COMPUTER SIMULATION OF DIGITAL SYSTEMS

A DISSERTATION

Submitted in partial fulfilment of the
requirements for the award of the Degree
of
MASTER OF ENGINEERING
in
ELECTRICAL ENGINEERING
(SYSTEM ENGINEERING AND OPERATIONS RESEARCH)

By

ARUNA SHARMA

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE (INDIA)
September, 1979

CERTIFICATE

Certified that the dissertation entitled 'COMPUTER CONTROL OF DIGITAL SYSTEMS' which is submitted by Aruna Sharma in partial fulfillment of the award of Degree of Master of Engineering in System Engineering and Operation Research of the University of Roorkee, Roorkee is a record of student's own work carried out by her under my supervision and guidance. The matter embodied in this dissertation has not been submitted for the award of any other degree or diploma.

This is further to certify that she has worked for a period of 8 months from 4th January 1979 to September 12, 1979 for preparing this dissertation at this university.

Dated September 12, 1979

(M.L. Vasanthi)
Reader
Department of Electrical Engg.
University of Roorkee,
Roorkee.

ACKNOWLEDGEMENT

The author deems it a privilege to have worked for his dissertation work under the guidance of Dr.M.K.Vasantha, Reader, Department of Electrical Engineering, University of Roorkee, Roorkee. The author wishes to express his deep sense of gratitude and indebtedness to Dr. Vasantha for his encouraging guidance and enlightening suggestions. The deep interest shown by him for solving the author's difficulties will be remembered with gratitude of no parallel.

The author is highly thankful to Dr.L.N. Ray, Professor and Head, Electrical Engineering Department, for providing computer facilities despite a large constraints on computer time.

Thanks are also due to those who helped the author directly or indirectly in preparing this dissertation.

Asharma

ROORKEE — ARUNA SHARMA

Dated Sept. 12, 1979

A B S T R A C T

In this dissertation, a study of the computer simulation of Digital systems using digital design functional language (Register transfer language) is made. It has been found that any type of digital system can be simulated by using either Arithmatic model approach or by Logic model approach. Simulation of INTEL 8080 microprocessor architecture is taken as a special problem.

The dissertation has been divided into five portions.

The first chapter describes the R.T.L. language used for the design, simulation and implementation of digital systems. In the second chapter study of digital simulation using Arithmatic or Logic approach is made and for these models subroutines are developed. Some examples are also given for its illustration. The chapter third deals the architecture of a hypothetical LLNC computer, and its simulation models. Chapter four gives the Arithmatic models and Logical models of an Associative memory of hypothetical Computer. In the last chapter INTEL 8080 microprocessor is simulated on an existing digital computer, indicating the importance of digital simulation techniques.

CONTENTS

Chapter Page

Chapter Page

CHAPTER-1

R.T.L-A LANGUAGE FOR DESIGN, SIMULATION AND IMPLEMENTATION OF DIGITAL SYSTEMS

1.1 INTRODUCTION

A system which converts the discrete or continuous signal to its analogous electrical signal is known as digital system and can be represented by figure 1.1.

The digital system design is separated into three aspects:(1)

i) component design,
ii) functional design, and
iii) system design.

All such digital systems can be classified into two general categories:

(a) combinational systems,

(b) sequential systems.

1.1.1 Combinational Systems

If in a system the TRUE/FALSE values of the output signals depend only on the current TRUE/FLASE values of the input signals, and there is no feed-back from the output signals to the input signals then the system is known as combinational digital system. Figure 1.2, illustrates a combinational digital system.

In designing combinational digital system all input and output signals to and from the system are expressed in TRUE/FALSE

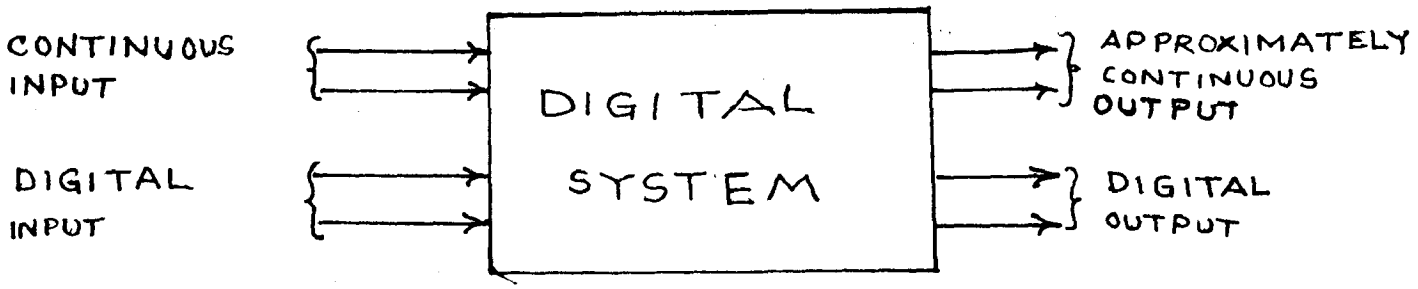


FIG 1.1 A GENERALIZED DIGITAL SYSTEM

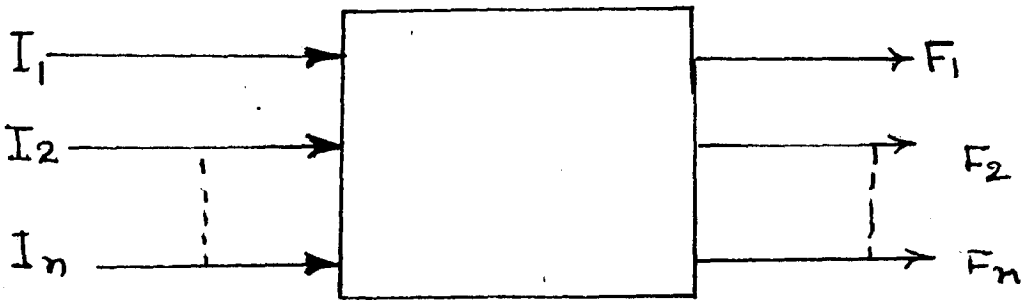


FIG 1.2 COMBINATIONAL SYSTEM

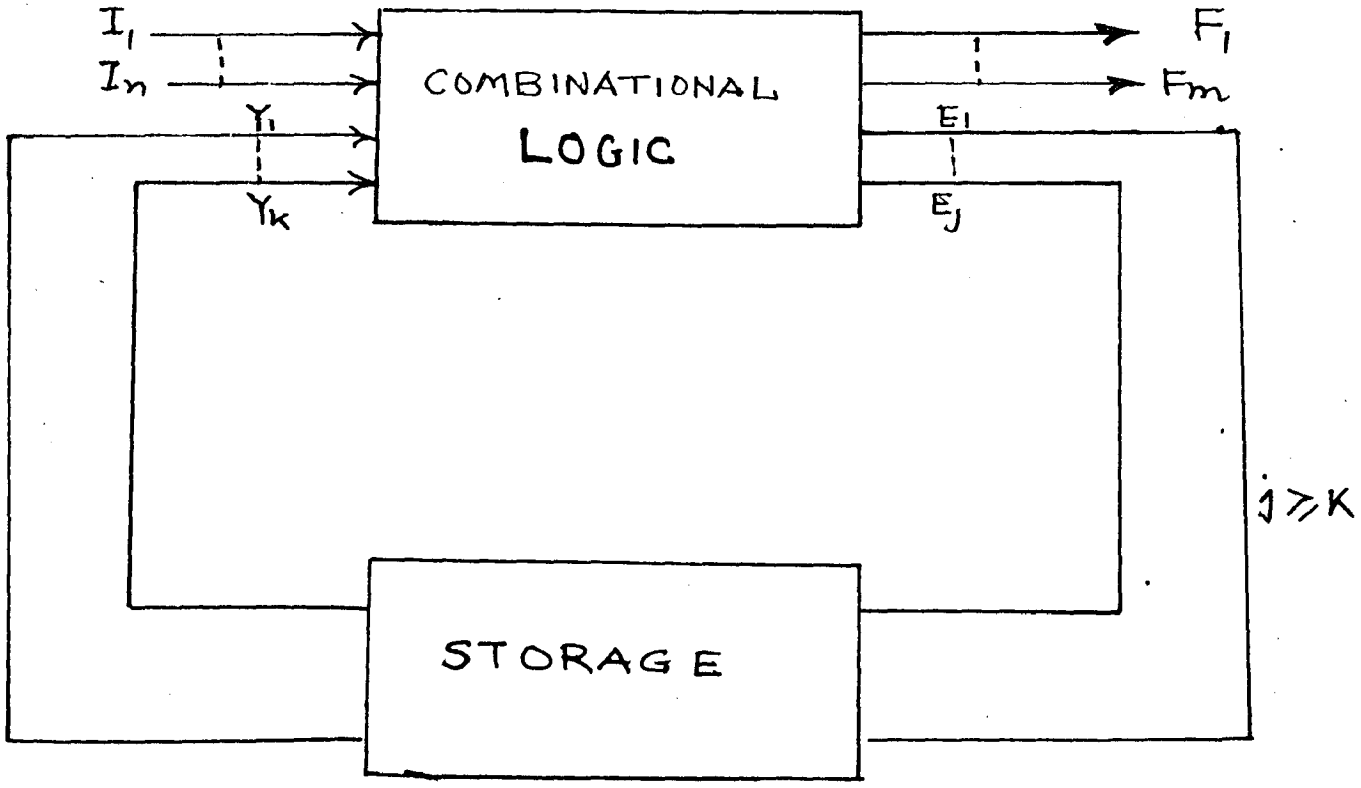


FIG 1.3. GENERALIZED SEQUENTIAL SYSTEM

form and represented by boolean variables through truth table. According to the digital system requirement Boolean logic expressions are formed. These expressions are minimized by using Karnaugh map technique (Desk Minimization) or by Quine-McCluskey technique (computer aided mini ization). The system is then constructed with suitable logical devices, such as AND, OR, NOT, NAND, NOR, gates.

1.1.2 Sequential System (5)

Sequential systems are characterized as having two properties:

1. There is atleast one feedback path from output of the system to the input of the system.
2. The system has a memory capability for holding past information, so that the previous input and output values can be used in determining the current output signals.

The generalized sequential system is shown in figure 1.3. During each present state, the values of the primary and secondary output variables are determined by the combinational operation upon the primary and secondary input variables. The computer aided sequential system design involves formulation of state diagrams, state table, state minimization, state assignments, simplifications using Quine-McCluskey technique before circuit implementation.

The digital system design at systems level on the other hand involves interconnection of different sub-tasks (which themselves are sequential in nature) using state controller.

1.2 STEPS IN LOGIC DESIGN OF SYSTEM LEVEL

The logic system design involves the following steps at system level.[0,1]

i) An algorithm is obtained from the given requirement so that the different subtasks may be performed in a logical sequence to arrive at the final result.

ii) From the algorithm, determine the number of registers needed and their capacities. Also find the flow of information between registers and the logical operations to be performed on the information.

iii) Break-up the algorithm into a sequence of micro-operations. Identify operations which could be carried out simultaneously and those which are to be sequential.

iv) Synthesize counter decoder circuit to generate the timing signals which control the identified micro-operations.

v) Using the control timing signals and the identified logic operations, complete the logic design.

Algorithm is a sequence of micro-operations. If well defined notations are used then this can be presented in a simplified way. These notations are called 'Digital System Design Language'.

1.3 DIGITAL SYSTEM DESIGN LANGUAGE

The digital system design language has facilities to specify registers and describe logical operations and information transfer between registers in a digital system. The three major sections of this language are

a) Declaration,

b) Micro-operations, and

c) Sequencing.

1.3.1 Declaration

The declaration describe the hardware blocks used, such as registers, counters, decoders, flip-flop, code-code convertor used in the given logical system. Declaration is nothing but a list of components used along with bit descriptions and its use. (1,3,8)

A register 'X' of 10 bits is represented by the following declaration:

'DECL' REG,X(0-9)

The ten bits of register X are referred to as 0,1,2,...9, counted from right to left (X_0 being LSB and X_9 being MSB)

(a) REG, A(0-15),B(1-3), C(0-31)

It declares that there are three registers named, A, B, and C. A is a 16 bit register referred to 0,1,2,...15;

B is a 3-bit register referred to as 1,2, and 3, from right to left. C is a 32-bit register referred to as 0,1,2,3,...31

Table 1.1 Declaration statements

S.No.	Declaration type	Meaning
(a)	REG,A(-3),B(0-3)	Two registers A and B. A is a 4-bit register referred to as 0,1,2, and 3 and B is 6-bit register referred to as 0,1,....5 from left to right
(b)	FLIP-FLOP, C,F.	C and F are two flip-flops.
(c)	CLOCK, 10	Standard length of register is 10-bit
(d)	SUBREG, ADDR(1-6)=1(10-15)	Sub-register, new name I is given to the register ADDR to the selected bit sequence.
(e)	COMPARE,Z(0-31)=A(0-15), B(0-15)	Comparator. 0-15 bits of Z register are identical to 0-15 bits of A register and 16-31bits of Z register are identic l to 0-15 of register
(f)	Decorder, Def(1-2)	2-bits of the Z register form the inputs to the decoder and the 4-outp ts are labelled as D(0), D(1) D(2) and D(3), indicating the four combinations 00,01,10 and 11 of the bits of Z.
(g)	C , P	P is a clock generating pulses at a pre-determined rate.
(h)	Memory, M(L,N)	Memory unit having L words with each word having N bits.
(i)	LIGHT,P,A(1-3)	
(j)	SWITCH,C,MS(1-3)	Two switches C and MS switch C has one position and switch MS has 3 positions

Table 1.1 continued

S.No.	Declaration type	Meaning
(k)	CK, P(100-6), Q(27.5 E-6) [2]	P and Q are two clocks. P with a one nsec. and Q with 27.5 nanosec. period and two phases Q(1) and Q(2).
(l)	DI, D1(10E-9)	Delay of 10 nano-sec is introduced by clock D1.
(m)	LO, D(4)=BVC	Boolean operation or Logical 'OR' operation between register B and C and the result is stored in D(4) register.
(n)	BLOCK, SN7400(O(4),I(4:2)	Eight inputs and four outputs block, ight be a quad 2-input NAND package.
(o)	DATA,A,B,C 001 11 1010	A has 001 as data, B has 11 as data and C as 1010 as data

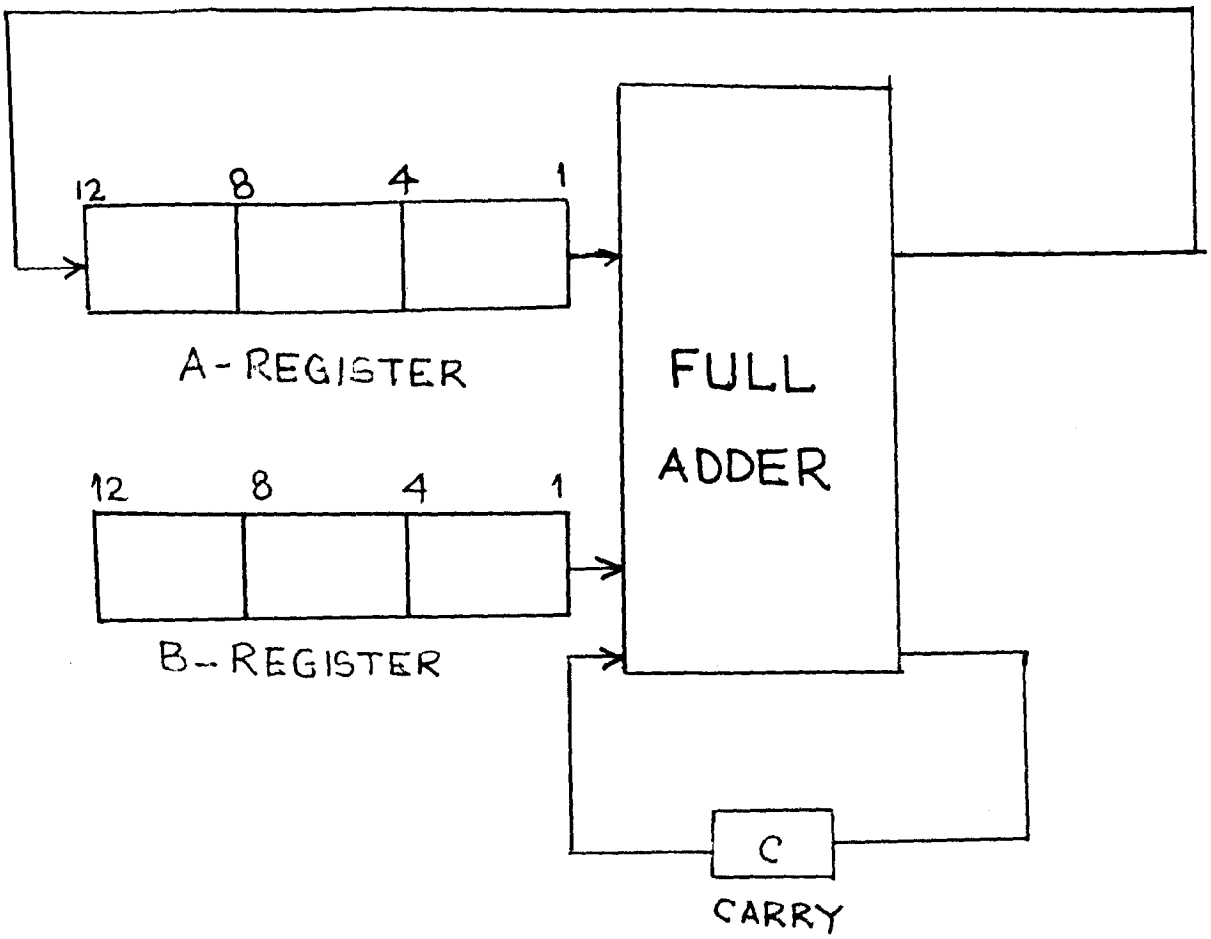


FIG. 1.4 6-BIT BINARY ADDER

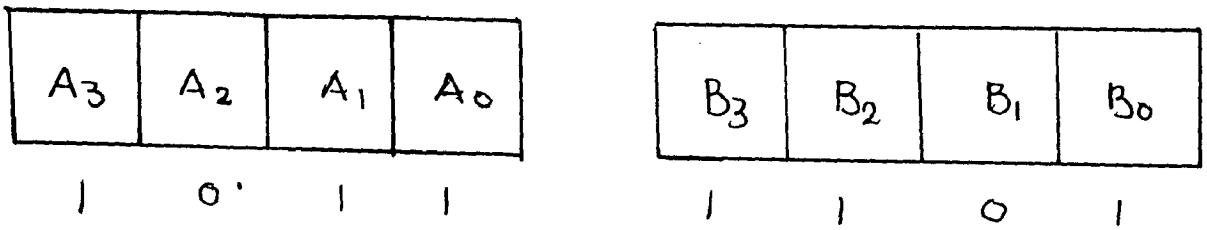


FIG. 1.5 MICRO-OPERATION $A_1 \leftarrow B_1$

from right to left.

Many types of declarations are given in the table 1.1, given on the coming page.

Example

Declarations for the 6-bit binary adder.

DECLARATIONS

SIZE, 6
REG, A,B
FLIPFL, C,
DMSA, A,D.
EL, FULL ADDER

Giving A and B are registers of 6-bit and C is a flip-flop. DMSA is taken from registers A and D.

1.4 MICRO-OPERATIONS-R.T.L. LANGUAGE DESCRIPTION

Micro-operations are statements that give the inter-connections and interactions of facilities and some or all of the ~~condition~~ conditions under which those connections are to be made and actions are to be performed. (1,9,0,3)

These are best described by register transfer logic (R.T.L.)

This language is similar to most of other programming languages. The R.T. level is a generalization of the switching circuit level.

It is a language for describing the information flow and processing between registers. The data flow and control operate in discrete steps.The elements are combined according to some

rule and then stored into another register.

The rules of transformation are:

(i) The contents of the register will be denoted by one or more letters with the first always being in upper case.

(ii) Parallel (not serial) transfers will occur between registers, i.e. all bits will be transferred at the same time.

(iii) The bits of each register are numbered from right to left with the least significant bit (L.S.B) labelled as zero.

(iv) The structural elements are arrays of identical subsystems belonging to switching level; (i.e. registers are made of flip-flop and gates driven by clocks).

Most R.T.L. include declaration statements for specifying the existence of basic elements, indicating their type and dimension, and associating a name with each element or collection of elements same as discussed earlier.

For example: Memory elements are commonly abstracted to basic type of memory or register. Then a statement such as

MEM,M(6,16)

declares the existence of 96 binary memory elements organized into a 2-dimensional array known as M, with 16 bit word length indexed from 0 to 5.

