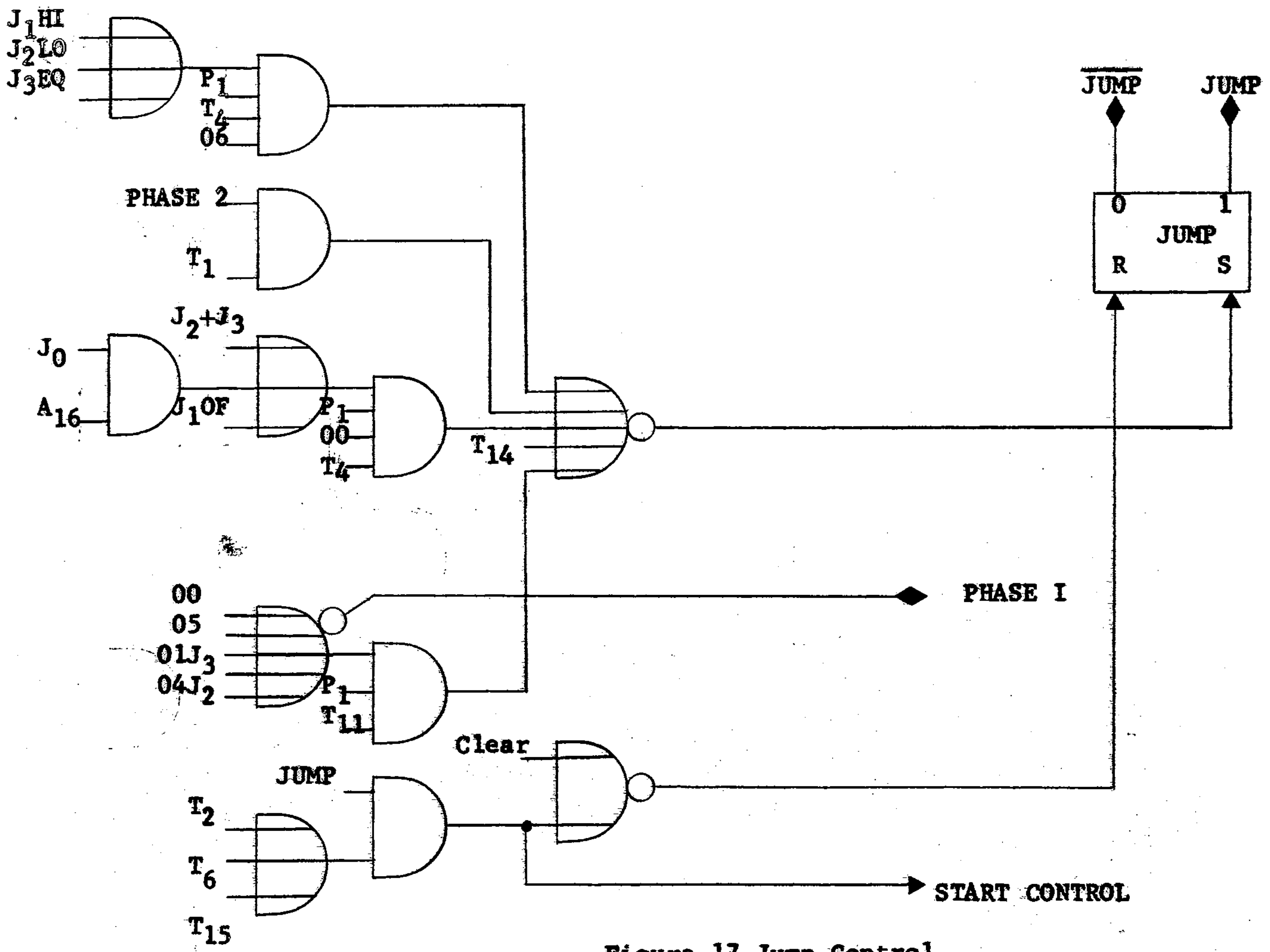


Figure 17 Jump Control

will reset and remain reset for 6 usec. Then the START control will set and the clock will start running again. This provides a delay between all phase changes that will allow all operations that should be completed in a phase complete their execution. As soon as all the controls settle down, the computer will start up again at the proper time and phase.

Variable Operations

There are several variations of the normal start cycle. These are the ORDER, CYCLE, and PULSE modes. If a Clear pulse is generated, followed by an O/S pulse, the START and ORDER controls will be set. The computer will start on an instruction cycle similar to the one using the START mode. The one basic difference is that at the end of an execution phase, the Start Control pulse will reset the START control and 6 usec later the START control will not set. Therefore, the computer will halt after this execution phase. If O/S is generated again, the computer will go through the next execution phase. This mode may be used to step the computer through an instruction one phase at a time. This is a great aid in trouble shooting the logic of the computer. If the clear pulse is followed by a C/C pulse, the START and CYCLE controls will set. The only difference between this mode and the Start mode is that the computer will repeat the same instruction over and over again instead of going to the next sequential instruction in a program. This action is controlled by the $\overline{\text{CYCLE}}$ level as shown in Figure 22. This mode is also useful in trouble shooting since it allows viewing repetitive waveforms of an instruction cycle. If the clear pulse is followed by a P/S pulse, the START and PULSE controls will be set. The $\overline{\text{PULSE}}$ level will be at a logic 0 turning off the clock, and all control pulses will come from P/S allowing the computer to be operated one pulse at a time. The overflow control should always be cleared with a Clear OF pulse before starting the computer on a program cycle.

The computer now has a timing and phase control unit. The design of the logic and control of the remainder of the units of the computer are dependent upon these units.

Registers and Their Control

The memory buffer and memory address registers have been discussed in Chapter II. There are several registers necessary for sequential operation, decoding of instructions, and arithmetic operations. These registers are the A,B,C, and D registers. The A and B registers are arithmetic registers, while the D register will act as a command holder, and the C register stores the address of the next sequential instruction.

The A register or accumulator is a 16 bit register that can be shifted either to the right or to the left. It serves as a storage register for the result of arithmetic operations. Data may also be read into it in parallel from the memory register. This register is a typical shift register, and the controlling pulse and data inputs are shown in Figure 18.

The B register is a 15 bit register that is capable of shifting to the right or the left. It also serves as a storage register for arithmetic operations. The controlling pulse and data inputs are shown in Figure 19.

The C register is a 9 bit register that stores the address of the next sequential instruction. This is accomplished as outlined by the mechanization chart by transferring the contents of the memory address

