"DESIGN AND APPLICATIONS OF A PROGRAMMABLE BINARY DECISION PROCESSOR"

A. S. K. DURRANI

A DISSERTATION

IN THE

FACULTY OF ENGINEERING

Presented in partial fulfillment of the requirements
for the degree of MASTER OF ENGINEERING at the Concordia
University, Montreal, CANADA

APRIL, 1979

ABSTRACT

DESIGN AND APPLICATIONS OF A PROGRAMMABLE BINARY DECISION PROCESSOR

A.S.K. DURRANI

A large number of the problems found in controlling electronic and electromechanical devices involve decision oriented tasks. There are, many ways to solve these types of problems. Originally, conceptually simple and easily maintained relays were used extensively. However, in hardwired relay logic systems, control sequence is determined by the way system components are wired together. To change the sequence requires the time consuming chore of rewiring parts of the system. Next came programmable controllers as a replacement for the random relay logic. They consists of a standard control circuit and a program memory. Most operate by stepwise evaluation of the encoded Boolean sum of products expressions describing the switching functions. Programming and implementation are extremely simple, and they permit to replace random logic by stored program control in a straight forward fashion. The evaluation of a Boolean expression, however, takes a large amount of program steps. The number of program steps required for each expression is an exponentially increasing function of the number of input variable. The concept of Binary Decision programs makes it possible to evaluate each switching function in a number of steps that does not exceed the number of input variables. The corresponding processor, henceforth, called the Programmable Binary Decision Processor (P.B.D.P.), whose structure will be described here, can therefore be used in cases where speed requirements render the usual programmable controller unsuitable. The PBDP executes a Binary decision program for all outputs in parallel as compared to a machine based on a serial execution. This difference leads to a faster execution

of multiple output functions. Moreover, the architecture of PBDP is considerably simpler than that of a programmable controller, resulting in a lower cost. The PBDP is applicable where a minimcomputer would be too expensive or too slow, too cumbersome to program or too complicated to understand.

ACKNOWLEDGEMENTS

The author would like to express his deepest appreciation to Dr. E. Cerny for his guidance and suggestions throughout the course of this work.

The author greatfully acknowledges the understanding accorded to him by his wife who has been a great source of inspiration throughout the course of his studies and not forgetting his two sons who have been cautioned a million times to lower the 'noise pollution'.

The author also wishes to acknowledge Mrs. K. Golwalla for her friendly offer in typing the manuscript.

TABLE OF CONTENTS

LIST OF FIGURES

PAGE

LIST OF TABLES

LIST OF SYMBOLS

AS	Address Register
BDP	Binary Decision Program
CLK	Clock
CMOS	Complimentary Metal Oxide Silicon
Cext	External Capacitor
CS	Chip Select
D	Data Pin
EPROM	Erasable Programmable Memory
f_1,Q	Output
ICU	Industrial Control Unit
IDS	Instruction Data Selector
IC	Integrated Circuit
I/O	Input Output
INH	Inhibit
FO	Frequency Output
HZ	Hertz
LED	Light emitting diode
LP	Link Platform
OPCODE	Operation code
OD	Output Disable
OL	Output Latch
PC	Program Counter
R/W	Read/Write
MOS	Metal Oxide Silicon

T_{CD}	-	Chip select delay time
T_A	-	Read access time
nS	-	Nano Second
uS	-	Micro second
MHZ	-	Megahertz
LSB	-	Least significant bit
MSB	-	Most significant bit
RAM	-	Random access memory

x

CHAPTER 1

INTRODUCTION

1.1 General

It has been shown by Shannon(1) that design and manipulation of switching circuits using Boolean Algebra is possible. He showed how relay switching circuits can be described by Boolean expressions and manipulated according to the rules of Boolean algebra. In 1938 his paper came as a great stimulus, and the algebraic (Boolean) method was further developed and many techniques, (such as minimization, hazard elimination etc.) were based on this method.

Since then, efforts were directed to find an alternate and more efficient way to represent, manipulate and realize switching functions. The realization of Boolean functions by binary decision programs as an alternate approach was most likely first proposed by Lee(2), for implementations using relay networks.

Both representations were aimed at the design of fixed (hard-wired) relay logic in a systematic fashion. At that time, the state of technology did not allow to consider the implementation of any of these descriptions in a stored machine, except in a computer, an uneconomical solution as compared to hard-wired logic.Compared to a stored machine or Programmable realizations, hard-wired logic has a definite speed advantage. However, in hardwired logic systems, control sequence is determined by the way system components are wired together. To change the sequence requires the time consuming chore of system redesign. But since the programmable realizations stores the control sequence in memory, changes can be made more quickly and easily. Therefore, hardwired logic although found to be faster than programmable logic lacks the important requirement of being flexible (In applications where speed is not the prime factor.).

The present-day availability of relatively cheap semi-conductor control (microprocessors) and storage (RAM, ROM) devices has drastically changed the situation. Programmable controllers are widely used in industry for implementing random logic by means of standard devices (e.g. program memory), especially in applications where the higher processing power of microprocessor is not required.

Present day Industrial programmable controllers, e.g., Motorola MC14500B Industrial Control Unit (ICU), operate by stepwise evaluation of the encoded Boolean sum of products expressions describing the switching functions (stored in Memory) required by the applications. In section 4.2. chapter 4 we will show, how a Boolean expression, such as, $Y = \overline{X}_1X_2 + X_1X_3\overline{X}_4 + \overline{X}_2X_4$ can be expressed as a program which can be loaded into memory after encoding in binary form. Programming and implementation are thus extremely simple, and they make it possible to replace random logic by stored program control in a straight forward fashion. The evaluation of a Boolean expression, however, takes a large amount of program steps. It can be shown that the number of program steps required for evaluating each expression is an exponentially increasing function of the number of input variables.

Binary Decision programs (2) make it possible to evaluate each switching function in a number of steps that does not exceed the number of Input variables.

Binary decision programs, as we will see are not algebraic in nature. They are, therefore, less easily manipulated by standard Boolean techniques. The programmable realizations based on Binary Decision Programs can be simplified not by minimizing its hardware, but rather by finding for it a faster and shorter decision program.

Early electronic switching circuits then in fact suppressed the idea of any machine realization based on Binary decision programs, and only recent technological changes renewed some interest in the approach. These were the availability of inexpensive and fast read-only memories, as well as the need for regularity and uniformity in the design of LSI circuits. The first one lead the way to the R.T. Boute's binary-decision machine(3), which serially executes binary decision programs, thereby evaluating local values of the corresponding Boolean function.

Recently Cerny, Mange, and Sanchez(4) have extended the concept of binary decision trees which permits simultaneous evaluation of multiple output functions, and they have proposed a minimization technique for binary decision trees realizing multiple output incompletely specified functions.

In this work, a Binary Decision processor is designed to execute a program for all outputs in parallel as compared to serial execution of the output functions on the R.T. Boute's machine(3). This difference leads to a faster execution of multiple output functions, although the relative memory requirements depend on their individual properties.

The method proposed in (4) provides a systematic approach to programming this new machine. In the discussion to follow, we have choosen to call this realization as "Programmable Binary Decision Processor" (PBDP). The P.B.D. Processor is applicable where a minicomputer would be too expensive or too slow, too cumbersome to program or too complicated to understand. For instance, simple industrial controllers(5), test machines(6) or any other programmable sequencer carrying out decision oriented tasks.

CHAPTER 2

BINARY DECISION PROGRAMS

2.1. Structure of a Binary Decision Program

A binary decision program is based on a single instruction:

X; A, B

Where

X is the input variable

A and B are jump addresses

The instruction says that, if the input variable X is zero, take the next instruction from program address A, and if X is one, take the next instruction from address B. Every binary decision program is made up of a sequence of instructions of this kind. Take, for example, the contact network shown in fig(1).

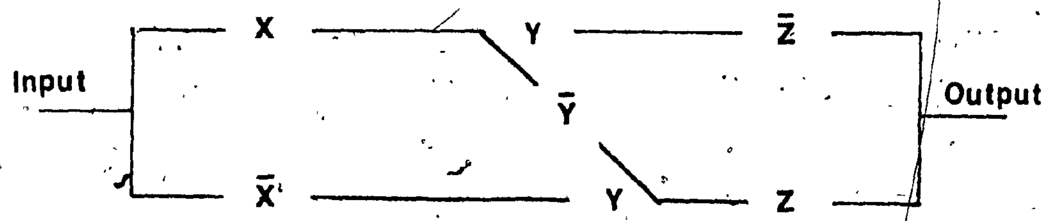


Fig. (1) Typical Contact Network

This circuit is described by the following binary decision program:

program:

PROGRAM ADDRESS	INPUT, VARIABLE NAME	JUMP ADDRESS IF VAR=0	JUMP ADDRESS IF VAR=1
(1)	(X)	(2)	(4)
(2)	(Y)	(6)	(3)
(3)	(Z)	(6)	(7)
(4)	(Y)	(3)	(5)
(5)	(Z)	(7)	(6)
(6)	OUTPUT = 0	GO TO ADD. (1)	
(7)	OUTPUT = 1	GO TO ADD. (1)	

The above program is actually a sequential description of possible events that may occur. We begin at program address 1 by examining the variable X. If X should be 0, we go to address 2 and examine Y. If Y is 0, we go to address 6, otherwise we go to address 3, and so forth.

A binary decision diagram based on the above program is shown in fig(2).

In this work, it is not our intention to carry on a detailed study of the Binary Decision programs, and as a concluding remark we can say that the concept of binary decision programs makes it possible to evaluate each switching function in a number of steps that does not exceed the number of input variables (4),(7),(8),(9).

2.2. Simplification of a Binary Decision Porgram

Cerny, Mange, and Sanchez[4] have recently extended the concept of

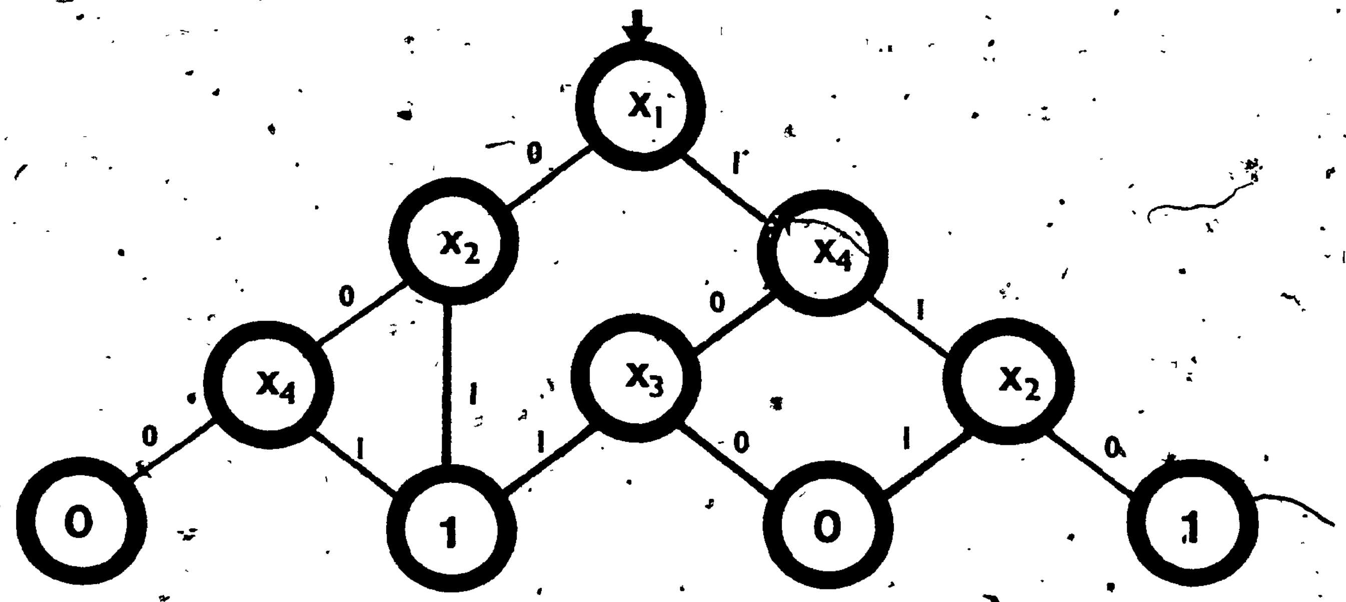


Fig. (2) Binary Decision Diagram

a binary decision tree to include trees representing multiple output incompletely specified functions and their minimization techniques. The following example is choosen to demonstrate that how a further simplification of an existing binary decision program could be achieved (10).

A comparator with two input variables and three possible outputs is shown in fig(3).

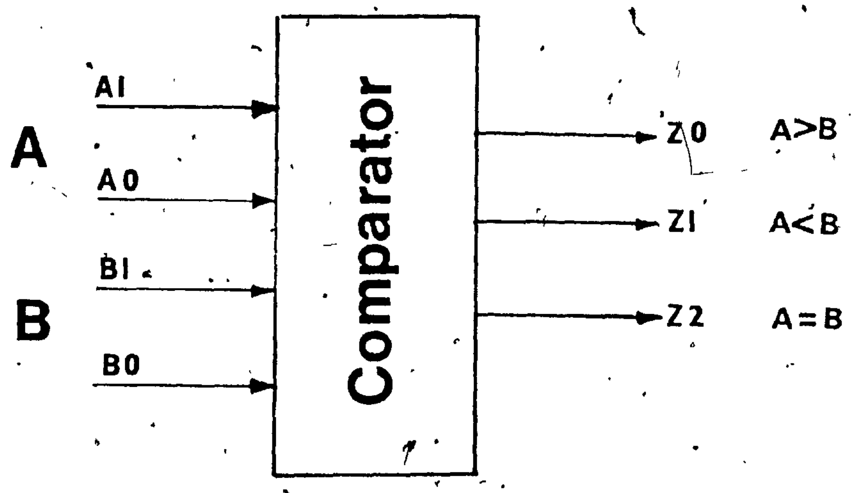


Fig. (3) Comparator Block Diagram.

The inputs to the comparator are A and B, both being two decimal numbers and each represented by two binary bits. Either A is greater than B (A>B), equal to B (A=B) or is less than B (A<B). Depending on any of these tests three possible outputs Z_0,Z_1 & Z_2 are obtained. Table 1 gives the binary representation of two inputs i.e. A and B

Table 1 Binary Representation of two Decimal numbers A & B

A_1	A_0	A
B_1	B_0	B
0	0	0
0	1	1
1	0	2
1	1	3

One possible truth table for these functions is shown in Table 2.