Micro-operations declare the existence of connecting and combining logic circuits more or less explicitly depends upon the R.T.L. Five types of micro-operations are :

(1) Basic Register-Transfer operations,

Table 1.2 Basic Register-Transfer Operations

S.No.	OPERATION	MEANING	EXAMPLES CONTENTS OF REGISTER		REMARKS
			BEFORE	AFTER	
1.	$A \leftarrow B$	The content of register B are transferred to register A	A = 1101; B = 0110;	A =0110 B =0110	The contents of register B remain same.
2.	$A_2 \leftarrow B_3$	Transfer the bit 3 of register B to bit 2 of register A	A = 0011 B = 1111	A =0111 B =1111	
3.	$A_2 \leftarrow A_3$	Transfer bit 3 register A to bit 2 register A	A = 0101	A =0001	
4.	$A_{1-3} \leftarrow B_{1-3}$	Transfer bits 1 through 3 from register B to A	A = 0011 B = 1100	A = 1101 B = 1100	The bits 1-3 of register A has the value 1-3 of register B and the register B remains unchanged.
5.	$A_{1,3} \leftarrow B_{1,3}$	Transfer bits 1 and 3 of register B to 1 and 3 of register A	A = 1111 B = 0010	A = 0110 B = 0010	
6.	$A \leftarrow BAC_E$	Transfer the group of bits called E from BAC register to register A with right justification	A = 0100 B = 1011	A = 0101 B = 1011	The group E is defined, the contents of this group is shifted register A and the register B remain unchanged

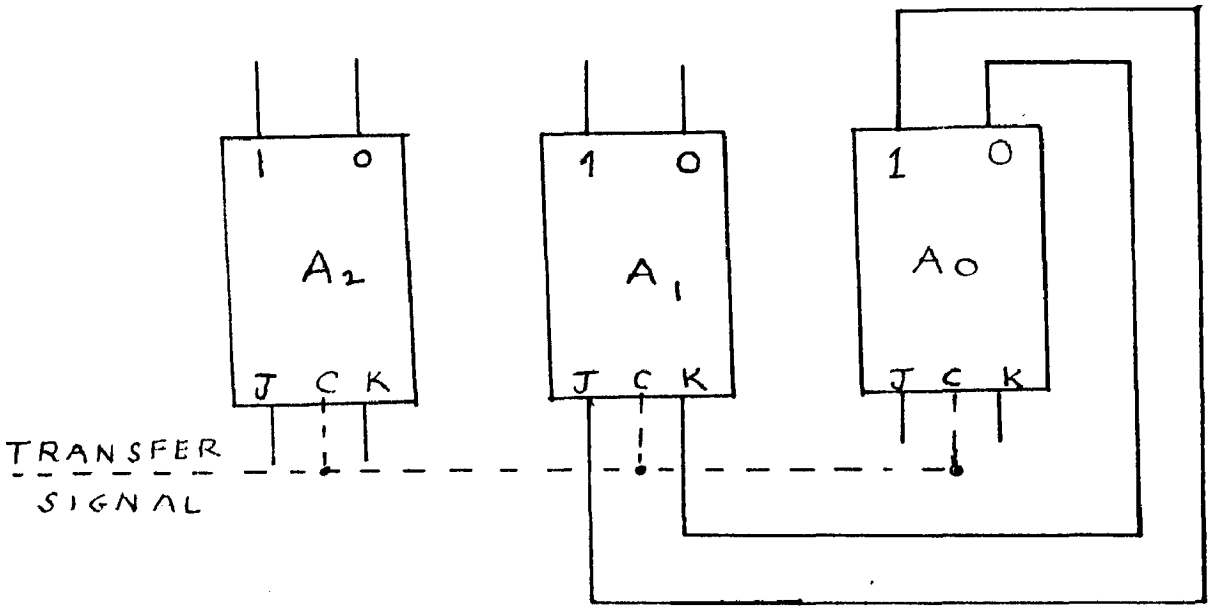


FIG 1·6 LOGICAL DIAGRAM $A_1 \leftarrow A_0$

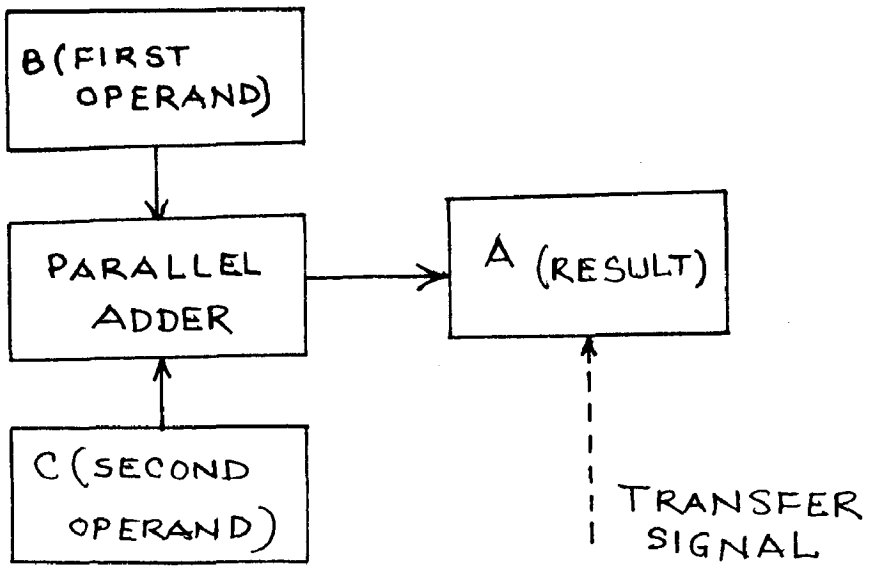


FIG 1·7 LOGICAL DIAGRAM FOR PARALLEL ADDITION

$A \leftarrow B + C$

(2) Arithmetic-operations.
(3) Logical operations.
(4) The Shift, Rotate and Scale Operations
(5) Memory and conditional operations.

1.4.1 Basic Register-Transfer Operation

This operation shows the transfer of bits of one register to the other place.

A has value 1011

B has value 1101

The operation $A_1 \leftarrow B_1$ states that transfer bit 1 of register B to bit one of register A. The content of register B remains unchanged after operation.

The operational statements such as $A_1 \leftarrow B_1$ or $A \leftarrow B$ (Data Transfer from Register B to Register A with contents of B remaining unchanged) is called a PRODUCTION. Many productions involving Basic Register Transfer operations are given in Table 1.2 coming on the next page.

After operation A has value 1001 while B has value 1101.

If R.T.L. micro-operation is known, its hardware implementation can be done with ease.

For example: The micro-operation $A_1 \longleftarrow A_0$ can be implemented as shown in figure 1.6.

1.4.2 Arithmetic Operations

Adders, subtractors and number complement operations are very common to digital systems. Whether number is 1's

Table 1.3 Arithmetic Operations

S.No.	Operation	Meaning	Before Execution	After Execution	Notes
1.	A ← 0	Clear register A	A = 1100	A = 0000	Register A has zero value
2.	A ← B+C	Transfer the arithmetic sum of B and C registers	A = 0001 B = 1100 C = 0011	A = 1103 B = 1100 C = 0011	Whatever the value of register A is immaterial and after operation it is the arithmetic sum of B and C
3.	A←B−C	Subtract C from B and transfer the result to A	A = 0123 B = 1420 C = 0011	A = 0117 B = 1420 C = 0011	The value of register A is immaterial before operation and register B and C remain unchanged after operation
4.	A←A+1	Increment A by 1	A = 0011	A = 0100	Register A is incremented
5.	A←Symbol	Load octal number 1500 into register A with right justification	A = 0001	A = 1500	A is a 12 bit register.

complement or 2's complement, arithmetic can be made with the assumption that arithmetic operators obtain n-bit result for n-bit operands while the relation operators give a single bit result. These operators usually can not be simulated or taken to imply circuitry.

For example A ← 0

states that the register A is cleared i.e. all bits of A are replaced by zero. Some common types of Arithmetic operations are given in table 1.3.

The implementation for the parallel addition of 2-operands B and C can be done as shown in figure 1.7. The result is stored in register A and the micro-operation is A ← B+C

1.6.3 Logical Operations

Logical operations are very common part of digital system operation. Operators AND, OR, and NOT are well known operations. If two operands of dimension n are given, they perform operation on bit-by-bit basis and given results in n-dimension. One of the operands has unit dimension, its extension to n-dimension is implied and a result of n-dimension is formed.

The following logical operations are valid for RTL.

Table 1.4 Logical Operations

S.No.	OPERATION	MEANING	EXAMPLES CONTENTS OF REGISTER		Remarks
			Before Operation	After Operation	
1.	$A \leftarrow B \vee C$	The OR operation of the corresponding bits of B with C is transferred to A	A=0100 B=1010 C=1000	A=1010 B=1010 C=1000	Register A is the 'OR'ed of register B and C, the value of register B and C remain unchanged
2.	$A \leftarrow B \wedge C$	Transfer to register A the AND combination of the corresponding bits of B with those of C	A=0111 B=1001 C=1111	A=1001 B=1001 C=1111	Register A is the 'AND' result of register B and C.
3.	$A \leftarrow B \wedge C_2$	Transfer to register A the AND combination of the corresponding bits 2 of C with individual bits of B.	A=1100 B=1001 C=1100	A=1001 B=1001 C=1100	The bit 2 ofregister C is AND ed with the individual bits of register B, the B and C remain unchanged.
4.	$A \leftarrow B \wedge C_1$	Transfer to register A the AND combination of the corresponding bit 1 of C with individual bits of B	A=1101 B=0011 C=1100	A=0000 B=0011 C=1100	
5.	$A \leftarrow B \vee C_2$	Transfer to register A the OR combination of bits 2 of C with the individual bits of B	A=0000 B=1101 C=0111	A=1110 B=1101 C=0111	The register A has value as the OR ed value 2nd bit of register C with the individual bits of register B. B and C remain unchanged.
6.	$A \leftarrow \bar{A}$	Complement the individual bits of A	A=1001	A=0110	All bits of register A are inverted

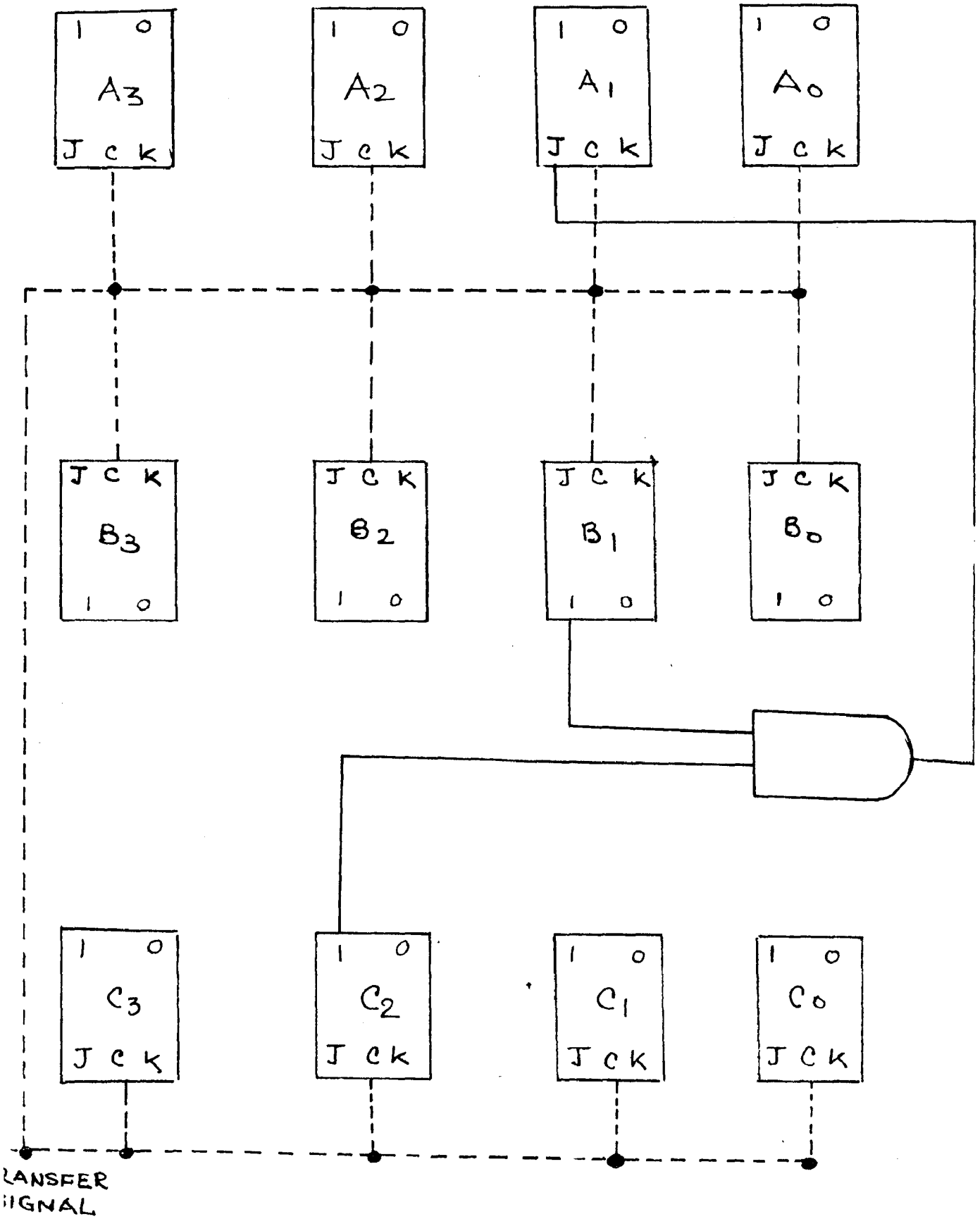


FIG 1·8 LOGICAL DIAGRAM FOR $A_1 \leftarrow B_1 \wedge C_2$

Table 1.3 Rotate, Shift and Scale Operations

S.No.	Operation	Meaning	Examples, contents of registers Before	After	Remark
1.	A←SrA	Shift register A right by one bit	A = 1011	A = 0101	The L.S.B. is lost
2.	A←SlA	Shift register A left by one bit	A= 1011	A = 0110	The M.S.B. is lost
3.	A←Sl3A	Shift register A left by 3-bits	A= 1100	A = 0000	
4.	A←rrA	Rotate register A right by 1-bit	A= 0101	A = 1010	The L.S.B. becomes M.S.B.
5.	A←rlA	Rotate register A left by 1-bit	A= 1100	A = 1001	The M.S.B. goes to L.S.B. Position and vice-versa
6.	A←rr2A	Rotate register A right by 2 bit	A= 1100	A = 0011	Position and vice-versa
7.	A←cc A	Scale register A right by 1-bit	A= 0110	A = 0011	The register A is first shifted right and the sign bit remains unchanged.
8.	A,B←Sl A,B	The concatenated register A,B is shifted left one bit	A,B=all 00 A B	A,B= 0100	The M.S.B. of A is lost and M.S.B. of B is also lost

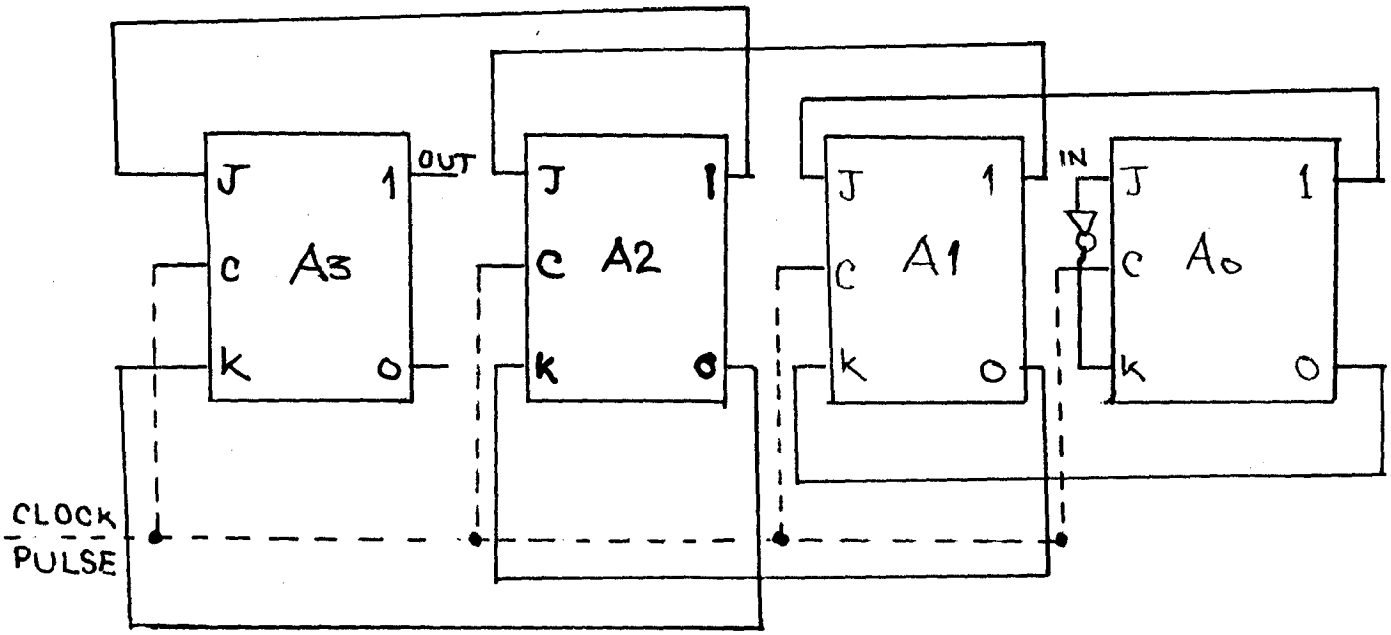


FIG. 1.9 LOGICAL DIAGRAM FOR A←SLA

1.4.4 Shift, Rotate, and Scale Operation

Shift, rotate operations of various registers are very important micro-operations of digital system through which certain major arithmatic operations (like multiply, divide) can be performed. Shift operations are register transfer operator. Counting is special case of addition and subtraction.

rr = rotate right

rl = rotate left

sl = shift left

sr = shift right

sc = scale.

The implementation of the shift operation A ← SLA is shown in Fig.1.9.

1.4.5 Memory and Conditional Operation

This operation gives how memory communicate under different conditions. The condition statement 'control' the information processing. These are used to select actions under a condition generated by a test network then network is described by boolean expression.

For example: IF(A≥0)B ← 0, C ← 0, is a conditional statement, states that registers B and C are cleared if the contents of register A is greater than zero or equal to zero.

The common types of productions under this group are listed in table 1.6.