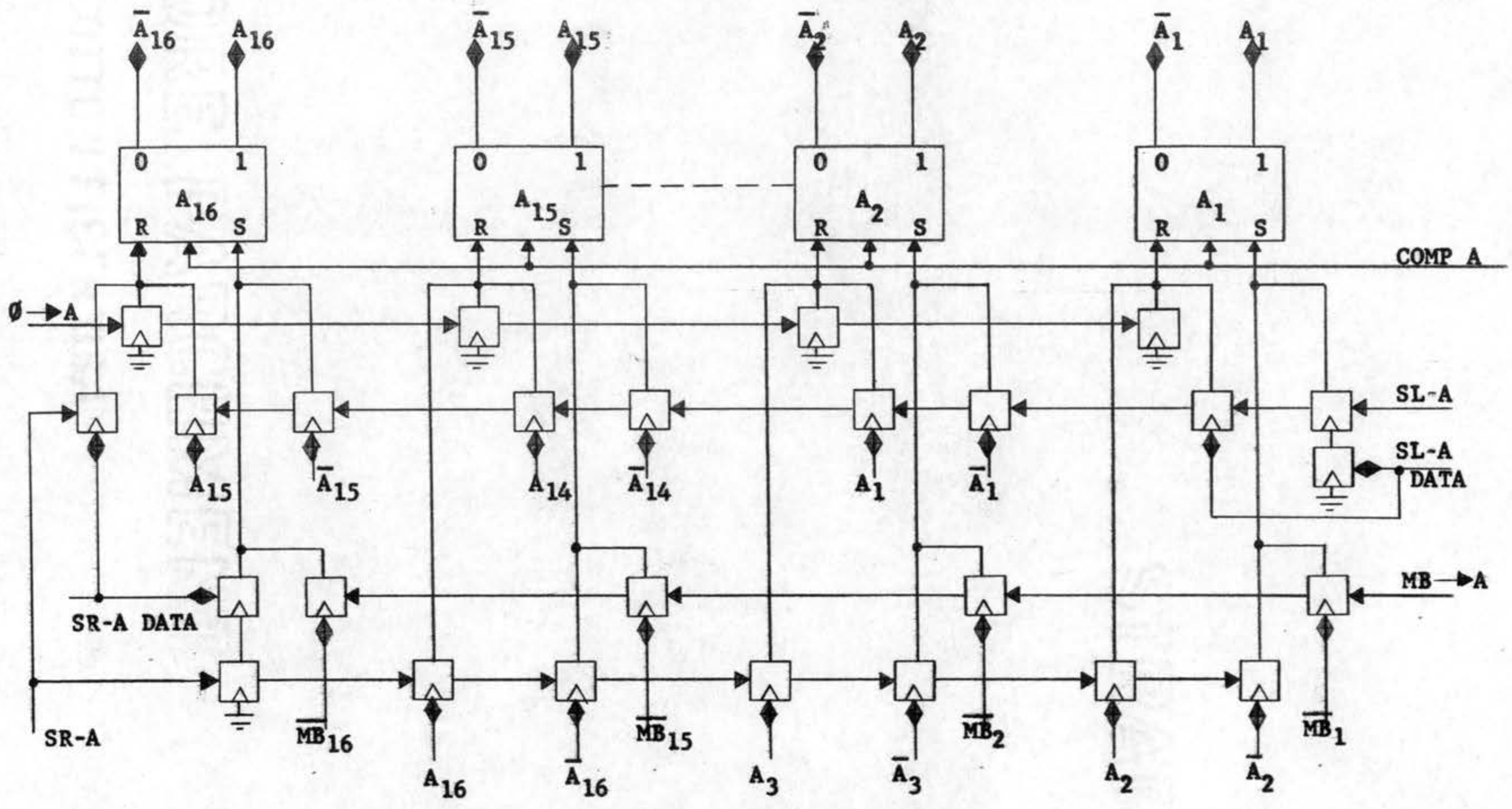


Figure 18. A Register

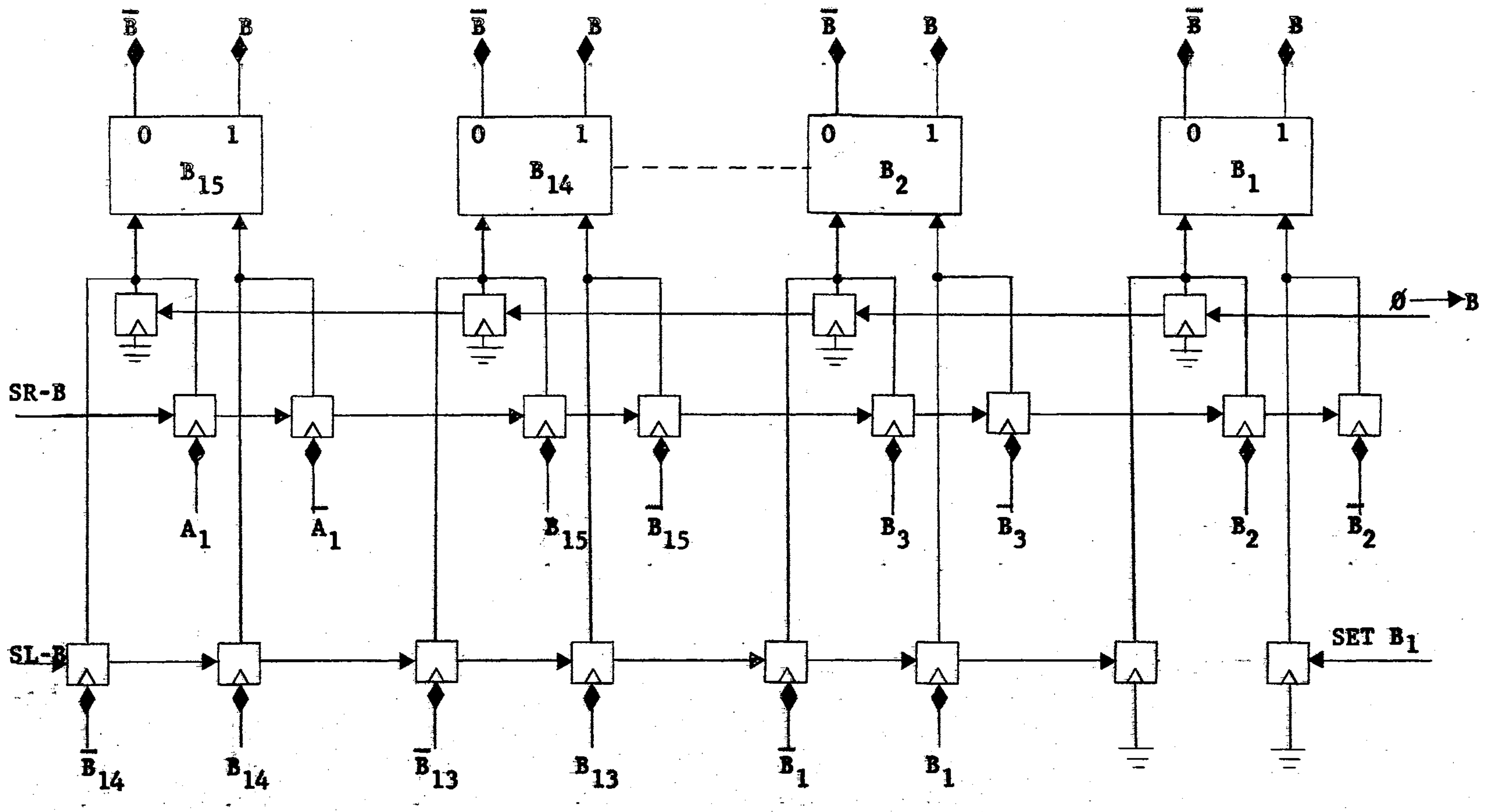


Figure 19. B Register

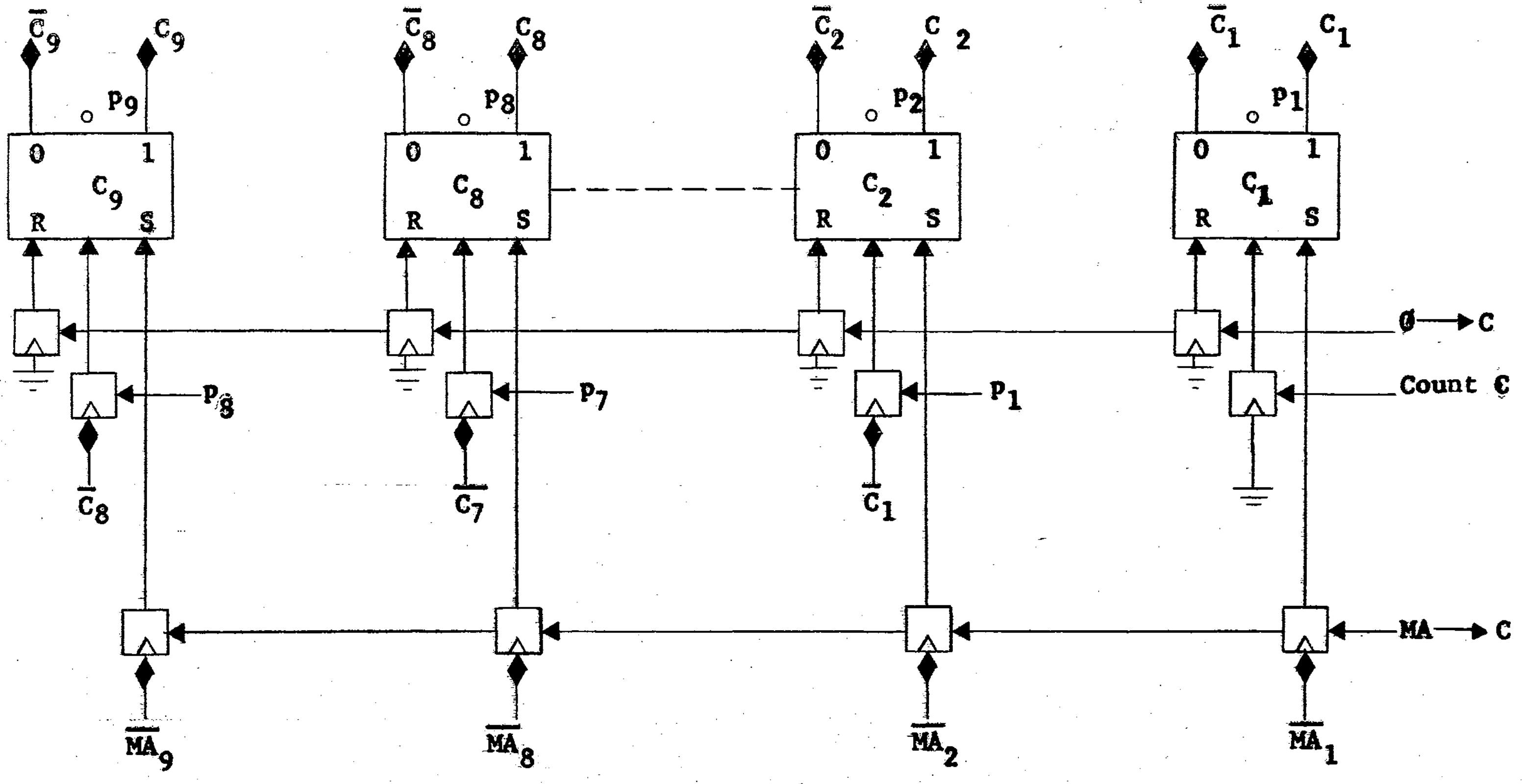


Figure 20. C Register Counter

register into the C register and then incrementing the C register by one. In order to accomplish this, the C register must also act as a binary counter. This counter is identical in its operation to all counters used in this computer. The pulse and data inputs are shown in Figure 20.

Since the OP CODE is 6 bits long, the register designed to store this code must be a 6 bit register. This 6 bit register, the D register, is shown with its pulse and data controls in Figure 21.

These various registers are controlled as outlined in the mechanization chart. The logic for these register controls is shown in Figure 22. At P_1T_0 the MB, C, and D registers will be cleared to zero in preparation for the selection and decoding of the instruction word indicated by the contents of the MA register. At P_1T_1, the contents of the MA register will be transferred to the C register. At P_1T_2, the MA register will be cleared to zero in preparation for receiving the address portion of the instruction word. Also the count in the C register will be advanced by 1. At P_1T_3, the 9 least significant bits of the MB register will be transferred into the MA register, and the OP CODE portion of the instruction word will be transferred into the D register. At this point the instruction word will have been selected and decoded. This information will remain in the MA and D registers until the instruction has been executed. At P_1T_7 the memory buffer register will be cleared. This is in preparation for the data word to be read from memory if this is an arithmetic or output instruction. At P_1T_9, the A register will be cleared if this is a CLA, CLR, INC, or DEC instruction. The A register will be transferred into the MB register in preparation for storage if this is a store instruction.

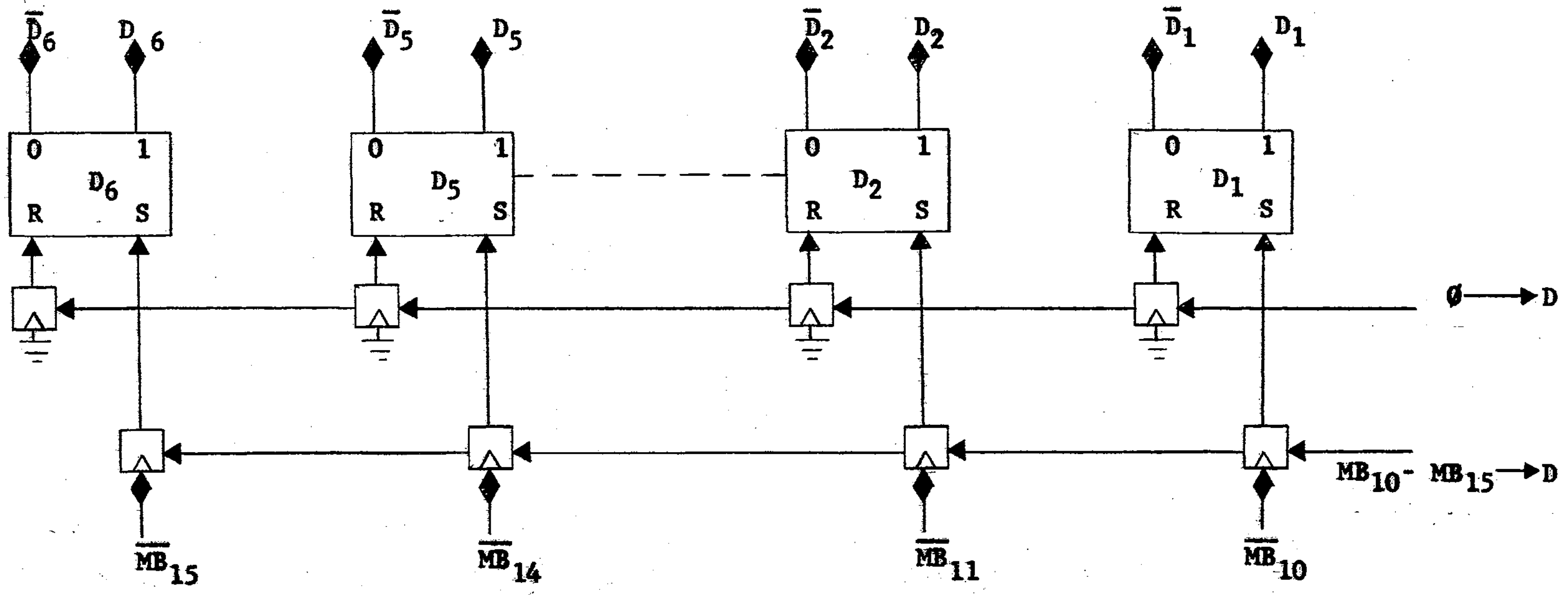


Figure 21. D Register

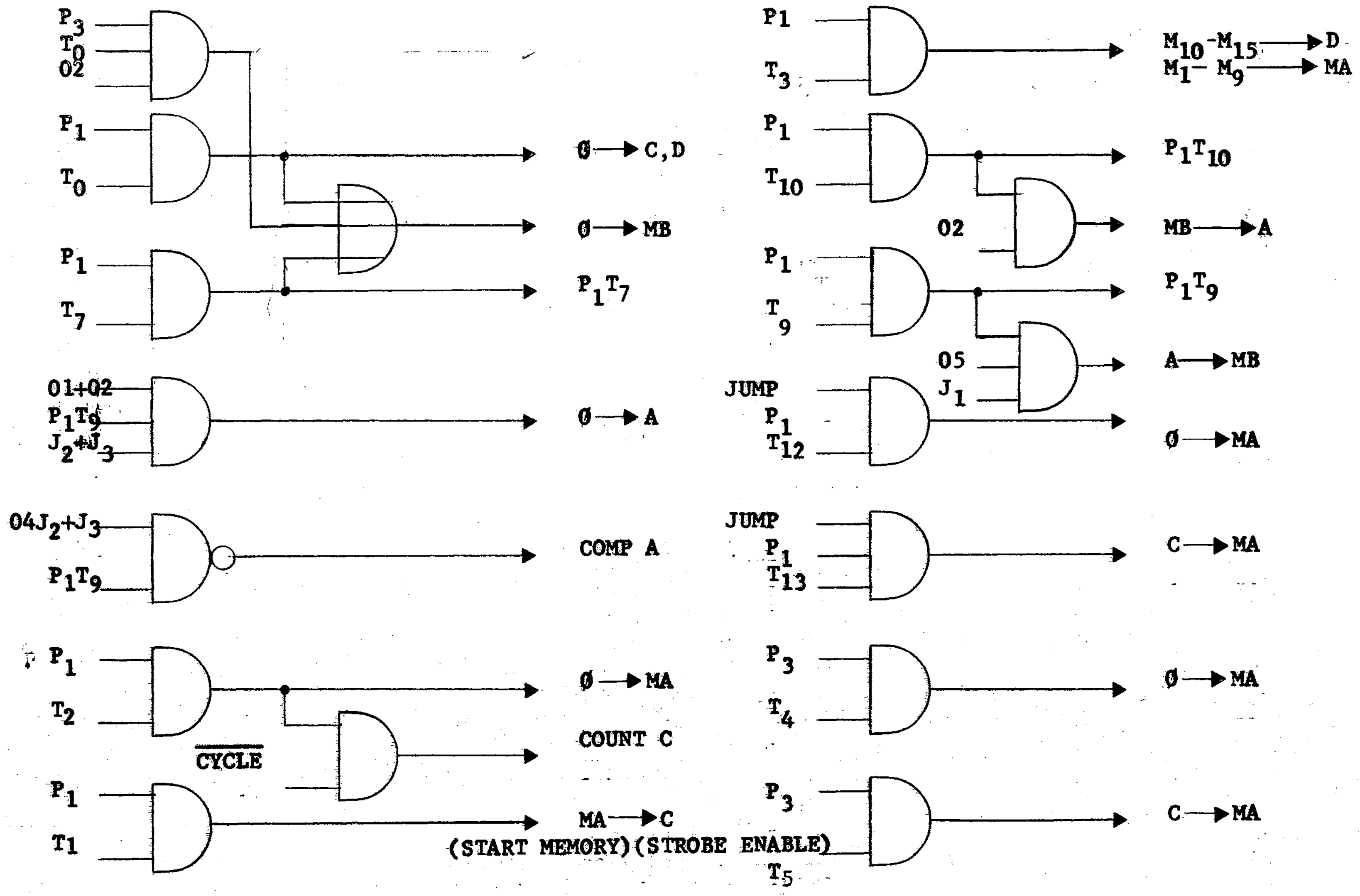


Figure 22. Register Controls

At P_1T_{10}, the MB register will be transferred into the A register if this is an INC or DEC instruction. At P_1T_{12}, clear the MA register if the JUMP flip-flop has been set. At P_1T_{13}, transfer the contents of the C register, the address of the next instruction, into the MA register if the JUMP control has been set.

During P_2, the serial arithmetic operations will be performed. After the completion of the execution of the arithmetic instructions, the control registers must be set up for the next instruction. At P_3T_4, the MA register will be cleared in preparation for the address of the next instruction. At P_3T_5, the contents of the C register, the address of the next instruction, will be transferred into the MA register. This concludes the design of the register controls.

Input Control

The input to a computer may take on several forms. Various input devices in common use are: the typewriter, teletype, punched cards, punched tape, magnetic tape, and the Flexowriter. All of these devices require input controls and buffering between them and core memory. Since none of these devices were available for test purposes, none of them were used.