NO.	A_1	B_1	A_0	B_0	Z	Z_2	Z_1	Z_0
0	0	0	0	0	2	1	0	0
1	0	0	0	1	1	0	1	0
2	0	0	1	1	0	1	0	0
3	0	1	0	0	2	1	0	0
4	0	1	0	1	1	0	1	0
5	0	1	1	0	1	0	1	0
6	0	1	1	1	1	0	1	0
7	1	0	0	0	1	0	1	0
8	1	0	0	1	0	0	0	1
9	1	0	1	0	0	0	0	1
10	0	0	1	1	0	0	0	1
11	0	1	0	0	0	0	0	1
12	0	1	0	1	2	1	0	0
13	0	1	1	0	1	0	1	0
14	0	1	1	1	0	0	0	1
15	1	1	1	1	2	1	0	0

Table 2 Comparator Truth Table

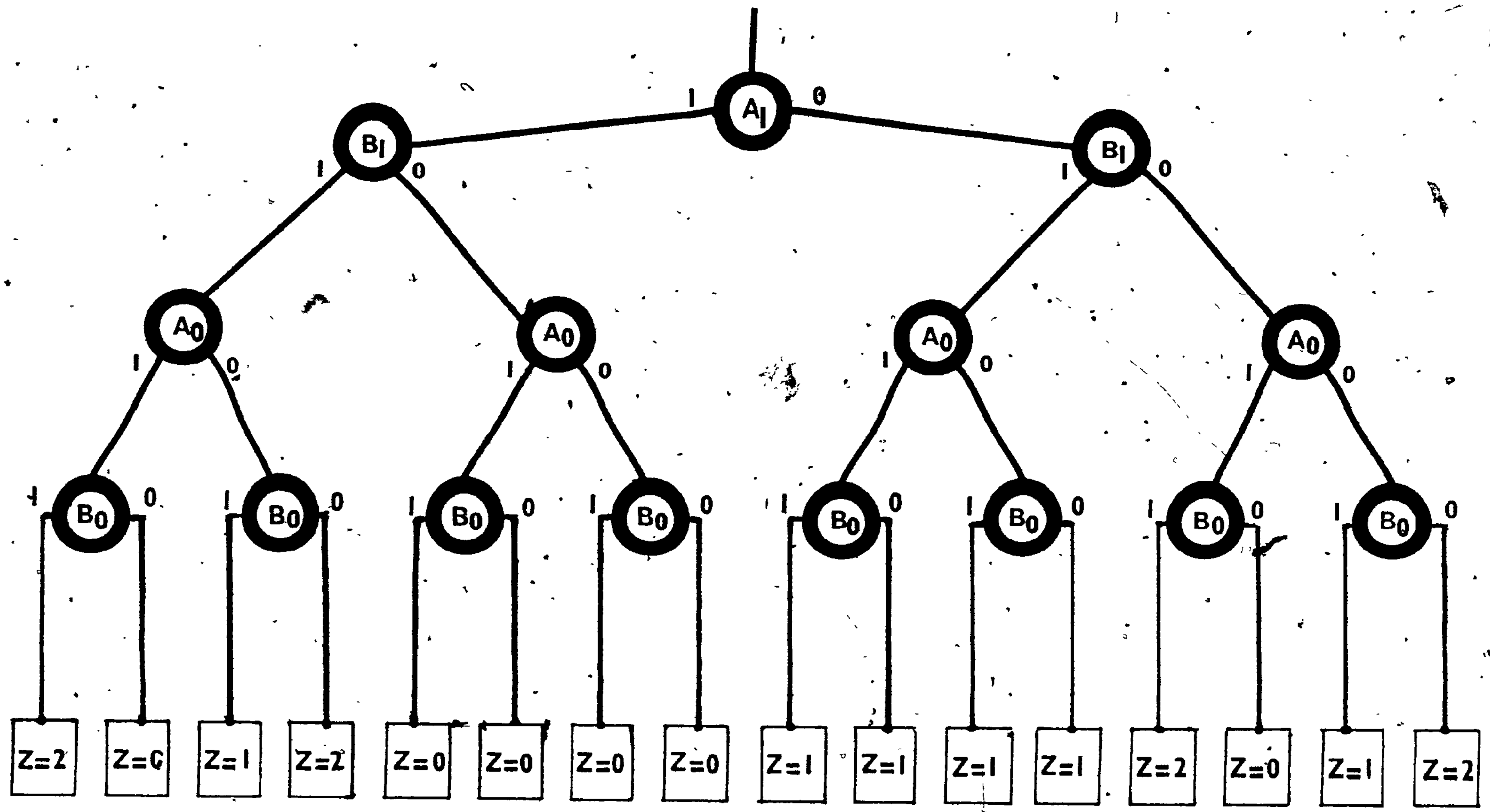


Fig. (4) Binary Decision Tree for the Comparator Example

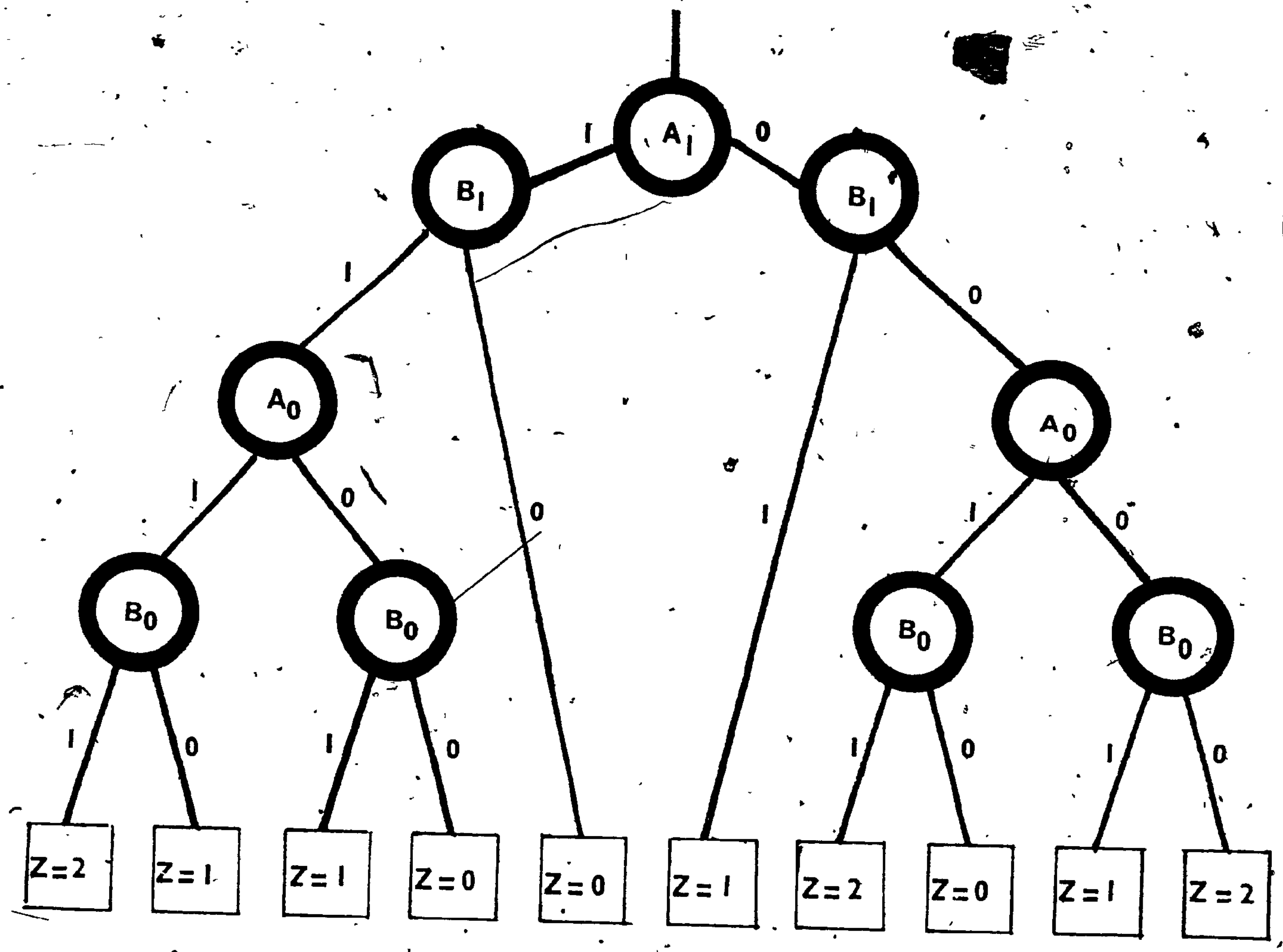


Fig. (5) Reduced Binary Decision Tree for the Comparator Example

By applying Shannon's Expansion theorem (1,7), a full binary decision tree is obtained as shown in fig(4). It can be seen from the diagram and also from Table 2, that for A_1; B_1 = 01, the value of Z=1, and is independent of values of A_0 and B_0. Similarly, for A_1,B_1 = 10, the value of Z=0, and is also independent of values of A_0 and B_0. Taking this fact into account a simplified binary decision tree is obtained as shown in figure(5). This greatly reduces the number of instructions to be used in a Binary decision program. In order to reduce the number of instructions in a program (decrease memory requirements) or number of multiplexers in a network, various optimization methods were developed (4,11,12).

CHAPTER 3

SYSTEM DESIGN

3.1. General

The purpose of this chapter is to briefly discuss the Boute's sequential machine(3), to list its various features, and then to describe the design of the Programmable Binary Decision Processor.

3.2. R.T. Boute's Machine

Boute's realization is based on the structure of a binary decision program and, therefore the evaluation of a Boolean expression does not require a large amount of program steps as compared to a conventional programmable industrial controller (5,13,14). For example, an expression with 8 variables may require about fifteen implicants, with an average of four literals per implicant. For such an example an industrial controller will require 60 program steps (3). Therefore, an industrial programmable controller will require more execution time.. If the same expression is translated into a binary decision program, the number of program steps are reduced, resulting in faster program execution. As Boute(3) pointed out that the main problem resides in complete lack of branching capabilities, which necessitates the complete evaluation of each implicant, even though most of them have the logical value "false" for given input configuration. Fig(7) shows a block diagram of Boute's machine which evaluates one output variable at a time.

The Blocks are:

. Logic gates, or Central-controller of the system.

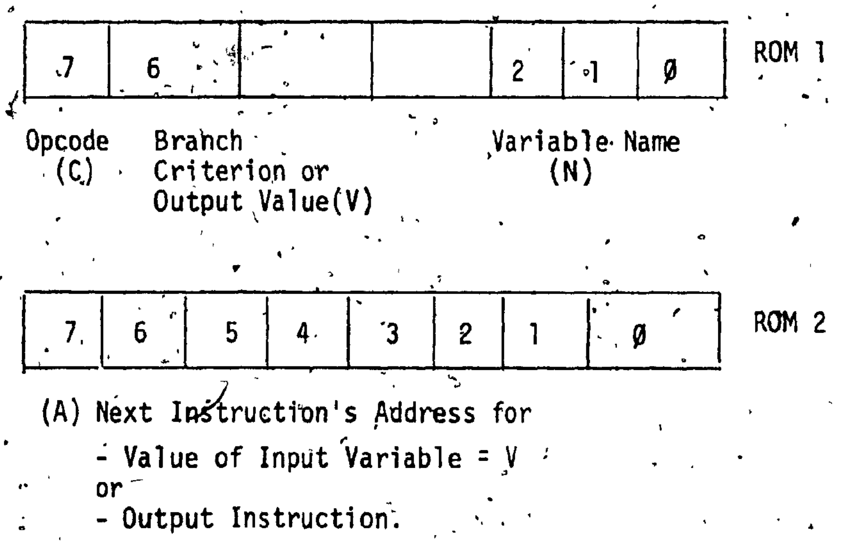
. The ROM'S, here, the steps of the programs are stored, the individual instructions and the addresses of the inputs and outputs.

. The presettable program counter, used to step/preset the machine through the sequence of program instruction.

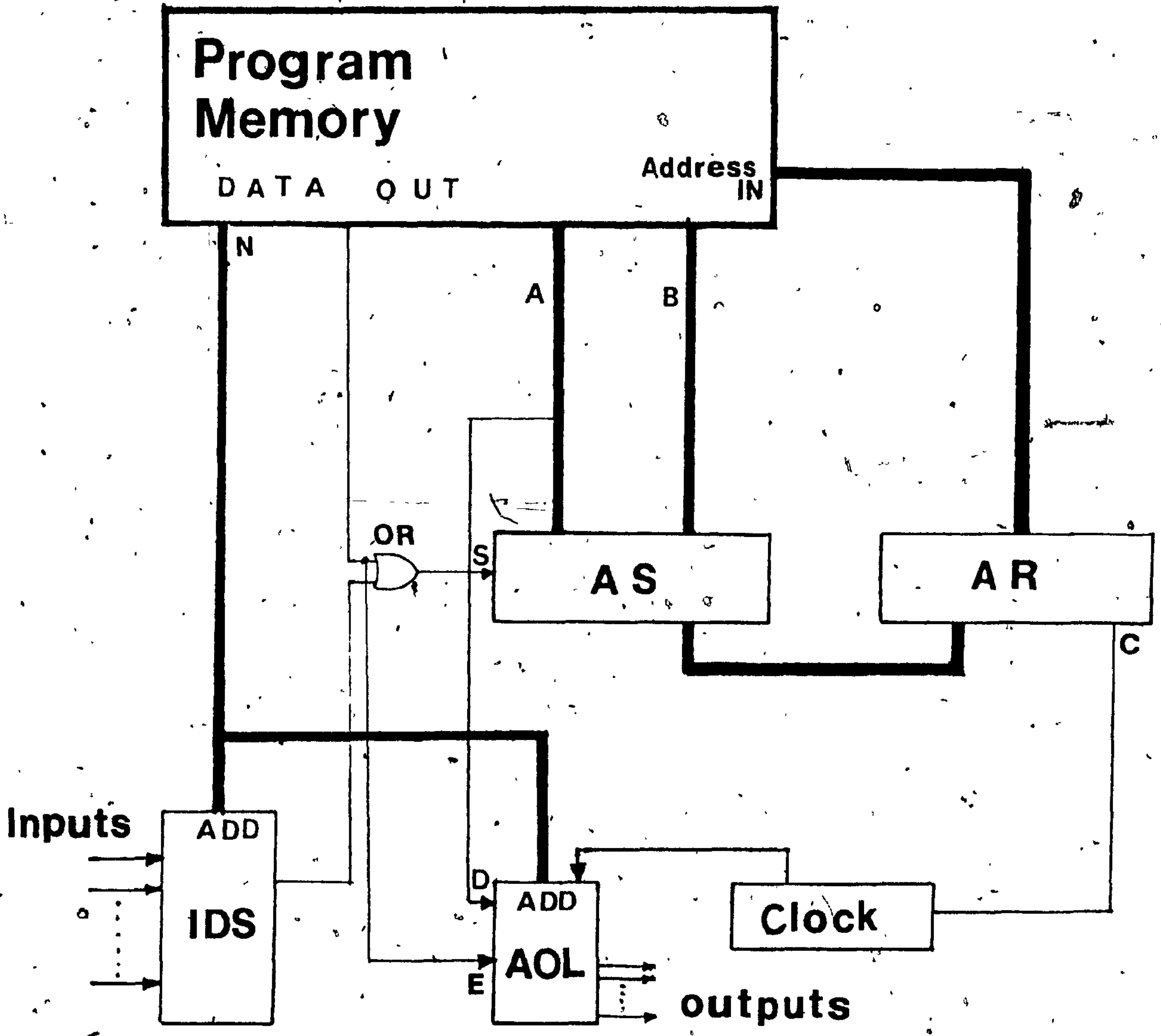
. Inputs and outputs, each individually selected by the machine, from information contained in the memory.

. System clock, generated by an external oscillator.

Boute's realization can be best explained by its instruction set (see figure(6), showing instruction formats); consisting of only a branching and an output instruction.


7
6
2
1
Ø
ROM 1
Opcode (C)
Branch Criterion or Output Value(V)
Variable Name (N)
7
6
5
4
3
2
1
Ø
ROM 2
(A) Next Instruction's Address for
- Value of Input Variable = V
or
- Output Instruction.

Fig(6): Instruction Formats, Boute's Machine



AS	=	Address Selector
AR	=	Address Register
IDS	=	Input Data Selector
AOL	=	Addressable Output Latch

Fig. (7) Boute's Binary Decision Machine

Let 4 tuple (C,N,A,B) describe a binary decision machine instruction, where:

C: OPCODE, e.g.

C = Ø for a branching Instruction

C = 1 for an output Instruction.

N: The name (number of the logical variable considered (input variable if C=Ø, output variable if C=1).

A,B: For a Branching Instruction:

A = address of the next instruction in case the tested variable (input)is "False" (Ø).

B = branching address for "True" (1).

For an Output Instruction:

A = logical value to be assigned to the (output) variable (Name).

B = the (unconditional) address of the next instruction.

Let us note here that it is not always necessary to specify the branching address explicitly. For example, in the case of a branching instruction, the branching address for variable (N) = "False" may be specified implicitly by the convention that it equals the memory address of the instruction plus one.

From the previous discussion we have noted that the Boute's machine has the architecture of a sequential machine which serially executes binary decision programs, thereby evaluating local values of the corresponding Boolean functions.

The Programmable binary decision processor is a modification of the Boutes sequential machine resulting in a Binary decision processor. which:

. Executes Multiple Outputs in parallel.

. Uses RAM's and, therefore, may be programmed repeatedly for developing and testing programs before they could be burned into ROM's or PROM's.

3.3. Instruction Repertoire and General Processor Organization

It can be seen from the structure of a binary decision program that only two commands are needed to execute the entire program, namely a branching instruction and an output instruction, that is, in its most elementary form. However, one could add provisions such as Sub.routine call instructions etc, but as stated in Chapter 1, the processor has to be as simple as possible for the intended purposes.

The following set of instruction formats are chosen for the processor. We will call them Instruction Register (IR) and Address Register (AR) formats. Both formats are shown in fig(8).

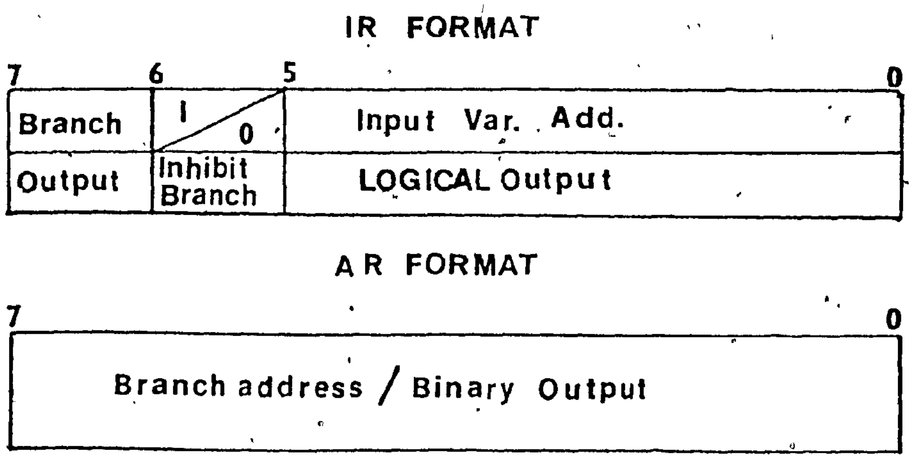


Fig. (8) Instruction Formats PBDP

Explanation of each term follows:

Input Variable Address: Bits (0-5) of IR are reserved for the address of the input variable.