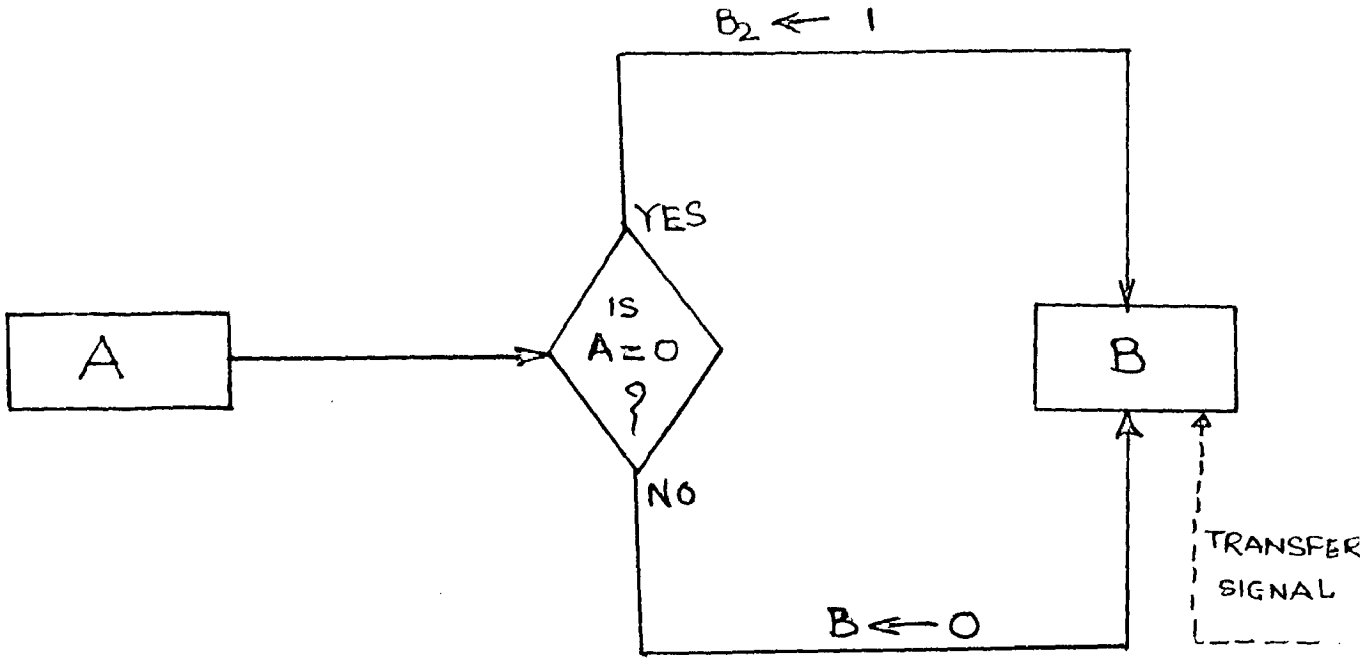


FIG 1·10 RTL TRANSFER DIAGRAM

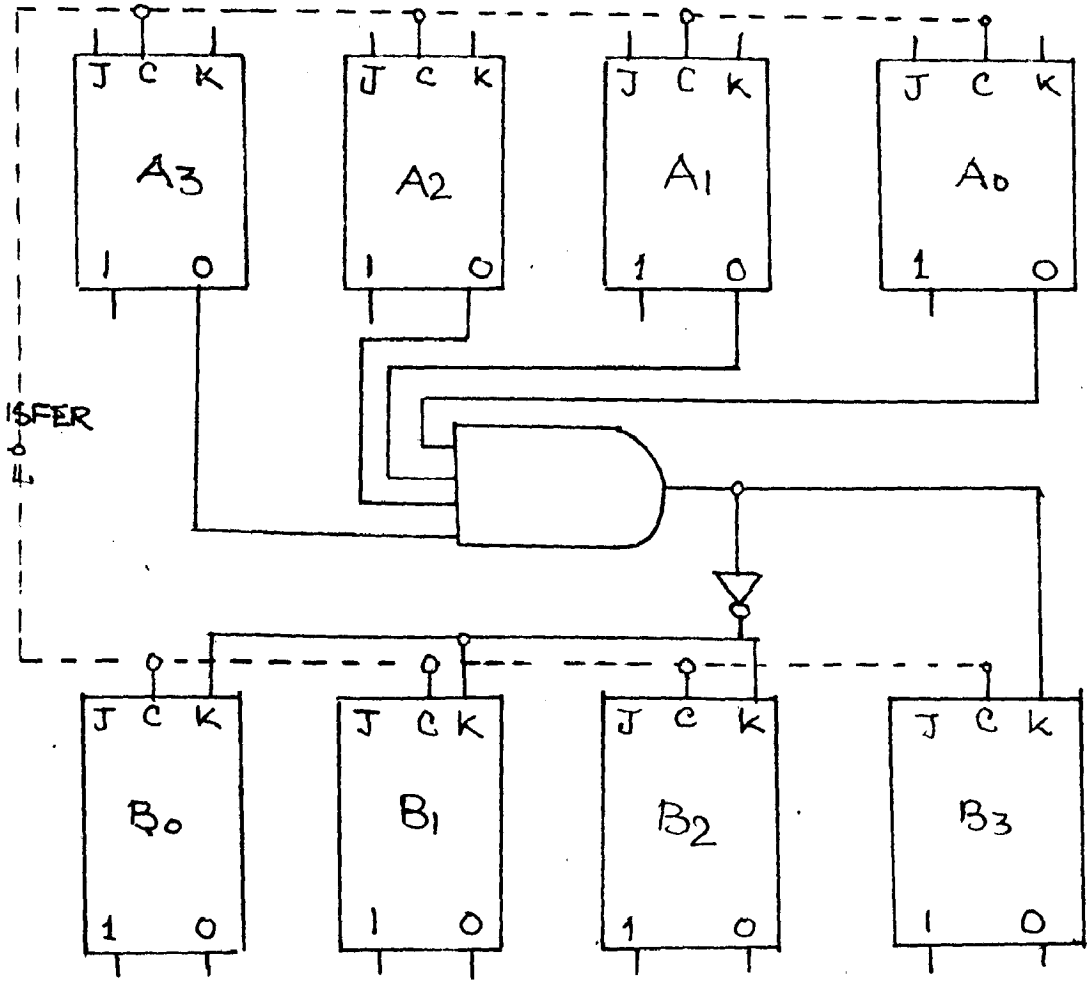


FIG 1·11 LOGICAL DIAGRAM FOR THE ABOVE RTL DIAGRAM

Table 1.6 Memory and Conditional Operations

S.No.	Operation	Meaning	Examples, contents of Registers		Remarks
			Before	After	
1.	A ← M(100)	Load register A from content of memory location whose address is 100	A=dddd M(100)= 1000	A=1000 M(100) =1000	The register A do not care and contents of M(100) remains unchanged.
2.	M(200)←A	Load the contents of register A into memory location whose address is 200. Contents of M addressed by MA goes to MB	A=4250 M(200)= 1690	A=4250 M(200)= 4250	The contents of register A remains unchanged.
3.	MB←M(MA)				
			MB=1020 M(MA)=4440	MB=4440 MB=4440	Contents of M(MA) remains unchanged
4.	IF(A_3=1)B←0	If bit 3 of register A is one, then clear register B.	A=1100 B=1110	A=1100 B=0000	B is made clear after operation
5.	If(A≠0)B←0 Else B_2←1	If A is not equal to zero, clear B, otherwise set bit 2 of B to 1.	A=0110 B=1100	A=0110 B=0000	If A = 0 then B will remain unchanged
6.	If(A,B_2=0,1) C← 0	If A is cleared, and if bit 2 of B is 1 then clear C.	A=0000 B=111 C=dddd	A =0000 B =1111 C =0000	Whatever the value of C is immaterial before operation.
7.	If((D_1=1 OR C=0) AND D=0) A_1←1	If bit 1 of B is 1 OR C is zero AND D is cleared then set bit 1 of A to 1.	A=0100 D=1010 C=0000 D=0000	A =0110 B =1010 C =0000 D =0000	

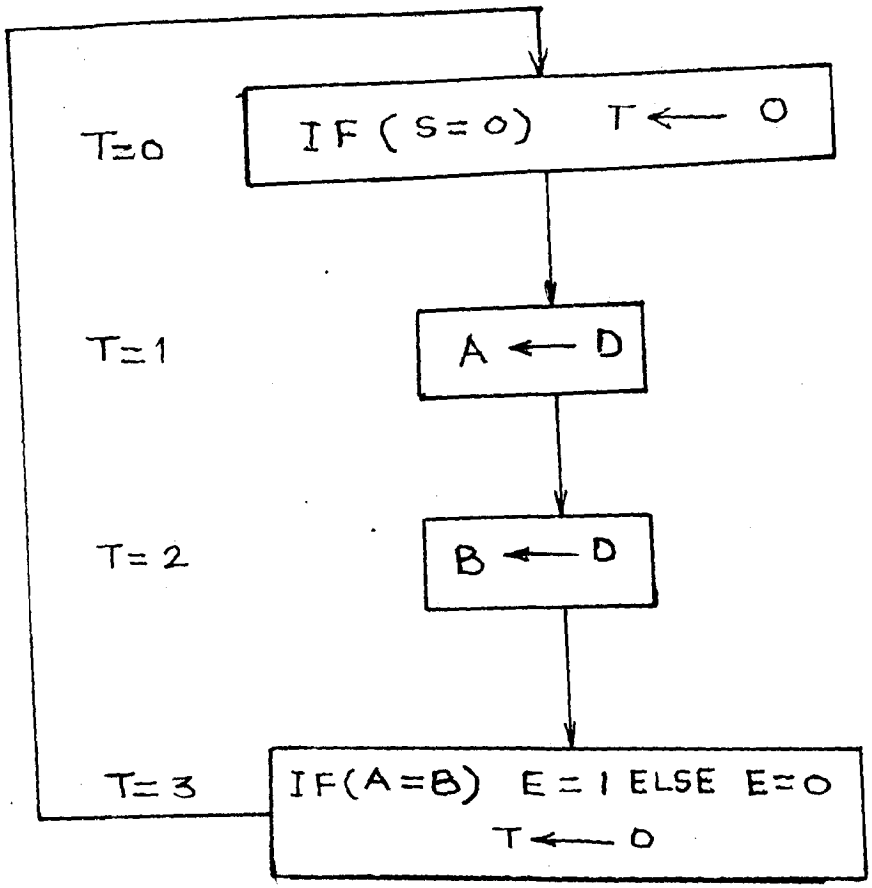


FIG 1·12 RTL PROGRAM FOR THE DATA STORACE AND COMPARISION

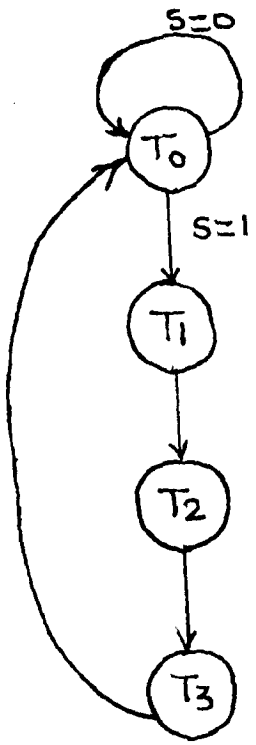


FIG 1·13 STATE DIAGRAM

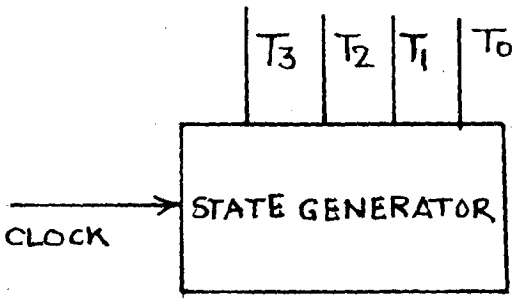


FIG 1·14 SCHEMATIC DIAGRAM OF STATE GENERATOR

The implementation of the conditional operation IF(A=0) $D_2 \leftarrow 1$ ELSE $D \leftarrow 0$ is as shown below.

1.9 MICRO TIMING AND SEQUENCE

The micro-operations are time sequenced by the control signals. Operations performed on a set of operands forms a a statement.

$$CPRO \leftarrow M(ADDR), \quad P \leftarrow P+1$$

states that contents of M(ADDR) replaces CPRO and clock is counted up simultaneously.

A sequence of statements will be numbered and each of them will be executed according to a three sequence.

This can be illustrated by the following example.

Let the two numbers are read from data source serially and then compared for equality. The result is stored in the next unit. The process is repeated for new data and so on.

The HDL program is written for the storage and equality of data. One number is written into register A from data source D and the other number in register B. Then number is checked for equality in the comparator and the process is repeated with new data as long as the start level S is TRUE.

Here T_0 through T_3 states occur in sequence and then the circuit return to state T_0. Figure 1.13 gives the state diagram.

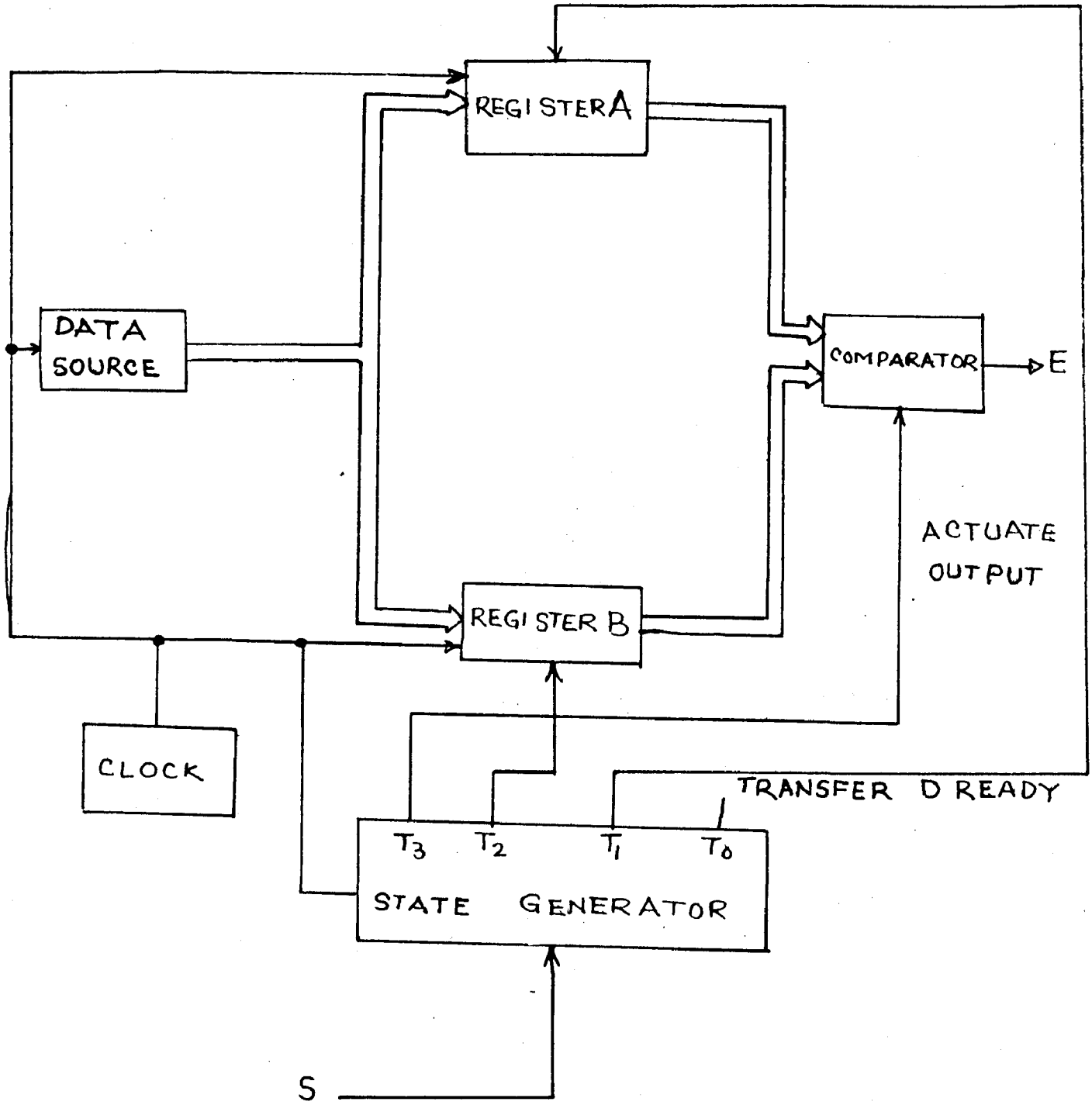


FIG 1.15 SYSTEM BLOCK DIAGRAM.

The R.T.L. program for the problem being known (fig.1.12) its hardware realization can be given as shown in fig.1.19.

When S is zero, state generator will have $T_0=0$, $T_1=0$, $T_2=0$ and $T_3=0$ outputs, and this time E=0 and registers A and B are disabled when state generator has output 0010 then register A is enabled and data is transferred to register A, B will remain disabled.When state-generator has output 0100, B is enabled and data is transferred to register B. When state-generator has output 1000 comparison operation is performed. If A=B then E=1,otherwise E=0; The same procedure will go on repeating with the new data and will continue till S = 1 and B has data.

Design of A 2-Digit BCD ADDER

Consider the following example for further details. The function of an adder is to produce the sum S of addend A and augend B considering the Carry C which may have been produced by the next lower decade. The adder generate a Carry C to the next higher decade.

The problem states that an adder is to design which adds two digits unsigned numbers stored in 8421 code. Storing each digit requires one 4-bit register and result may have 5 digits. The RTL program for the design of 2 digit BCD adder is shown in figure 1.15. The addend and augend digits are read into register X and Y respectively. The addition operation is done in 1 digit BCD adder. Result is stored in register Z. The process is repeated till start level S remains TRUE.

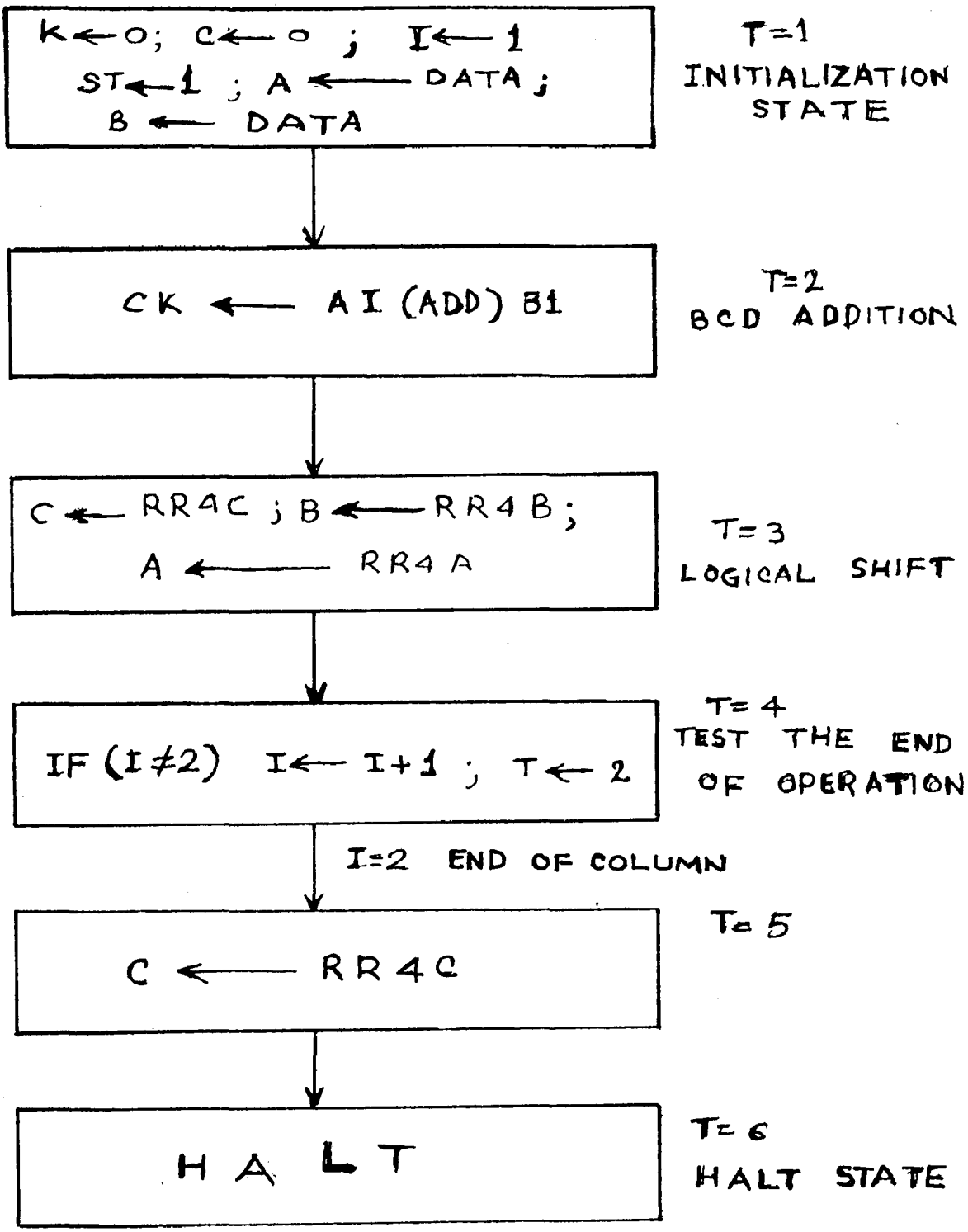


FIG. 1.15. RTL PROGRAM FOR 2-DIGIT BCD ADDER

The states T2 to T4 occurs in sequence and in the state T5 the result in BCD code is obtained. The figure 1.16 shows the block diagram for the circuit that realizes the RTL program of figure 1.15. The figure illustrates how addition operation is performed. The declaration statements used in the program are given below.