There were registers available for test purposes in the laboratory facilities of the Federal Aviation Agency Academy, in Oklahoma City. These registers were set and reset by push button control. The registers that were available with this feature, were the memory buffer, memory address, accumulator, and B registers. Information such as programs and data was written into memory, by inserting the address of the word into the memory address register, and inserting the instruction of data word into the memory buffer register. Once information was loaded into memory, the logic of the computer was tested. The testing of the logic will be discussed in Chapter V.

Output Control

A digital printer was also available for test purposes in the Federal Aviation Agency Academy Computer Laboratory. This printer was connected to sample the contents of the memory address and memory buffer registers upon signal from the computer's memory and printer control logic. Any address and instruction or data word or the entire contents of memory can be printed upon signal.

Printer and Memory Control

The design of the Printer and Memory Control must take into consideration the read-write cycle delay time, and the printer delay time. The mechanization charts outline the design of this unit. The memory control unit has been discussed in Chapter II, and the available printer has its own control unit. That is, the printer will print the contents of the MA and MB registers upon signal, and it will signal the printer control unit when the print cycle is complete.

The logic for Core and Printer Control is shown in Figure 23. The main features of these controls are initiating the memory and print cycles. A print cycle consists of starting the printer at a predetermined time in the memory cycle. At this time the contents of the MA register will be known, and the contents of memory are determined by what is printed. Since there is about a 200 milli-second mechanical delay when printing, the master clock of the computer

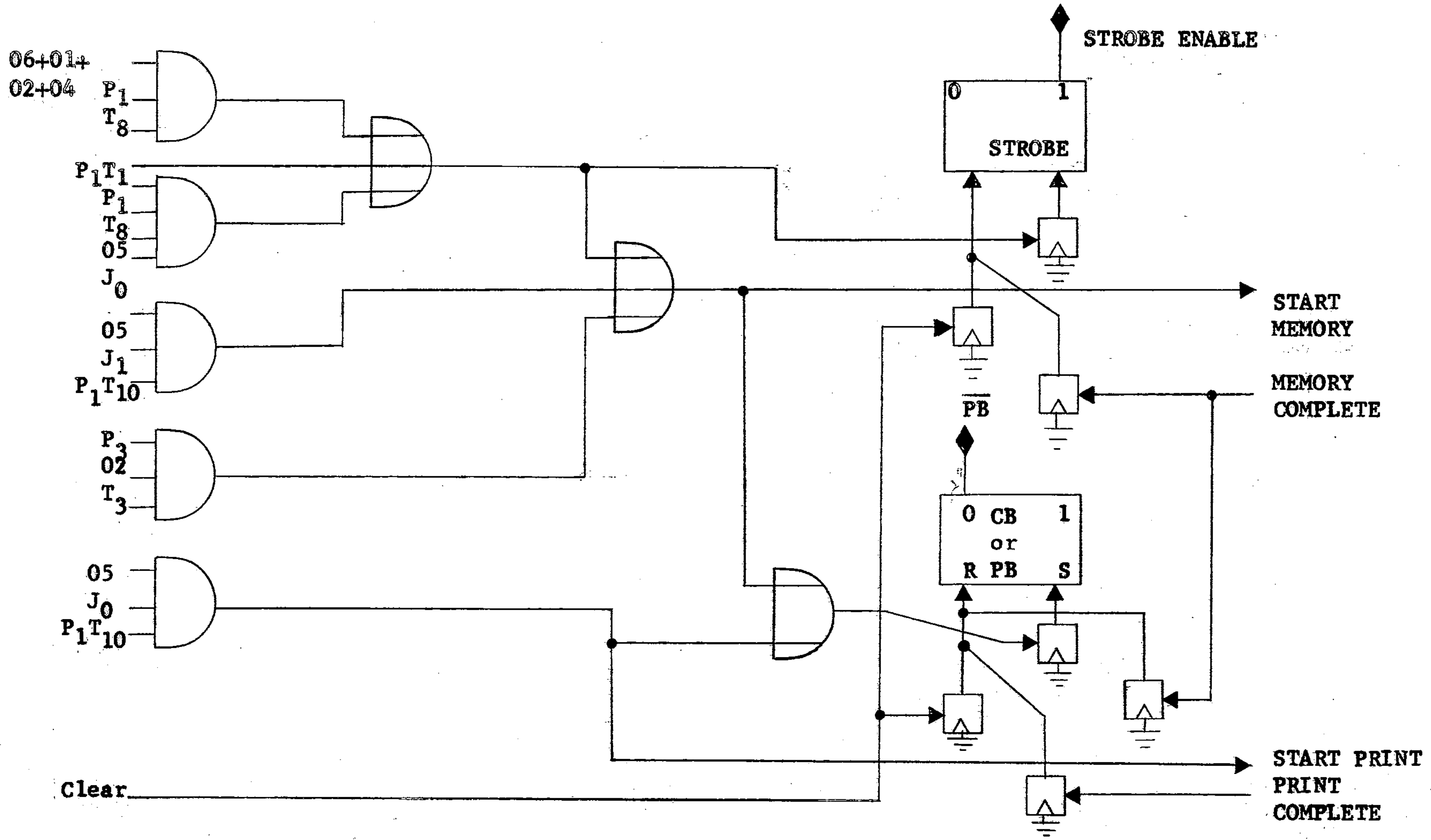


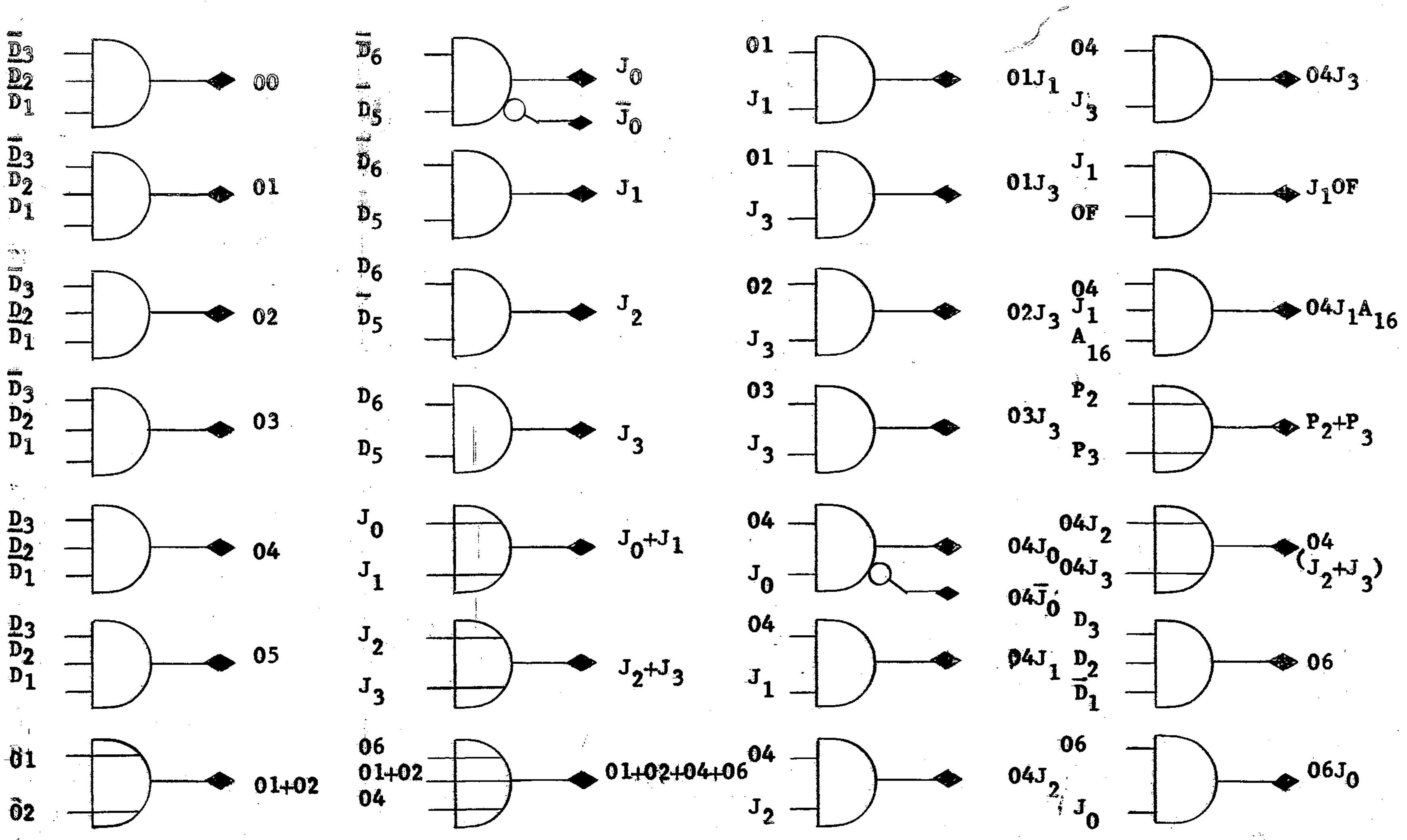
Figure 23. Core and Printer Control

must be turned off during the print cycle. This may be easily accomplished by setting the PB or printer busy flip-flop each time a print cycle is initiated. The Strobe and PB controls will be reset initially by the operator initiating a Clear pulse to these flip-flops. According to the mechanization chart, a print cycle should be initiated at P_1T_{10} if PNT has been decoded. When the print cycle is initiated, the PB control is set furnishing a $\overline{PB}$ level to the Master Timing Control unit turning off the clock. The printer will print the contents of the MA and MB registers, and then it will initiate a print complete pulse. (After a 200 milli-second delay) This print complete pulse will reset the PB control. This will remove the control level controlling the clock, allowing it to start again at P_1T_{11}.

The memory cycle is a little more complicated to control, since the logical design must take into consideration the type of memory cycle being executed. The Core control must determine if the cycle should be a read or a write cycle. This is determined by the STROBE control. If the cycle is a read cycle, the STROBE control will be set, and a STROBE ENABLE level will be fed to the memory control logic telling the memory control unit that this is a read cycle. The Memory Control unit was discussed in Chapter II. If the cycle is a write cycle, the STROBE control will not be set. This signals the Memory Control unit that this is a write cycle. Since there is a 10 usec delay in the read-write cycle, the CB control will be set any time a memory cycle is initiated. This furnishes a $\overline{PB}$ level to the Master Timing Control turning the clock off until the memory cycle is complete. The memory cycle is complete when the Memory Control unit initiates a memory complete pulse to reset the CB and STROBE controls. The clock will start running again at the timing pulse following the start of the memory cycle. The mechanization charts outline the following memory cycles. The first memory cycle is the selection cycle common to all instructions. This cycle occurs at P_1T_1, and the logic gate generating this pulse is shown in Figure 22. This memory cycle is a read cycle since it is required to set the STROBE control. There should be another memory cycle at P_1T_8. This is also a read cycle, and the information read from memory is used in the arithmetic operations ADD, SUB, CLA, INC, DEC, MUL, DIV, and CMP. There is also a read cycle at this time to determine what should be printed during the execution of the PNT instruction. There is a write cycle at P_1T_{10} if a STO instruction is being executed. At P_3T_3, a write cycle should be initiated to complete the execution of the INC or DEC instructions.

Instruction Decoding

The design for part of the instruction cycle, the selection cycle, has been completed. Part two of the instruction cycle is the decoding of the instruction. According to the mechanization chart, the selected instruction word should be stored in the D and MA registers at P_1T_3. This information will remain in these registers until the completion of the execution of the instruction. The D register could be labeled the "Decoding Register". The design of the command decoders is very simple, and is shown in Figure 24. Both the J-modifier and the OP CODE are decoded. Certain combinations of the J-modifier and the OP CODE that are used frequently are also decoded in Figure 24. The contents of the MA register are decoded by the X and Y decoders in the memory control unit, and by the data controls of the arithmetic units.



$\overline{D}_3$ $\overline{D}_2$ $\overline{D}_1$ 00
$\overline{D}_3$ $\overline{D}_2$ D_1 01
$\overline{D}_3$ D_2 $\overline{D}_1$ 02
$\overline{D}_3$ D_2 D_1 03
D_3 $\overline{D}_2$ $\overline{D}_1$ 04
D_3 $\overline{D}_2$ D_1 05
$\overline{01}$ $\overline{02}$ 01+02
$\overline{D}_6$ $\overline{D}_5$ J_0 $\overline{J}_0$
$\overline{D}_6$ D_5 J_1
D_6 $\overline{D}_5$ J_2
D_6 D_5 J_3
J_0 J_1 J_0+J_1
J_2 J_3 J_2+J_3
06 01+02 04 01+02+04+06
01 J_1 $01J_1$
01 J_3 $01J_3$
02 J_3 $02J_3$
03 J_3 $03J_3$
04 J_0 $04J_0$ $04\overline{J}_0$
04 J_1 $04J_1$
04 J_2 $04J_2$
04 J_3 $04J_3$
J_1 OF J_1OF
04 J_1 A_{16} $04J_1A_{16}$
P_2 P_3 P_2+P_3
$04J_2$ $04J_3$ 04(J_2+J_3)
D_3 D_2 $\overline{D}_1$ 06
06 J_0 $06J_0$

Figure 24. Operation Decoders

The Logical Design of the Arithmetic Unit

The computer should now be capable of selecting an instruction, decoding an instruction, and in a few cases even executing a few simple instructions. The main purpose of a general purpose digital computer is to process data. In order to do this, an arithmetic unit will be required that is capable of addition, subtraction, multiplication, division, and even some variations of these instructions. The basic design of this computer consists of a serial binary full adder and subtracter. The other listed operations will be simple variations of addition and subtraction.