Logical Output Values: Bits (0-5) of IR are reserved for logical output values if it is an output instruction.

Operation Code:

If bit 7 of IR is set and if bit 6 is set then next instruction is taken from the next location and the Bits 0-7 of AR of the current instruction is taken as Output together with bits 0-5 of IR. If bit 6 of IR is cleared then output consists of bits 0-5 only and the next instruction address is taken from the AR of the current instruction.

If bit 7 of IR is cleared and bit 6 is set then branch takes place (address in AR) if addressed variable is true, else the next instruction is taken. If bit 6 of IR is cleared then branch takes place if addressed variable is false, else the next instruction is taken from the next location.

Branch Address

Bit (0-7) of the address register (AR) are treated as the Branch address or output values.

3.4. Execution Sequence Summary

The execution sequence of the PBDB may be summarized in the following two phases.

Loading Phase - In this phase the binary decision program is loaded into the memory.

Executing Phase - Executing phase can be best explained with the aid of the PBDP instruction execution flow chart as shown in figure (9). The flow chart is summerized as the following:

. Initialize the program Counter (PC)

. Initialize the output Latches (OL)

. Load PC onto the Memory address lines.

. Make binary decisions and act accordingly.

3.5. Program: Execution Sequence.

The following program is based on the flow chart as shown in figure (9).

```
BEGIN
    PC ← Ø                          Initialize the program Counter (PC)
    OL ← Ø                          Initialize the output latches (OL)
    IR ← PC                         Load PC into Instruction Register (IR)
    AR ← PC                         Load PC into Address Register (AR)
    IF  IR(7) = 0 then              See Note (1)

      BEGIN
          IF (IR(6)=Xi) then        See Note (2) (Xi=Addressed Input
                                                  variable)
              PC ← AR
                 else
                    PC ← PC + 1     Increment PC
```

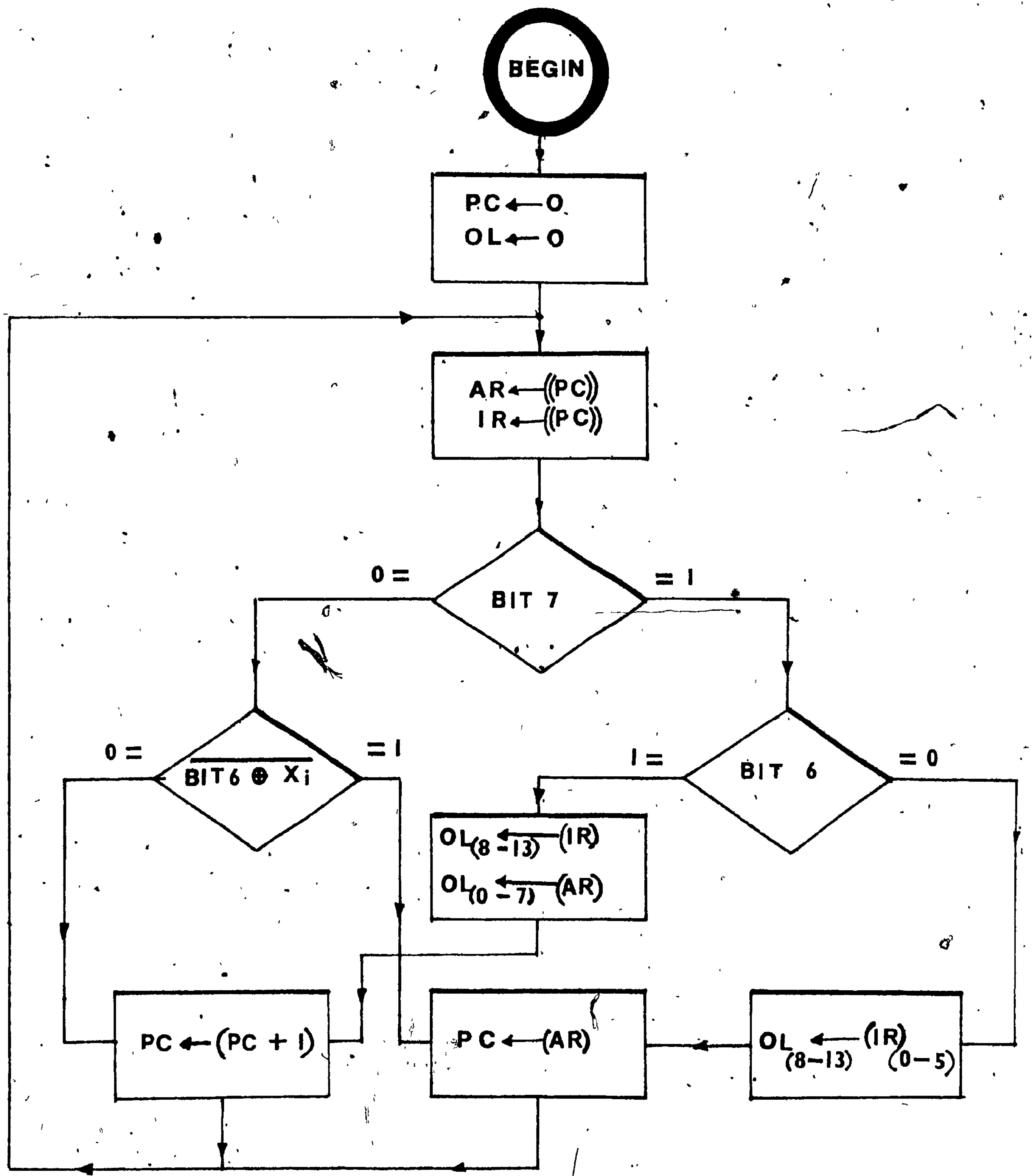


Fig. (9) PBDP Instruction Execution Flow Chart

```
END

      else
      BEGIN
         IF (IR(6)=0) then
                  BEGIN
                     OL(8-13) ← IR(0-5)          See note (2)
                     PC ← AR
                  END

      else
        BEGIN
         OL(8-13) ← IR(0-5)                      Load(IR) bits 0 to 5 into
                                                 OL
         OL(0-7) ← AR(0-7)                       Load(AR) bit 0 to 7 into
                                                 OL(0-7)
         PC ← PC + 1
        END
     END
END
```

Comments: Note (1). If the Bit 7 of the IR is set to 0 then this is treated as a jump instruction and depending on the status of the input variable appropriate action takes place.

Note (2). Exclusive NOR bit 6 of IR with Xi (2^6 possible inputs: i=0 to 63). If the result is 1, then the tested variable is equal to IR_6 and a jump address is loaded into the memory. If, on the other hand, the result is a logical 0 then the program counter PC is simply incremented by one. If bit 7 of IR is one, then the content of the IR is loaded into the output latches (OL) i.e. bits 8 to 13. Moreover, if bit 6 of IR is one, then also memory data from AR is loaded into OL.

3.6. Hardware System Design

The block diagram in fig. (10) is a block diagram of the Programmable Binary Decisions Processor (PBDP). The components are composed of Standard TTL/CMOS parts, except for the memory. The system operates on the principle of a stored program processor. A set of instructions reside in the memory. The system "fetches" an instruction from the memory and executed them. After executing an instruction, depending on the logic states, another instruction is fetched from the memory, and the process is repeated ad infinitum.

As an example, a typical instruction might be, to read the logic level (logic 1 or logic 0) of an input variable, to carry out the binary decision, to jump or to display output, as appropriate.

Using an instruction format as shown in figure (8) a program is loaded into the memory. The operation of the system is as follows: The system memory supplies the control circuit with an opcode as shown in block diagram figure (10), also, it addresses the input selector. The level of the selected input is then transferred over to the control circuit.

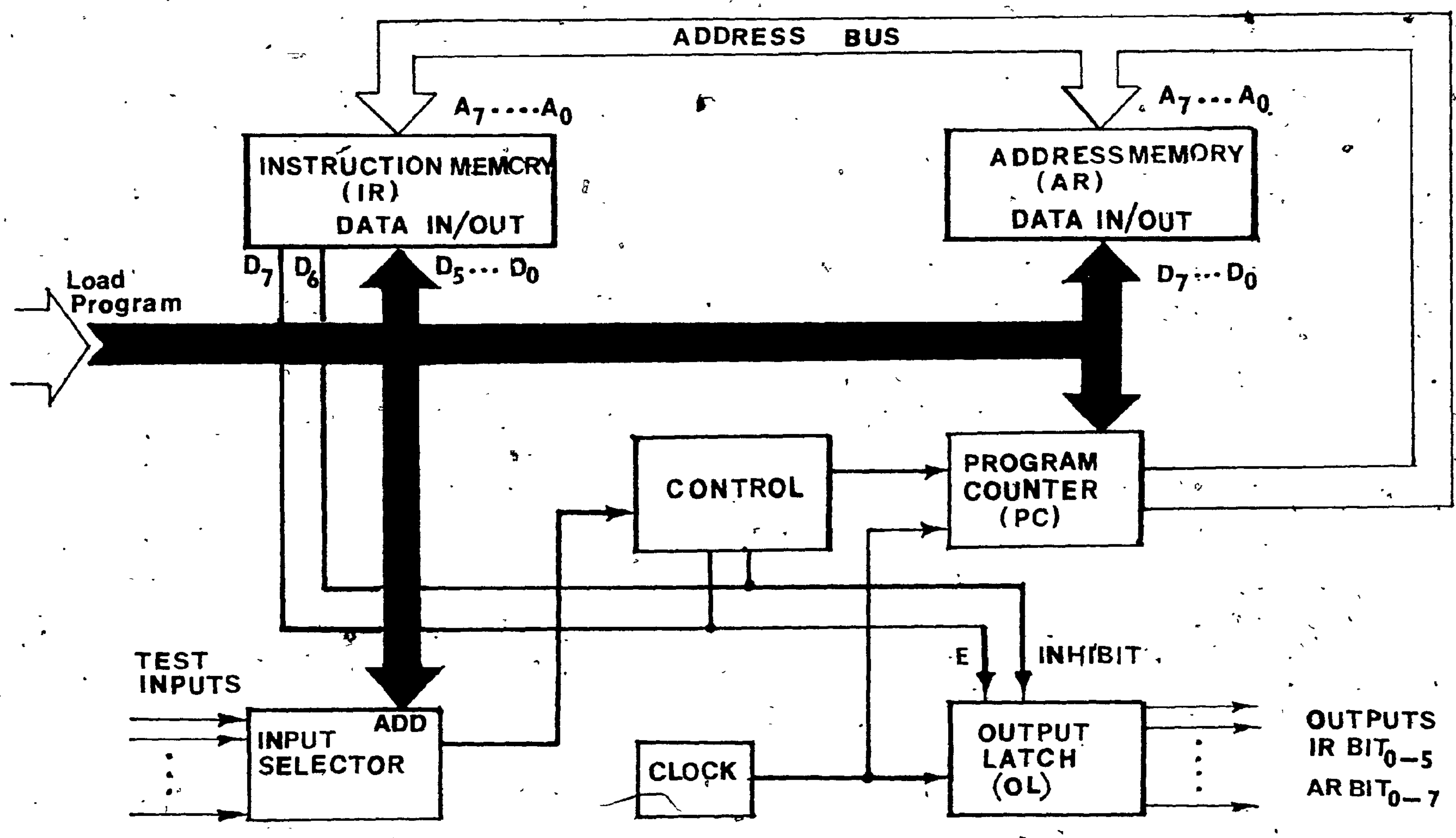


Fig. (10) Architecture of PBDP

In case of a jump, the control circuit generates a Load Command. The PC will be loaded by the jump address. Similarly, in case of an output instruction, the control circuit generates an output enable command. The action of this command (as was shown in figure (9)) is to load the output latches by the content of the IR and optionally the AR register.

3.7. System Block Components

As already shown in figure (10), the blocks are:

. Memory System
. Memory Display Circuit
. System Control Circuit (SCC)
. Program Counter (PC)
. Input data multiplexer
. Output latches and Display
. System clock.

Detailed design/description of each block follows.

3.7.1. Memory: System

In fig (12) IC35 and IC36 are two memory elements, namely Instruction Register (IR) and Address Register (AR). Each is a 2048 bit Static Ram. Organized as 256x8 bit words, they feature a common input/output structure. The choice of this type of memory elements makes it possible to load and retrieve up to 256 program steps. Each memory chip has a separate output disable function to control its bus structures I/O pins. Fig(15) shows the logic symbol of the memory chip.

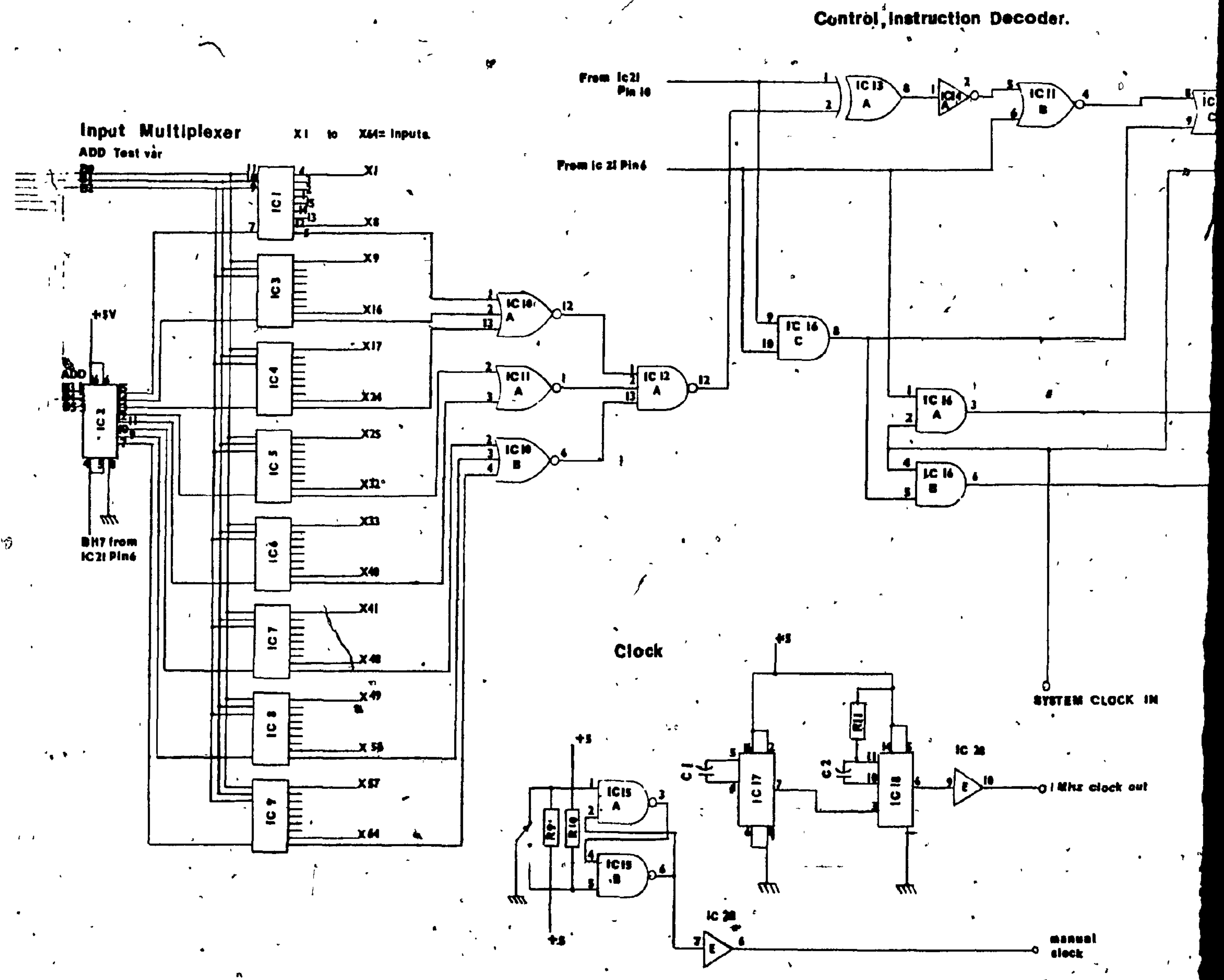
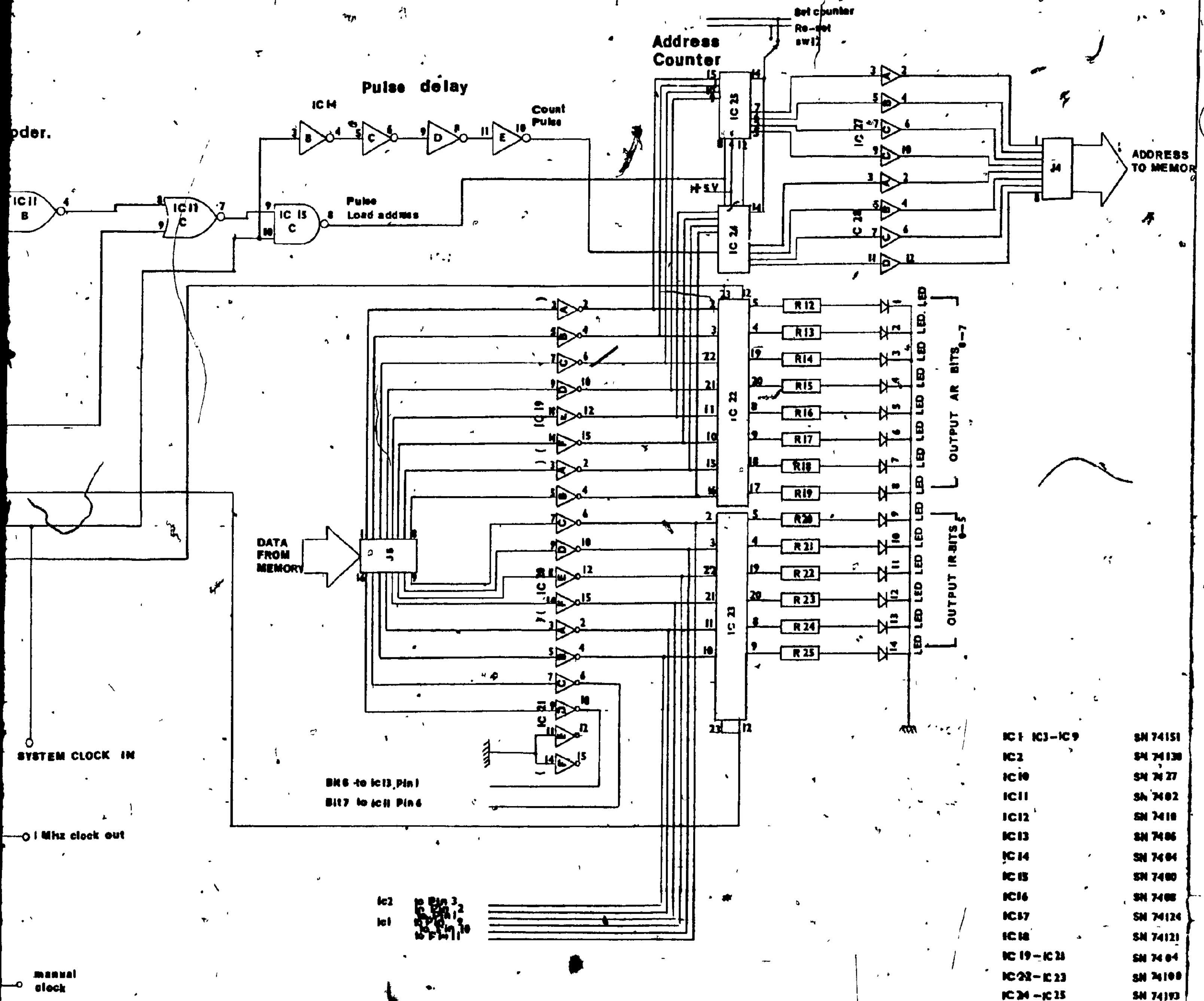