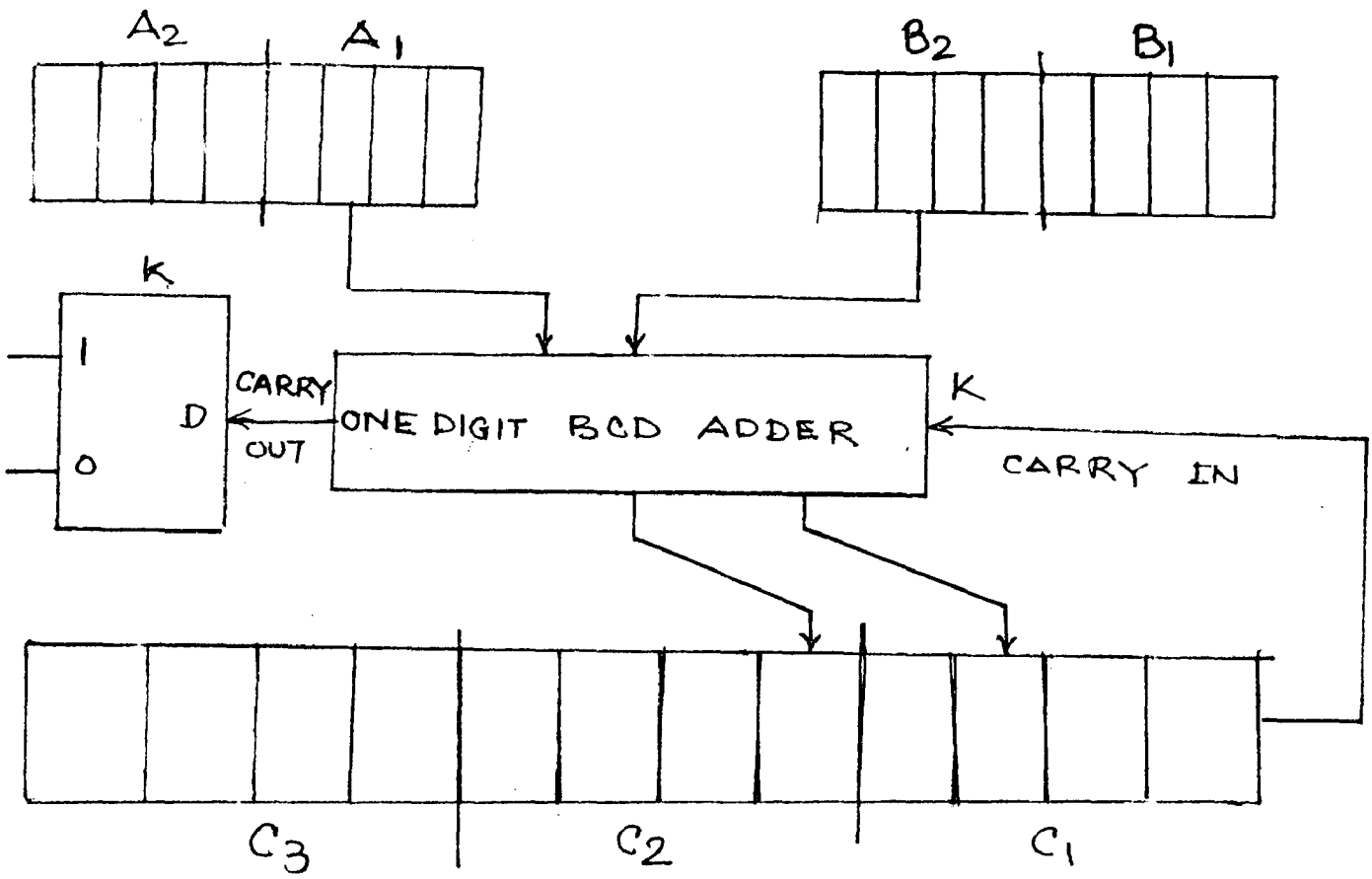
```
'DECL' CLARG, 4
       REG, A1,A2,B1,B2,C1,C2,C3
       FLPFL, K,AS
       CONREG, A=A1.A2, B=B1.B2, C=C1.C2.C3,
               CK = C2(4).C1.
```

The RTL production representing a one digit BCD adder CK← A1(ADD)B1 can be realized using fortran logical statements. The following program represents 1-digit BCD adder:

```
C C     ONE DIGIT BCD ADDER K(4),K
C ----- A1=Augend, B1=addend, K=CORRECTION VALUE
C ----- K=CARRY BIT, K=NO.OF BIT/CONREGISTER,CC=RESULT REGISTER
        LOGICAL=K, A1(4),B1(4),K(4),CK(5).
        K=.FALSE.  ←CALL ZADD(A1,B1,CK,K,K)
        IF((CC1.AND.CC2).OR.(CC1.AND CC3).OR.K) GO TO 50
        CALL EX(CC,C,K)
50 ---- CALL T ADD (CC,K,CK,K,K)
        RETURN
        END
```

The hardware realization for a 1-digit BCD adder for the above program is given in figure 1.17. This shows how a 1-Digit BCD adder do operations.

On the similar basis other systems can be designed and the usefulness of RTL language can be seen. Based on a properly predefined set of RTL primitives (or modules) the complete



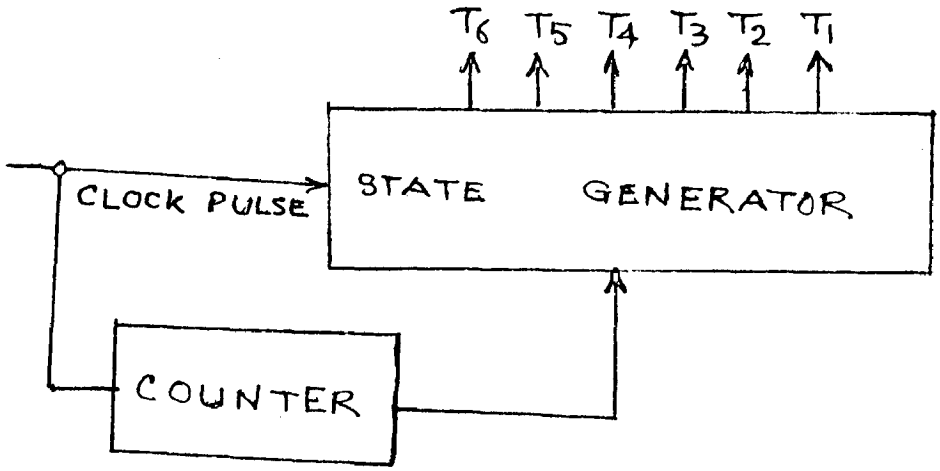


FIG 1.16 BLOCK DIAGRAM OF A 2-DIGIT BCD ADDER.

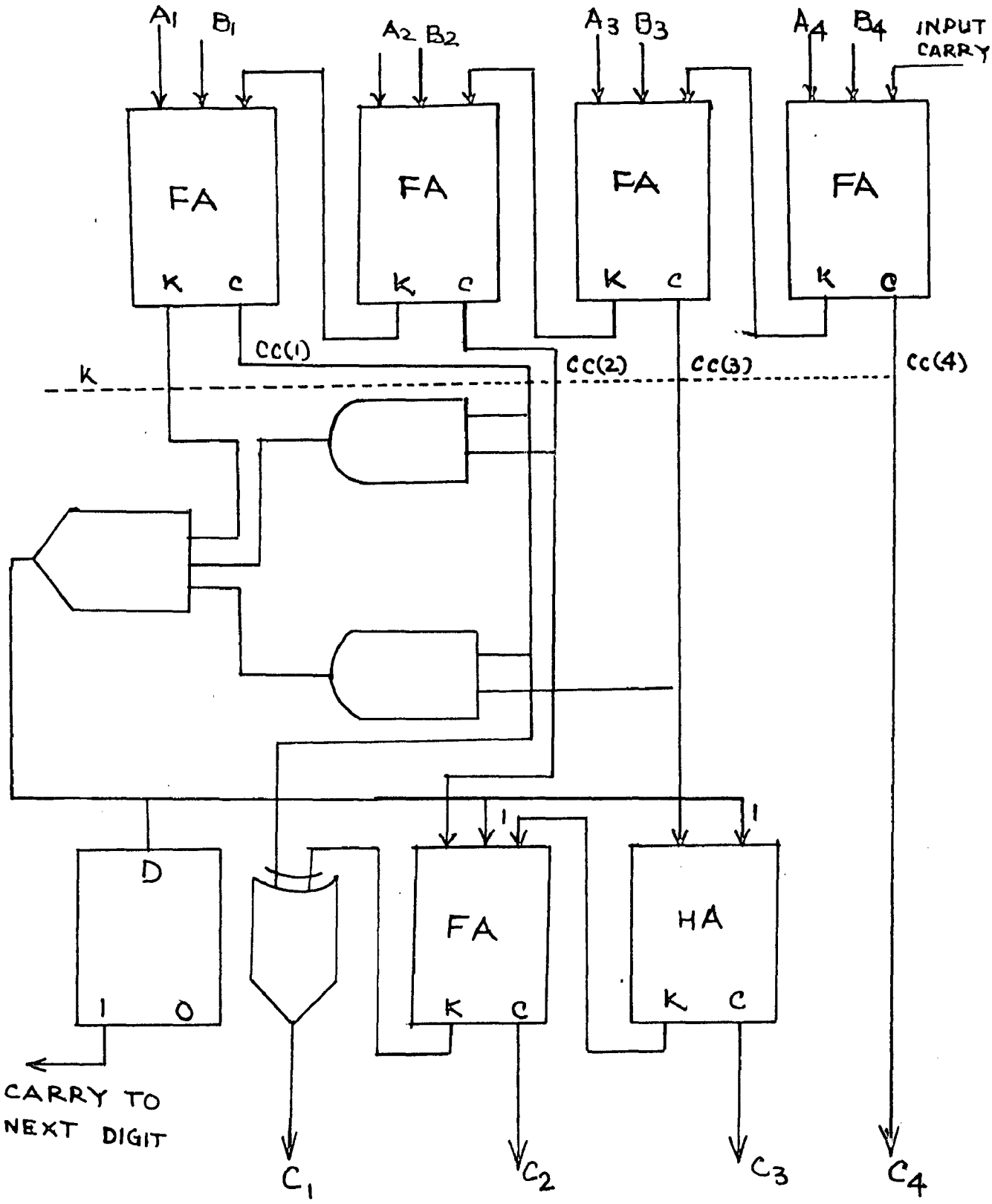


FIG. 1·17 SINGLE DIGIT BCD FULL ADDER

description of a logic system can be given in a very concise manner.

The benefits of the above approach provide a unique specification, a means of validation of the design and the element required for documentating and manufacturing the system.

The advantages of RTL can be summarized as follows:

(i) The ability to verify archi-integrity of a design.

(ii) It acts as a common specification language between hardware and software designers.

(iii) Simulation of the declared design in a digital computer to validate a concept while providing for a performance baseline as the source of data to debug hardware.

(iv) Minimum cost printed circuit board layout and manufacture.

(v) The grouping of RTL statement gives saving in states similar to the redundant states.

(vi) A direct conversion from RTL to assembly language, machine language and a direct conversion from RTL to hardware implementation.

(vii) A direct conversion from RTL program to FORTRAN IV programme for digital simulation on a digital computer.

1.6 LIMITATIONS OF RTL LANGUAGE

Although the register transfer level is a perfectly valid level of design, it has not been fully defined and understood due to some limitations. For example hardware constraints (such as environment) of the time of electronic system. Construction and the environment at the time of operation of the system. Some limitations are also imposed due to some practical rules. For example, conventional, single, unified memory only allows one location to be operated on at a time so the R.T.L. production like R(C) ← A; B ← R(B) or R(B) ← R(A); is not a valid R.T.L. production. More than two operands can not be added simultaneously. Multiplication and division is not an RT defined for R.T.L. Simultaneous as register exchange A ← B; B ← A is not valid since memory can not be read and written simultaneously even if the same address is employed.

In addition to these above general limitations there are some specific limitations for digital computers such as memory can not be read directly into two different buffer registers because in a digital computer only one buffer is assumed to exist. $\begin{pmatrix} MB \leftarrow M(MA) & T = 0 \\ MC \leftarrow M(MA) & T = 1 \end{pmatrix}$

MA ← SRSA is also invalid RTL production.

R(MA) ← A data paths do not exist for these
PC ← A separate productions.

When RTL flow charts are developed these above limitations must be taken into considerations, otherwise it will not give the correct/desired output.

CHAPTER -2

SIMULATION OF DIGITAL SYSTEMS AT SYSTEM-REGISTER AND GATE/LOGIC LEVEL

2.1 INTRODUCTION

Simulation is a process where by it is possible to model either mathematically or functionally the behaviour of a real system, experiments can then be conducted on the model and related back to the actual system.

The three levels of digital simulation are:

1. System level;
2. Register Transfer level;
3. Gate or logic level

2.1.1 SYSTEM LEVEL

System level simulation of digital system consists of the moment of unsophisticated-level general purpose simulation languages.

2.1.2 REGISTER-TRANSFER LEVEL

At the register transfer level microprograms are obtained using R.T.L. flowgraphs.

2.1.3 GATE OR LOGIC LEVEL

At the gate/(logic level) the actual logic gates or modules and their interconnections are functionally modelled in the computer. Each signal line is restricted to binary

values , and time is usually quantized to gate propagation delays.

Register-transfer level simulation is generally useful in evaluating and optimizing the architectural design of a digital system, rather than in uncovering races, hazards, illegal states and other critical timing considerations.

2.2 STEPS USED IN THE D.T.L. SIMULATION

The following steps are used in the simulation of a digital system:

(1) Establish a realistic internal process that reflects the implementation this D.T.L. description will achieve,

(2) Establish a convenient and consistent method of the input and output,

(3) provide facilities to control (initialize,start and stop) and monitor the model,

(4) Produce an efficient simulation structure.

The major use of simulation in digital system design is to verify and check design, both at system level and logic-gate level prior to actual manufacture. New computer design, algorithm and concepts can be tested before hardware implementation.

Table 2.2 gives some Arithmetic equivalents for R.T.L. productions.

Table 2.2 Arithmetic Model Equivalent for R.T.L. Statements

S.No.	R.T.L.PROGN.	FORTRAN equivalent	Remarks
1.	A←B	A = B	
2.	$A_2 \leftarrow B_3$	A(2)=B(3)	A and B are arrays
3.	A ← 0	A = 0	
4.	A ← B∘C	A = B∘C	
5.	A ← A∘1	A = A∘1	
6.	A ← $\bar{A}$	A = -A	
7.	A ← Π(B)	A = Π(B)	
8.	Π(B) ← A	Π(B)=A	Π (B) = contents of memory location Π(B) in an array
9.	T ← 6	GO TO 6	
10.	IF(A≥1)B ← 0	IF(A.GE.1) B=0	

2.3 SIMULATION OF DIGITAL SYSTEM

The R.T.L. flow chart of the given logical system can be translated into digital computer simulation programs using two different approaches, viz.,

(1) Arithmetic Model,

(2) Logical Model

2.3.1 Arithmetic Model Approach for Digital Simulation

This approach uses only FORTRAN IV arithmetic statements to represent R.T.L. production in the R.T.L. flow chart of a given logic system. The data are represented in the decimal digits in INTEGER MODE and not by TRUE-FALSE MODE (binary number representation). A R.T.L. production A← B is represented by an equivalent arithmetic statement A=B in the corresponding arithmetic model. A conditional operation R.T.L. production like IF(A<1) B←0 ELSE B← 1 is realized by the following arithmetic statements

```
IF(A.LT.1.0) B = 0.0
B = 1.0
```

Arithmetic Model

It is to be noted that there is no arithmetic model equivalent statements for logical, shift, Rotate and scale, R.T.L. productions.

Arithmetic model achieves conclusions. It sacrifices many hardware details to achieve this conclusions. If one is

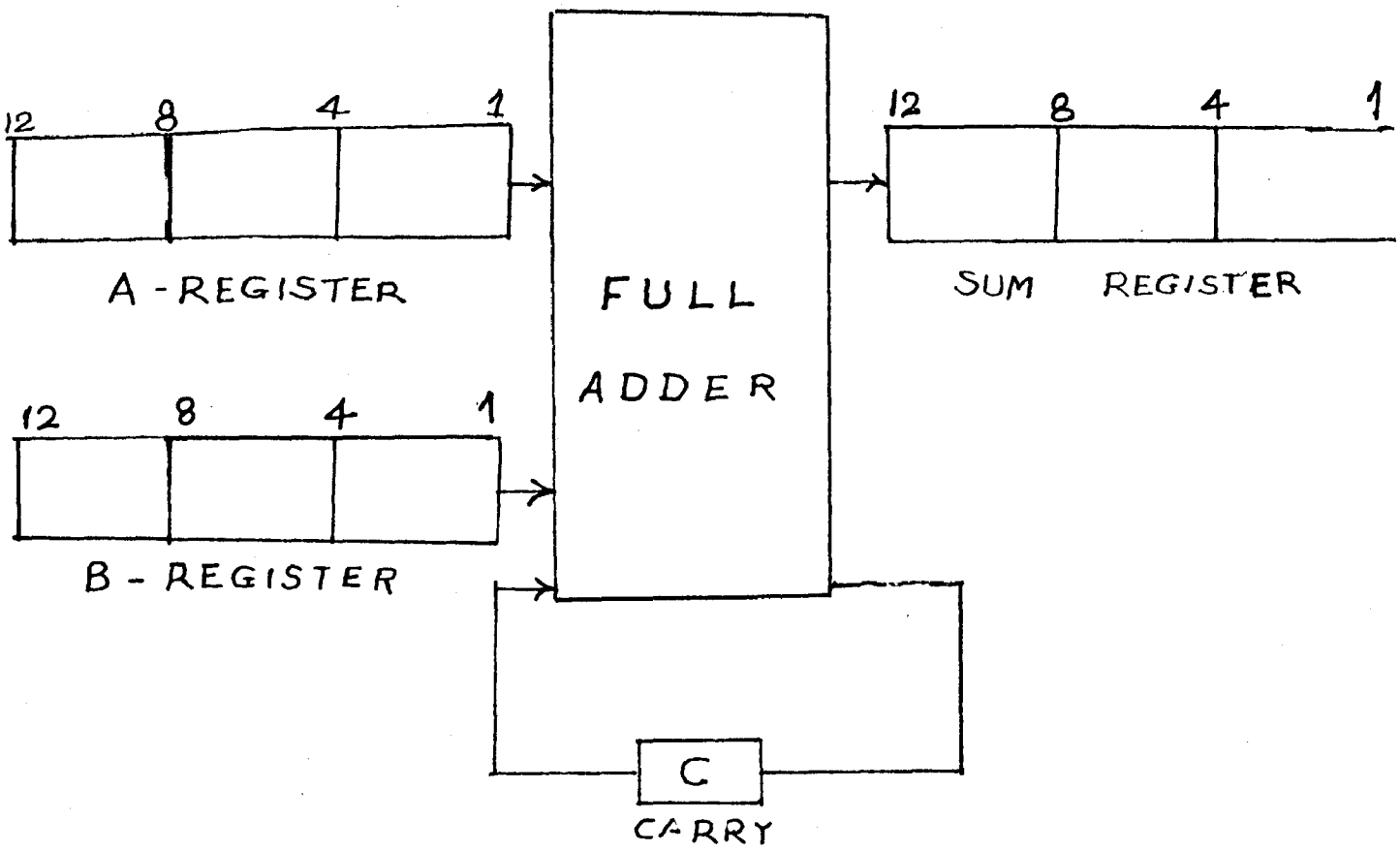


FIG. 2.1. BLOCKDIAGRAM FOR A 12-BIT FULL ADDER

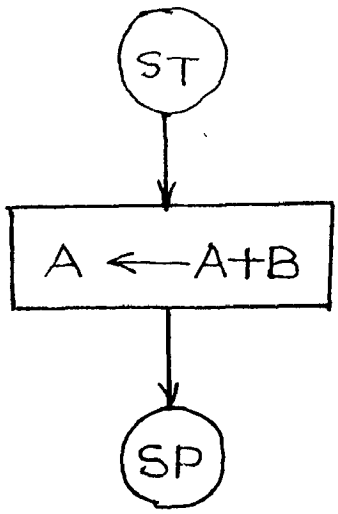


Fig. 2.2. RTL FLOW CHART

```
C   C ARITHMATIC MØDEL SIMULATION
          ØF  12  BIT  ADDER
          READ  (5, 10)  A, B.
          A  =  A + B
          WRITE  (6, 10)  A, B
          STØP
10        FØRMAT  (2 I4)
          END
```

FIG 2.3 ARITHMATIC MODEL PROGRAM

willing to sacrifice bit-by-bit details in order to simulate a complicated logic system, then the programmer can simulate the logic system by relatively short-programs compared to logic model approach. The memory requirement is very much less as well as computer time utilized for running an arithmetic model simulation program. Arithmetic model operates accessarily on decimal numbers. In contrast many computers operate on octal system or hexadecimal system Thus when computer architecture program (refer Chapter 4) are written to check machine language program, the above properties of arithmetic model are to be correctly interpreted. Thus when properly used, the arithmetic model can produce very helpful and concise simulation programs.

As an example of arithmetic model simulation consider a full adder (serial N=12 bit, full adder). A concise R.T.L. flow chart is written in Fig.2.2. The arithmetic model program will be as shown in figure 2.3.

It is to be noted that the arithmetic model statement A=A+B is necessarily decimal in nature. Therefore A and B should be decimal equivalents of bits in the register A and B (taking care of sign bit).

2.3.2 Logic Model Approach for Digital Simulation

This approach uses FORTRAN IV logical statements to represent R.T.L. productions in the R.T.L. flow chart of a given logic system. Binary number representation is used for the data representation. An R.T.L. production D← Ā

is represented by an equivalent logic statement (subroutine) CALL CMP(A,B,N) in the corresponding logic model. The D.T.L. production shows that A and B are N-bit registers. All the N-bits of register A are complemented (inverted) corresponding to the original value and the result is stored in register B.

It is to be noted that bit-by-bit operation is performed by the corresponding logical model. The arithmetic equivalent statement is given by B ← -A.

A conditional operation D.T.L. production D.T.L. production like IF(A=1) B ← A ELSE B←0 is realized by the following two logic statements if A and B are single bit variables.

IF(A)B = .TRUE.

B = .FALSE.

If A and B are N bit registers and C is a single bit register, then D.T.L. production IF(A=B)C←1 will be simulated by a corresponding subroutine call statement

CALL [illegible] (A,B,C,N)

Table 2.2 shows the correspondence between few statements in D.T.L. and logic model. The F.T.L. productions for shift, rotate are also discussed in this.