The design problem for a serial binary adder may be stated as follows. Design an arithmetic unit that is capable of determining the sum of two 16 bit binary numbers. These numbers will be stored in the A and MB registers of the computer. The sixteenth bit will represent the sign bit. A 0 represents positive numbers, while 1 represents negative numbers. All negative numbers will be stored in the computers core memory in their two's complement form. The carry resulting from the addition of two binary 1's will be stored in a flip-flop. The A register will store the Augend and the Sum resulting from the addition. The MB register will store the Addend. As outlined in the mechanization chart, addition will start at P_2T_0, and will be complete at P_2T_{15}. Binary addition is summarized in Table IV. The Augend = X and the Addend = Y.

TABLE IV

BINARY ADDITION

X	+	Y	=	Sum	Carry
0	+	0	=	0	0
0	+	1	=	1	0
1	+	0	=	1	0
1	+	1	=	0	1

Since the two registers will be added in binary form, and a carry of either 0 or 1 must be considered, there will be three variables to consider in this design problem. These three variables may each be either 0 or 1. There are 8 possible combinations of these three variables, and these 8 combinations represent all of the possible addition problems that this computer will have to solve. These 8 combinations with the resulting sum and carry are shown in Table V.

TABLE V

BINARY ADDITION TRUTH TABLE

Minterm	X	Y	C	Sum	Carry
m_0	0	0	0	0	0
m_1	0	0	1	1	0
m_2	0	1	0	1	0
m_3	0	1	1	0	1
m_4	1	0	0	1	0
m_5	1	0	1	0	1
m_6	1	1	0	0	1
m_7	1	1	1	1	1

The equation for sum is equal to:

$\text{Sum} = m_1 + m_2 + m_4 + m_7$

If this function were mapped on a Karnaugh map, no simplification would result since there are no adjacencies on the map.[2]

The equation for carry is equal to:

$\text{CARRY} = m_3 + m_5 + m_6 + m_7$

This function may be simplified in Figure 25 as follows.

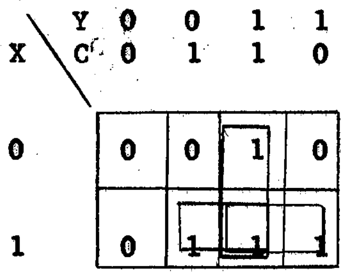


Figure 25. Karnaugh Map Simplification

$\text{CARRY} = XY + XC + YC$

Since the carry bit will be stored in a CARRY flip-flop, and the input to this circuit will come from the CARRY flip-flop, this circuit may be considered a sequential circuit. Since C is stored in the CARRY flip-flop, if C is already set, as it is for minterms 3, 5, and 7, then these minterms may be considered don't care minterms. Then Figure 25 may be redrawn as shown in Figure 26.

[2]Watts S. Humphrey Jr., Switching Circuits with Computer Applications, New York, 1958, p. 91.

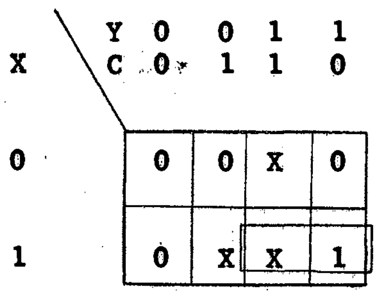
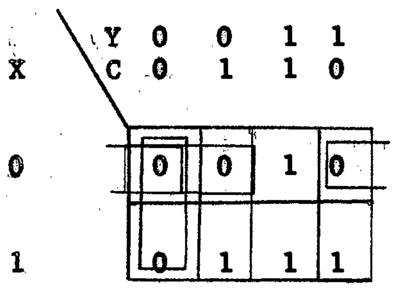


Figure 26. Karnaugh Map Simplification

CARRY may then be simplified to:

CARRY = XY

Any time the bits coming from both the A and MB registers are both 1, the CARRY flip-flop must be set. The logic for determining when the CARRY flip-flop should be reset is shown in Figures 27 and 28.



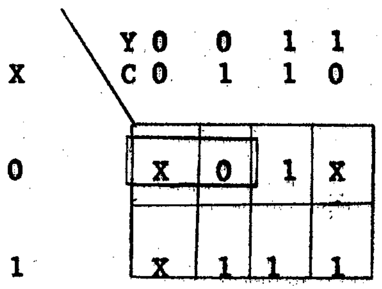


Figure 27. Simplification Figure 28. Simplification

The CARRY flip-flop should be reset as follows:

$\overline{CARRY} = \overline{X}\ \overline{Y} + \overline{X}\ \overline{C} + \overline{Y}\ \overline{C}$

If this is considered as a sequential circuit, when C is already reset, then it does not need to be reset.[3] Figure 27 may be redrawn as shown in Figure 28.

[3]Watts S. Humphrey Jr., Switching Circuits with Computer Applications, New York, 1958, p. 236.

$\overline{\text{CARRY}}$ may then be simplified to:

$\overline{\text{CARRY}} = \overline{X}\ \overline{Y}$

The design equations for the arithmetic unit adder may be summarized as follows.

$\text{Sum} = \overline{X}\ \overline{Y}\ C + \overline{X}\ Y\ \overline{C} + X\ \overline{Y}\ \overline{C} + X\ Y\ C$

$\overline{\text{CARRY}} = \overline{X}\ \overline{Y}$

$\text{CARRY} = X\ Y$

The Binary Subtracter

The binary subtracter must meet the same specifications as the binary adder. The only difference is that the difference between the two registers A and MB must be found and stored in the A register instead of storing the sum. The minuend (X) is stored in the A register, and the subtrahend (Y) is stored in the MB register. Binary subtraction is summarized in TABLE VI.

TABLE VI

BINARY SUBTRACTION

X - Y =	Difference	Borrow
0 - 0 =	0	0
0 - 1 =	1	1
1 - 0 =	1	0
1 - 1 =	0	0

Again there are three variables, X, Y, and B. The borrow will be stored in the BORROW flip-flop. The 8 possible combinations of these three variables and the difference and borrow are shown in TABLE VII.

TABLE VII

BINARY SUBTRACTION TRUTH TABLE

Minterm	X	Y	B	Difference	Borrow
m_0	0	0	0	0	0
m_1	0	0	1	1	1
m_2	0	1	0	1	1
m_3	0	1	1	0	1
m_4	1	0	0	1	0
m_5	1	0	1	0	0
m_6	1	1	0	0	0
m_7	1	1	1	1	1

The equation for difference is equal to:

Difference = $m_1 + m_2 + m_4 + m_7$

This equation is the same as that for the sum and is not reducible.

The equation for setting the BORROW flip-flop is equal to:

BORROW = $m_1 + m_2 + m_3 + m_7$

This function may be simplified in Figure 29 as follows.

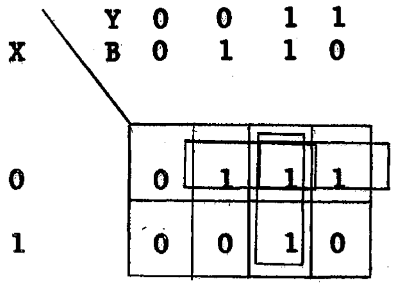


Figure 29. Karnaugh Map Simplification

BORROW = $\overline{X}\,B + \overline{X}\,Y + Y\,B$

Since the borrow bit is stored in the BORROW flip-flop, and the input to this circuit will come from the BORROW flip-flop, this circuit may also be considered a sequential circuit. Since B is stored in the BORROW flip-flop, and if B is already set, as it is for minterms 1, 3, and 7, then these minterms may be considered don't care minterms. Figure 29 may be redrawn as shown in Figure 30.

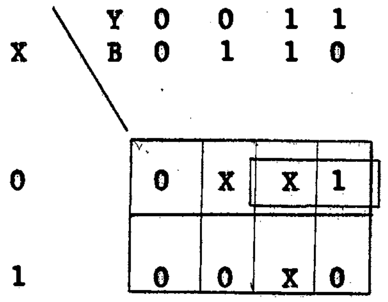


Figure 30. Karnaugh Map Simplification

BORROW may now be simplified to:

BORROW $= \overline{X}\ Y$

The equation for resetting the BORROW flip-flop is equal to:

$\overline{\text{BORROW}} = m_0 + m_4 + m_5 + m_6$

This equation may be simplified as shown in Figure 31.

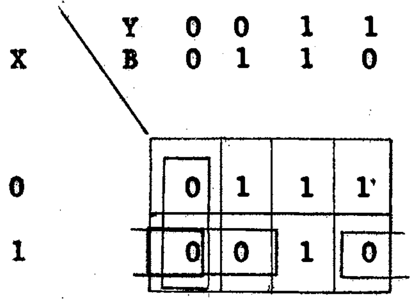


Figure 31. Karnaugh Map Simplification

$\overline{\text{BORROW}} = \overline{Y}\ \overline{B} + X\ \overline{Y} + X\ \overline{B}$

Any time the BORROW flip-flop is already reset, the minterm input can be called a don't care minterm, minterms 0, 4, and 6 are don't care minterms. Figure 31 may be redrawn as shown in Figure 32.

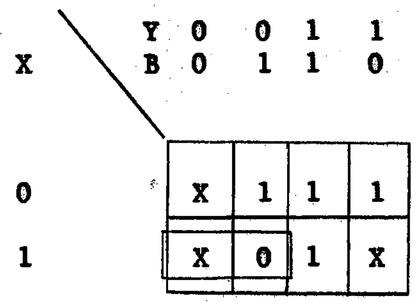


Figure 32. Karnaugh Map Simplification

$\overline{\text{BORROW}}$ may now be simplified to:

$\overline{\text{BORROW}} = X\ \overline{Y}$

The equations for the design of the adder and subtracter arithmetic unit may be summarized as follows.

$\text{Sum} = \overline{X}\ \overline{Y}\ C + \overline{X}\ Y\ \overline{C} + X\ \overline{Y}\ \overline{C} + X\ Y\ C$

$\text{Difference} = \overline{X}\ \overline{Y}\ B + \overline{X}\ Y\ \overline{B} + X\ \overline{Y}\ \overline{B} + X\ Y\ B$

$\text{CARRY} = X\ Y$

$\text{BORROW} = \overline{X}\ Y$

$\overline{\text{CARRY}} = \overline{X}\ \overline{Y}$

$\overline{\text{BORROW}} = X\ \overline{Y}$

Since the sum and difference equations are made up of the same minterms, the same logic can be used for sum and difference as shown in Figure 33.

If the equations for CARRY and BORROW are examined, it will be noticed that they differ only in the variable X. The same may be noticed for the equations for $\overline{\text{CARRY}}$ and $\overline{\text{BORROW}}$. One flip-flop can be used for both carry and borrow if the controls shown in Figure 33 are used. If the logic diagrams are examined closely, it may be seen that they satisfy the simplified derived functions. The ADD and SUB control levels originate in Figure 34. The Arithmetic Unit should act as a subtracter if

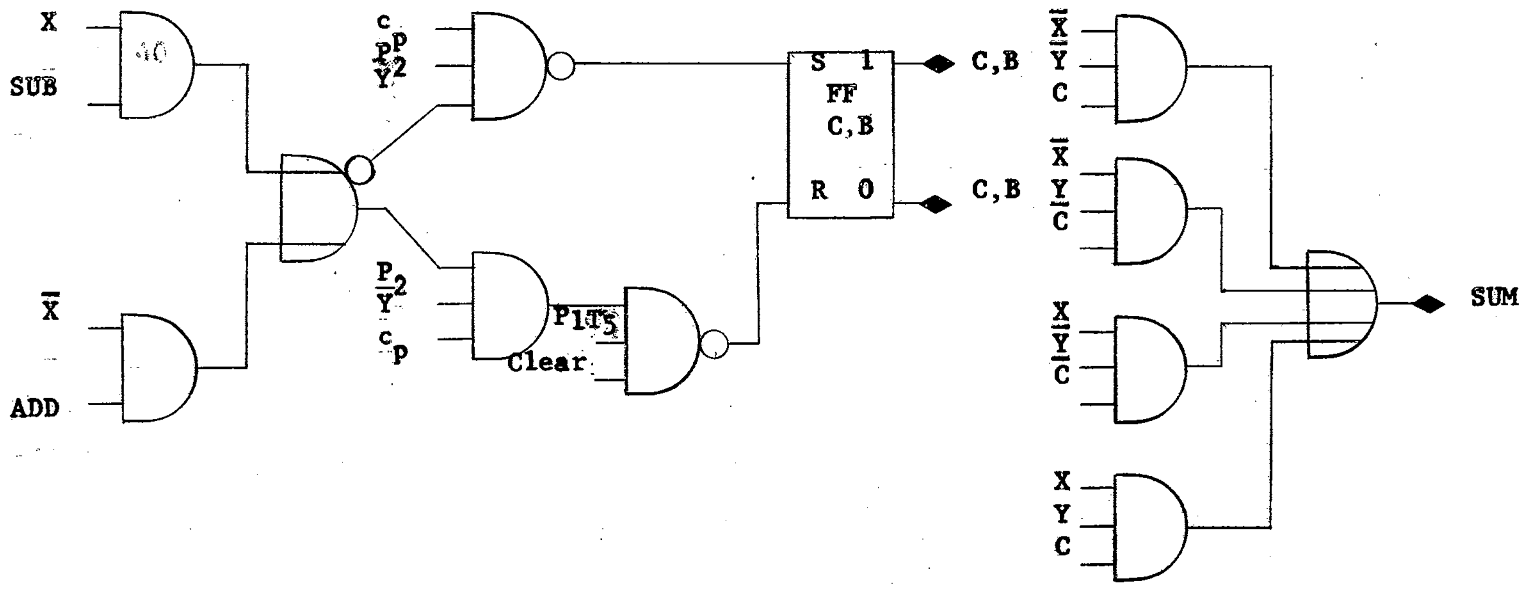


Figure 33. Serial Binary Adder and Subtracter

the instruction decoded is a SUB, DEC, DIV, or a CMP instruction. The Arithmetic Unit will act as an adder during the execution of any other instruction. The Arithmetic Unit will act as an adder during certain conditions of execution of the DIV and CMP instructions. These conditions will be explained in the sections on comparison and division. This arithmetic unit CARRY control will clear initially with the initiation of the Clear pulse. According to the mechanization chart, the CARRY control should also clear at P_1T_5. The logic for this is shown in Figure 33. This arithmetic unit will either add or subtract in P_2 if the computer logic reaches this phase.