Fig.II. PBDP Circuit Diagram

1 oF



IC1 IC3–IC9	SN 74151
IC2	SN 74130
IC10	SN 7427
IC11	SN 7402
IC12	SN 7410
IC13	SN 7486
IC14	SN 7404
IC15	SN 7400
IC16	SN 7408
IC17	SN 74124
IC18	SN 74121
IC 19–IC 21	SN 7404
IC22–IC 23	SN 74100
IC 24 –IC 25	SN 74193
IC27 –IC 28	CD.4050

Diagram

2OF2

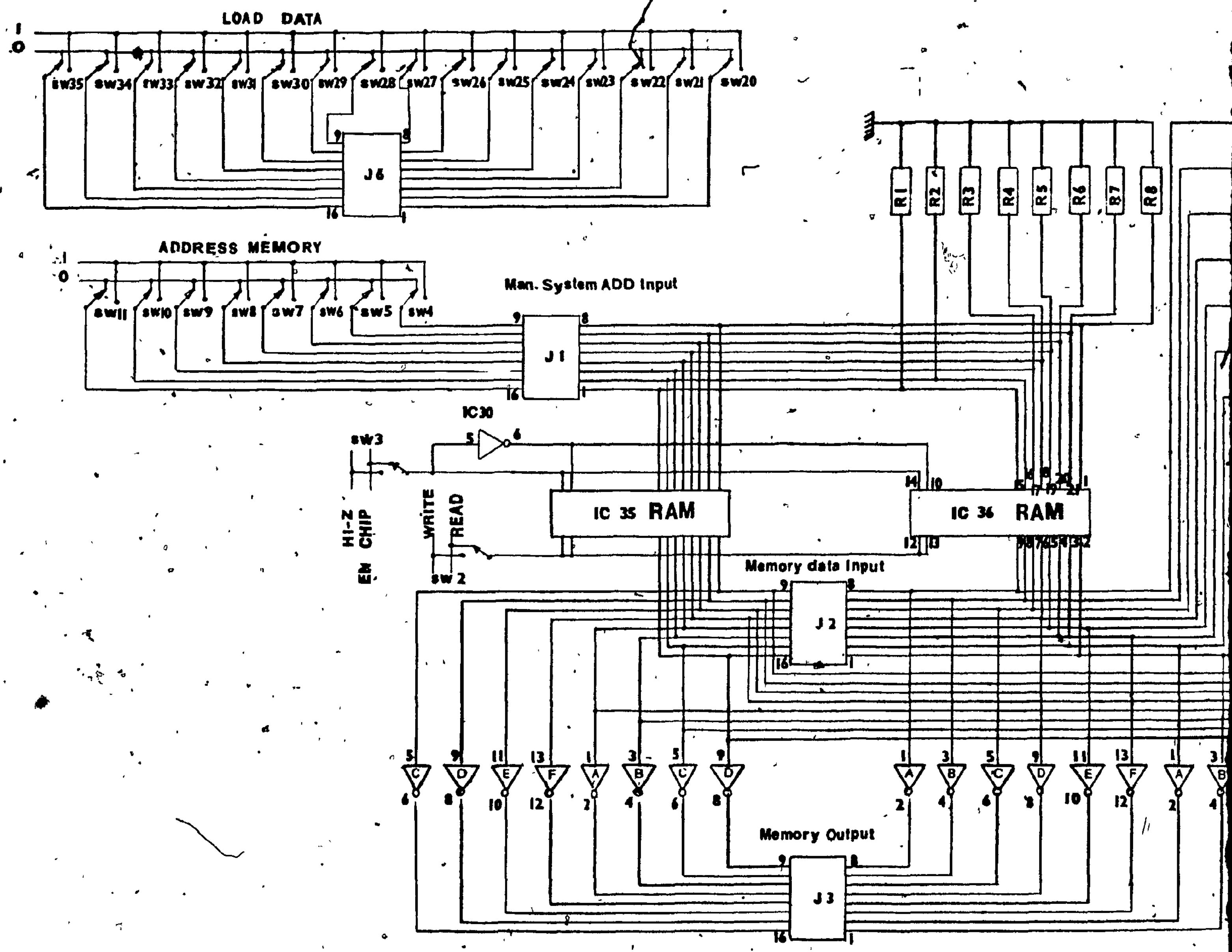
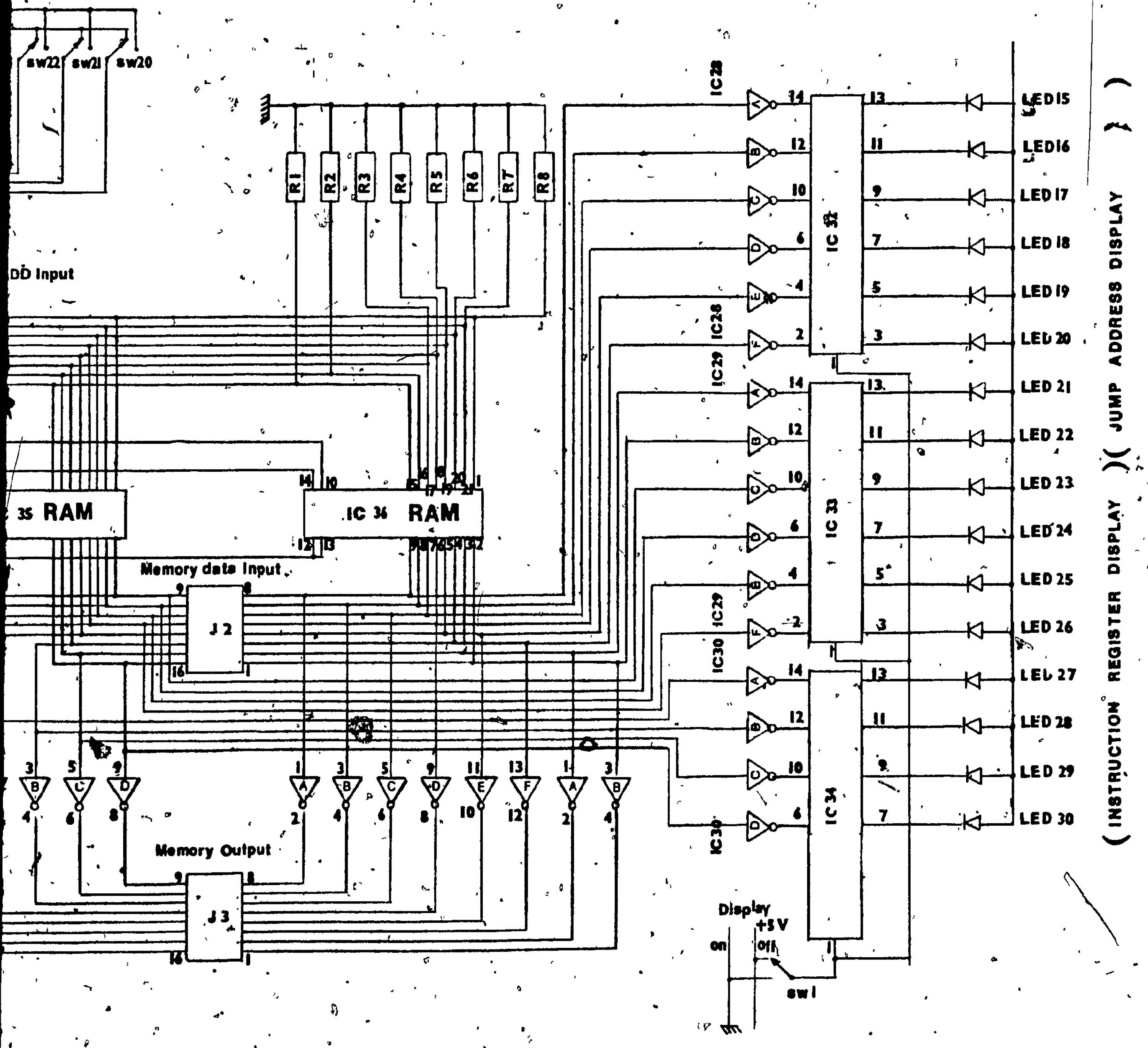