The logic simulation approach is not concise. Bit-by-bit detailed operations are performed. It requires large memory and large computer running time for a given logic

```
DECLARATION :        SL REG , 12
                     REG , A, B, 5
                     FLPLF , C
                     EL, SINGLE BIT FULL ADDER
```

FIG·2·4. DECLARATION STATEMENT FOR 12 BIT FULL ADDER

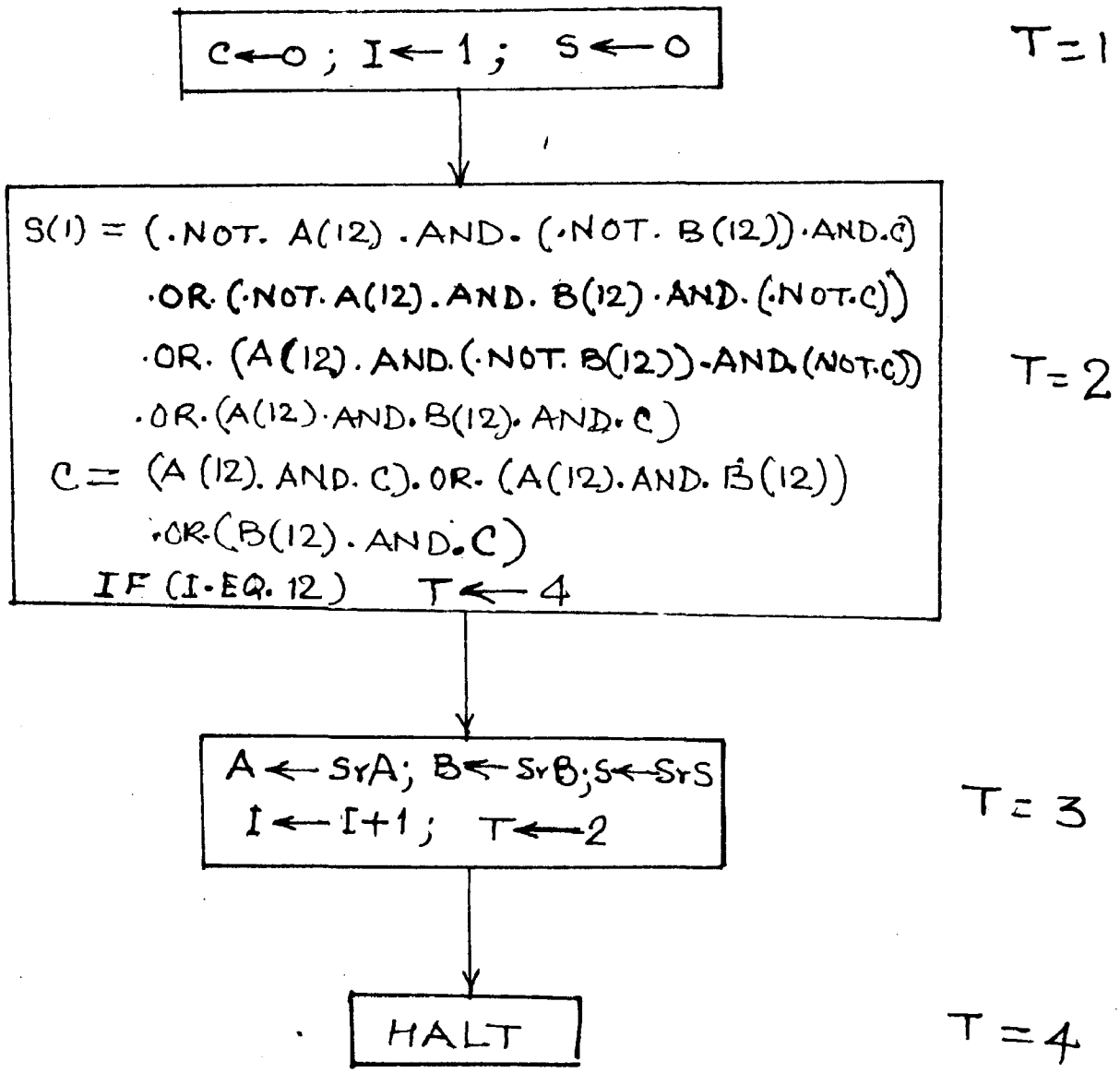


FIG 2·5. R·T·L· FLOW CHART FOR FULL ADDER

system. Logical model necessarily operates on binary number (TRUE-FALSE MODE) in contrast with any form of input.

As an example of logical model simulation, consider a full adder (N=12 bit) as shown in figure 2.1 in the arithmetic model simulation approach. The declaration and R.T.L. flow chart are shown in figure 2.4 and figure 2.5 respectively.

For the above R.T.L. flow graph a logical model simulation program is developed as shown in figure 2.6.

```
C   C  LOGICAL MODEL SIMULATION OF SERIAL ADDER
       LOGICAL * A(12), B(12), S(12),C,P,G,D
  10   READ(5,20, END=200)A,B
       WRITE (6,25)A,B
C ---- INITIALIZATION UPTO STATEMENT NUMBER: 30
  (1)  C=.FALSE.
       I=1
  30   CALL FALSE (S,12)
C ---- SIMULATION OF SERIAL BIT FULL ADDER
C      UPTO ST.NO.40
  (2)  P = .NOT.A(12)
       G = .NOT.B(12)
       D = .NOT.C
  S(I) = (P.AND.G.AND.C).OR.(P.AND.B(12).AND.D)
 1     .OR.(A(12).AND.G.AND.D).OR.(A(12).AND.
 2     B(12).AND.C)
 40    C=(A(12).AND.C).OR.(A(12).AND.B(12)).OR.
   1     (B(12).AND.C)
C ---- NOW SUM DECISION AS %. ALL BITS ARE
   $   ARE ADDED IN ST.NO.30
  90   IF(I.E.12) GO TO 4
```

```
      3 ------  CALL SR (A,A,12)
                CALL SR(B,B,12)
                CALL SR(C,C,12)
                I=I+1
                GO TO 2
      (4)------ WRITE(6,60)C,C
                GO TO 10
    200 ------  STOP
     20         FORMAT(12L2/12L2)
     25         FORMAT(20X,'A=', 12L2/20X,'B=' ,12L2)
     60         FORMAT(20X, 'S=' ,12L2/20X,'C= ',L2)
                END
                SUBROUTINE VALUE (X,N)
                LOGICAL*1X(N)
                DO 5 I=1,N
        5------ X(I)=.FALSE.
                RETURN
                END

                SUBROUTINE SR(A,B,N)
                LOGICAL*1A(N),B(N)
                N=N-1
                DO 5 I=1,N

        5------ B(N-I+1)=A(N-I)
                B(1) = .FALSE.
                RETURN
                END
```

Figure 2.6 LOGICAL MODEL SIMULATION PROGRAM

The following facts should be observed for the program written in figure 2.6.

(1) The simulation of a 12 bit, 2's complement serial adder employs BOOLEAN LOGIC statements as given in Fig.2.6

(2) It is to be noted from the program that it has one-to-one correspondence with the R.T.L. flow chart shown in figure 2.5.

(3) The simulation program is devided into four sections as shown in figure 2.6, corresponding to four different states of state controller in R.T.L.flow chart.

(4) The encircled number in the simulation program correspondo to the states in the R.T.L. flow graph. The initialisation during state 1 in R.T.L. flow chart (Fig) are translated into three successive statements starting from statement No.1 in Fig.2.6.

(5) The single bit full adder simulation (State Fig of RTL) is found in the successive statements starting from (2) upto statement number 4, where the Boolean expressions for the sum, S, and Carry, C outputs are realised. The logical decision is made in statement number 50, during state T = 2.

(6) All the R.T.L. productions during state Fig are translated into logical model simulation statement starting from statement No.3 upto and excluding statement No.4.

(7) The adder outputs are printed in the last/fourth section of the program, whenever RTL flow chart enters HALT state.

(8) It is clear from the simulation program that bit-by-bit operation is performed by the logical system. And it also indicates the type of logical system, for example, here this shows that it a serial adder.

2.4 THE SIMULATING DEVELOPMENT

It is advantageous to split a large computer program into number of small subprograms, each subprogram performing a well defined subtask of the total task. This has many advantages.

Each subprogram can be separately programmed, tested and then interconnected. A subroutine may be called by a main program several times and intern subprogram may call other subroutines.

It should be observed that between the caller and the called program two kinds of transfers, control transfer and information transfer should take place.

2.4.1 Examples: Considering the above conditions the subroutines for different number of operations have been developed.

For example: The R.T.L. production

IF (A=C) B ← 1 ELSE B ← 0 is very popular for many digital system design. The simulation of this R.T.L. production can be realized either by arithmetic model or by logical model. The arithmetic model for the R.T.L. production is realized by the following two arithmetic equivalent statements, where A, C and B are decimal integer numbers.

```
IF(A.EQ.C)D=1.0
D = 0.0
```

In the logical model realization the NEXT production test the algorithm for DUAL Routine.Steps used in DUAL operation are given below:

(1) Get the Boolean equation from the following algorithm

Input		Output
A	C	B
0	0	1
0	1	0
1	0	0
1	1	1

Boolean equation

$B = AC + \overline{A}\overline{C}$ *** (1)

$\overline{B} = (AC + \overline{A}\overline{C})$ *** (2)

(2) Find the complement of A and C registers according to equation no (1)

(3) Find the 'AND' ed value of A with C and $\overline{A}$ with $\overline{C}$

(4) Find the 'OR' ed value according to equation (1).

(5) TEST for zero by complementing the equation (1).

IF(.NOT.D) A = C ELSE A = C

the subroutine for TEST EQUAL is given below.

```
      SUBROUTINE TEST EQUAL (A,C,D,N)
C     A IS N-BIT LONG, C IS N BIT LONG REGISTER
C     D IS SINGLE BIT REGISTER
C     N = NO. OF BITS
C ****IF A=C, SINGLE BIT=1
      COMMON D(500),E(500)
      LOGICAL*1 A(N),C(N),D,B
      CALL COM1 (A,D,N)
      CALL COM1 (C,B,N)
```

```
      CALL AND (D,E,E,N)
      CALL AND (A,C,D,N)
      CALL OR (D,E,D,N)
      CALL COM(D,D,N)
      CALL TSTZR(D,D,N)
      RETURN
      END
```

In the digital system conversion from one radix to other

ry popular operation. For example conversion from Decimal to
al is the primary operation in a digital computer. The

ithm for conversionhas been discussed below.

et D = Decimal number, L = Logical number

ivide D by 2. Let DIL be the quotient and DI the remainder.

eplace L by DI attached to the left of L

f DIL = 0 go to halt state, Else continue
et D = DIL

o to step 2.

is binary equivalent of D

top

he subroutine for this is as follows:

```
      SUBROUTINE DTOL (D,L,N)
      D=DECIMAL, L=LOGICAL-T,F,
      N=BITS/L, BIT 1 = SIGN
      INTEGER D,DIL,DI
      LOGICAL *1 L(N)
      CHECK FOR NEGATIVE
      J = 0
      IF(D.GE.0) GO TO 10
      D=-D
      J=1
```

```
C ------     CONVERSION PROPER
      20    CALL FALSE (L,N)
            DO 20 I=1,N
            II = N-I+1
            DIV=D/2
            DI = D-2*DIV
            D = +DIV
            IF(DI.EQ.0) GO TO 20
            L(II) = .TRUE.
            IF (DIV.EQ.0) GO TO 40
      20    CONTINUE
      40    IF(J.NE.1) GO TO 25
            CALL ZERO (L,N)
      25    RETURN
            END
```

2.4.2 The R.T.L production MB←M(MA), is very popular in computer, which states that the memory buffer register,MB, is loaded with the contents of memory location whose address is given by MA.

The mar arithmetic statement is MB=M(MA), if suppose M(MA)=110 then MB=110 whatever its previous value is immaterial. For performing this operation the LOAD(M,MA,MB,L,N) routine is developed, stating that the contents of memory location whose address is given by K-bit register MA, is loaded into N-bit register MB. The contents of location MA remains unchanged. L is thought per memory.

Steps used in developing the routine are:

(1) Conversion of location from logical to decimal is performed.

(2) The contents of that memory location from bit 1 to N is transferred to register B.

The subroutine for the above R.T.L. production is given below:

```
          SUBROUTINE LOAD (N,B,A,L,M)
C MEM     MEM, M,LOCATION B IS LOADED IN REGISTER A
C-------  L = WORDS/MEM.M = BITS/WORD
          LOGICAL*1 A(M), B(N),M(L,M)
          CALL LOCD (B,J,N)
          DO10 I = 1,M
   10     M(J,I) = A(I)
          RETURN
          END
```

2.6.4 The R.T.L. statement $(A \leftarrow \bar{A})$ is very common operation in any digital system. The contents of accumulator A is complemented. For this the subroutine COMPLEMENT is developed.

SUBROUTINE COMPLEMENT (A,B,N) has a meaning that A is an N-bit value register, and its complement is also N-bit long and is stored in the register B. The subroutine for this is given below:

```
          SUBROUTINE COMP (A,B,N)
C MEM     B<A', N = BITS,A IS UNCHANGED
          LOGICAL* 1A(N),B(N)
          DO 10 I= 1,N
10        B(I) =.NOT.A(I)
          RETURN
          END
```

2.4.9 In digital computer memory is filled from some input device such as from cards, from magnetic tape etc. which stores data. The F.F.L. production for this is FILL MEMORY M(L,N) from cards. The input may be in binary or octal or hexadecimal code. The algorithm for FILL MEMORY is discussed below

Step 1= READ all the datas from data cards

Step=2 Convert the number/data into binary code if it is not in binary

Step=3 Fill the locations from binary numbers one by one.

The sub-routine is given below.

```
                SUBROUTINE FILLM(M,L,N)
C ***           FILL FIRST L LOCATIONS IN MEMORY,M,
      1         FROM DATA CARDS
C ------        SPECIAL NUMBER/CARD, FORMAT I10
C ------        M=BITS/WORD
                IMPLICIT INTEGER(Ø)
                LOGICAL*1 L1,M(L,N)
                DIMENSION Ø(50)
                READ(5,10)(Ø(I),I=1,L)
                DØ20 J=1,L
                CALL ØZZL(Ø(J),L1,N)
                DØ20 I=1,N
20              M(J,I) = L1(I)
                RETURN
10              FORMAT (9I10)
                END
```

2.4.6 If the data input is in the octal mode, then for the FALLIBILITY a conversion from octal to binary is performed. This conversion is very popular whenever input is in octal mode. The D.S.L. statement for this is OCTAL/T-F conversion.

The algorithm used is illustrated step-by step as follows.

Step-1 Let ∅=Octal number, D=Null (meaning of null is b is an empty register)
D = number of bits/F
BIT 1 is sign bit is to be checked.

Step 2- Divide octal number by 10, let ∅1 be the remainder. ∅1 is the integer octal number.

Step 3. Replace b by ∅11,∅12,∅13 as an octal number is equivalent to 3 bit group. The power of each binary bit is 2 to 0 from left to right

∅1 = ∅11.∅12.∅13
∅11 = (D-1-I, = 2^2
∅12 = (D-I) = 2^1
∅13 = (D-1-I) = 2^0

Step 4- If ∅1=0, go to step 6, else continue
Step 5- ∅=∅/10
Step 6- GO TO Step 2.
Step 7- 'b' is the binary equivalent of octal number.
Step 8- END

Following the above steps the subroutine is developed named as OCTAL TO LOGICAL CONVERSION viz. OTOL(∅ ∅,L,D).
where ∅∅ represents the octal number, L is logical number

```
      CALL SCRT (L,N)
11    RETURN
      END
```

2.4.7 The REL statement A ← 0 is very common statement and is frequently used in the almost all digital system for SUBROUTINE FALSE(A,N) makes reset of the system all the values of register A, i.e. from 1 to N as zero. For example if register A has value TFTFTFTF, then after this call subroutein A will have value as FFFFFFFF.

(a) Before operation

(b) After CALL FALSE SUBROUTINE

Figure 2.3 CALL FALSE (A,N)

The REL statement A ← 0 gives the above operation as shown in diagram 2.3. And its Fortran equivalent logical statement is CALL FALSE (A,N)

SUBROUTINE

```
      SUBROUTINE FALSE(A,N)
C ***  A=TFF....F,N=DIM
      LOGICAL nl A(N)
      DO 20 I=1,N
10    A(I)=.FALSE.
      RETURN
      END
```

2.6.6 If the data input is in the octal mode, then for the MILLIMICRON a conversion from octal to binary is performed. This conversion is very popular whenever input is in octal mode. The F.F.L. statement for this is OCTAL/T-F conversion.

The algorithm used is illustrated step-by step as follows.

Step-1 Let ϕ=Octal number, D=Null (meaning of null is b is an empty register)
N = number of bits/w
BIT 1 is sign bit is to be checked.

Step 2= Divide octal number by 10, let ϕ1 be the remainder. ϕ1 is the integer octal number.

Step 3. Replace b by ϕ11,ϕ12,ϕ13 as an octal number is equivalent to 3 bit group. The power of each binary bit is 2 to 0 from left to right

ϕ1 = ϕ11.ϕ12.ϕ13
ϕ11 = (N-1-1, = 2^2
ϕ12 = (N-1) = 2^1
ϕ13 = (N-1-1-1) = 2^0

Step 4= If ϕ1=0, go to step 8, else continue
Step 5= ϕ=ϕ/10
Step 6= GO TO step 2.
Step 7= 'b' is the binary equivalent of octal number.
Step 8= STOP

Following the above steps the subroutine is developed named as OCTAL 2D LOGICAL CONVERSION viz. OTOL(ϕ ϕ,L,N).
where ϕϕ represents the octal number, L is logical number

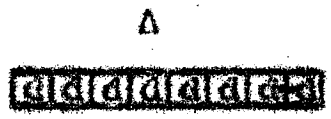
is sign bit.

broutine for this is given as follows:-

```
THIS (OTOL) OCTAL TO LOGICAL CONVERSION
      SUBROUTINE OTOL(ØØ,L,N)
C     ØØ=OCTAL; L=LOGICAL T,F;
      N= BITS/L; BIT 1 = SIGN
      INTEGER Ø,ØØ1, ØØ
      LOGICAL*1 L(N)
      Ø = ØØ
      CHECK FOR NEGATIVE
      J=0
      IF(Ø.GE.0) GO TO 10
      Ø = -Ø
      J=1
      CONVERSION PROPER
10    CALL FALSE(L,N)
      K= N-2
      DO 8 I = 1,K,3
2     ØØ1=Ø-(Ø/10)*10
      Ø=Ø/10
1     IF(ØØ1.EQ.0)GO TO 8
      GO TO (3,4,3,4,3,4,3),ØØ1
3     L(N+1-I)= .TRUE.
4     GO TO(6,5,5,6,6,5,5),ØØ1
5     L(N-I)=.TRUE.
6     GO TO (8,8,8,7,7,7,7),ØØ1
7     L(N-1-I)=.TRUE.
8     CONTINUE
      IF(J.NE.1)GO TO 11
```

```
            CALL SCOM (L,N)
   11       RETURN
            END
```

2.4.7 The RTL statement A← 0 is very common statement and is frequently used in the almost all digital system for SUBROUTINE FALSE(A,N) makes reset of the system all the values of register A, i.e. from 1 to N as zero. For example if register A has value TTTTTTTT, then after this call subroutain A will have value as FFFFFFFF.



(a) Before operation

(b) After CALL FALSE SUBROUTINE

Figure 2.3 CALL FALSE (A,N)

The RTL state ent A← 0 gives the above operation as shown in diagram 2.3. And its Fortran equivalent logical statement is CALL FALSE (A,N)

SUBROUTINE

```
            SUBROUTINE FALSE(A,N)
C ===       A=TTT....T,N=8INS
            LOGICAL nl A(N)
            DO 20 I=1,N
10          A(I)=.FALSE.
            RETURN
            END
```

2.4.8 For example in a shift register counter the value of all counter is shifted circular left/right depending on the use of counter. The Rotate right program gives a circular rotation of number. For example register A is 4-bit value as shown in figure 2.4. The result is stored in register B.

The B.T.L. statement in B-NET and the fortran equivalent logical model call statement is

CALL RR (A,B,N)

SUBROUTINE ROTATE RIGHT

```
      SUBROUTINE RR(A,N)
C     REGISTER A IS ROTATED RIGHT BY
C     ONE BIT; N = NUMBER OF BITS
      LOGICAL*1 A(N),B(N),C
      M=N-1
      DO 20 I=1,M
      CALL SR(A,B,I)
      B(1)=C
      CALL L (B,A,N)
   20 CONTINUE
      RETURN
      END
```

Similarly for other BTL statements subroutines are developed and tested as given in Appendix-A. Table 2.2 shows a set of BTL statements and their FORTRAN counter parts.

Table 2.2 Logic Model Equivalents and Arithmetic
Model equivalent for Typical B.S.I. statements

SL	FORTRAN ARITHMETIC EQUIVALENT	FORTRAN LOGIC EQUIVALENT	REMARKS
A←X+Y	B=X+Y	CALL ADD(X,Y,C,A)	1's complement addition
A←B∧C	does not exist	CALL AND (B,C,A,N)	
A←$\overline{A}$	B=-A	CALL COM(A,B,N)	1's complement
A ← B	A=B	CALL EQ(B,A,N)	
J=I ← L=I	does not exist	CALL EQUAL(B,L, N,A,J,K)	For one bit transfer: J=I, L=I, (J,K,L and I are integer constant terms).
A←0	A=0	CALL FALSE(A,N)	
TILL MEMORY array (M(L,N)) from cards)	does not exist	CALL FILLM(M,L,N)	L locations filled (L in decimal)
A←M(B)	A=M(B)	CALL LOAD(M,B,A, L,N)	M(B) = contents of memory location B. M(B) is an array.
M(B)←A	M(B)=A	CALL LOADM(A,M,B, L,N)	
T/Octal conversion	does not exist	CALL LOCT(L,B,N)	Logical To Octal
A=T/Decimal Conversion	does not exist	CALL LODE(L,B,N)	Logical To Decimal
A=BVC	does not exist	CALL OR (B,C,A,N)	
Octal/T=F Conversion	does not exist	CALL OTOL(O,L,N)	Octal To Logical
Print Memory array,N)	does not exist	CALL PRINT (M,L,N)	
B ←SLA	does not exist	CALL SL(A,B,N)	Shift left by one-bit

D←Cm	does not exist	CALL CM(A,D,N)	
S←X+Y	S=X+Y	CALL ADD(X,Y,0,C,N)	2'S complement Carry out
A←-A+1	A= -A+1	CALL TCOM(A,N)	2'S complement.
IF(A=B)C←1	IF(A.EQ.B)C=1	CALL TEST EQ(A,B, C,N)	C contains one bit.
IF(A=0)B←1	IF(A.EQ.0)B=1	CALL TESTZR(A,B,N)	B contains one bit
IF(A>B)D←0	IF(A(3).EQ.0) B(4)=0	IF(NOT.A(3)) B(4)=FALSE	
IF(A>=1)	IF(A(3).EQ.1) GO TO 9	IF(A(3)) GO TO 9	
A1←(B.VC2)∧D4	does not exist	A(2)=(.NOT.B(2).OR.(C(2)) .AND.B(4)	
D←(A∧B)VC	does not exist	CALL AND (A,B,D,N) CALL OR (D,C,D,N)	

The advantage of using REL statement flow chart is that one can simulate complicated system by means of these sub-routines. The benefits of using digital simulation techniques are savings in coding time and cost caused by eliminating fundamental design errors. Other advantages are:

(a) Validity of operational specifications can be checked.