Arithmetic Overflow

An arithmetic overflow indication may be stored in an Overflow flip-flop as shown in Figure 35. This control may be reset initially by using the Clear OF control in the Master Timing Control unit. The mechanization chart outlines the design for setting and resetting the Overflow flip-flop OF. If a JOF instruction has been decoded, OF should be reset at P_1T_4. The OF control should also be reset at P_1T_{14} if any arithmetic operation is to be performed in P_2. The mechanization chart also requires that OF be set at P_2T_{15} if an arithmetic overflow has been detected. The condition for overflow is detected at the time in P_2 that the sign bits of the two words are added together with the carry from the most significant bit. Since there are three variables, there are 8 possible combinations of these three variables. The combinations and the resulting overflow condition are listed in TABLE VIII for the ADD cycle.

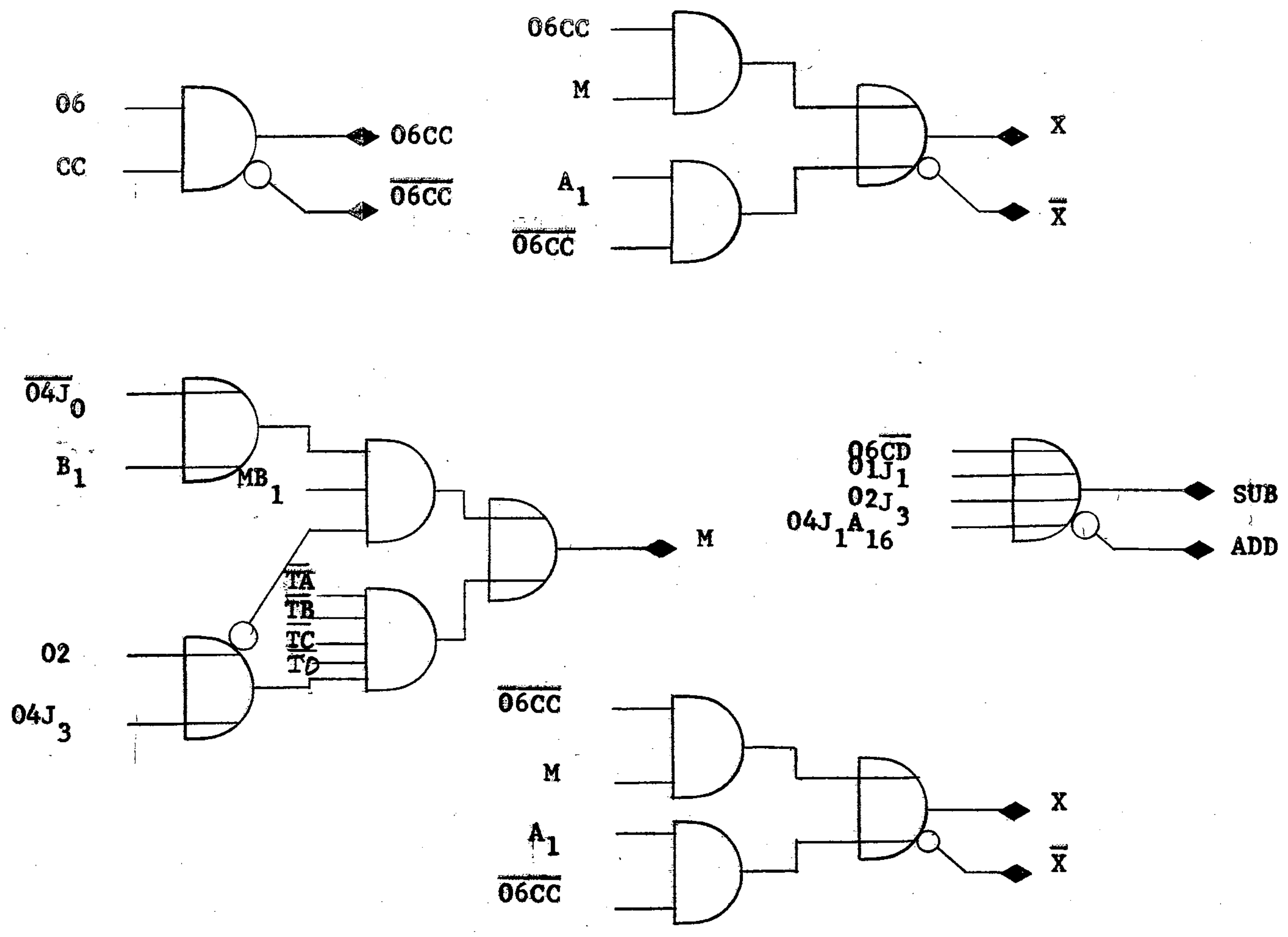


Figure 34. Arithmetic Unit Data Control

TABLE VIII

ADDITION OVERFLOW

Minterm	S_X	S_Y	C_M	OVERFLOW
m_0	0	0	0	0
m_1	0	0	1	1
m_2	0	1	0	0
m_3	0	1	1	0
m_4	1	0	0	0
m_5	1	0	1	0
m_6	1	1	0	1
m_7	1	1	1	0

If the arithmetic unit is adding, the overflow equation equals:

$\text{Overflow}_{\text{ADD}} = \overline{X}\,\overline{Y}\,C + X\,Y\,\overline{C}$

The logic for this is shown in Figure 35.

The 8 possible combinations of the sign bits and the carry are listed in TABLE IX for the SUB cycle.

TABLE IX

SUBTRACTION OVERFLOW

Minterm	S_X	S_Y	C_M	OVERFLOW
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	0
m_4	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	0
m_7	1	1	1	0

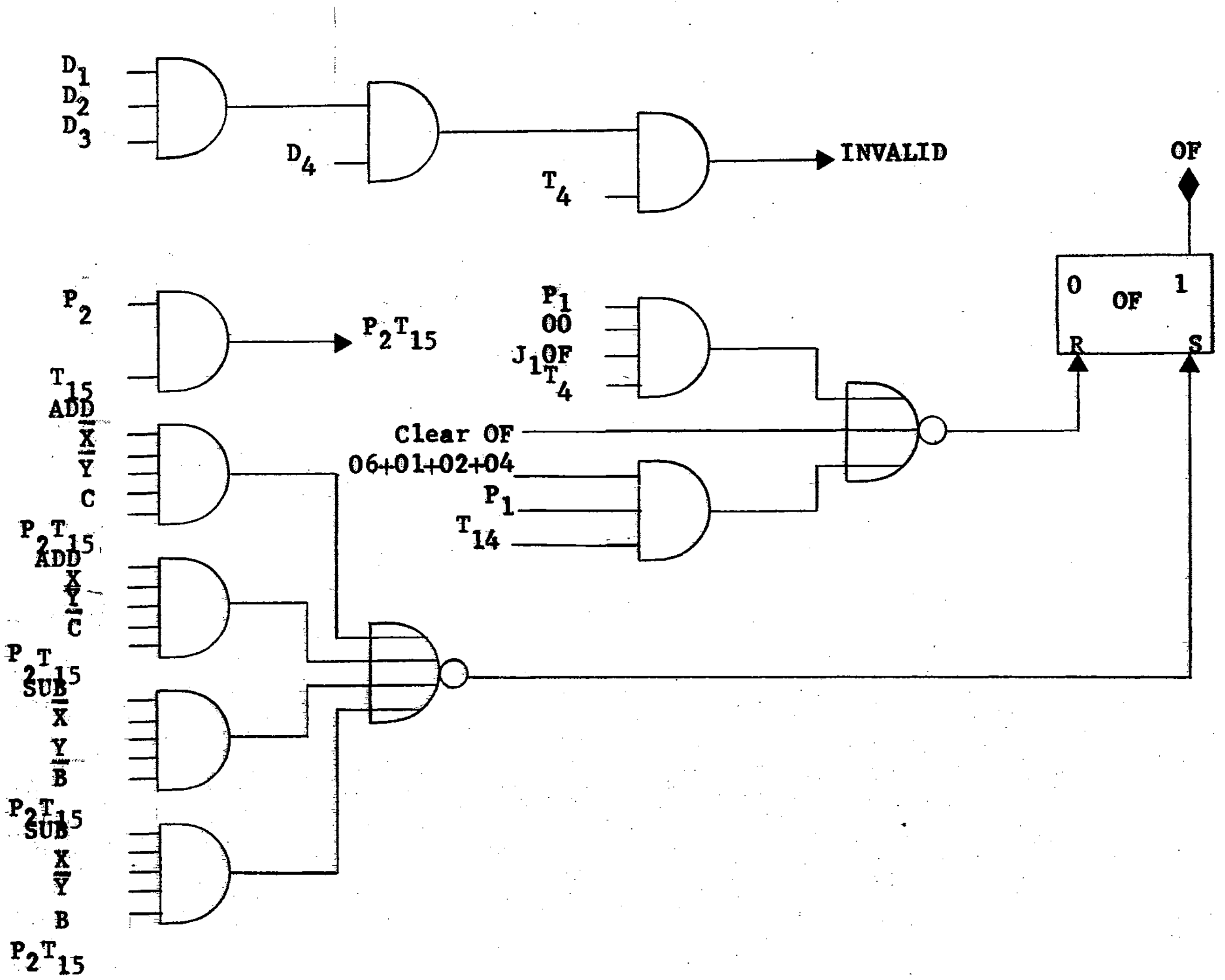


Figure 35. Arithmetic Overflow

If the arithmetic unit is subtracting, the Overflow equation equals:

$\text{Overflow}_{SUB} = \overline{X}\ Y\ \overline{C} + X\ \overline{Y}\ C$

The logic for this is shown in Figure 35.

Invalid Operations

The invalid operations decoder is also shown in Figure 35. Any time an invalid operation is decoded during the decoding cycle, the Master Timing Control unit is signaled to reset the START control and the computer will stop. In order to restart, the operator must reset all controls. The logic for this may be designed by considering all of the invalid codes. OP CODES 07 through 17 are not used, therefore, D_4 can never be a logic 1 and the combination $D_1 D_2 D_3$ can never occur.

Shift Control

The shift instruction is used to shift the contents of the A and B registers to the right or left by a predetermined amount. Shifting a register is a fairly simple operation to control, but the amount of shift is a little more difficult to control. A shift register may be shifted by pulsing the shift inputs the required number of times. This can be controlled by pre-setting a counter to tell the control unit the amount of the shift. Since the registers are 16 bits long, the counter should be a scale of 16 counter. A conventional up counter is shown in Figure 36, and is called the Operations Control Counter. This counter is referred to in the mechanization charts as COUNT. This counter is used to control the ARS, ALS, LRS, LLS, MUL, DIV, and the CMP instructions. The counter along with the control