Fig.12 MEMORY CIRCUIT PBDP

1 of



ORY CIRCUIT PBDP

2OF2

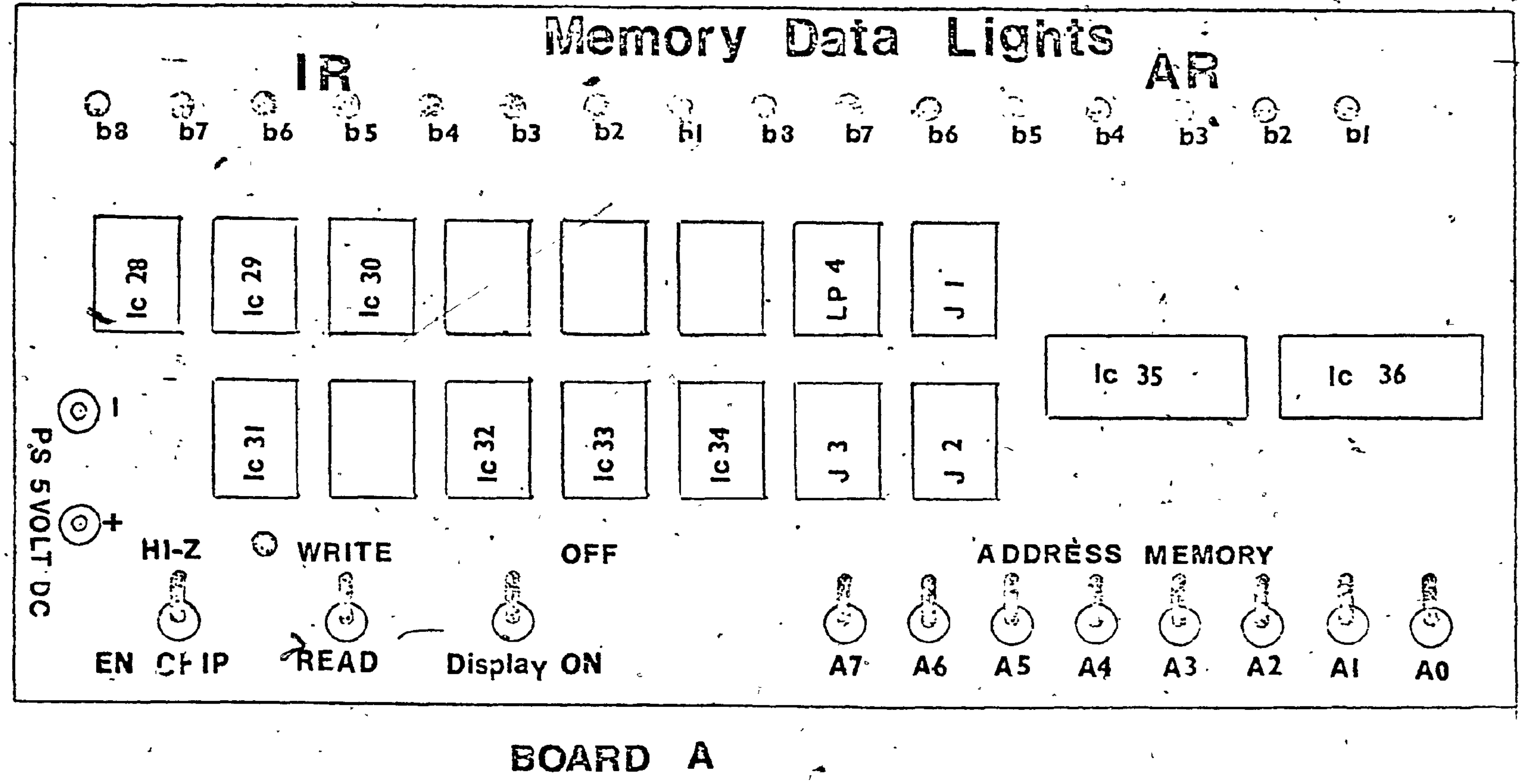


Fig. (13) Layout Board A

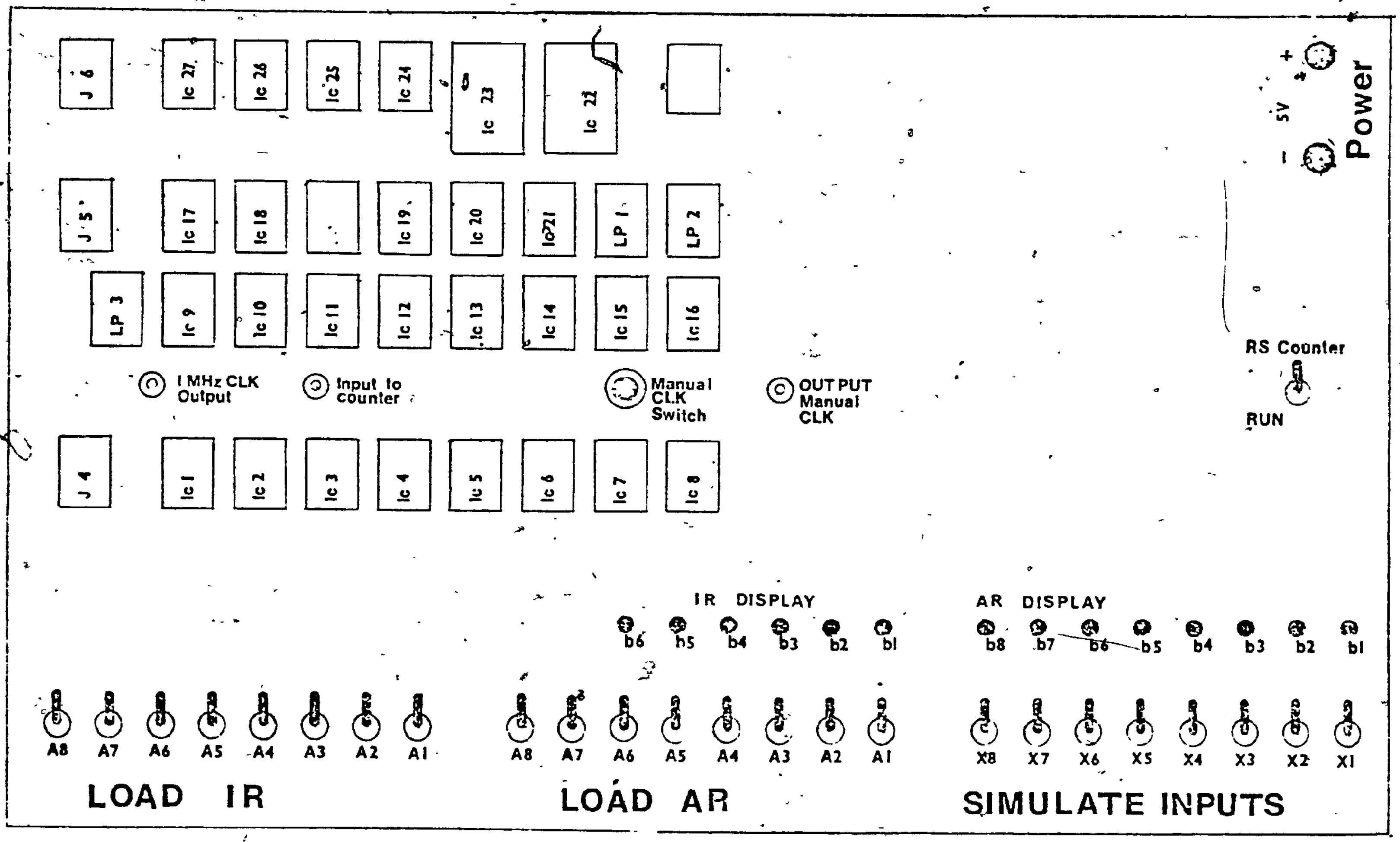


Fig.(14). Layout Board B

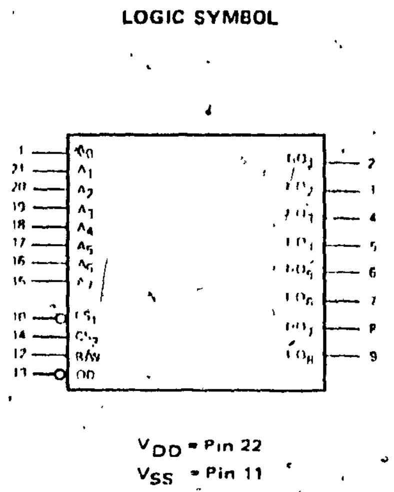


Fig.(15) Logical Symbol of 3539

This memory eliminates the need for a three state buffers. There are two chip select pins on the 3539. And it uses a multiplexed input/output structure. A truth table and a logic diagram of the memory are shown in Table 3 and fig (16), respectively. From the truth table and the logical diagram it can be seen that the I/O network is controlled by Read/Write (R W), output disable (OD) & two chip select (CS1 &CS2) inputs. Memory Timing diagram is shown in Fig (17).

Table 3. Memory Truth Table

CONTROL INPUTS				3539 OPERATING MODE				I/O BUS MODE		
$\overline{CS}_1$	CS_2	$\overline{OD}$	$R/\overline{W}$	Selected	Deselected	Write	Read	Input	Output	HI-Z
H	X	X	X		●					●
X	L	X	X		●					●
L	H	L	L	●		●		●		
L	H	H	L	●		●			●	
L	H	L	H	●			●			●
L	H	H	H	●			●		●	

X = Irrelevent state
L = LOW(-V_{DD})
H = HIGH(-V_{SS})

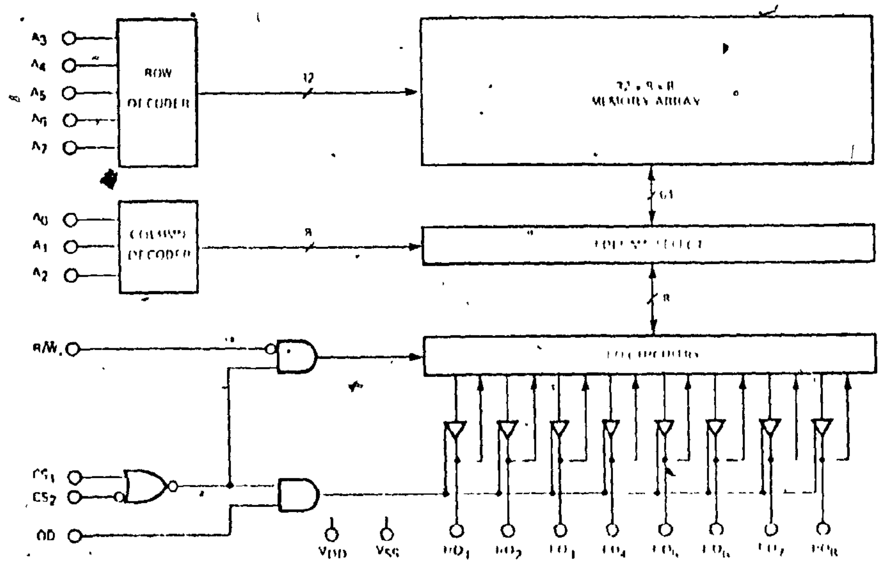


Fig.(16) Memory Logic Diagram

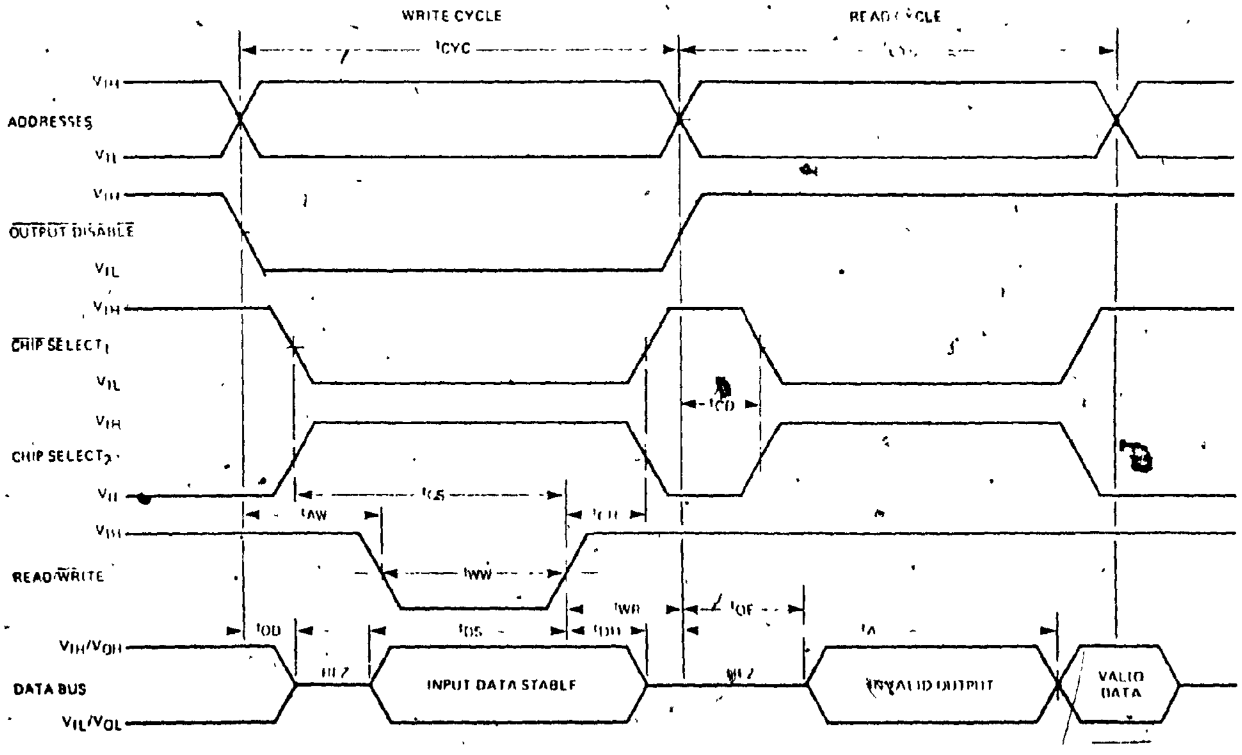


Fig. (17) Memory Timing diagram

3.7.2. Memory Display Circuit

In Fig (12), the display circuit consists of IC's (28-34) and LED's (15-30). The circuit is so designed that it will monitor all data in and out of the memory.

IC 28 to 30 are CMOS, CD4049, hex Invertor/buffers. In this application they are primarily used as an interface between the MOS Memory Chips (IC35&IC 36) and the display drivers IC 33 and IC 34. Considering the manufacturers data sheet for the Display drivers DS8869 (National), we need a logic low level to turn the indicator on. Therefore, all memory data is inverted before it is applied to the input of the display circuit. The output circuit of the DS8869 can sink up to 20mA which is a sufficient current to drive the led indicators.

3.7.3. System Control Circuit (SCC)

Design of the control circuit is based on the flow chart as shown in Fig (9), Section 3.5. It is required from the flow chart that the control circuit should follow the steps as outlined below. (See Fig. 18).

Step (1): Test the bit 7 of the Instruction register. If bit 7=0, then follow step (2) and if bit 7=1, then follow step (3).

Step (2): Exclusive or bit 6 of the instruction register with the input variable under test. Depending on the results of the test either generate a command to increment the program counter by one or generate a pulse to pre-set the counter with the jump address. No output should be displayed at this point.

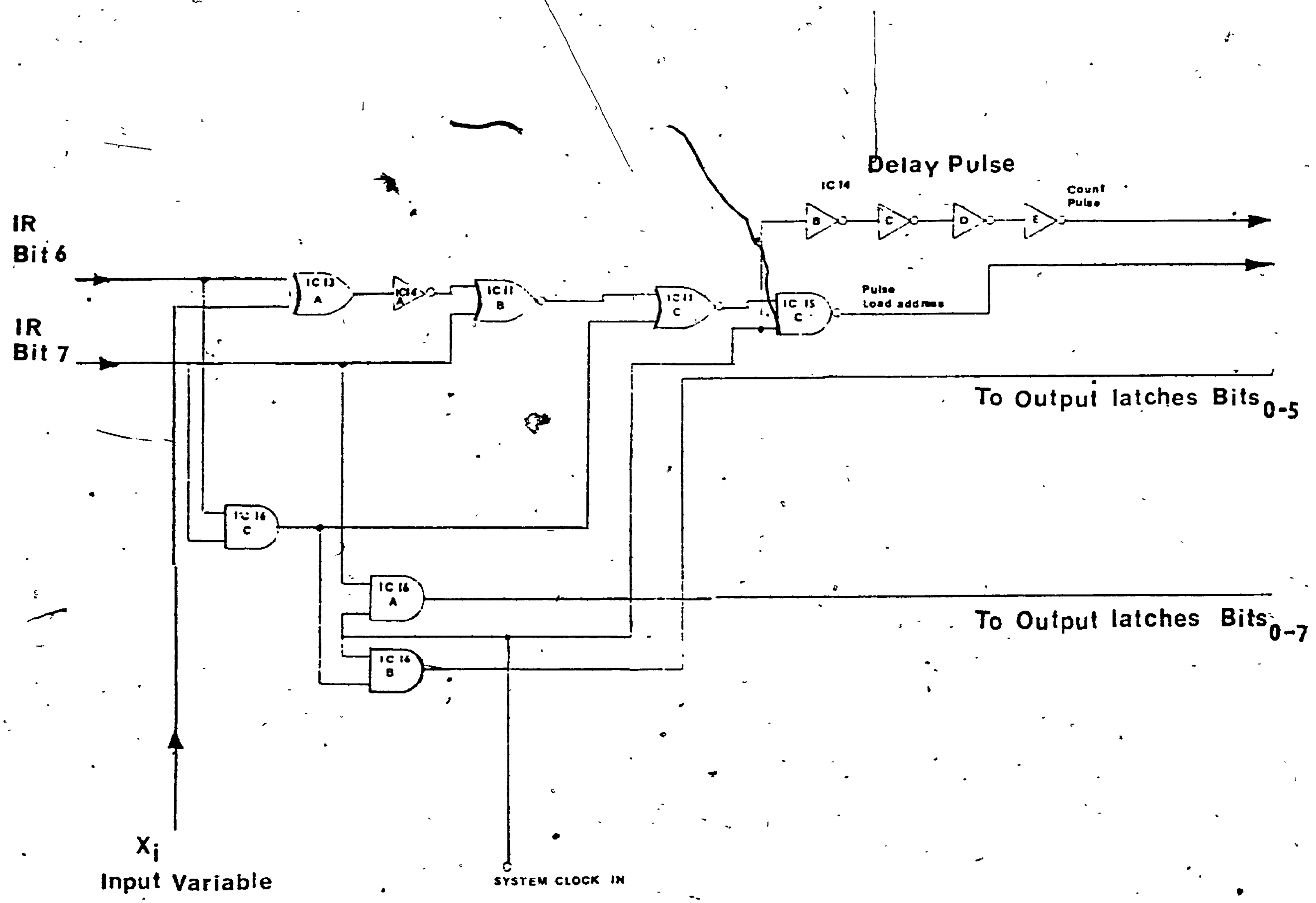


Fig. (18) Control Circuit

Step (3): Test bit 6 of the IR, if bit 6=0, then bits 0-5 of the IR should be displayed and if bit 6=1, then both, bits (0-5) of the IR & bits(0-7) of the AR should be displayed.

3.7.4. Program Counter (PC)

The Program counter is composed of two SN74193 binary up counters, IC24 & IC25 chained together to create 8 bits of memory address. This gives the system the capability of addressing 256 separate memory words. The counters are configured to count up on the rising edge of the system clock signal and reset to zero or preset to a branch address by applying an appropriate signal to the counter control inputs.

The program counter supplies the memory with the address bits. The program counter normally increments on the rising clock edge of the each clock pulse, sequencing through the programmed instructions in memory. In a non-branching application, the count sequence of the program counter is not altered. Therefore, the program statements are executed in order, until the program counter "wraps around", and the sequence is repeated. This is known as "looping control structure". Timing diagram for the counter circuit are shown in fig (19).