(b) During the early design stage corrections and modifications can be made

(c) Alternative procedures and designs may be evaluated.

(d) It serves as design documentation.

CHAPTER -3

BASIC COMPUTER ARCHITECTURE
(Hypothetical Mini Computer)

3.1 INTRODUCTION

In early days of computer age first hard-ware was designed and fabricated. Then software was developed. A digital computer is an electronic programmable calculating machine which works on a binary principle. Electronic circuits used to represent combinations of binary digits, each of which can be in one of the two states, on or off, high or low, open or closed, etc. Logically these two states are called as TRUE OR FALSE. These two states are represented by electrical voltages. The use of binary system in an electronic machine lies in the elimination of erroneous interpretation while transporting information from one point of the machine to another.

Digital computers are sub-categorised according to cost of peripheral devices and software used. These are given below:

(i) the large, general purpose, data processing systems,
(ii) the smaller, cheaper general purpose systems,
(iii) the special purpose systems.

3.1.1 The large Machine- These are not 'on line' machines. They are generally used to run multiple jobs and requires a controlled environment to run in, such as an air conditioned

rocs. These systems fall into two groups. One is data processing and the other is scientific processing.

3.1.2 The Smaller Machine

These are general purpose machine used for the more specific operations. For example micro computers, minicomputers and midicomputers. They are more rugged, requiring no special environmental conditions. They are used in a wider range of applications, many of which involves 'real-time' or 'online' operation such as a control of a steel mill.

3.1.3 Special Purpose Systems

The special purpose machines is designed for a single type of application and frequently has part of its program provided. For example: The navigation computers.

All digital computers are capable of executing a limited number of defined operations by giving it an instruction. Each instruction is simply a binary word, a particular combination of zeros and ones is decoded by the control unit in the CPU. The operation of digital computer is well defined by considering basic computer organization.

3.2 SCHEMATIC DIAGRAM OF A DIGITAL COMPUTER

Figure 3.1 shows the basic block diagram of a digital computer. The architecture of the classical computer is centered around four basic building blocks. These are:

1. The arithmetic/logic unit (ALU).
2. The control unit.

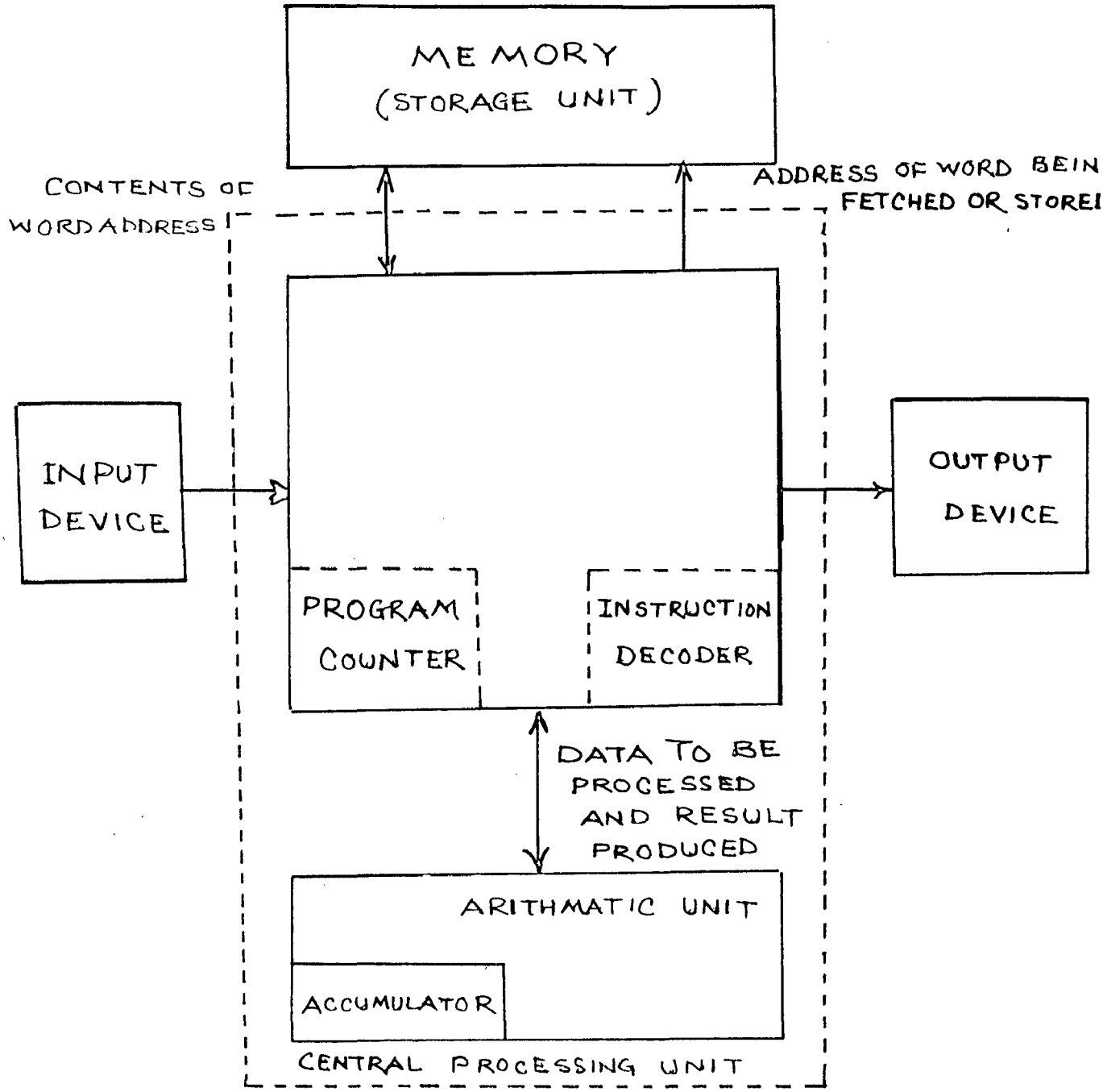


FIG. 3.1. BASIC COMPUTER ORGANIZATION
(BLOCK DIAGRAM)

3. The memory unit.

4. The input/output unit.

The input device converts the information received from the input machine such as punch cards, magnetic tape into electrical signals and sends it to the control unit within the control processing unit. From there the information is sent to memory. After the program and data have been transferred to the memory, the control unit begins allowing one program at a time, in sequence to be executed. The data is processed in the arithmatic unit and is stored in the storage unit. The memory address register, MAR will give the address of this located data. Accumulator always contains one of the data. The output unit reads the r e lt from the memory under the supervision of control u.it.

Suppose for example computer is ordered to perform a particular operation by giving an add instruction. It will perform the operations in the following sequence:

(i) the number at the input terminal is transferred to CPU-accumulator memory,

(ii) this number from accumulator is transferred to a particular memory location,

(iii) the second number at the input terminals is transferred to theCPU accumulator.

(iv) the number in memory location is added to the to the number presently in the accumulator and the result is lost in the accumulator.

(v) The number in the accumulator is outputted to the output unit. The computer cycles stops.

This explains a computer organization.

2.3 ORGANIZATION OF LINC COMPUTER

For the detail-architecture of a digital computer, and machine-language description LINC (the Classic LINC Laboratory Instrument Computer) computer is considered for example. (3)

The LINC is a 12-bit minicomputer. All the information is processes and stored in its memory will be represented by 12-bit numbers. It is to be noted that the leftmost bit is the sign bit and the other 11-bits forms the magnitude of the number in the 1's complement system.

A schematic-layout of a LINC computer is given in figure 2.2. An instruction is fetched from the memory into the control unit where it is decoded, the control unit then causes the action defined by the instruction to be executed. When the instruction has been executed, the next instruction is fetched from memory, and so on.

All arithmetical and logical operations are performed by the decoded instruction, by the arithmetic and logic unit. The communication between the computer and the outside world takes place via the input/output system.

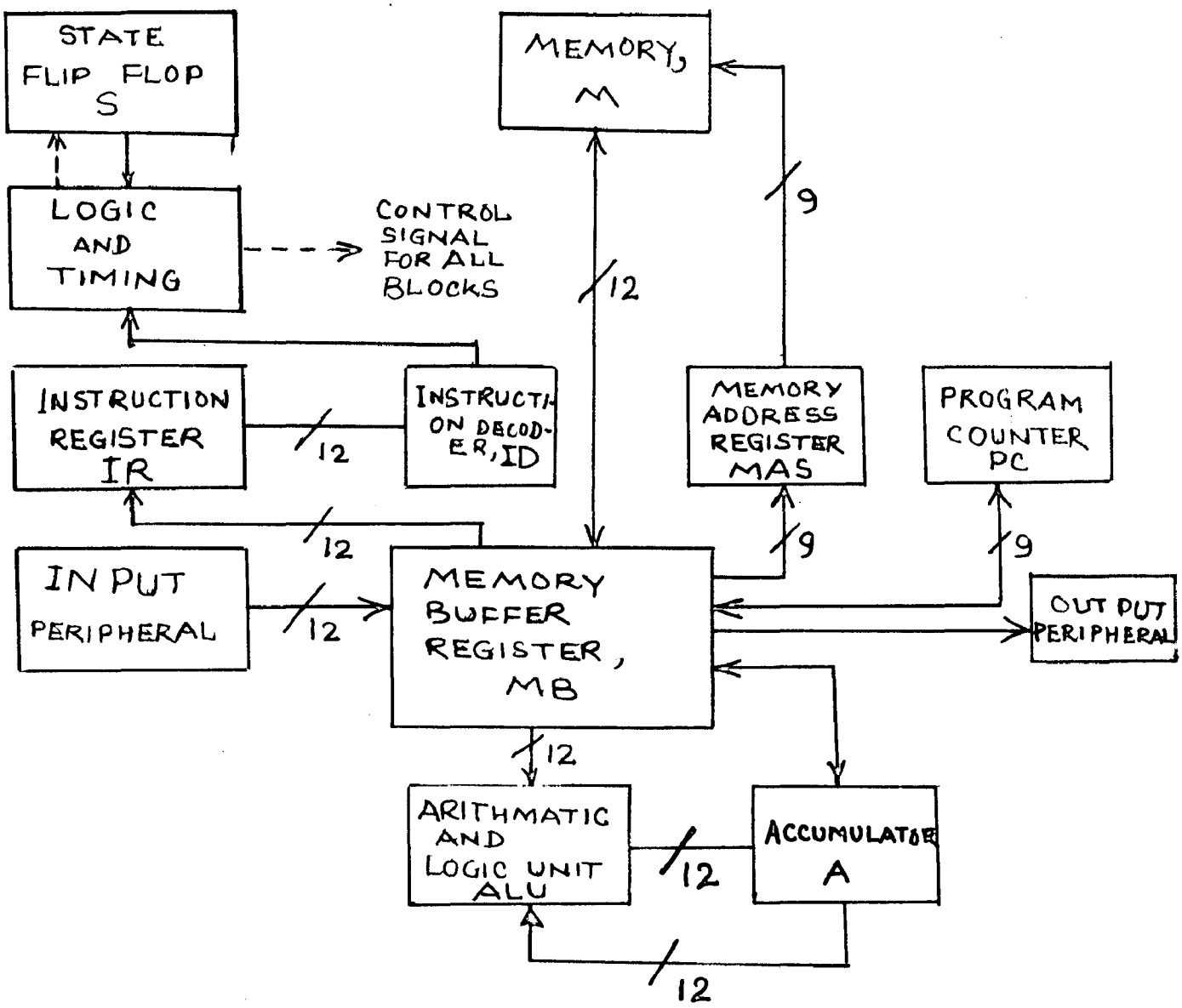


FIG 3.2 ORGANIZATION OF LINC COMPUTER (BLOCK DIAGRAM)

With this basic overall picture in consideration, each of these blocks has been discussed in the following steps:

(1) Memory Unit - The memory unit is the main storage area of the computer. The important features of memory are as follows:

(a) Information is stored by the word or byte. Any operation on individual bit is performed by passing the whole word to the ALU.

(b) Each location in the memory can be accessed in the same time as any other, each location is said to have a unique address.

(c) Memory cycle time is the time taken by the digital computer to complete a memory access.

(d) The size of the memory is restricted by the range of addresses as determined by the machine instruction set.

(e) Different types of memory can be mixed together in some machines, but this is uncommon.

(f) Some memories are volatile and some are non-volatile in nature.

If memory is seen from the CPU side it looks like a block box with 2 input ports and two output ports.

(2) Memory Buffer Register- It is register serves as an interface between the memory and other units. If the information

is to be written into memory, the CPU must simply transfer the appropriate word into the MDR. It is then automatically copied to the appropriate memory location.

(3) Memory Address Register (MAR)- The specific word of information to be accessed, either read or write, is located by the contents of the MAR i.e. it gives the location of data received or outputted in the memory.

(4) Instruction Register- The instruction register is a part of instruction decoder and hence is a part of control unit. The instruction register receives those bits of the current instruction which convey operator information i.e. content of MDR goes to the instruction register, IR.

(5) Instruction Decoder(ID)- The instruction decoder and controller is a part of control unit. It decodes the contents of the instruction register.

(6) Program Counter- The program counter in combination with Instruction decoder and controller forms a control unit. It is also a word length register. It indicates during the execution of one instruction, the address at which the next instruction to be executed is stored.

(7) Control Unit- It is the heart of central processing unit. It comprises of two parts, the instruction decoder and controller and the Program Counter.

The following gives step-by-step operation of control unit:

(i) contents of PC goes to MAR, and initiate the memory read cycle, thereby loading the MBR.

(ii) Restore the memory location just read if magnetic core is being used.

(iii) Increment the PC

(iv) Contents of MBR is transferred to BR

(v) Decode the instruction

(vi) Execute the instruction

(vii) Repeat the same from (i) etc.

Step (i) and (ii) gives read/restore memory cycle, this implies that processor clock is controlling the memory cycle time. This timing sequence is shown in figure 3.3.

(D) Arithmetic and logic Unit - It is that part of computer where all logical and arithmetic operations are performed. It is constructed exclusively of high speed electronic components. Any operation is executed by a particular set of command signals which are generated by ID at the appropriate time as determined by the specific instruction being executed. Some of the typical operations are executed as follows:

(i) Add,

(ii) Subtract

(iii) Complement and negate

(iv) Increment,

(v) Decrement,

(vi) Shift.

(9) Accumulator- The ALU itself is not a store. Accumulator is a prominent part of ALU. It always holds one operand for each arithmetic or logic operation and the result is always stored in it.

(10) Logic and Timing Circuit- This unit gives the control signals for the start of operation between the various registers at the proper times.

(11) Status- In order to keep track of the code in which the computer is operating the control unit employs a one-bit register called the state flip-flop.

(12) Input and Output Unit- Peripherals either transmit data to the computer (input) or receive data from the computer (output) along the 12-bit wires.

3.4 INSTRUCTION SET FOR LEC COMPUTER

Machine language instructions directly control the various functions of a digital computer. The instruction set can be divided into a number of types of functions for example:

(1) Transfer- move data between memory locations and the CPU registers.
(2) Perform arithmetical, logical and shift type operations.
(3) Branch and control.
(4) Transfer data to and from the peripherals, example I/O.

(9) Various control operations, example halt, not or clear a register, etc.

These instructions are usually put into one of three classes:

(a) Memory Reference Instructions - which involve operation on data stored in memory.

(b) Non-Memory Reference Instructions: which either operate on data stored inside the CPU or require no data.

(c) Input/output instructions.

The basic format for memory and non-memory reference instruction are given in Figure 3.4.

For example: The memory reference instruction ADDXXX states that the content of the accumulator be added to the contents of location XXX and leave the result in the accumulator. Using 1's complement addition. The instructions commonly used for LAIC are listed below in table 3.1

Table 3.1 A Brief List of Pseudo-LAIC INSTRUCTIONS

Mnemonic	Machine Code No. (Octal)	Meaning
ADD XXX	2XXX	ADD the contents of the accumulator to the contents of location XXX and leave the result in the accumulator using 1's complement addition.

ANZ	0A3X	Skip the next instruction if the accumulator is positive i.e. sign bit equals zero
CLA	0011	Clear the accumulator
COM	0017	Complement the contents of the accumulator.
HLT	0000	The machine is ordered to Halt.
JMP XXX	6XXX	JUMP to location XXX to obtain the next instruction
MUL XXX	5XXX	MULtiply the contents of the acc. by the contents of location XXX and leave the least significant 11 bits in the accumulator.
NOP	0016 or YYY	No Operation. This merely gives delay before the next instruction is executed. YYY is unassigned octal
ROR	0500n	Rotate Right
STC	4XXX	Store the contents of the accumulator into location XXX and then Clear the accumulator

---

Observe the following facts concerning the table:

(1) The underlined uppercase letters in the last column give the key to the mnemonics.

(2) Certain instructions, such as ADD,SKC, have both an operation part and an address part.

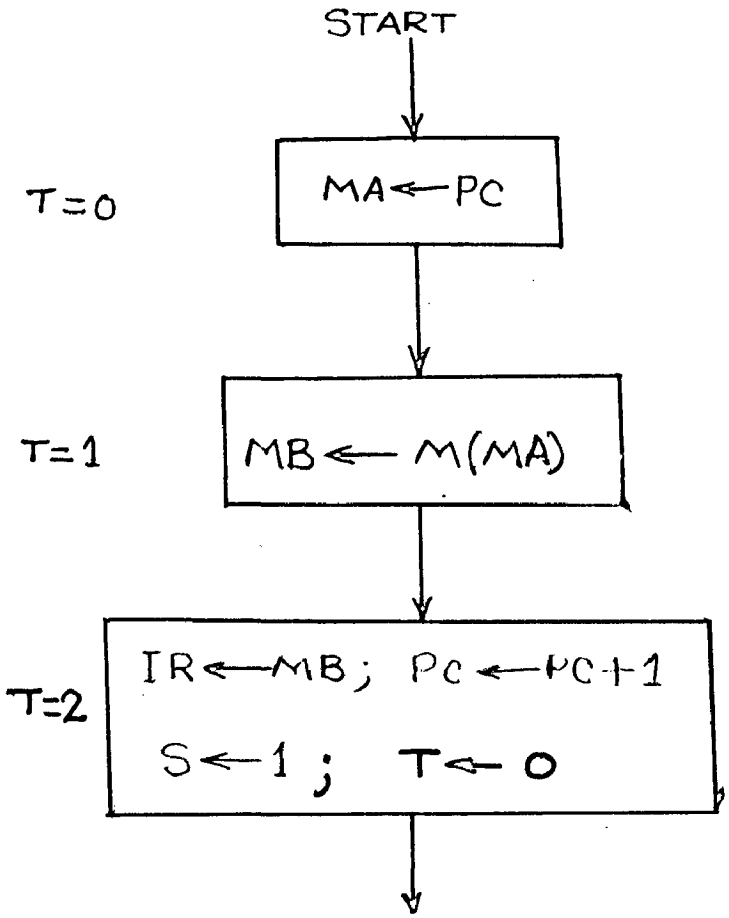


FIG. 3.5 RTL DIAGRAM FOR FETCH CYCLE

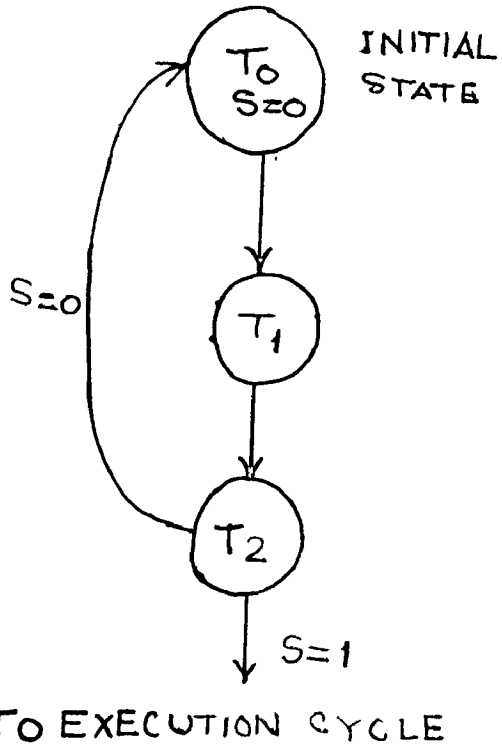


FIG 3.6 STATE DIAGRAM FOR FIG. 3.5

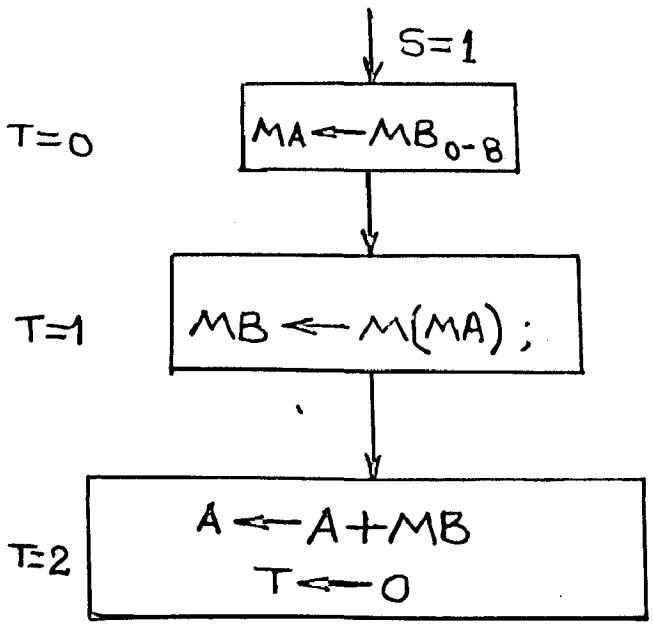


FIG. 3.7 RTL DIAGRAM FOR ADD INSTRUCTION

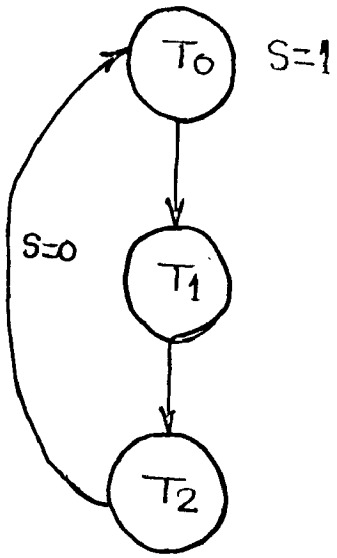


FIG. 3.8 STATE DIAGRAM FOR FIG. 3.7

(3) Other instructions, such as CLA, HLT, have only an operation part.