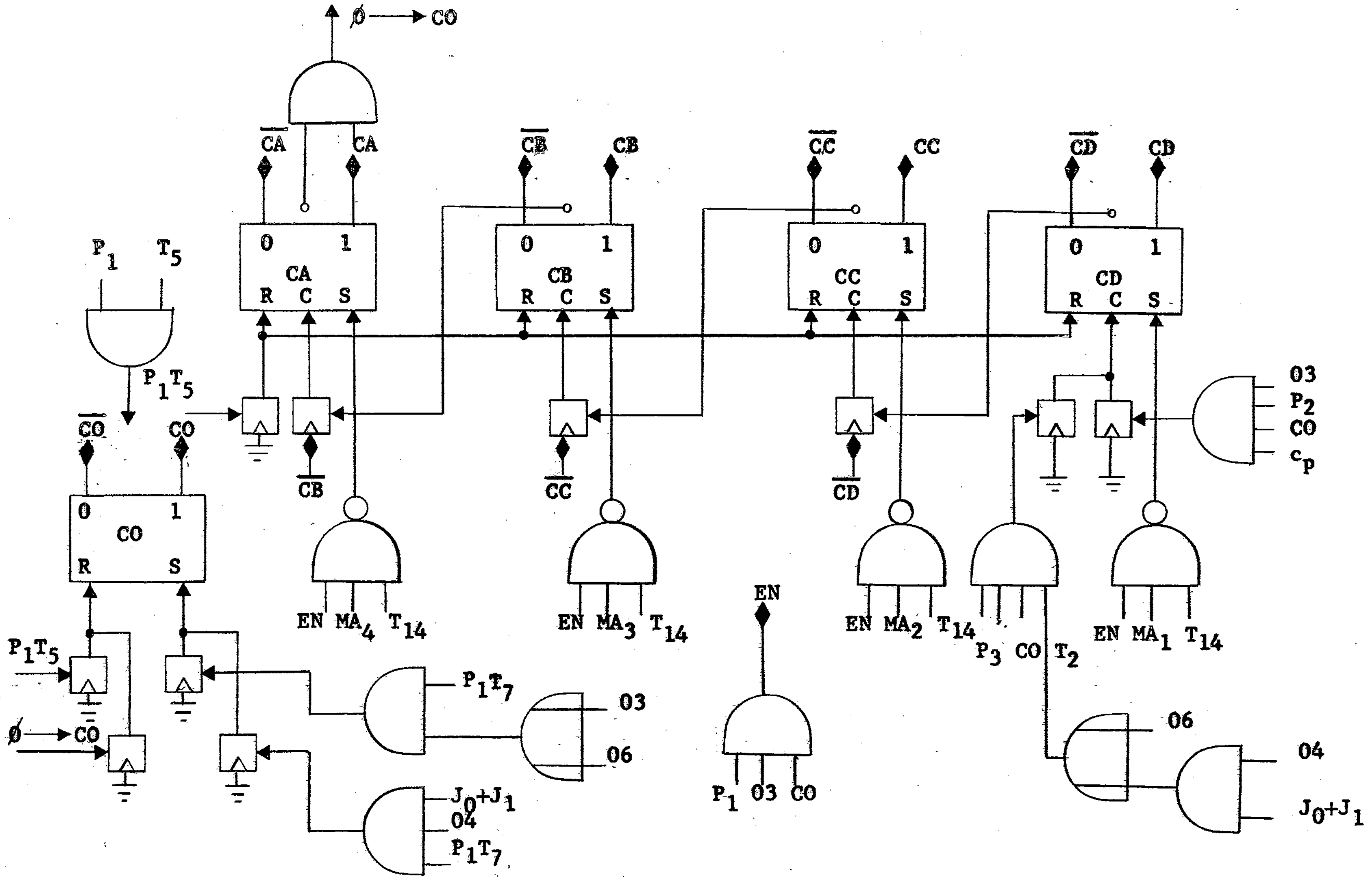


Figure 36. Operations Control Counter

flip-flop CO is always cleared at P_1T_5, as indicated in the mechanization chart. The mechanization chart requires that CO be set at P_1T_7 if the instruction being executed is an ARS, ALS, LRS, LLS, MUL, DIV, or a CMP instruction. CO is used as a control level. The counter should be preset at P_1T_{14} to the count pre-determined by the count stored in the address portion of the instruction word stored in the MA register, if the instruction being executed is a shift instruction. This logic is shown as controlling the set inputs to the counter in Figure 36. The clock pulses in P_2 are gated into the count input if the instruction is an ARS, ALS, LRS, or LLS instruction. The CO control will reset when the counter resets to 0000. This removes the pulse input to the counter and shifting will stop. The other controls in this figure control MUL, DIV, and CMP, and they will be discussed in those sections. The control of the register shifting is shown in Figure 37.

The mechanization chart requires that the MB register circulate its information in a loop in P_2. This is accomplished as the SR-MB pulse generated in Figure 37. In P_2, if a shift instruction is being executed, CO is set allowing a predetermined number of pulses to be directed as shift left pulses to A (LA), shift left pulses to B (LB), shift right pulses to A (RA), or shift right pulses to B (RB). The J-modifier gates the shift pulse to the proper register as outlined in TABLE X.

The A register must also shift right 16 bits through the arithmetic unit if the instruction being executed is any arithmetic instruction. The Data to the A register will come from the arithmetic unit (SUM), if the instruction being executed is any arithmetic

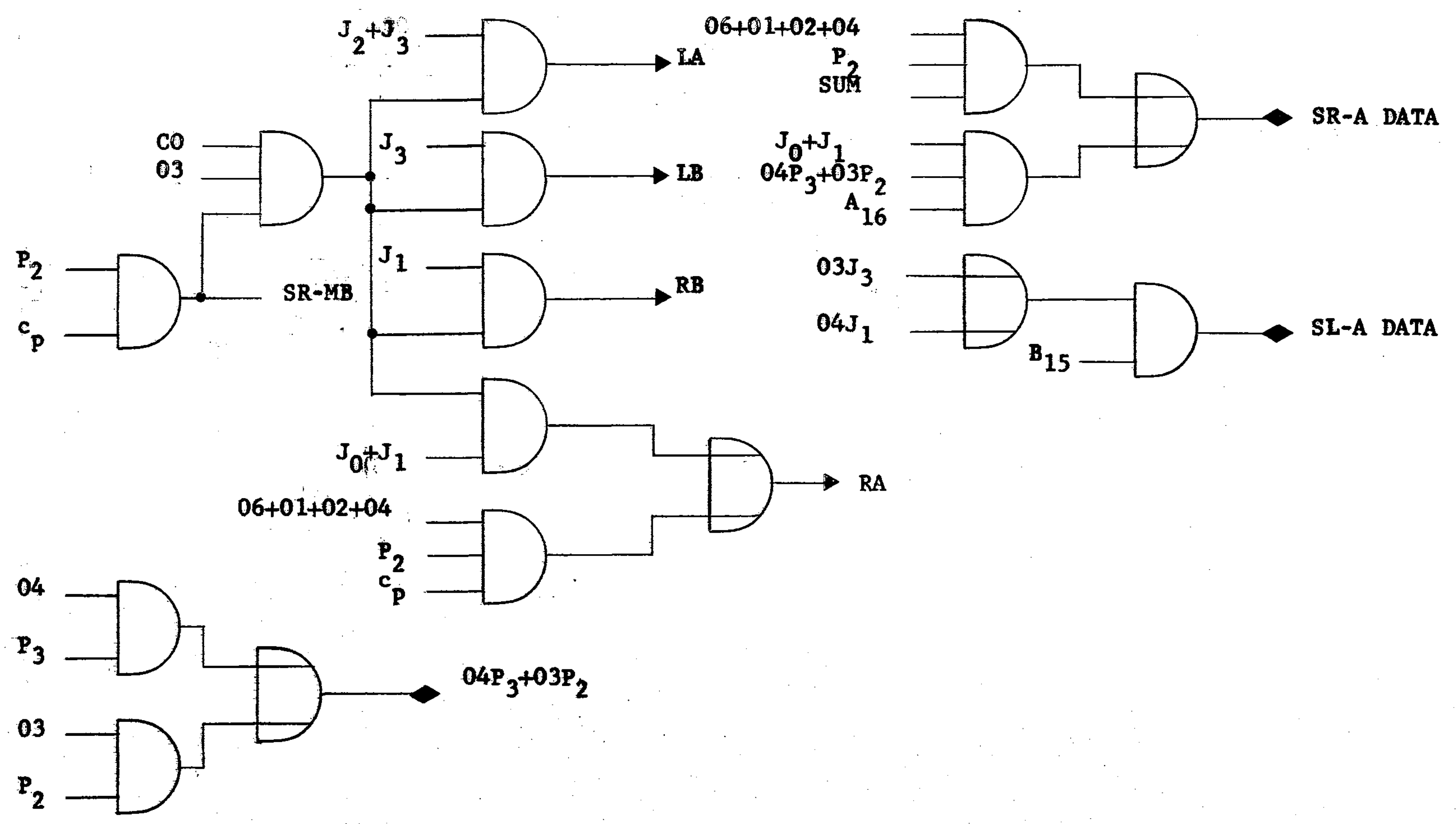


Figure 37. Data Control

instruction. This logic is also shown in Figure 37 as SR-A DATA.

TABLE X

SHIFT CONTROL

J-modifier	A Register	B Register
0	Shift right	No shift
1	Shift right	Shift right
2	Shift left	No shift
3	Shift left	Shift left

Multiply Control

Multiplication as it is performed using pencil and paper and binary numbers is nothing more than a combination of forming a partial product and shifting and adding. The multiplication table for binary numbers is very simple, as shown in TABLE XI.

TABLE XI

BINARY MULTIPLICATION

X	x	Y	=	Product
0	x	0	=	0
0	x	1	=	0
1	x	0	=	0
1	x	1	=	1

Binary multiplication might be summarized by stating that 1 times anything in binary is equal to anything. An example problem will illustrate this.

Binary		Decimal
1101	Multiplicand	13
x 101	Multiplier	x 5
1101	Partial Product	65
0000	Partial Product	
1101	Partial Product	
1000001	= $(65)_{10}$	Product

Each time the multiplier is equal to 1, the multiplicand is simply copied down. The next partial product is shifted to the left and copied down. In this example, the partial product is 0000 and actually did not need to be copied down. The third partial product is the multiplicand shifted to the left one bit and copied down. If the computer can be made to simulate these steps, it will be able to multiply.

The multiplier should first be loaded into the B register using the CLA and the LRS commands. The least bit of the multiplier then must be tested to see if it is equal to 1, and then add the multiplicand to the A register. Once this is accomplished the logic of the computer should shift the A and B registers to the right 1 bit. (The least significant bit of the multiplier is no longer needed). This shifts the least bit of the product into the most significant bit position of the B register. Then the next bit of the multiplier is sampled and a partial product formed by adding either 0 or the multiplicand to the A register and then shifting right. This continues until all 16 bits of the multiplier have been sampled. The product will be stored in the B register after the execution of the instruction.

The mechanization charts outline the operations just described,

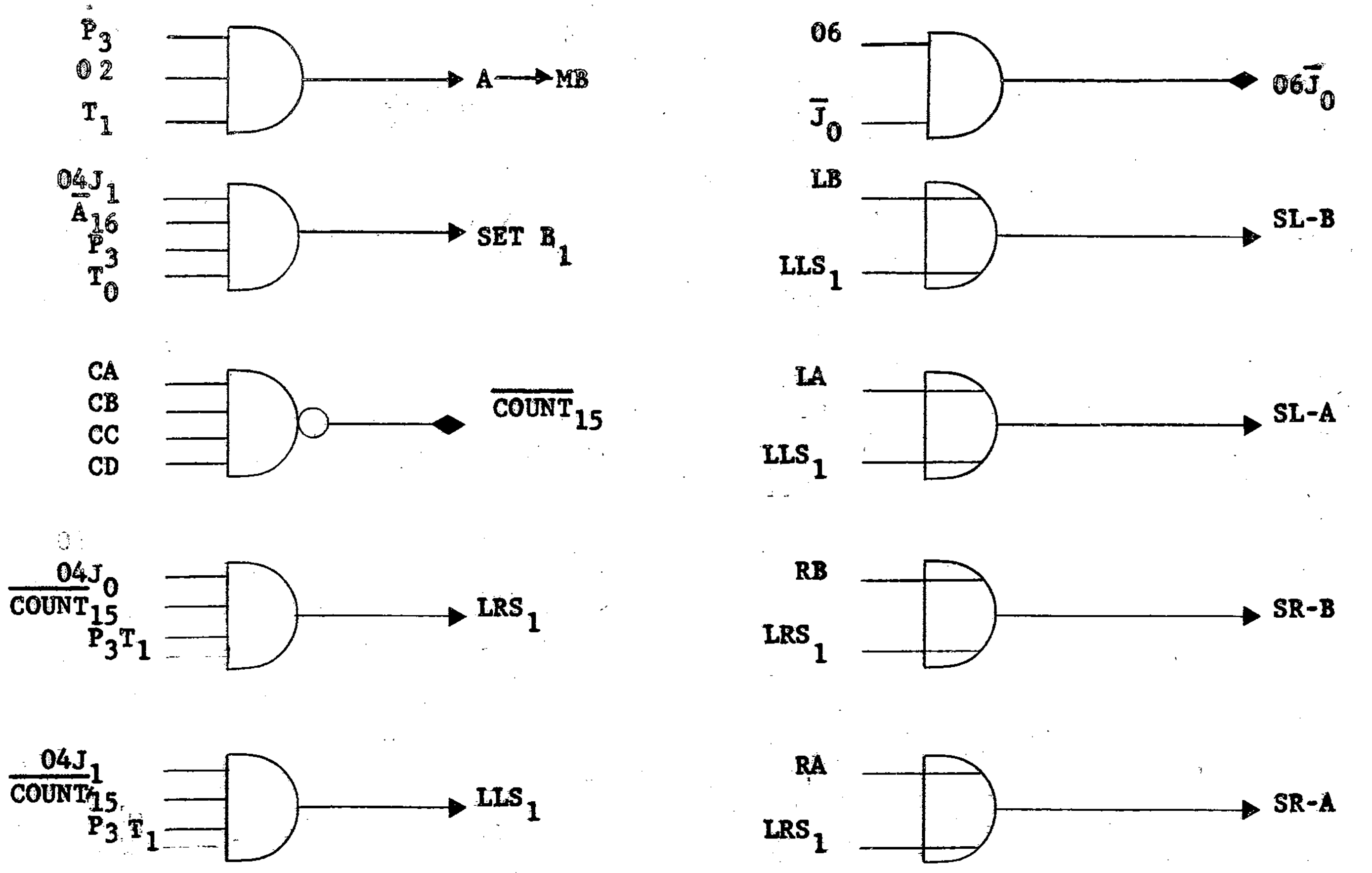


Figure 38. Shift Control

and they will be used to design the logic required. The CO control is set at P_1T_7, and the multiplicand will be read from memory at P_1T_8, and the OF control will clear at P_1T_{14}. The design for these operations has been discussed in earlier sections of this chapter. During P_2, the logic will direct the arithmetic unit to act as an adder and the multiplicand will be added to the A register if the decoded command is MUL. The computer must step through P_2 16 times to execute the MUL command. If an overflow is detected at P_2T_{15}, the OF control will be set. P_3 is used to control the logic for the MUL command, and also to decide if the command has been executed. At P_3T_1, the A and B registers will shift right 1 bit if this is a MUL command and the contents of the COUNT register are not equal to 1111. The logic for this is shown as LRS_1 in Figure 38. As has been explained in the section on control of the JUMP flip-flop, the JUMP control will be set at P_3T_1 if the COUNT is not equal to 1111 and this is a MUL command. This logic is shown in Figure 17. Each time the computer logic steps through P_2 in the execution of the MUL command, an indication is stored as an advance in COUNT at P_3T_2. If the command has not been executed, the control logic will jump phase and repeat P_2. The Phase Counter, and the Timing Counter must be reset at this time to go back to P_2T_0. The logic for this is shown in Figures 14 and 16. The computer will continue repeating P_2 until the count stored in COUNT is equal to 1111. The last time through the cycle, the JUMP control will not set, and the control will not jump phase at P_3T_2. Instead, the cycle will continue through P_3 and finish the execution of the instruction as explained in earlier sections of this chapter. The only difference between MUL, and ADD is the modification of control that causes the computer logic to add and then shift until the command is executed instead of adding only. The data control logic is shown in Figure 37, and is labeled SR-A DATA.