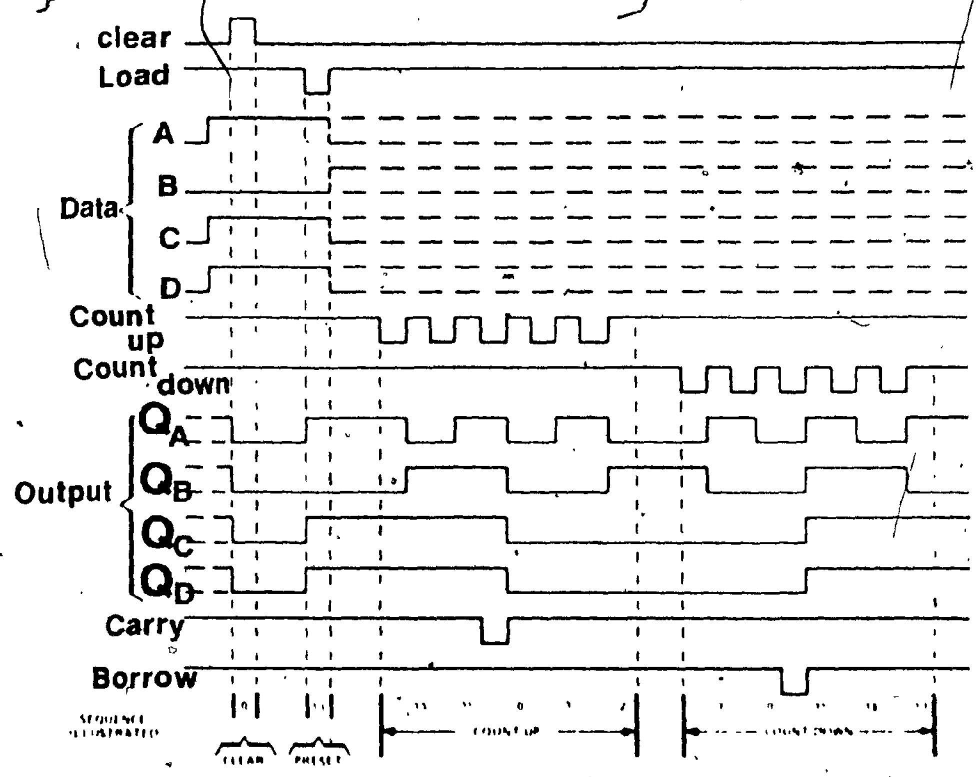
3.7.5. Multiplexer/Decoder Circuit

Based on the instruction format of figure(8), the designed multiplexer circuit is shown in Figure (20). The first six bits of the instruction register (IR) are the address of the input variable and when used as the input to the multiplexer will select up to 64 input variables. The circuit consists of IC2,IC1,IC3-9, IC10 and IC11. IC1, and IC3-9 are

typical clear, load, and count sequences

Illustrated below is the following sequence

1. Clear outputs to zero
2. Load (preset) to binary thirteen
3. Count up to fourteen, fifteen, carry, zero, one, and two
4. Count down to one, zero, borrow, fifteen, fourteen, and thirteen



NOTES: A. Clear overrides load, data and count inputs

B. When counting up, count down input must be high; when counting down, count up input must be high

Fig. (19) Counter Timing Diagram

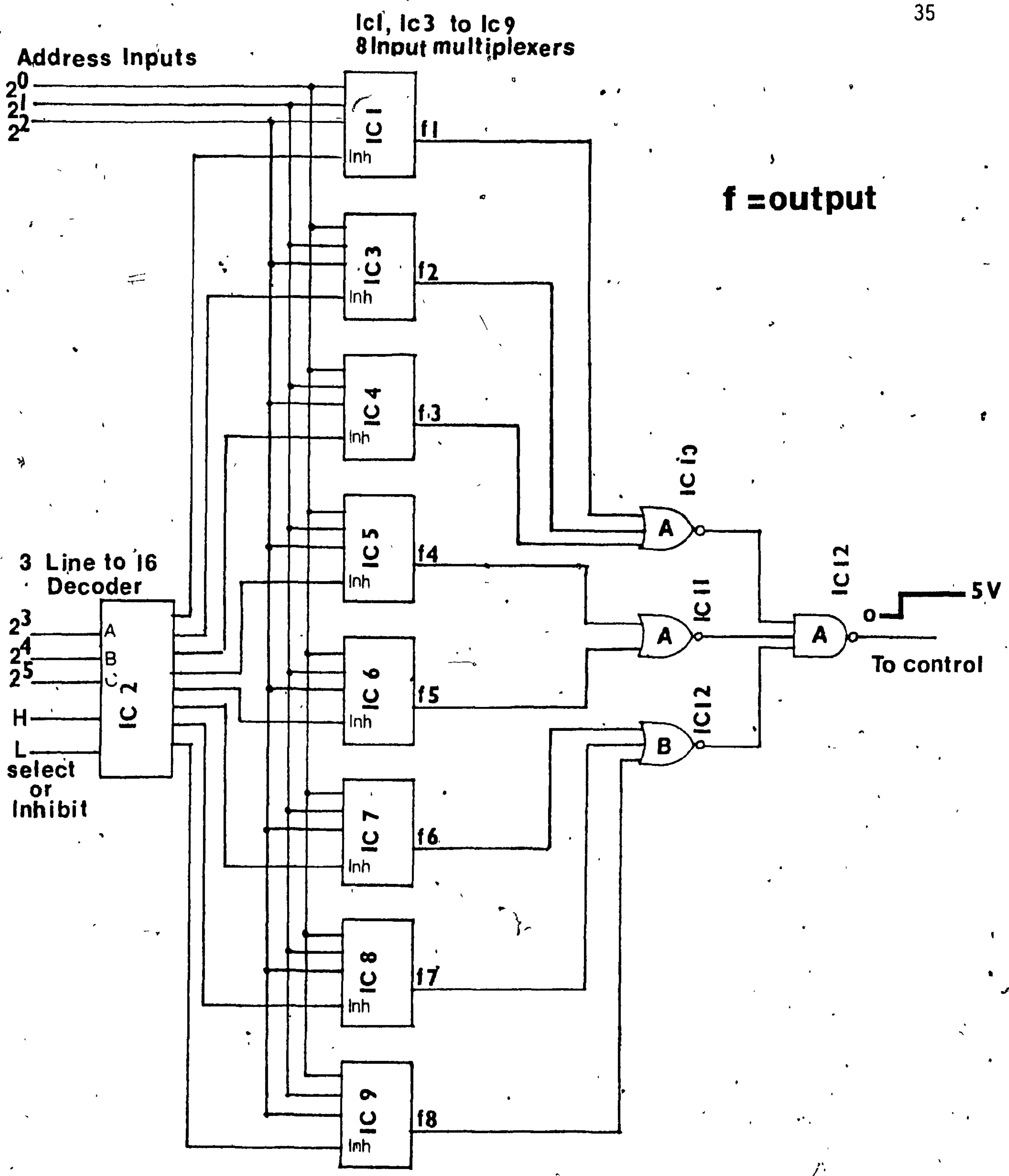


Fig. (20) 64 Input Digital Multiplexer

8 line-to-one line multiplexers. The device gates one of eight inputs to the output, depending on the three input code. Eight of these multiplexers are incorporated to multiplex all the inputs. IC1 is used as a 3 to 8 line decoder and select only one of the multiplexer depending on the input code. The first three bits of the address code are used as a common input to all 8 to 1 line multiplexers. The multiplexer output is routed via IC10 and IC11 to the comparator circuit.

The existing system is capable of accepting only 8 inputs, however, when all the system components are present and a simple wiring change is made, the system will select up to 64 inputs.

3.7.6. Output Display Circuit

The output circuit is so designed that either the bits 0-5 of the Instruction Register (IR) or bits 0-7 of the address register (AR) or output of both register together can be displayed. IC22 and IC23 are the two main elements of the output display circuit. Both chips are 8 bit bistable latches. These latches are used as temporary storage of binary information between processing units and input-output or indicator units.

LED's 9 to 14 are allocated to display the contents of IR and similarly LED'S 1-8 display the contents of AR. Information present at data (D) input is transferred to the Q output when the enable (G) is at a logic high level and the output will follow the data input as long as the enable remains high. As soon as the enable goes low, the information (that was setup at the data input at the time the transition occured) is retained at the Q output until enable is permitted to go high.

3.7.7. System Clock

The various AC characteristics of the memory chips are given in the manufacturer's data sheet. There, delays, set up times and hold times must be considered with those of the other logic elements of the memory system to determine the clock frequency.

In Read mode a new address appears at 17 to 26 ns corresponding to the delay limits of SN74193 counters (IC24,25), chip select delay time T_{CD} of 100ns now becomes between 117 and 126 ns. The latest time for new data to appear is given by the manufacturer data sheet as 650 ns. Since the delay due to address counter is very small and, therefore, for all practical purposes can be ignored. As long as the time greater than T_{CD} is not reached, the memory output is at a HI-Z. Therefore, taking the T_A as the major limiting factor, the system clock should be designed to exceed the period. A clock period of Ius is choosen. See waveform in figure (21).

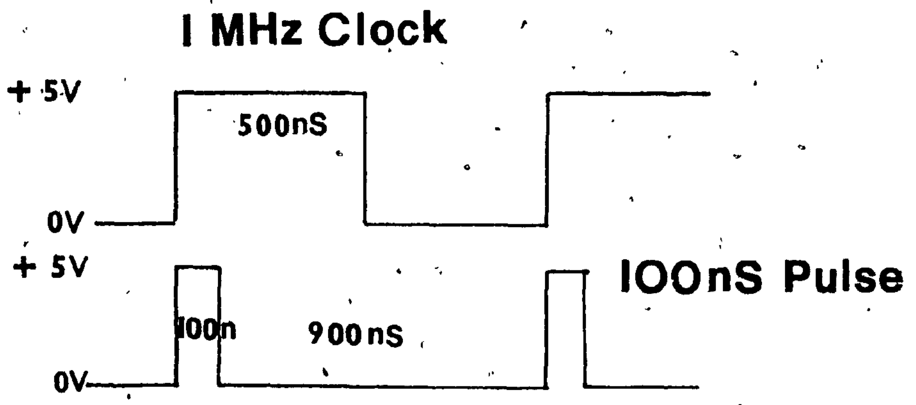


Fig. (21) System Clock Waveforms

The clock circuit is composed of IC17 and IC18. IC17 is a dual voltage controlled oscillator. The output frequency is simply established by a single external capacitor whose value is calculated as follows:

$$FO = \frac{5 \times 10^{-4}}{Cext}$$

Where

FO = output frequency in HZ
Cext = external capacitor in Farads.

The pulse synchronization gating section of the oscillator ensures that the first output pulse is neither clipped nor extended. Duty cycle of the square-wave output is fixed at approximately 50%. The clock output from IC17 is fed to the input of IC18. IC18 is designed to trigger at the rising edge of the clock and hence it produces a pulse of 100ns duration.

3.8. Functional Description of switches, display lights, Input/Output Sockets:

Display Lights (Fig. 13 and Fig. 14)

(i) Chip-Status Light: The Chip Status light (located on Board A) shows the current state of the RAM's. 'ON' state indicates that the chips I/O ports are at High Impedance (HI-Z) and neither Read nor write operation can take place.

(ii) Memory Data Lights: The memory data lights (located on Board A) show the content of the memory location currently addressed by the program counter (PC) and the clock signal. After data has been loaded into memory, it is displayed by the memory data lights. The lights are also useful in verifying programs entered in memory.

(iii) Output Display lights: The output display lights (located on Board B) show the state of the output variables. The first 8-bits display the content of output buffer loaded by the address register (AR) and the remaining 6-bits display the output buffer loaded by the Instruction Register (IR).

Functional Switches (Fig. 13 and Fig. 14)

(i) Hl-Z/EN.CHIP. Selects or de-selects the RAM's.

(ii) Write/Read. Selects the Memory Mode of Operation.

(iii) Display On/Off. Turns the data display On or Off.

(iv) Address Memory. These switches set the manual address at which the data into the RAM locations is to be loaded. Switches are designated from A0 to A7 on Board A and create 8 bits of memory address. This gives the system the capability of addressing 256 separate memory words.

(v) Simulate Input. These switches set the desired binary input for processing. Switches are marked from X1 to X8 i.e. up to 8 inputs may be simulated. The system is designed to accept up to 64 inputs.

(vi) Manual CLK. Advances the address counter one cycle per depression. (i.e. the single step push button toggles the clock signal.)

(vii) Load AR Data. These switches simulate the binary encoded program for entering in the AR section of the memory.

(ix) Load IR Data. These switches simulate the binary encoded program for entering in the Instruction Register (IR) section of the memory.

(x) RS Counter/RUN. When set to RS, it clears the program counter and when set to RUN the system will sequence through the program in memory or the program may be "single stepped" using the single step push button.

SOCKETS (Fig. 13 and Fig. 14)

J1 : Accepts the manual/system address. (8 bits).

J2 : Accepts data for loading into memory (16 bits).

J3 : Supplies outputs from the RAM's (16 bits).

J4 : Supplies output of the address Counter (8 bits)

J5 : Accepts output from the Memory. (16 bits).

J6 : Supplies binary output to program the memory.

1 MHZ CLK output	:	1 MHZ clock output
Output manual CLK	:	Single step clock output
Input to counter	:	Accepts clock input up to 1 MHZ

3.9. Program Loading Procedure

To load a Binary Decision Program into the memory the following procedure is adopted.

- Short Circuit the memory address Input socket. On Circuit Board (A) this socket is marked as J1. A link platform is provided to short circuit the socket J. When the system is OFF and if it is desired to address the memory then insert the link platform (LP) into socket J1, this connects the Address toggle switches directly to the RAM address lines. To select an address, set the toggle switch, according to the binary bit. The right most switch is the LSB of the binary address.

. On the board B, there are two sets of eight toggle switches and each of them represents one of the memory registers namely, Instruction Register (IR) and Address Register (AR). Each register is eight bit long hence eight toggle switches in either set represent the eight bits of each register. Now the desired data can be loaded into memory by manipulating the toggle switches.

. The data along with proper address is now ready for actual entry into the memory. To complete the load cycle set the Read/Write Switch to position "Write" and then also set the Chip select switch to position "En-Chip".

. For loading the next set of data set the Chip select switch to position "HI-Z." Set address, set data and now set Chip select switch to position "En-Chip".

When Chip select switch is in position "HI-Z" the memory input/output parts are at high impedance (HI-Z).

. To check the status of data at any memory location, select address and set the Read/Write switch to position Read. The LED'S will indicate the appropriate data at the selected address.

3.9.1. <u>Program Execution: (1MHZ Clock Operation)</u>

After the program is loaded into the memory, the following steps should be followed to run the program.

. Re-set chip select switch to position "HI-Z."

. Take out the link paltform (LP) from J1. This isolates the address switches from the memory address bus. Connect socket J1 and socket

J4 via a ribbon type connector. Now the memory is ready to accept any addresses directed by the logic control circuit.

. Re-set the Address Counter by setting the toggle switch to position Re-Set. Now the pointer is set to the memory address 0.

. Connect the 1 MHZ clock to the "system clock in"

. Set the Counter Switch to position "Set". The program is now executing.

3.9.2. Program Execution: Single Step Operation

. Manual clock may be used for single step execution of a program by connecting the manual clock output into the "System Clock In". When we single step through the execution of a Binary Decision Program, we are not obliged to step it at a particular rate. The processor will remain in a WAIT state after each single step clock pulse until we decide to apply another such pulse. With this provision it is easier to check a BDP's operation before we execute at the full clock rate of 1 MHZ.

CHAPTER 4

APPLICATION

4.1. General

The various examples outlined in the following paragraphs are just a few of the many possible applications of the programmable Binary Decision processor. The advantages of a Binary Decision program as compared to a Boolean function program were presented in Chapter 2 and now we will see with the aid of the following examples their speed advantage resulting in the saving of memory spaces. Computers and microcomputers may also be used, but they tend to over complicate the task and often require highly trained personnel to develop and maintain the system.

4.2. PBDP as a Programmable Controller

In many applications industry uses programmable controller as a solid state, Boolean logic calculator that uses a memory to store a predetermined control sequence. The controller activates certain outputs in response to predetermined logic, which is triggered by various inputs. A typical programmable controller is shown in Fig. (22).

Relay control panels with hardwired logic can do the same thing for simple and dedicated tasks. Computers, too, can control sequences, but their cost may not be justified for simpler tasks.

Programmable controllers as used in industry for implementing random logic are not designed to handle Binary Decision Programs and hence do not offer the advantages provided by a machine such as the Programmable Binary

Decision processor. We will show with the aid of an example that a conventional Industrial Controller is not only slower but requires more program steps than the corresponding programmable Binary Decision processor.

Many of the logic structures found in the controls industry consists of branches of several series relays, in parallel with another branch of series relays. Fig.(23) shows an example of this structure.