(4) The JMP and ADD are branch instructions.

(5) Print instruction is particular to particular computer.

(6) For the one or more negative operands, the algebric sign for the result for arithmetic operations is placed in the leftmost bit.

3.3 RTL Description for the Instruction

The operation of an instruction takes place in two parts. These are

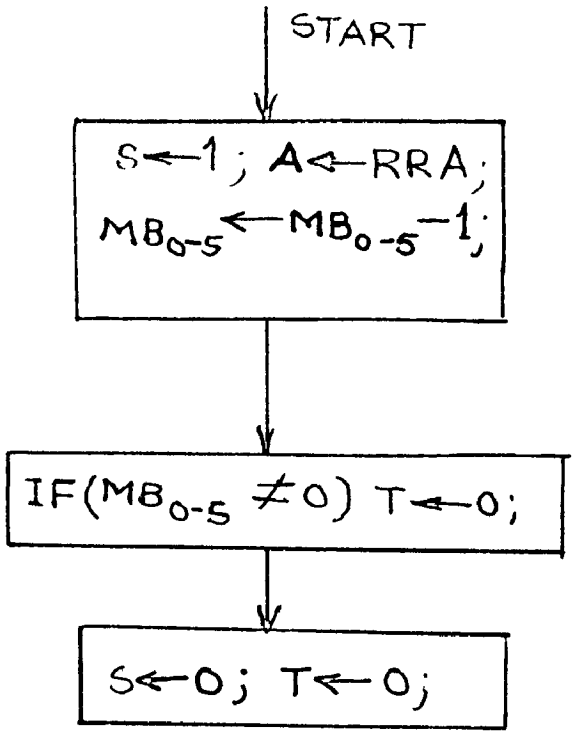
(a) Fetch cycle or Instruction Cycle.

(b) Execution cycle.

3.3.1 Fetch Cycle

A fetch cycle is the primary cycle for each instruction. This cycle obtains the instruction from the memory.

How fetch cycle operates, can be made clear by RTL diagram and state diagram as it is in Figure 3.5, 3.6 respectively T_0 through T_3 are the states in which operation is taking place At T_0 state the Program Counter fetches the address of the instruction to the memory address register Then at T_1 state the contents of the memory whose address is given by MA is transferred to the memory buffer register, MB. At T_2 state the instruction part of MB is transferred to instruction register, and this is immediately decoded, PC is incremented, the state register is set to execution.



FIG·3·9 RTL DIAGRAM FOR ROTATE RIGHT INSTRUCTION

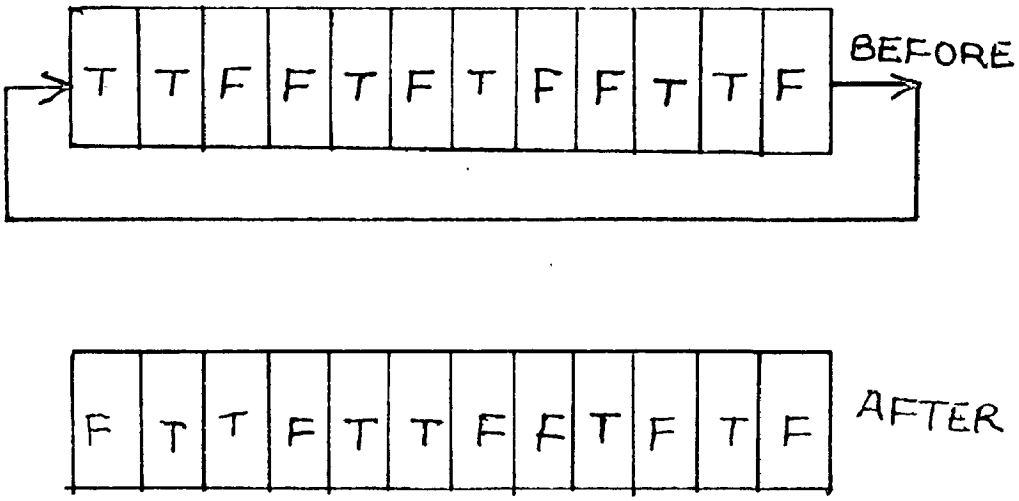


FIG. 3·10 ROR OPERATION

3.3.2 The Execution Cycle. The execution of the instruction takes place in this cycle. The execution cycle is different for different instruction. The RTL representation for the execution cycle of several instructions are given below.

3.3.3 RTL diagram for the ADD Instruction

From state T_0 to T_3 the execution operation takes place. At T_0 state the address section of the memory buffer is transferred to the memory address register. At T_3 the contents of the memory location whose address is given by MA is transferred to the buffer register, MB, at last at T_2 the contents of accumulator is added to the contents of the memory buffer register and is stored in accumulator, and the process is repeated for the new instruction cycle starting with the fetch cycle. The state diagram is similar to the fetch cycle.

3.3.4 RTL Program for the ROR

By rotate-right means the least significant bit of the accumulator becomes the most significant bit in one clock pulse and the L.S.B. position is occupied by the bit next to the L.S.B. This operation repeated till all the bits less by one bit has rotated right. For example.

Accumulator is 12 bit word length

$$A \leftarrow ror_4\ A$$

meaning that accumulator is rotated right by 4 bits and the

result is stored in the accumulator.

Similarly the RTL program for the other instructions can be written. The following table gives the RTL program for the common instructions.

Table 3.2 RTL Program for the execution of common instruction

States	ADD	CMA	JMP	STC	ROR
T=0	$IR \leftarrow M[PC]$	$A \leftarrow \bar{A}$;	$PC \leftarrow IR_{0-8}$	$IR \leftarrow IR_{0-8}$	$A \leftarrow shr A$; $IR_{0-9} \leftarrow IR_{0-9}$
T=1	$MD \leftarrow M[IR]$	———	———	$IR \leftarrow A$; $A \leftarrow 0$	$IF(IR_{0-9} \neq 0)$ $T \leftarrow 0$
T=2	$A \leftarrow A + MD$ $T \leftarrow 0$	$S \leftarrow 0$;	$S \leftarrow 0$; $T \leftarrow 0$;	$M[IR] \leftarrow MD$; $S \leftarrow 0$; $T \leftarrow 0$;	$S \leftarrow 0$; $T \leftarrow 0$;

3.6 HARDWARE REALIZATION OF R.T.L. FLOW CHART FOR ADD INSTRUCTION

It is to be noted that for a given R.T.L. flow chart and R.T.L. program a logical diagram can be drawn as shown in figure 4.11. This logic diagram shows how logical operations are performed in sequence by an instruction.

For example: Consider ADD instruction. The R.T.L. program is given in table 3.2 and its R.T.L. flow chart is drawn in figure 3.7 by using R.T.L. productions in group according to the requirement of the logic system. Its logical diagram or hardware implementation is shown below.

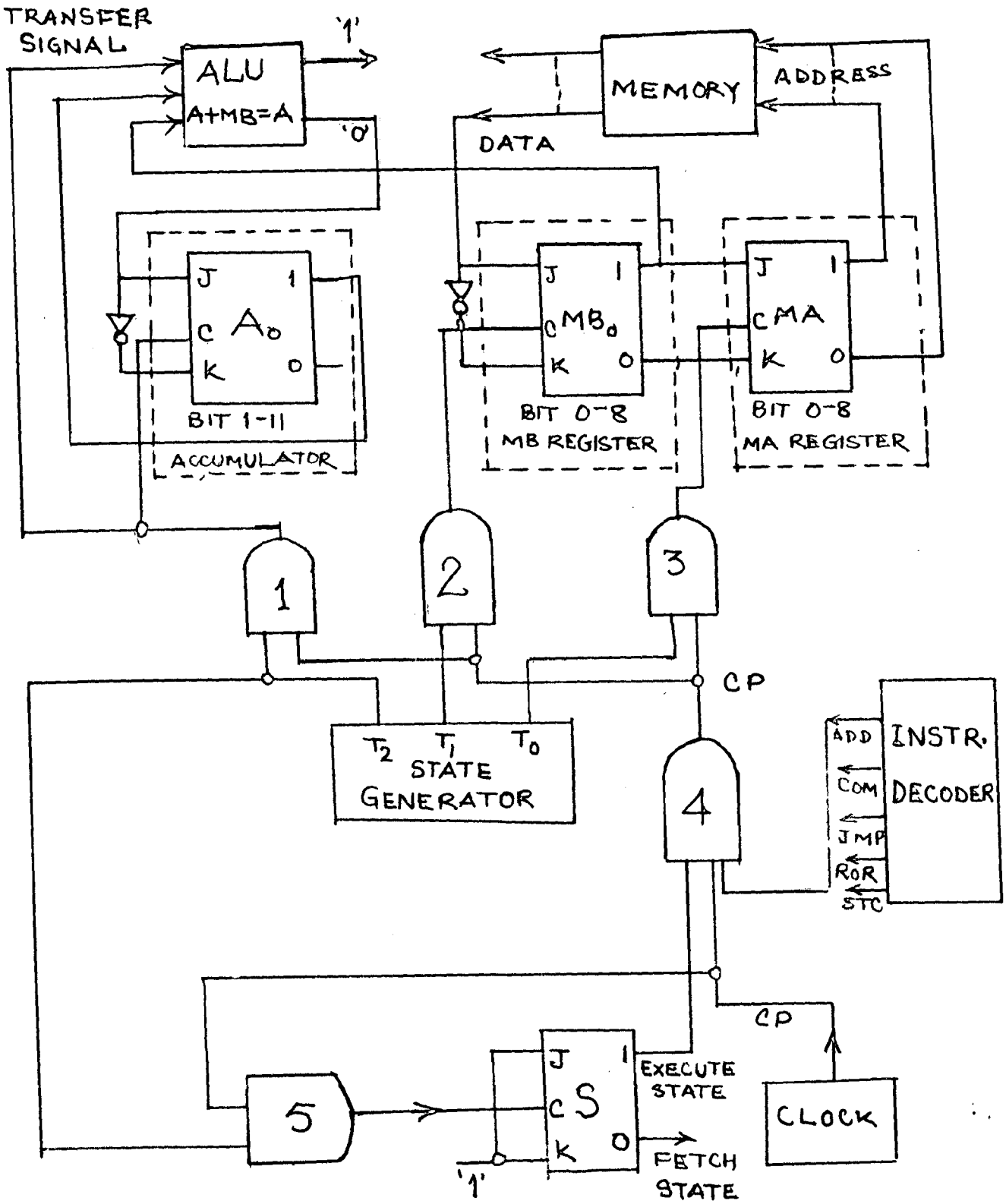


FIG. 3·11. LOGIC DIAGRAM FOR THE ADD INSTRUCTION EXECUTION

It is to be noted that for the ADD instruction execution, an 'OR' gate is used for the 'OR' operation performance. Only this gate inputs are moved in the accumulator itself.

Similarly the logic diagram for the other instructions can be drawn.

It is to be noted from the above discussion that the LINC computer can be simulated following any of the two simulation techniques as discussed in Chapter (2) corresponding to the requirements. If the bit-by-bit informations are of not importance then use a very simple and concise approach named Arithmetic model approach and if Detail operation of a logical system is to be seen then logical model approach is used.

3.7 SIMULATION OF LINC COMPUTER

The following section is devoted for the simulation of LINC computer using the two model approaches.

3.7.1 Arithmetic Model Approach for the Simulation of LINC Computer

The arithmetic model is developed from the above R.T.L. flowgraphs and R.T.L. program. By using the FORTRAN arithmetic equivalent for the above R.T.L. production group. This model is very helpful and useful for the simulation of a fairly high level, example, the shift operation does not even appear explicitly.

The simulation program for the arithmetic model approach is given below:

```
C[illegible]    SIMPLE COMPUTER SIMULATION
                DIMENSION M(60)
                INTEGER PC,A
C[illegible]    MEMORY CLEAR, INSTRUCT READ IN,PRINT
    2           DO 5 I=1,60
    5           M(I)=0
                WRITE(6,80)
                READ(5,72,END=70)PC,N,NL,NH
                READ(5,79)(M(I),I=20,N)
                WRITE(6,73) PC,N,NL,NH
                WRITE(6,74)
                WRITE(6,79)(M(I),I=NL,NH)
C[illegible]    FETCH CYCLE
    9           MA=PC
                MD=M(MA)
                PC=PC+1
C[illegible]    DECODER PART OF FETCH
                IF(MD.LT.2000.) GO TO 110
                IF(MD.LT.4000.) GO TO 120
                IF(MD.LT.6000.) GO TO 130
                IF(MD .LT.8000.)GO TO 140
  110           IF(MD.EQ.0.) GO TO 150
                IF(MD.EQ.17.) GO TO 112
                IF(MD.EQ.17.) GO TO 114
                IF(MD.EQ.0514) GO TO 115
                PRINT 100
                GO TO 2
```

```
C####      ADD INSTRUCTION
 120       MA=MB-2000.
           MB=M(MA)
           A=A+MB
           WRITE(6,77)PC,MA,A
           GO TO 5
C####      STC INSTRUCTION
 130       MA=MB-4000.
           MB=A
           A =0
           M(MA)=MB
           WRITE(6,77)PC,MA,A
           GO TO 5
C####      JMP INSTRUCTION
 140       PC=MB-6000.
           WRITE(6,77)PC,MA,A
           GO TO 5
C####      CLR INSTRUCTION
 112       A=0
           WRITE(6,77)PC,MA,A
           GO TO 5
C####      COM INSTR.
 114       A=-A
           WRITE(6,77)PC,MA,A
           GO TO 5
C####      APO INSTRUCTION
 115       IF(A.LT.0.) GO TO 116
```

```
            PC=PC+1
  126       WRITE(6,77)PC,MA,A
            GO TO 5
C***        HALT INSTRUCTION
  150       WRITE(6,77)PC,MA,A
            WRITE(6,82)
            WRITE(6,79)(M(S),I=1,MS)
            GO TO 2
   70       STOP
   72       FORMAT(6I3)
   73       FORMAT(5X, 'PC=',I3,'M= ',I3,
     $      'HL= ', I3, 'ML=  ',I3)
   74       FORMAT('0',5X, 'INITIAL MEMORY CONTENTS')
   75       FORMAT(10X,9I10)
   77       FORMAT(5X,'PC= ',I3,5X,'MA = ',I3,
     $      5X,'A= ',I4)
   79       FORMAT(9I10)
   80       FORMAT('0')
   81       FORMAT ('0', 5X, 'FINAL MEMORY CONTENTS')
  100       FORMAT(0, 10X,'PROBLEM IS NOT POSSIBLE TO SOLVE')

            END
```

(1) The model operates on decimal numbers.

(2) Only Arithmetic Fortran statements are used.

(3) In the decoding section, after decoding all the known instructions GO TO 2 STATEMENT CAUSES ANY UNDECODED instruction to cause the current simulation to terminate and a new program to be read in.

(4) The computer memory is represented as a 60 element array, M(1) through M(60).

(5) Do-operation causes the current simulation to continue and a new program to be read in.

(6) This is a very concise program and take less time in comparison with logical model.

The above program is tested on IBM 360 and result of the program is given in Appendix B.

3.7.2 LOGICAL MODEL APPROACH FOR THE SIMULATION OF COMPUTER

The logical model simulation technique is very helpful for describing the bit-by-bit operation of a complicated logical system; for such as digital computer. It has been discussed in section 2.3 that how a logical model is developed from its corresponding R.T.L. flow graphs. The same technique is also used here for the realization of logical model of LMC computer.

The logical model for the LMC computer is given below:

```
C     LOGICAL MODEL SIMULATION FOR SIMPLE
     $ COMPUTER
       LOGICAL*1 A(12),PC(9),IR(12),IRC(12),MA(9)
     0 MB(12),M(60,12),D,P(9),N(6),MD(12),B(12)
C     MEMORY READ IN, AND PRINT
  2 ------ READ(5,72,END=70)L,M,MA
```

```
      READ(5,1)PC(9)
      CALL FALSE(I,L,N)
      WRITE(6,100)
  100 FORMAT(20X, 'INTEGRAL')
      CALL FALSE(K,L,N)
      CALL FALSE(P,N1)
      P(N)=.TRUE.
      CALL FALSE(I,G)
      R(G)=.TRUE.
C     FETCH CYCLE
    9 CALL ADD(PC,MA,N1)
      CALL LOAD(I,MA,MB,L,N)
      CALL EQ(MB,IR,N)
      CALL TADD(PC,P,PC,D,N1)
C     DECODING OPERATION OF FETCH
      CALL COM(IR,IRC,N)
      IF(IRC(1).AND.IRC(2).AND.IRC(3)) GO TO 10
      IF(IRC(1).AND.IR(2).AND.IRC(3)) GO TO 20
      IF(IR(1).AND.IRC(2).AND.IRC(3)) GO TO 30
      IF(IR(1).AND.IRC(2).AND.IR(3)) GO TO 40
      IF(IR(1).AND.IR(2).AND.IRC(3)) GO TO 50.
C ------ INSTRUCTION FETCHED CANNOT BE DECODED.GO TO NEX PROGRAM
      PRINT 200
      GO TO 2
   10------IF(IRC(4).AND.IRC(5).AND.IRC(6)) GO TO 60
      IF(IRC(4).AND.IR(5).AND.IR(6)) GO TO 61
      IF(IR(4) AND IRC(5) AND IRC(6)) GO TO 62
```

```
C           INSTRUCTION WHICH CANNOT BE DECODED
            PRINT 200
            GO TO 2
   60       IF(IRC(7).AND.IRC(8).AND.IRC(9)) GO TO 63
            IF(IRC(7).AND.IRC(8).AND.IR(9)) GO TO 66
C           PRINT 200
            GO TO 2
   66       IF(IRC(10).AND.IRC(11).AND.IR(12))GO TO 67
            IF(IR(10).AND.IR(11).AND.IRC(12)) GO TO 68
            IF(IR(10).AND.IR(11).AND.IR(12)) GO TO 69
C           PROGRAM CODE IS NOT POSSIBLE TO SOLVE
            PRINT 200
            GO TO 2.
C           EXECUTION CYCLE FOR THE DECODED INSTRUCTIONS
C ---------- ADD INSTRUCTION EXECUTION
   20 ------ CALL EQUAL (MD,4,12,MA,1,9)
            CALL LOAD(1,MA,MB,1,1)
            CALL ADD (A,MB,A,D,1)
            PRINT 870,PC,MA,A
            GO TO 5
C ---------- STC INSTRUCTION
   50 ------ CALL EQUAL (MD,4,12,MA,1,9)
            CALL EQ(A,MB,1)
            CALL FALSE(A,1)
            CALL LOAD(MD,1,MA,1,9)
            PRINT 870,PC,MA,A
            GO TO 5
```

```
C ————————  JMP INSTRUCTION
     50     CALL EQUAL(MB,4,12,PC,1,9)
            PRINT 870,PC,MA,A
            GO TO 5
C ————————  NOR INSTRUCTION
     61     CALL NR(A,A,N)
            CALL COM(R,R,6)
            CALL ADD(MB,R,MB,6)
            CALL EQ(MB,MB,6)
          0 IF (MB(1).AND.MB(2).AND.MB(3).AND.MB(4).AND.MB(5).AND.
            MB(6)) GO TO 61
            GO TO 5
            PRINT 870,PC,MA,A
C ————————  APO INSTRUCTION
     62     IF(A(1)) GO TO 100
            CALL TADD(PC,P,PC,D,N)
    100     PRINT 870,PC,MA,A
            GO TO 5
C ————————  HLT INSTRUCTION
     65     PRINT 870,PC,MA,A
            PRINT 871
            CALL PRIMER(N,L,N)
            GO TO 2
C ————————  CLA INSTRUCTION
     67     CALL FALSE(A,A,N)
            PRINT 870,PC,MA,A
            GO TO 5
C ————————  NOP INSTRUCTION
     68     PRINT 870,PC,MA,A
            GO TO 5
```

```
C ------        CON INSTRUCTION
      69        CALL CON(A,A,I)
                PRINT 67,,PC,MA,A
                GO TO 5
      70        STOP
       1        FORMAT (9L1) X
      72        FORMAT (3I3)
     670        FORMAT(5X, 'PC=--',3L2,3X,'MA=  ',
          $           3L2,3X,'A=  ',I2L2)
     200        FORMAT(1OH, 'INSTRUCTION CAN NOT BE DECODED'.)
     671        FORMAT ('0',55X, 'FINAL')
                END
```

The following facts should be seen concerning the program:

1. The logical model is divided into five sections:
   (a) Fill memory, (b) FETCH cycle, (c) Decoding cycle, (d) Execute cycles, and (e) Output print and format statements.