Divide Control

The divide instruction is usually the most difficult and time consuming instruction any general purpose digital computer will have to execute. However, if the non-restoring division technique is used, the operation is no more time consuming than the multiply instruction.[4] If the process of division is examined, on pencil and paper, a method for machine division can be developed. For example, the quotient that results from dividing 110 into 1100 may be determined as follows.

```
          11             10
    110 / 1100     110 / 1100

          110            110
          ---            ---
          0000           0000

           110            000
           ---            ---
         - 110          + 000
```

The quotient should be 10. This is determined on pencil and paper by a process of inspection, and guessing. In the first part of the example, the second trial division did not "go" as indicated by the negative difference between the divisor and the dividend. This tells the mathematician that the divisor will "go" some amount less than the guessed amount. In this case, the divisor will go 0 times as indicated in the second part of the example problem. Binary division may be summarized by recording a 1 in the quotient each time the divisor

[4] Thomas C. Bartee, Digital Computer Fundamentals, New York, 1960, p. 203.

"goes" into the dividend, and by recording a 0 each time the divisor does not "go".

Division in this computer will be accomplished by first placing the dividend into the B register by using a CLA and a LRS instruction. This will leave the A register cleared, and the dividend in the B register. During the execution of the DIV instruction, the divisor is read out of memory in P_1 and will be stored in the MB register for the remainder of the instruction. Therefore, three registers will be used in the divide instruction. They are the A, B, and MB registers. The basic steps required in the execution of the DIV instruction may be summarized as follows.

1. For the first step, the divisor will be subtracted from the A register.
    - a. Since the result for this first step will always be a negative difference, the A and B registers will be shifted to the left 1 bit.
    - b. For the next step, the divisor will be added to the A register.
    - c. If the result is positive, the divisor "went" into the dividend, and the least bit of the B register will be set to 1, and both registers will be shifted to the left 1 bit.
    - d. If the result is negative, the divisor did not "go", and the registers will simply be shifted to the left 1 bit.

After each trial, the computer control logic will examine the sign of the A register. If the sign is negative the cycle will revert to step b. If the sign is positive, the cycle will revert to subtracting the multiplicand from the A register, and then to steps c. or d. The computer control will repeat these trials 16 times for the 16 bits in the words. At the completion of the execution cycle, the quotient will be stored in the B register, and the remainder will be stored in the A register. The steps for the design of the computer logic to execute the DIV instruction are listed in the mechanization charts. The selection and decoding cycles are the same for all instructions. At P_1T_7, the CO control will be set, as shown in Figure 36. At P_1T_8, the divisor is read from memory and placed in the MB register as shown in Figure 23. The control logic will then go into P_2. At P_2T_0, the control logic will start subtracting the divisor in MB from the dividend in A. The logic that controls this is shown in Figure 34. The computer control will produce a SUB level since this is an $04J_1$ command and the sign bit A_{16} is 0 or positive. Therefore, the arithmetic unit will solve the problem X - Y and store the difference in the A register. Since the A register was cleared at this time, the sign bit of the A register will be 1. The computer control will go into P_3, and the following operations will occur. Since the count in COUNT is not 1111, the A and B registers will shift left 1 bit at P_3T_1. Nothing happened at P_3T_0, since the sign bit was negative. The JUMP control will be set at P_3T_1, since the count is not 1111 in the COUNT register. At P_3T_2, the count in COUNT will be advanced by 1, and the computer control will jump back to P_2 as described in the section on MULTIPLY CONTROL. The next time through P_2, the computer control in Figure 34 will tell the arithmetic unit to ADD X + Y, since the sign bit is negative. Then the computer control will advance to P_3, and either set or not set the least significant bit of the B register to 1 at P_3T_0, depending upon the sign bit in the A register. At P_3T_1 the A and B registers will shift left 1 bit, and at P_3T_2 the computer control will jump to P_2 as explained above. The computer control will make 16 trips through P_2 before the count stored in COUNT becomes equal to 1111. The subtract or add cycles will be controlled by the logic in Figure 34, and the quotient will be developed in the B register. After the COUNT reaches 1111, the control logic will advance past P_3T_2 and complete the execution cycle as described in the section on MULTIPLY CONTROL.

Arithmetic Unit Control

The execution of the arithmetic instructions such as ADD, CLA, SUB, INC, and DEC is only a simple variation of the same control logic. Since this is true, the design of all of these instructions will be explained in this one section. The CP1 instruction must be explained before explaining the CP2 instruction. Therefore, the CP1 instruction will be explained here. Since the CLR instruction is very simple, the design of CLR will also be shown here. All instructions have the same selection and decoding cycles as explained previously. P_1 is the same for all arithmetic instructions that do not use the operations control counter, and none of the instructions in this section require the use of this counter.

At P_1T_8, a memory cycle is initiated for all of these arithmetic instructions, and the logic for this is shown in Figure 23. At P_1T_9, the A register must clear if the instruction is CLA, CLR, INC, or DEC. This completes the execution of the CLR instruction, since the A register has been cleared. The A register is cleared in preparation for adding if this is a CLA instruction. It is also cleared for INC and

DEC in preparation for incrementing or decrementing the contents of a particular memory address. Also at P_1T_9, the A register will be complemented if this is a CP1 or a CP2 instruction. The logic for clearing A and complementing A is shown in Figure 22. The execution of the CP1 instruction is also complete at this time. At P_1T_{10}, the contents of MB are transferred into the A register in parallel if this is an INC or a DEC instruction. The logic for this is shown in Figure 22. The control logic will advance the computer to P_2. In P_2, the computer control will decide if the arithmetic unit will act as an adder or subtracter, and it will also decide what data will be used in the arithmetic unit. The logic for these controls is shown in Figure 34. The SUB level will be a logic 1 if the command decoded is SUB or DEC. Otherwise, ADD will be a logic 1. The data as shown in Figure 34 is determined by the particular instruction decoded. X will come from the A register for all of the instructions decoded in this section, since none of them will decode as 06CC. Y is variable, since it depends upon the particular instruction decoded in this section. M in Figure 34 is equal to the contents of the MB register if the instruction decoded is not MUL or not DEC or not INC or not CP2. In other words, the data will come from MB for the CLA, ADD, and SUB instructions. The data for the INC, DEC, and the CP2 instructions is equal to 1. Therefore, in P_2, MB will be added to A for the ADD or CLA instructions and MB will be subtracted from A for the SUB instruction. Also in P_2, 1 will be added to the A register if the instruction is CP2 or INC, and 1 will be subtracted from A if the instruction is DEC. The execution of all of these instructions is complete in P_2 except for INC and DEC. The computer control logic will advance the computer to P_3.

The P_3 mechanization chart outlines the logical design for the completion of these instructions. At P_3T_0, the MB register must be cleared. At P_3T_1, A will be transferred into the MB register in preparation for storing the incremented or decremented word if INC or DEC had been decoded. The logic for this is shown in Figures 22 and 38. At P_3T_3, a write cycle is initiated to store the incremented or incremented word into core storage. The logic for this is shown in Figure 23. At P_3T_4, MA is cleared, and at P_3T_5, C is transferred into MA in parallel. The logic for this is shown in Figure 22. This is the address of the next sequential instruction. At the end of P_3, the computer control will jump phase back to P_1 and the next instruction cycle will start at P_1T_0 of the instruction whose address is in MA.

Compare Control

The compare instruction is very useful to the user of the digital computer. It will be particularly useful in this computer since sign control must be determined by the programmer. The compare instruction (CMP) will turn on the HI indicator if the A register is greater than the contents of memory, the LO indicator if the A register is less than the contents of memory, or the EQ indicator if the A register is equal to the contents of memory. This instruction has been mechanized using the Operations Control Counter, and the add and subtract logic.

The selection and decoding cycles are the same as all other instructions. At P_1T_5, the HILO control will clear if the instruction decoded is a CMP instruction. The logic is shown in Figure 39. The Operations Control Counter, the CARRY flip-flop, and the CO control must also clear at this time. At P_1T_7, the MB register clears in preparation for the data to be compared. CO will be set at this time as shown in Figure 36. At P_1T_8, a memory cycle is initiated to bring the data out of memory. This logic is shown in Figure 23. Since this is an arithmetic instruction, OF will clear as shown in Figure 35. The computer control logic will then advance the computer into P_2 as explained in previous sections.

There are a number of possible methods that could be used to compare 2 binary numbers. The method selected for this computer is outlined below.[5]

1. Subtract MB from A. If the result is negative, MB is greater than A, and the LO indicator should be set. If the result is positive, another trial must be made. Zero in this computer is positive.

2. If the result of 1 was positive, A is either greater than or equal to MB. MB must be added to A to restore the original value in A.

3. Subtract A from MB, and if the result is negative, A is greater than MB. If the result is positive, A is equal to MB.

The first time through P_2 using the CMP command, MB will be subtracted from A. Since the 06CC level will be 0, MB will be subtracted from A. The logic for these controls is shown in Figure 34. The difference will be stored in the A register. The COUNT stored in the Operations Control Counter at this time is equal to 0000. At

[5]Fundamentals of Computers, Section 1, Symbolic Logic, Federal Aviation Agency, Oklahoma City, Oklahoma, 1962, p. 7-33.

P_3T_0, the LO indicator will set if the sign bit in the A register is negative. This logic is shown in Figure 39. If the difference were positive, the JUMP control would set at P_3T_1. This logic is shown in Figures 16 and 17. At P_3T_2, the computer control will jump back to P_2 if the result of the first subtraction was positive. In this case, the A register must be restored to its original value. Also at this time, the JUMP control will clear if it has been set, and the COUNT will advance to 0001 in the Operations Control Counter. The logic for these operations is shown in Figures 14, 16, 17, and 36. The control logic in Figure 34 will cause the arithmetic unit to add A + MB since $06\overline{CD}$ is equal to 0. Then the computer control will cause the computer to advance into P_3. Since this cycle was a restore cycle, the computer control will simply cause the computer to jump back to P_2 by setting the JUMP control at P_3T_1, and by jumping back to P_2 at P_3T_2. The JUMP indicator will be cleared, and the COUNT will be advanced to 0010 in the Operations Control Counter. The logic for these operations is shown in Figures 39, 14, 16, and 17. The A register has been restored to its original value. Since the first trial resulted in a positive difference, the next trial will require subtracting A from MB. The 06CC level is equal to a logic 1, and the control logic in Figure 34 will cause the arithmetic unit to subtract A from MB. The difference will be stored in the A register. The computer control logic will then cause the computer to advance to P_3. At P_3T_0, the HI indicator will be set if the sign bit of A is negative, as shown by the control logic in Figure 39. At this time, all possibilities have been considered, and the computer control will exit from P_3 into P_1 and start the instruction cycle of the next instruction as indicated by the contents of the MA register.