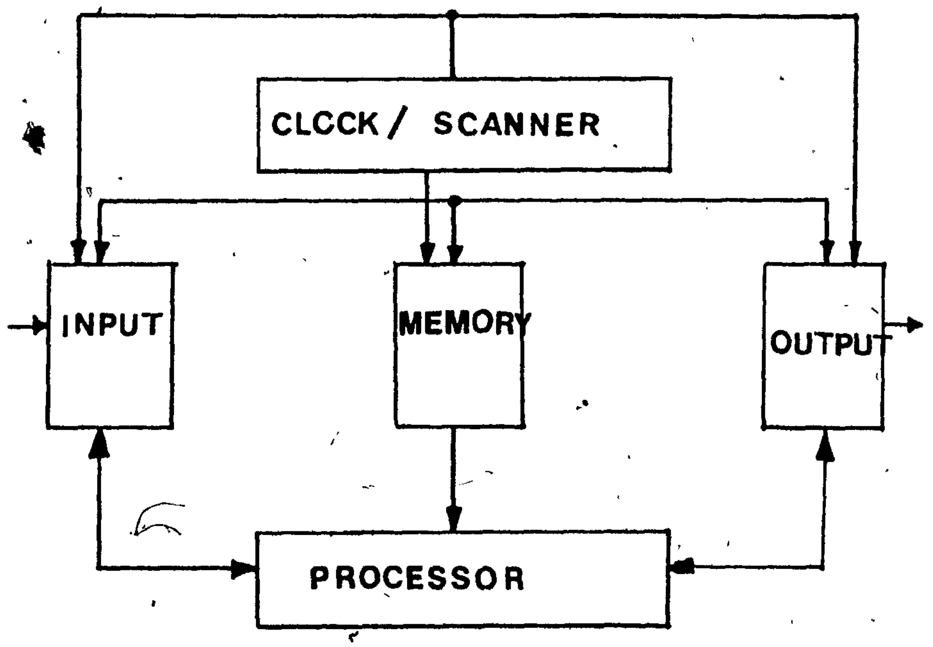


Fig. (22) Block Diagram of a Typical Programmable Controller

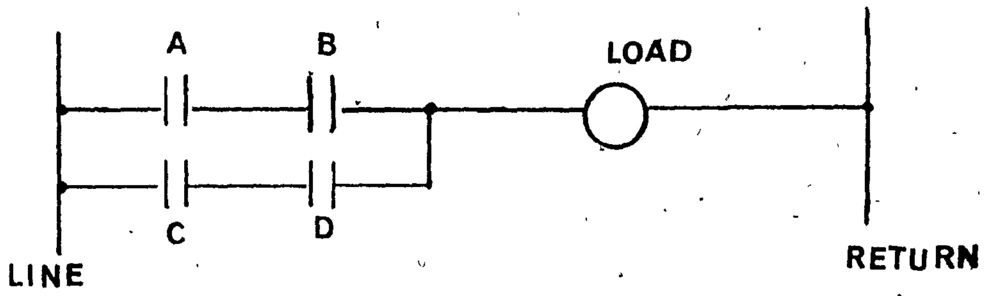


Fig. (23) Relay Ladder Logic

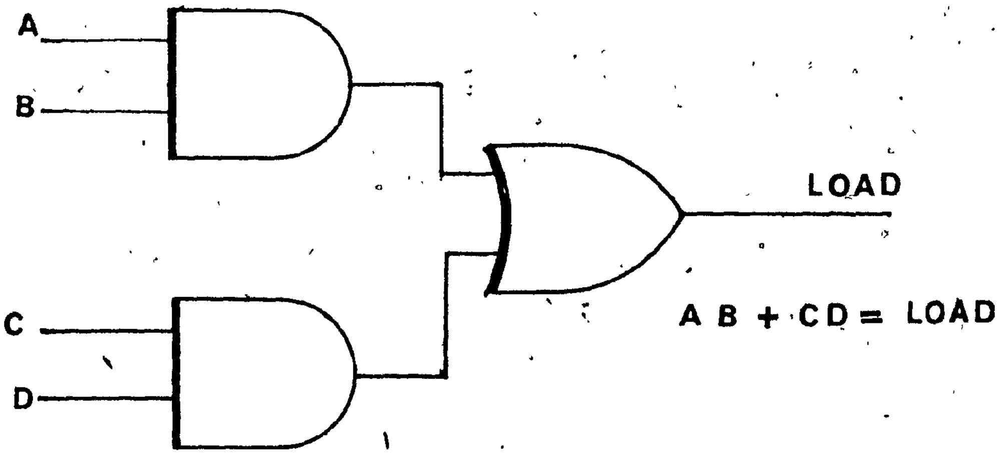


Fig. (24) Solid State Equivalent Circuit

Thus, when A and B or C and D are closed (logic 1), load is energized (logic 1). This is just one of the many examples to be found in Control Industry.

4.2.0. As an example consider the following switching function representing the circuit in Fig (24).

$$Y = \overline{X}_1 X_2 + X_1 X_3 \overline{X}_4 + \overline{X}_2 X_4$$

The above switching function could be taken as an industrial control problem, where X and Y are simply inputs and output functions (the status of inputs could be taken as the states of various relays if desired). To program an industrial controller a program such as shown below will be used:

```
AND CI          (1)    (AND, COMPLEMENT, INPUT)
AND I           (2)    (AND, INPUT)
  OR
AND I           (1)
AND I           (3)
AND CI          (4)
  OR
AND CI          (2)
AND I           (4)

                END              (9 steps)
```

The above program, after encoding in binary form, is loaded into the memory. The Boolean controller fetches these instructions in consecutive order, decodes and executes them.

In the following paragraphs we will show the application of the corresponding Programmable Binary Decision Processor to solve the above problem using a Binary Decision Program.

4.3. Application Example: (1) Industrial Control

For the previously described switching function in section 4.2. a binary decision program can be written as the following:

(Assume R1, R2, R3 & R4 are inputs and Y is the output)

```
BEGIN

    If R1=0  then  if R2=0  then if R4=0  then Y=0
                                          else Y=1
                   else if R4=0  then if R3=0  then Y=0
                                          else Y=1
                                 else if R2=0  then Y=1
                                          else Y=0
END
```

Table (4) is constructed to represent the Binary Decision Program of example (1) for a machine like PBDP with a simple branch address.

Table (4): BDP INSTRUCTION REPRESENTATION

ADDRESS	Instruction Register (IR)		Output Register (AR)
	NAME OF VAR. TESTED	BRANCH IF	*BRANCH ADDRESS
0	R1	1	5
1	R2	0	3
2	(Output Inst.)	(1)	0
3	R4	1	2
4	(Output Inst.)	(0)	0
5	R4	1	8
6	R3	1	2
7	(Output Inst.)	(0)	0
8	R2	1	7
9	(Output Inst.)	(1)	0

* In this example Bits 0-7 of AR are used as BRANCH ADDRESS, however, these bits may also be used as output variables if needed.

Now we are ready to encode the instruction in Table (4) into binary notation as shown in Table (5)

Table (5) BDP Binary Notation

ADDRESS	OP CODE	(IR) ADD.INPUT VAR(OP)	(AR) JUMP ADDRESS
000000	01	000000	00000101
000001	00	000001	00000011
000010	10	000001	00000000
000011	01	000011	00000010
000100	10	000000	00000000
000101	01	000011	00001000
000110	01	000010	00000010
000111	10	000000	00000000
001000	01	000001	00000111
001001	10	000001	00000000

Once coded as in Table(5), the sequence can be programmed using the toggle switches on the processor board.

Explanation of the program:

Instruction #1: Tests the logic value of input R1. If R1=0 process the immediate Instruction else jump to specified address.

Instruction #2: Tests the logic value of input R2. If R2=1 then process the immediate Instruction else jump to specified address.

Instruction #3: Is an output Instruction and it displays the variable 1, program returns to #1.

Instruction #4: Test the logic value of input R4. If R4=1 then a jump to the specified address will be executed else the immediate instruction is processed. In this case an address of an output instruction is specified.

Instruction #5: Again an output Instruction. Program will loop back to #1.

Instruction #6: Tests the logic value of R4. If R4=1 then jump to specified address else take immediate instruction for processing.

Instruction #7: Tests the logic value of input R3. If R3=0 takes the immediate instruction and process else a jump at the specified address is made.

Instruction #8: Output Instruction. After display program loops back to instruction #1.

Instruction #9: Tests the logic value of input R2, if R2=1, a jump is executed at the given address else the immediate instruction is executed.

Instruction #10: Output Instruction. Data is displayed and program loops back to #1.

4.3.1. Detailed procedure for entering verifying, single stepping and running the example program

Entering the Program into RAM

Power to system is OFF.

A. Connect J6 to J2.

B. Short Circuit (S/C) J1 by inserting the link Platform.

C. Set the program address.

D. Set Data at the address.

E. Set HI-Z/En Chip to HI-Z.

F. Set Read/Write switch to position 'write'.

G. Set Display ON/OFF switch to 'ON'.

H. Switch on power.

I. Set HI-Z/En Chip momentarily to En-Chip. The data pattern will be displayed by the data lights. Displayed data is now stored in the Memory.

Repeat C, D and I untill the entire program has been entered.

Caution: Do not switch OFF.

Verifying the program entered in RAM:

A. Disconnect J6 from J2. All data display lights will come on.

B. Set the address.

C. Set Read/Write switch to write.

D. Set HI-Z/En Chip to En-Chip. The entered data at the desired address, will be displayed by the data lights. The entire program may be verified by sequencing the data address one by one.

Single Stepping the Program:

A. Set HI-Z/En-Chip to HI-Z.

B. Set Read/Write to Read.

C. Connect Output Manual 'CLK' to 'Input to Counter' by a patchcord.

D. Disconnect S/C LP from J1.

E. Connect J4 to J1.

F. Connect J3 to J5.

G. Simulate Inputs momentarily.

H. Set RS-Counter/RUN to RS-COUNTER.

I. Set HI-Z/En-Chip to En-Chip.

J. The processor may now be sequenced throught the program entered in the memory using the single step switch. Each depression of the single step push button advances the clock by one cycle. Output data and memory data are displayed for each depression.

Running the Program:

A. Set HI-Z/En-Chip to HI-Z.

B. Connect 1 MHZ clock to Input to Counter. Momentarily.

C. Set RS-Counter/RUN to RS Counter.

D. Set HI-Z/En-Chip to En-Chip. Program is now running.

The example above not only shows how the BD Processor may be used to a variety of industrial switching purposes but also shows the speed advantage of the BDP over a Boolean representation controller judged from the number of program steps in each case.

Therefore, in applications, where a complex decision depending on many variables is to be made and made repeatedly and we wish to arrive at the decision quickly without having to go through a large amount of computation we will have to resort to the services of a BDP type of a machine.

4.4. Application Example(2). Magnitude Comparison

In this example (10), we have designed a binary decision program based on the binary decision tree as shown in fig.(4), Chapter 2.

A and B are two decimal numbers and each is represented by two binary bits. These are shown in table (2), Chapter 2. Magnitude comparator compares A and B to see which is greatest or if they are equal. Only three results are possible. Program in Table (6) is derived from the flow chart of the comparator as shown in fig.(5), Chapter 2. A binary diagram based on this program is shown in figure (25).

Table (7) shows the binary form of the program in Table (6), now the sequence can be programmed using the toggle switches on the processor board.

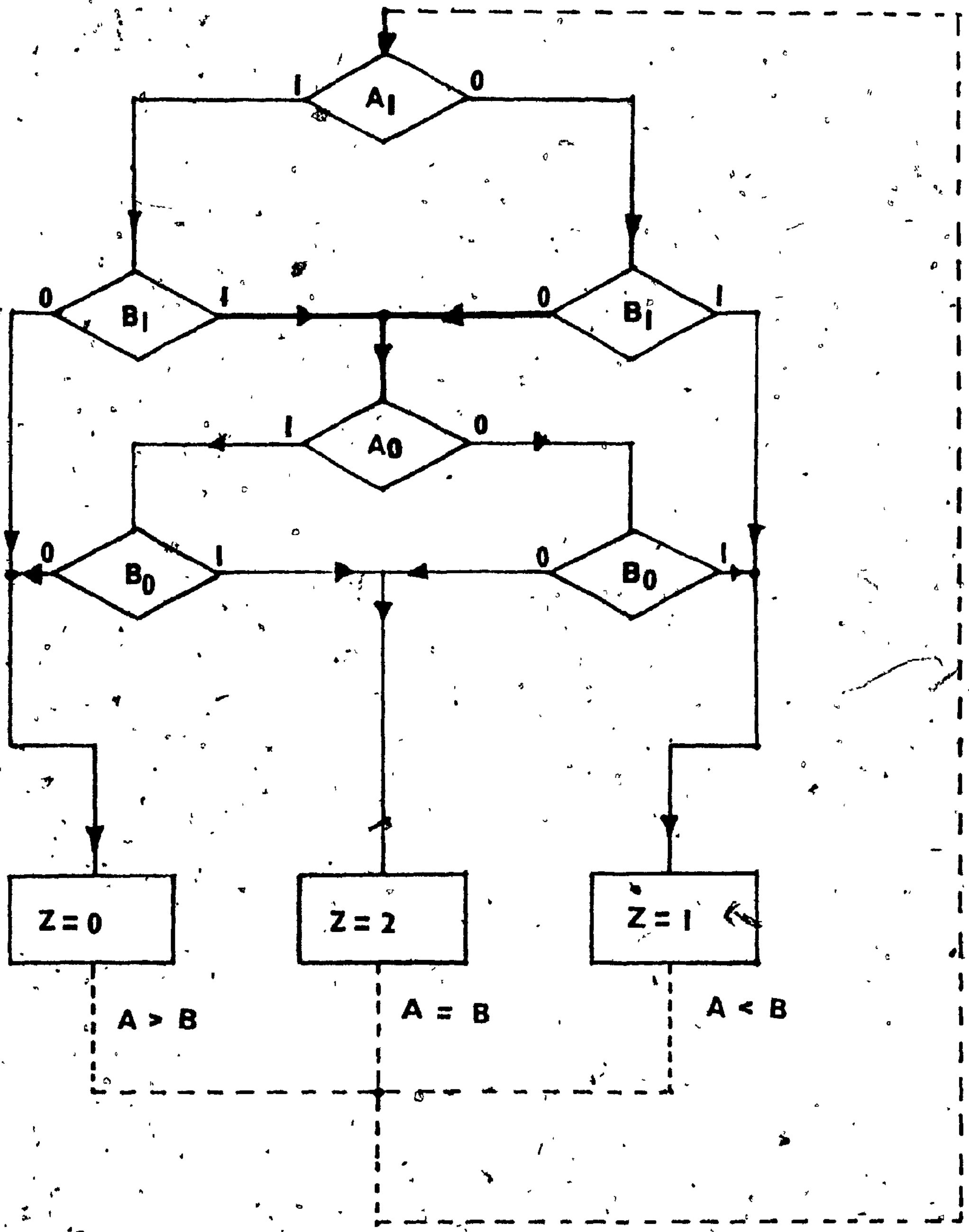


Fig (25) Binary Decision Diagram
Magnitude Comparison

Table (6) BDP Instructions Representation

ADDRESS	INSTRUCTION REGISTER (IR) VAR.TESTED	BRANCH(IR)	OUTPUT REGISTER (AR) OR BRANCH ADDRESS
0	A1	1	3
1	B1	0	5
2	(Output Inst.)	(Z=1)	0
3	B1	1	5
4	(Output Inst.)	(Z=1)	0
5	A0	1	8
6	B0	1	2
7	(Output Inst.)	(Z=2)	0
8	B0	0	4
9	(Output Inst.)	(Z=2)	0

Table (7) BDP Binary Notation

ADDRESS	(IR) OP CODE	AD. INPUT VAR.(OP)	(AR) JUMP ADDRESS
000000	01	000000	00000011
000001	00	000010	00000101
000010	10	000001	00000000
000011	01	000010	00000101
000100	10	000001	00000000
000101	01	000001	00001000
000110	01	000011	00000010
000111	10	000010	00000100
001000	00	000011	00000000
001001	10	000010	00000000

4.5. Application Example (3). Auto Safety Control Circuit

In this example (15), the PBDP is used to solve a very practical problem namely that of Auto safety involving the car ignition switch and the car seat belt. The problem is stated as the following:

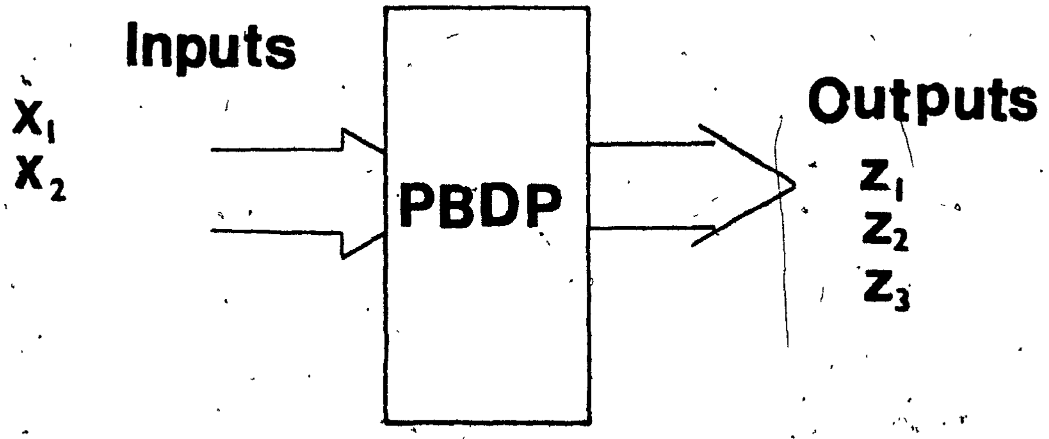
(a) If the driver is buckled up and the ignition switch is turned ON, the car starts.

(b) If the ignition switch is ON but the driver is not buckled up then both alarms, audio and visual must turn ON (the engine does not fire).

(c) If the ignition switch is ON and in the meantime the driver buckles up then the audio alarm is turned OFF, but the visual alarm remains ON. The engine will not fire until the ignition switch is turned OFF and again ON, with the driver remaining buckled up.

(d) If the engine is running and the driver unbuckles himself then only the visual alarm should be turned ON.

Figure (26) is the block representation of the auto safety example.



where,

X_1 = Input (giving state of ignition of switch)

X_2 = Input (seat belt status)

Z_1 = Output (Ignition Current)

Z_2 = Output (alarm light)

Z_3 = Output (Audio alarm)

Figure (26) Auto Safety: Block representation

A state diagram of a sequential machine conforming to the auto safety specification is shown in Fig (27).

By applying state reduction technique (16) a simplified state diagram is obtained in Fig (28).

Following is a Binary Decision Program based on the state diagram as shown in Fig (28).

<u>B.D.P. Auto Safety Example</u>

```
ADDRESS                 INSTRUCTIONS

BEGIN:      IF     X1    then   a1     else    a2.
   a1:      IF     X2    then   b2     else    c1.
   b2:        OUTPUT  100,  GO   TO  BEGIN.
   b0:      IF     X1    then   b3     else    a2.
   b3:      IF     X2    then   b1     else    b2.
   b1:        OUTPUT  110,  GO   TO  b0
   a2:        OUTPUT  000,  GO   TO  BEGIN.
   c0:      IF     X1    then   c3     else    a2.
   c3:      IF     X2    then   c2     else    c1.
   c2:        OUTPUT  010,  GO   TO  c0
   c1:        OUTPUT  011   TO   TO  c0
```

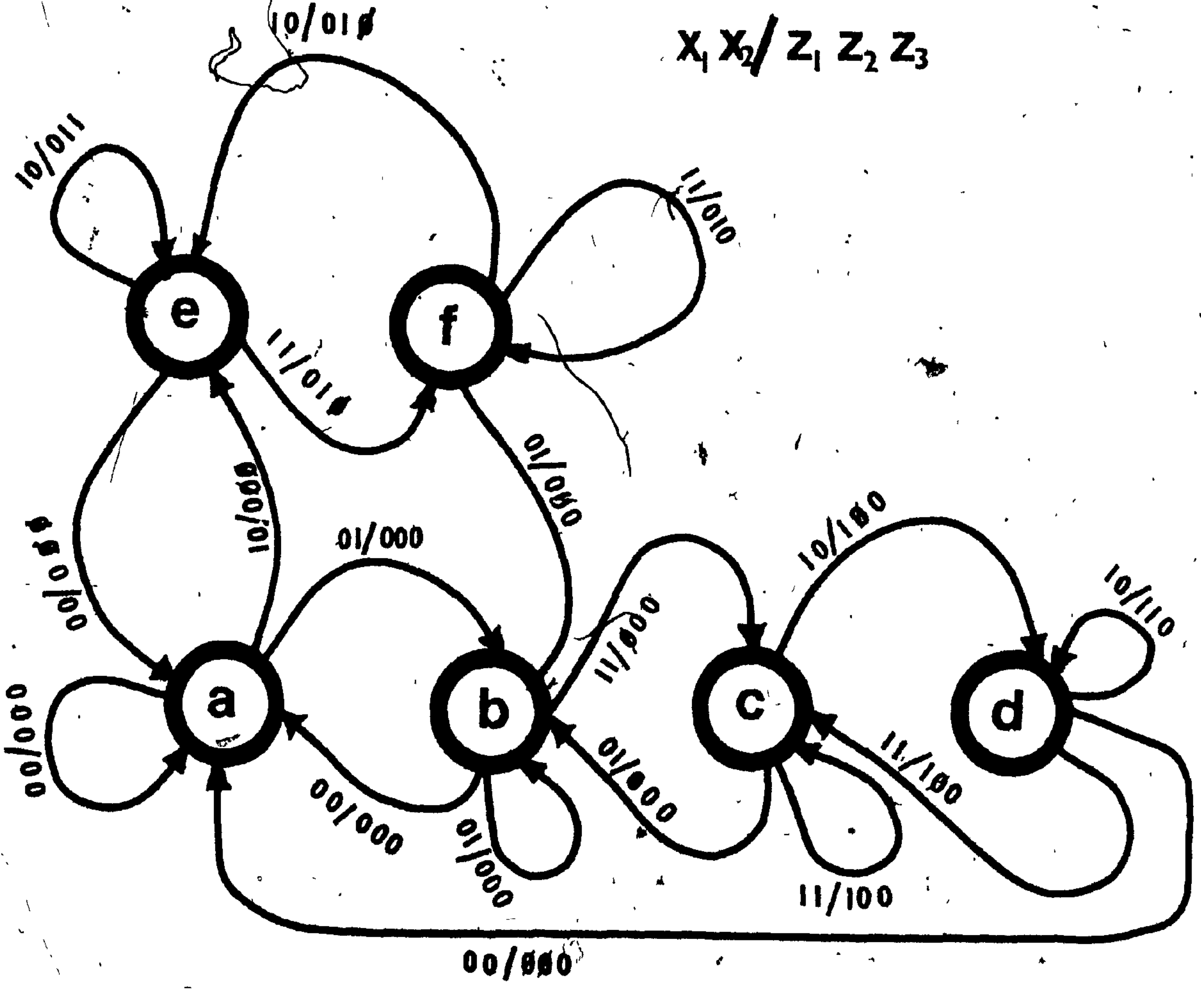


Fig. (27) State Diagram: Auto Safety Problem

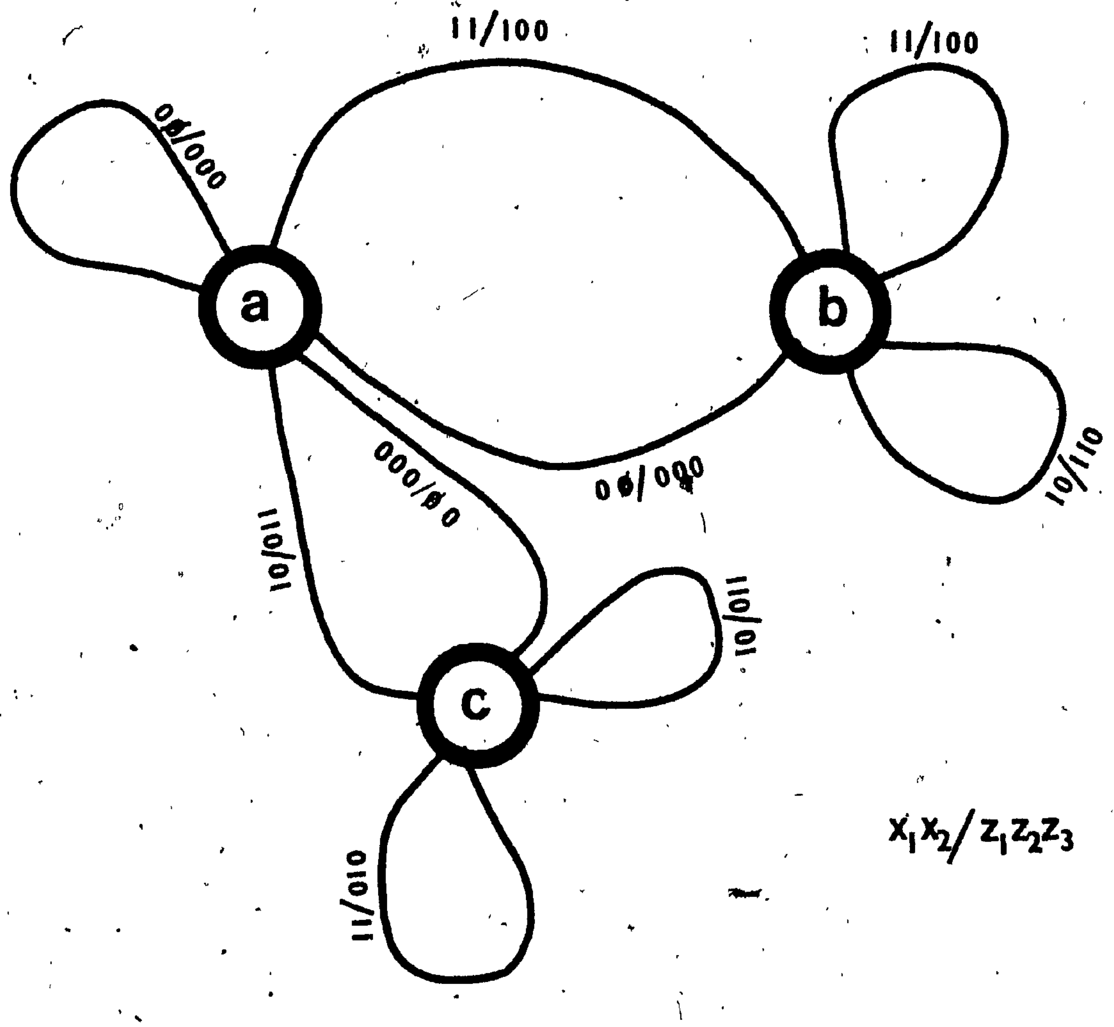


Fig. (28) Reduced State Diagram: Auto Safety Problem

Each state is represented as a loop in Binary Decision Diagram as show in fig. (29).

The Binary Decision Program may now be encoded into Binary Notions as shown in Table ~~(8)~~.

Table (8): Binary Notation

ADDRESS	(IR) OP CODE	ADD. INPUT VARIABLE	(AR) JUMP ADDRESS OR OUT PUT VAR.
000000	00	000000	00000110
000001	00	000001	00001010
000010	10	000100	00000000
000011	00	000000	00000110
000100	00	000001	00000010
000101	10	000110	00000011
000110	10	000000	00000000
000111	00	000000	00000110
001000	00	000001	00001010
001001	10	000010	00000111
001010	10	000011	00000111

The program shown in Table (8) is now ready for loading.

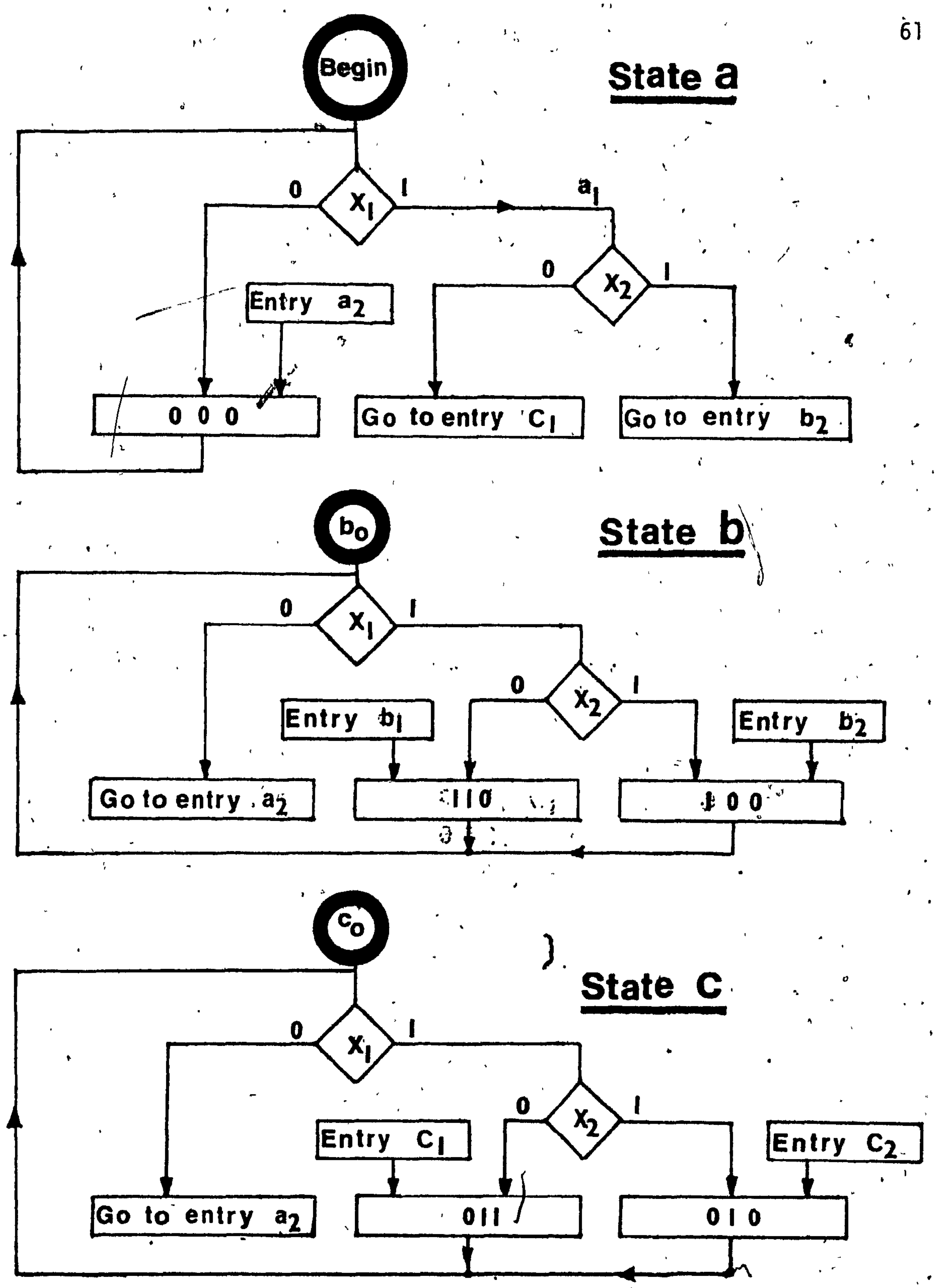


Figure (29) Binary Decision diagram for state a,b, & c.

CHAPTER 5

CONCLUSION

5.0. General

A Programmable Binary Decision Processor (PBDP) was designed to execute multiple output decision programs. The PBDP executes a Binary Decision Program for all outputs in parallel as compared to the serial execution of the output functions, on the Boute's machine (3).

The PBDP uses Read/Write Memory, which can be programmed, erased, and re-programmed repeatedly and, therefore, it makes possible to test a Binary Decision Program before it is burned into ROM's or EPROMs to perform a dedicated task.

The corresponding PBDP is developed, as the concept of binary decision trees was extended to include multiple output Boolean functions by Cerny, Mange & Sanchez (4). As mentioned in Chapter 3, Boute's realization, although, the first of its kind, based on a Binary Decision Program, is a serial executing machine & hence not suitable for executing programs based on multi output functions. Therefore, the corresponding PBDP has a definite speed advantage over Boute's realization.

In chapter 4, three applications of the Processor are demonstrated. These are just a few of the many possible applications of the PBDP. Example 1 and example 2 are based on a single output and multi output functions respectively. In example 3 the PBDP is used as a state machine where each state is represented by a loop, and then each loop is translated into a Binary Decision Program.

Various improvements and additions to the existing PBDP are possible. For example, timers and counters may be added to suit some specific application. As the PBDP can be used as a state machine, therefore binary decision programs may also be written to implement counter functions if needed.

Output expansion is possible by dividing the output bits into disjoint groups of bits such that each is a part of an output instruction field and can be viewed as an address of an output group while the second part could specify the values of the bits in that group. Thus, one can evaluate multiple output functions in serio- parallel manner.

As a possible application of B.D.P., Cerny and Moreau (6), have proposed a method for test verification of digital devices. Device input-output relationship is modelled by a decision program, executed then by a decision processor (specialized). Test sequences must be either pre-determined or random. For each test vector applied the decision processor checks the validity of the device's output.

REFERENCES

1. Shannon, C.E. 'A Symbolic Analysis of Relay and Switching Circuits'. A.I.E.E. Trans., 57,1938, PP.713.

2. LEE, C.Y. 'Representation of Switching Circuit by Binary Decision Programs', BSTJ July 1959, PP.985-999.

3. Boute, R.T. 'The Binary Decision Machine as a Programmable Controller', Euromicro Newsletter, 2(1976), PP.16.

4. Cerny, E., Mange, D., Sanchez, E. 'Synthesis of Minimal Binary Decision Trees', (To appear IEEE TC, July 1979)

5. Gregory, V., Dellande, B., 'MC14500B Industrial Control Unit Handbook'.

6. Cerny, E., Moreau, T. 'Test Evaluation with a Decision Machine'. (Manuscript accepted for the 9th Conference on Fault Tolerant Computing, FTCS-9, June 1979, Madison WI)

7. Shanon, C.E. 'The Synthesis of Two-Terminal Switching Circuits', B.S.T.J., vol. 28, JAN, 1949, PP.59.

8. Muller, D.E. 'Complexity in Electronic Switching Circuits, I.R.E. Trans, EC6, 1956, PP.15.

9. Cerny, E. 'Controllability and Fault observability in Modular Combinational Circuits', IEE TC, VOL. C-27, #10, October 1978.

10. Mange, D. 'Arbres de decision pour systems logiques Cables au programmés' Tirage a part du Bulletin ASE/VCS, t.69(1978) No.22, PP.1238 1243.

11. Thayse, A. 'Optimization of Binary Decision Algorithms', M.B.L.E. Research Laboratory (Bruxelles), Report R348 May 1977.

12. Mange, D., Sanchez, E. 'Synthese des Fonctions Logiques avee des Multiplexeurs', submitted for publication to Digital processes, May 24, 1977.

13. Webb, R.V. 'Sequential Controllers used for Industrial Automation', 2nd Conference on Industrial Robot Technology, Birmingham (Eng.) March 1974, PP. B4/47-56.

14. Smith, D.L. 'The problem with programmable Controllers', IEEE Transactions IECL, vol. IECL-21, No 2, PP.50-52 (May 1974).

15. Sanchez, E. 'Programmes et machines de decision binare', Seminar presented at 'EPF-Lausune, chaire de systèmes logiques, June 1978.

16. Fredrick. J'H., Gerald.R.P. 'Introduction to switching theory and logic Design-2nd Edition 1968 John Wiley & Sons.