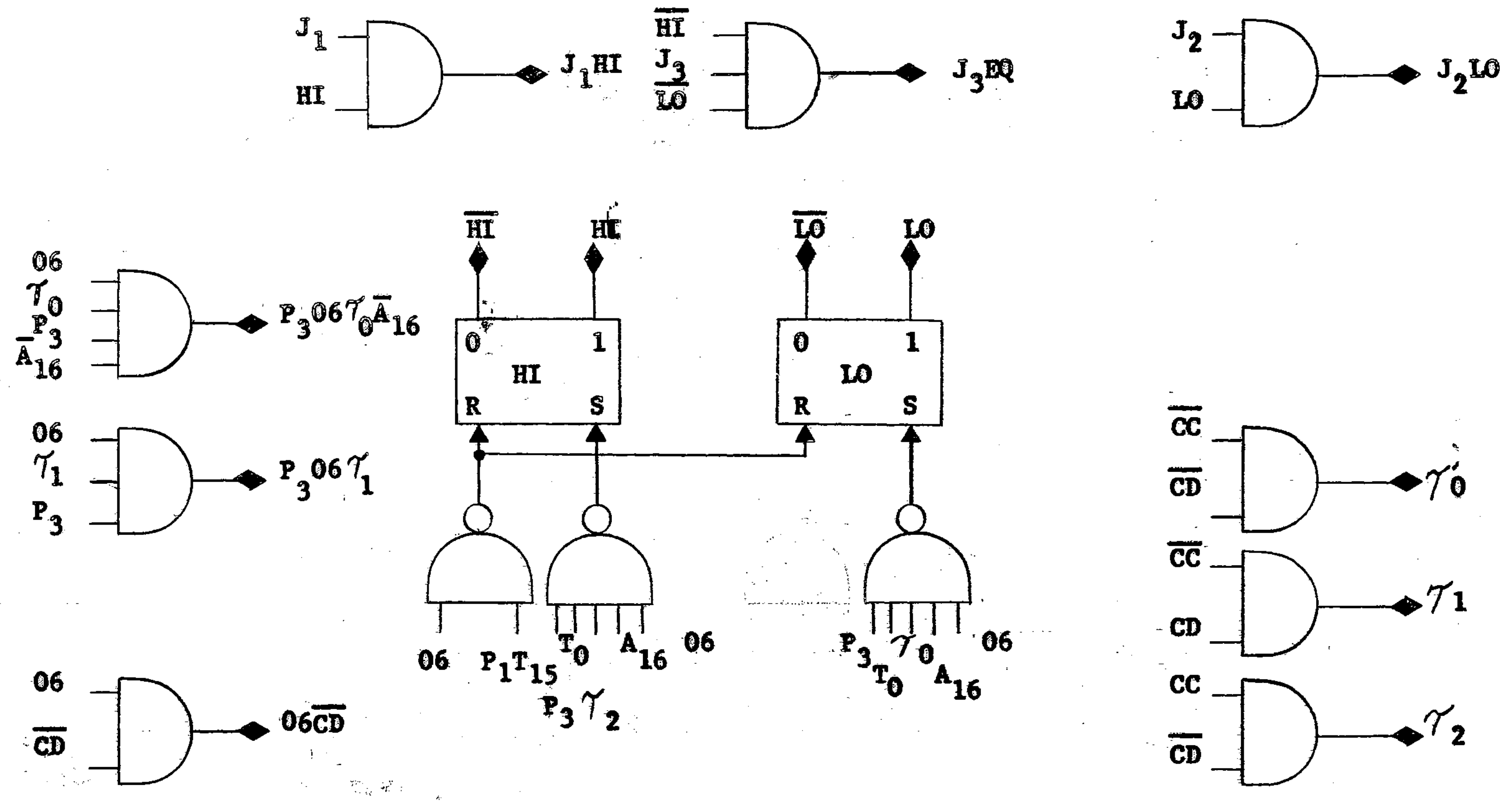


Figure 39. Compare and Controls

The contents of the HILO indicators show whether A is greater than, less than, or equal to MB. These indicators may be tested by the JHI, JLO, and JEQ instructions to determine the result of the compare. The method for testing these indicators was discussed in the section on JUMP CONTROL.

Store Control

The design of the store control logic is very simple. At P_1T_9, the contents of A must be transferred into MB, and at P_1T_{10}, a WRITE memory cycle is initiated to store the contents of MB into core memory. The JUMP control will be set at P_1T_{11}, as explained in the section on JUMP CONTROL and the computer control will jump phase and start on the next instruction cycle at P_1T_{15}.

Loading

This concludes the logical design of the computer. The loading of the various pulses and levels must now be considered. If not, the computer may not operate properly or may become marginal at the upper frequency limits of operation. The 1 and 0 levels of each flip-flop can drive 14 base loads. The And/Nand and Or/Nor modules are capable of driving 8 base loads. The master clock is capable of driving 8 base loads, and each pulse generator can drive 8 base loads. The loading of each module in the computer will have to be determined by simply counting the loads on each module. This is not quite as time consuming as it may seem since most modules are loaded by only one or two base loads maximum. As the computer is assembled, careful check must be kept of the loading, and as the load limit is exceeded in each case, a pulse or level amplifier must be inserted into the logic.

CHAPTER V

SUMMARY AND CONCLUSIONS

The methods used for the design of this computer may be summarized as a procedure for putting logical thought into action. These logical thoughts were obtained using several procedures. These procedures are listed as follows.

1. Intuitive Methods
2. Boolean Algebra
   a. Karnaugh Mapping
   b. Sequential Circuit Analysis

The mechanization charts were found to be a very useful tool for listing the ideas, intuitive reasoning, and the results of Algebraic simplification. They were a definite guide in organizing the many pages of logic that must be designed for even a very simple digital computer.

Commercially available modules were used to test the logical design of the various units of the computer. The results of these tests were very gratifying. It was found that this computer design was very useable up to master clock frequencies of 500 kilocycles per second. The designed logic starts becoming marginal at frequencies higher than this. This should have been expected since the modules used were 500 kilocycle modules.

The methods outlined in this thesis can be used to design varying types of digital devices. These methods can be used to design control devices for many manufacturing processes, such as the processes used in the building of aircraft, missiles, and the processing of such products as gasoline.

Proposed Sophistications

There are many changes and additions that might be made to the logical design of this computer. After each examination of the logic, a new or better way of doing one or more of the operations can usually be found. Of course, in the design of any device a point is reached where the changes in design must stop and a particular design must be chosen. There are a number of instructions that can easily be added to this design. Some of these are logical add, logical subtract, logical multiply, and indexing of instructions. The problems associated with input and output equipment were not considered in any great detail. The problems with these devices can be as difficult or even more difficult than the problem of designing the arithmetic unit and its control.

Applications could be found for this computer in the field of education. The units are easily constructed from commercially available modules. This computer can be easily programmed and could be used to teach programming, logical design, and the analysis of digital machines.

A SELECTED BIBLIOGRAPHY

Bartee, Thomas C. Digital Computer Fundamentals. New York: McGraw-Hill Book Company, Inc., 1960.

Burroughs Corporation. Digital Computer Principles, New York: McGraw-Hill Book Company, Inc., 1962.

Humphrey, Watts S., Jr. Switching Circuits with Computer Applications. New York: McGraw-Hill Book Company, Inc., 1958.

Smith, Charles V. L. Electronic Digital Computers. New York: McGraw-Hill Book Company, Inc., 1959.

APPENDIX

SYMBOLS USED IN THE LOGICAL DESIGN

The following AND/NAND and OR/NOR symbols were used in the logical design of the computer. They represent types of hybrid gates.[6] The AND/NAND gate is shown in Figure 40, and the OR/NOR is shown in Figure 41.

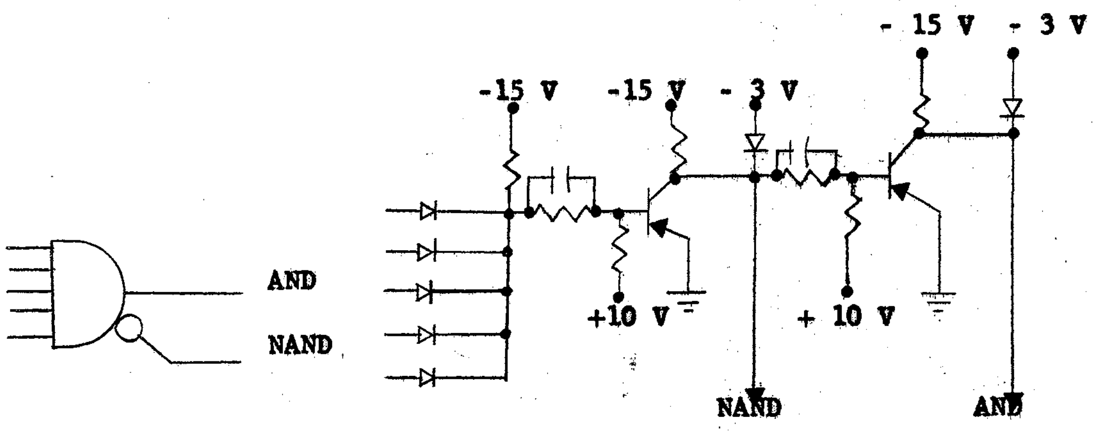


Figure 40. AND/NAND Circuit

[6]Burroughs Corporation, Digital Computer Principles, New York, 1962, p. 158.

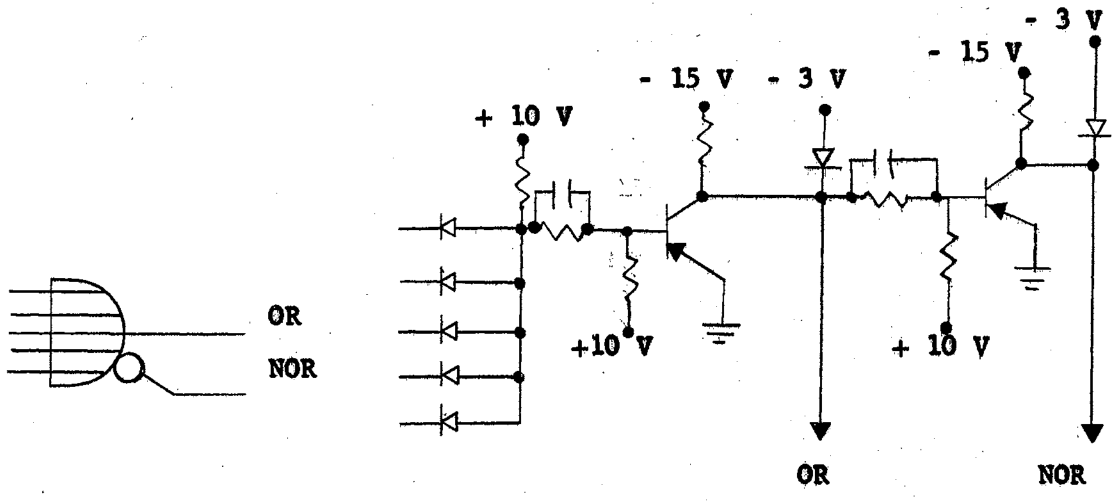


Figure 41. OR/NOR Circuit

The symbol shown in Figure 42 was used as an inverter, and was also used to gate pulses and levels into the set, reset and complement inputs of the bistable multivibrators.

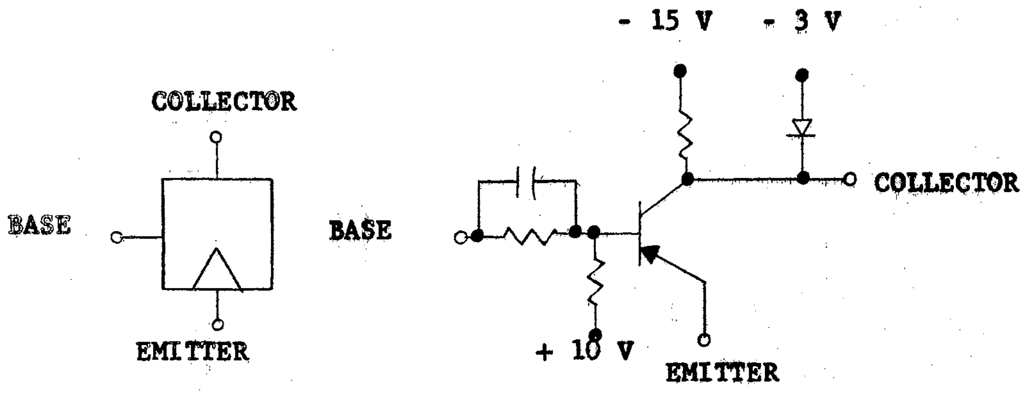


Figure 42. Inverter Circuit

The following symbols in Figure 43 are used as pulse and level indicators.

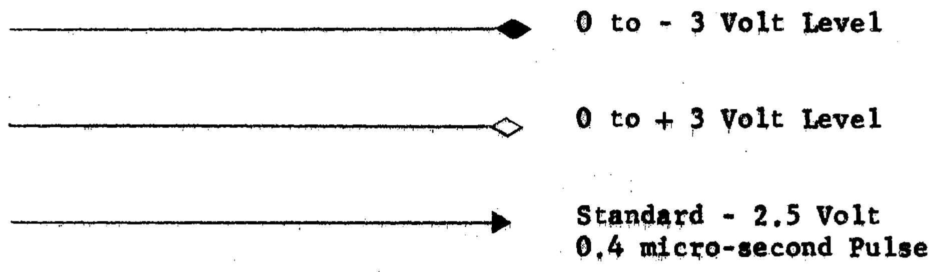


Figure 43. Level and Pulse Indicators

VITA

David Wayne Fleming

Candidate for the Degree of

Master of Science

Thesis: THE LOGICAL DESIGN OF A SIMPLE MAGNETIC CORE DIGITAL COMPUTER

Major Field: Electrical Engineering

Biographical:

Personal Data: Born at Harrah, Oklahoma, July 14, 1931, the son of D. A. and Beulah Mae Fleming.

Education: Attended grade school and high school in Ponca City, Oklahoma. Graduated from Ponca City High School in 1949. Received the Bachelor of Science degree from the Oklahoma State University, with a major in Electrical Engineering, in May 1958; completed requirements for the Master of Science degree in May, 1963.

Professional experience: Entered the United States Navy, in 1951, and was an Electronic Technician until discharged in 1955. Was employed as a Graduate Assistant in the Electrical Engineering Department, Oklahoma State University in 1958. Was an Instructor in the Electrical Engineering Department of Oklahoma State University, from January 1959 to May 1960. Is presently employed by the Federal Aviation Agency, Oklahoma City, Oklahoma, as the Technical Assistant to the Branch Chief of the Communications Equipment Branch. During this period from May 1960 to the May 1963, was responsible for the analysis and design of a training digital computer. Was an instructor in the Department of Engineering Extension, Oklahoma State University, from September 1961 to May 1963.