(a) In the fill memory section, memory is filled from the cards containing Machine language data in OCTAL numbers.

(b) The second section gives the fetch cycle of computer, how instructions are fetched into memory.

(c) The third section describe the bit-by-bit decoding of of the fetched instruction in sequence.

(d) The fourth part describes the execution of the decoded instructions. Here bit-by-bit operation of the instruction is described.

(e) The fifth and the last part of the program gives the output print and format statements.

2. The memory is represented by a matrix M(L,N) with L as the number of locations in decimal and N as the number of bits, L and N decimal numbers.

3. The logical rotate operation is also described here.

4. The contents of program counter from bit 1 to bit 9 is transferred to the memory addressed register in statement number 5.

5. The first three bits of memory address register represents the OP code of the instruction.

CHAPTER-4

ASSOCIATIVE MEMORY

4.1 INTRODUCTION

The memory systems that responds to request based on contents are known as content addressed memories (CAM) or associative memories. The associative memories are more complex systems, involving more logic with each memory cell and hence are used only in special purpose computers.

The CAD mode of memory operation differs from the location-addressable mode of operation in the following ways:

(i) the address is not needed for storing and retrieving information.

(ii) During the retrieval process, information is simultaneously scanned from all specified memory locations and processed to determine if it is associated with the desired information.

(iii) All associated information in the specified section can be retrieved , i.e., it is a multivalue output process.

4.2 MODES OF OPERATION

The CAM has three modes of operations

4.2.1 Read-Write mode, 4.2.1 Read mode
4.2.3 Search mode (or Match mode)

4.2.1 READ-WRITE MODE

In the read-write mode a word can be written into a location, overwriting any previously stored data or can be read out from a selected location. The read action may be destructive or non-destructive. Since this is the common memory type it is referred to simply as RAM rather than read-write-RAM.

4.2.2 READ-ONLY MODE

In read only mode a RAM (random access memory) can be made with data loaded into it during its fabrication. The data stored in any location can be read but it cannot be altered. This type of memory is called ROM.

4.2.3 SEARCH MODE

In the search mode the contents of each word of memory are compared with a given data word. If a match is detected, then a flag bit one of which is associated with each word is set. This technique is also useful for matching a part of a word to set the indicator flag as shown in the figure 4.1. Only the bit in the search register which correspond to ones in the mask register will be compared with the memory contents. All memory locations which have 0101 in the left hand 4-bits will have the associated search result flag set irrespective of the contents of rest of bits. The word select flag is cleared when the word is excluded from the search.

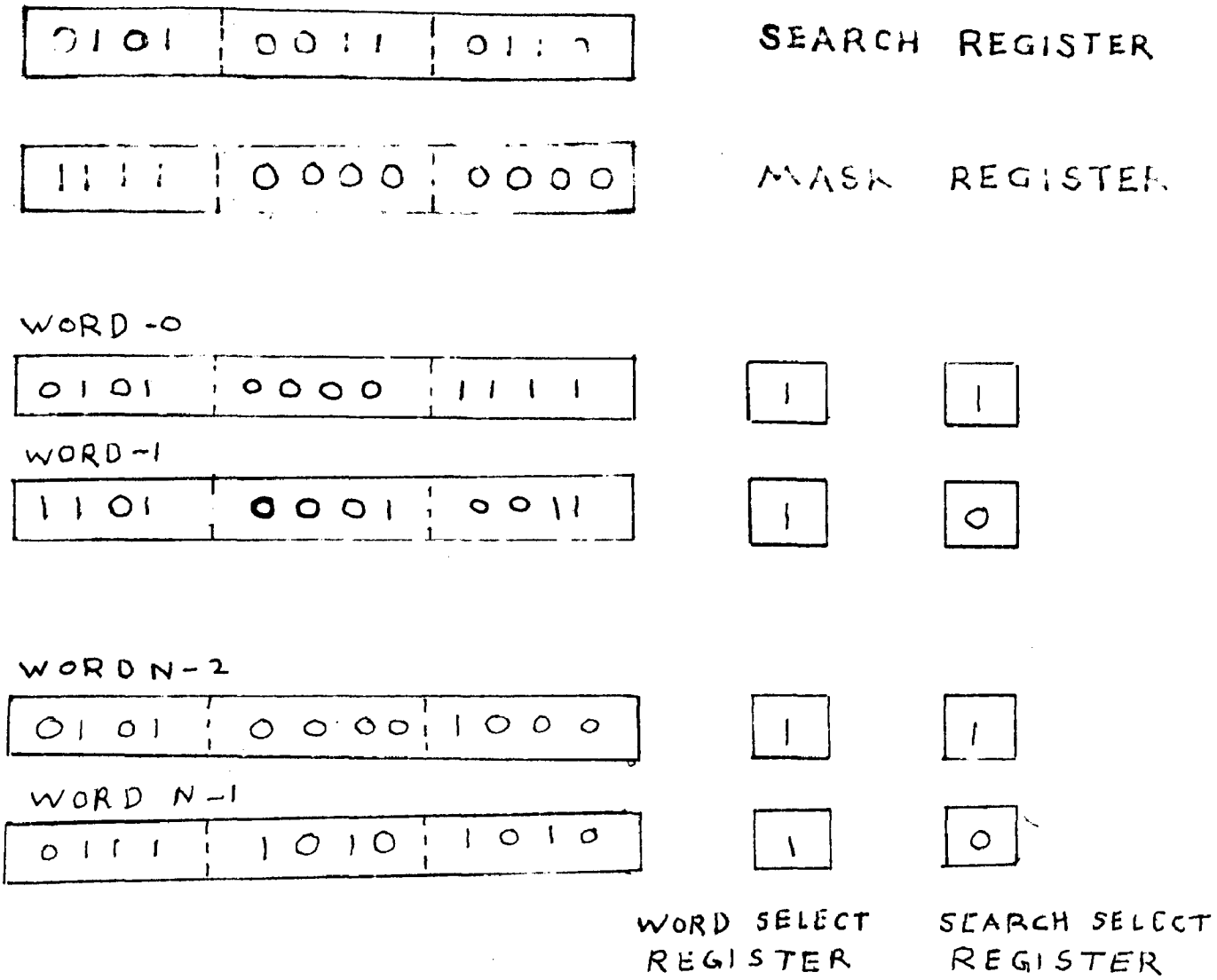


FIG 4.1 Content Addressable Memories

the mask and search registers are n-bit long and the word select and search select registers are of N-bit length.

From the above discussion the simplest algorithm for an ideal CAM operating in the content addressable mode will have the following necessary steps:

(i) Search for match.

(ii) retrieve or store the desired information,

(iii) alter the priority structure and manage the memory.

The algorithm is illustrated by the following example of LSIC computer associative memory.

6.3 EXAMPLE

Consider a problem for searching a given memory location in LSIC computer pm memory.

The memory in LSIC computer has L locations, word length being N-bits-12 bits represented by M(1), M(2), M(3)——M(L). It is desired that the contents of a given memory location X be compared with all the other memory locations and if an identical number is found in location Y, then the contents of the last location of memory, viz. M(L) should be made equal to Y in binary. If no match is found, then the last location M(L) should be made zero.

The associate memory organization for the problem has L locations and each location has 12 bits, the given memory location 'X' is stored in a register called 'descriptor', which characterises the desired information. The memory

trico then to locate this information in its memory storage. It responds either with its desired information or atleast with its address. Using the above defined algorithm the R.T.L. flow chart for the ASM computer associative memory can be written as illustrated in figure 4.2.

4.3.1 The R.T.L. Flow Chart for the Associative Memory

T = 1 : MA ← MD

T = 2 : MB ← M(MA)

T = 3 : C ← MD, MA ← 1

T = 4 : IF(MA=MB) M ← 0, MA ← MD, T ← 7

T = 5 : IF(MA≠MD) MA ← MA + 1, T ← 4 ELSE
MB ← M(MA)

T = 6 : IF(MB≠C) MD ← MA, MA ← MD ELSE
MA ← MA + 1, T ← 4

T = 7 : M(MA) ← MB

T = 0 : MA≤P

Figure 4.2 R.T.L. flow chart

It has been noted from the above R.T.L. flow graph that all the memory locations (1,2,...,L) are converted to binary number. Since memory is represented by a matrix M(L,N) where L is the number of locations in decimal and N number of bits/word in decimal. The given location K, where K is in decimal has to be changed into equivalent binary number.

(decimal) L $\xrightarrow{\text{ASM}}$ LD(binary)

(decimal) K $\xrightarrow{\text{ASM}}$ KD(binary)

The contents of location MD having address MA is stored in register C in state three and the MA is initialized for the further operation.

The states 4,5 and 6 form a loop for the checking of memory location at a time unit either a match is found or end of memory is reached i.e. at T=4, the memory addressed location is checked for the last location number and at the fifth state the memory address (MA) is compared with the given location MD, if it is NOT the contents of buffer register is compared with the contents of the descriptor register C, and for the NOT value the location MA is made equal to MD and this memory buffer value is loaded in the last memory location and the operation is halted.

Assuming this to be a correct program, when tested on digital computer it was not giving satisfactory results. It was noted that it always stopped at one particular value of memory location. After debugging it is found that the R.T.L. production (MA ← MA+1) is missing. On making this correction, when the program was tested it gives the required result. If without running this program hardware realization would never give the correct results and it was not possible at all to check where the error is committed and a lot of time would be wasted. Hence before realizing the hardware, it is a good practice to simulate the associative memory system by any of the following techniques:

(1) Arithmetic model simulation technique,
and (2) Logical model simulation technique.

4.32 Arithmetic Model Simulation for Associative Memory

If the detail operations of system are of not much concern, then bit-by-bit operation is sacrificed and the associated memory system can be simulated by a relatively short/ concise program. The arithmetic model is employed for the simulation of associative memory system. Each R.T.L. production is represented by its corresponding arithmetic model equivalent as given in table 2.1 of Chapter 2.

The arithmetic model for simulation program is listed below:

ARITHMATIC MODEL

1 2 3 4 5 6 7 8 9 10

```
C     C     SIMULATION OF ASSOCIATIVE MEMORY CIRCUIT-A.M.
            DIMENSION M(600)
            READ(5,10) L,X
C ------    L = TOTAL NO.OF LOCATION, MEMORY ADDRESS IN DECIMAL
            READ(5,20)(M(I),I=1,L)
C ------    ALL MEMORY LOCATIONS ARE FILLED WITH DECIMAL
          $ NUMBER DATA FROM CARDS.
            PRINT 70, (M(I),I=1,L)
C ------    ALL MEMORY LOCATIONS ARE PRINTED FOR THE CHECKING
            OF MEMORY DATA
C ------    REL SENSE-LATCH CLEARED FROM HERE
            LI=L+1
   (1)      M=X
   (2)      MD=M(MA)
C ------    CONTENTS OF LOCATION X IS STORED IN REGISTER C
   (3)      C = MD
            MA = 1
```

```
C ————————  CHECK FOR THE LAST MEMORY LOCATION
      (4)   IF (MA.E .LE)GO TO 50
      (5)   IF(MA.EQ.K) GO TO 40
C ————————  MD=M(MA)
C ————————  CHECK FOR THE EQUIVALENCE OF MD AND C
      (6)   IF(MD.EQ.C) GO TO 50
            MA=MA+1
            GO TO 4
   50 ————  MD=0
            MA=1
            GO TO 7
   40       MA=MA+1
            GO TO 6
   50 ————  MD=MA
            MA=1
        (7)— M(MA)=MD
            WRITE(6,60)L,K,(M(I),I=1,L)
        (8) STOP
   10 ————  FORMAT (I3,D3.0)
   20 ————  FORMAT(1X15)
   60 ————  FORMAT(' L=....', I3, 'K=——', D3.0/(1X15)
   70       FORMAT(15I5)
            END
```

The following points are seen for the arithmetic model calculation.

(1) Only FORTRAN arithmetic equivalent statements are used for the corresponding P.T.L. productions.

(2) It accepts data only in the decimal integer mode and not in the binary (TRUE/FALSE) code.

(3) The memory is represented by an array of elements. For example:M(1),M(2)....M(600)

(4) The arithmetic model is divided into subsections. In the first section the memory is filled from the data cards.

(5) In the section two the R.T.L. states 1 to 3 shows the fetch cycle for the given location I_p and memory address register is initialized at one.

(6) The section three gives all the operation of the system and the result of the operation is stored in the last memory location (LZ) as defined in the program.

(7) The fourth section is output print and the format statements. The format statement shows the datas are in decimal so for I_p format I5 is used.

(8) The Format F5.0 is given for the location of I which shows that I can have any location out of 600 location of memory.

(9) In the fifth section the physical end of the program is realized.

The above properties of the arithmetic model must be correctly interpreted for digital simulation of system (A.M.) It produces a very helpful and concise simulation program.

4.3.3 Logical Model Simulation Approach for Associative Memory

The simulation of an associative memory system of LRC

computer can be done by the logical model approach. Bit-by-bit detailed operations of the associative memory is realized from the M.T.L flow graph as shown in figure 4.3. The logical model simulation program is given below:

```
C  C     SIMULATION OF ASSOCIATIVE MEMORY CIRCUIT-LOGICAL
     $   MODEL
         INTEGER D,T
         LOGICAL*1 MD,LD,LMD,MA,MC,C,M,S,E
         DIMENSION MD(16),LD(22),LMD(12),MA(12),MC(12),
     $   C(12), M(20,12)
         READ(5,10)L,N,D
         LL=L+1
         CALL DTOL(L,LD,M)
         CALL DTOL(N,MD,M)
         CALL DTOL(LL,LMD,M)
C ------ THE FOLLOWING STATEMENT READS DECIMAL
     $   DOG FROM DATA CARDS,CONVERTS THEM TO LOGICAL
     $   AND STORE L MEMORY WORDS WITH D AS D
         CALL FILL D(M,L,D)
         WRITE(6,90)
        CALL PRINT(M,L,D)
C------- SIMULATION OF MTL STATEMENTS STARTS AT
     1   1 AND ENDS AT STATEMENT No.7
  1      CALL RD(MD,MA,M)
     2   CALL LOGA(D,MA,MC,L,M)
      3  CALL WR(MC,C,M)
         CALL FALSE(MA,M)
         MA(M)=.TRUE.
         CALL RD(MA,MB,M)
```

```
4 ——— CALL [illegible](MA,LED,T,N)
      IF(T)GØ TØ [illegible]
5 ——— CALL [illegible](MA,MD,T,N)
      IF(T) GØ TØ 30
      CALL LOA[illegible](M,MA,MB,L,N)
6 ——— CALL [illegible](MB,C,T,N)
      IF(T) GØ TØ 40
      CALL [illegible](MA,MB,MA,C,N)
      GØ TØ 4
20 ——— CALL FALS[illegible](MD,N)
      CALL [illegible](LD,MA,N)
      GØ TØ 7
30 ——— CALL [illegible](MA,MB,MA,[illegible],N)
      GØ TØ 4
40 ——— CALL [illegible](MA,MD,N)
      CALL [illegible](LD,MA,N)
7 ——— CALL LØ[illegible](MD,N,MA,L,N)
      WRITE (6,60)
      CALL [illegible](M,L,N)
8 ——— STOP
10 ——— FORMAT (I4,F4.0,I2)
50 ——— FORMAT ([illegible], 'INTERNAL MEMORY CONTENTS')
60 ——— FORMAT ([illegible], 'INTERNAL MEMORY CONTENTS')
      END
```

In analyzing the beginning of the program following points should be observed:

(1) Memory is represented by a matrix M(L,N) where L is the number of locations, here L=100 and N is the number of bits per word = 16 bits/WORD.

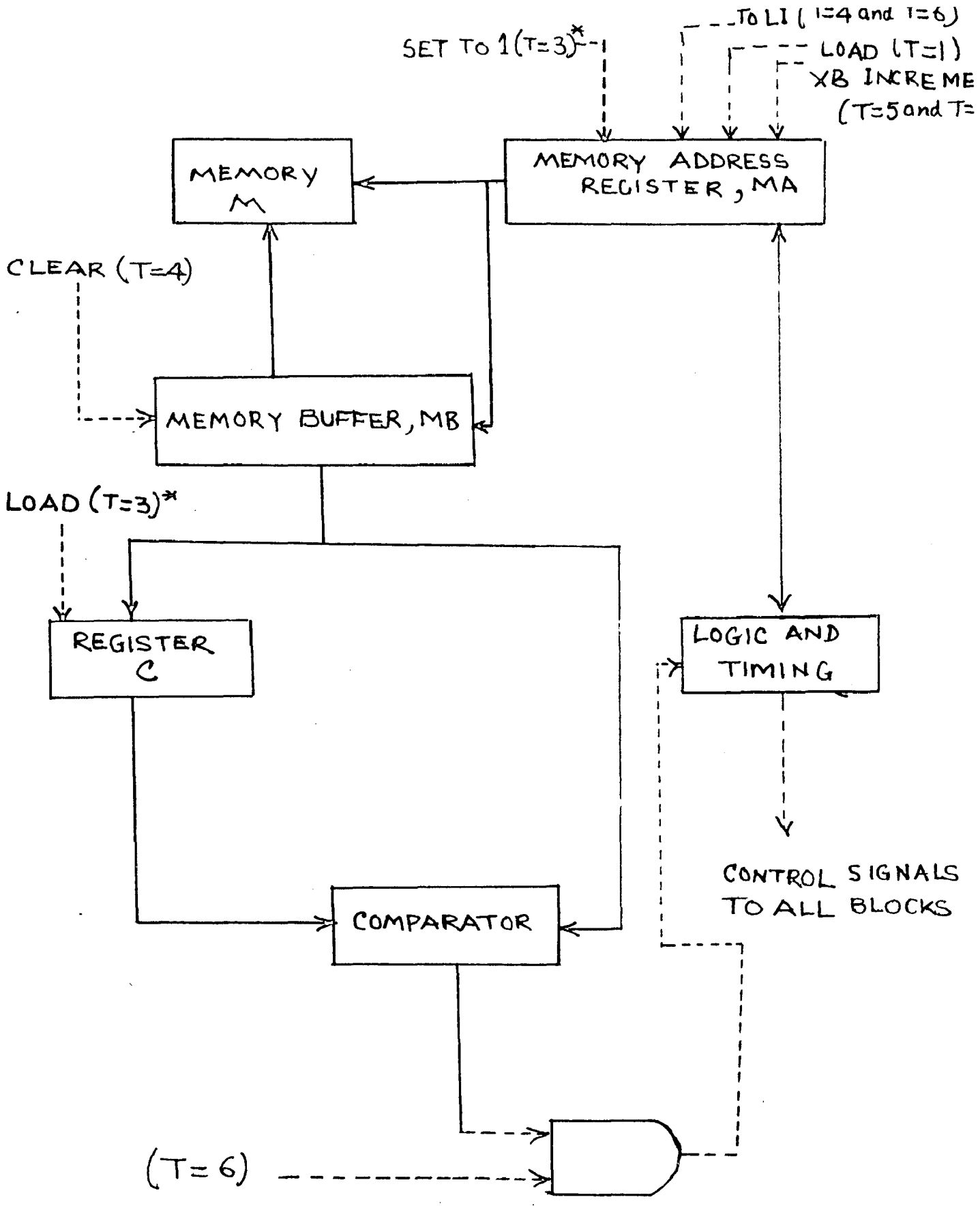


FIG. 4.3. HARDWARE CIRCUIT OF ASSOCIATIVE MEMORY

(2) The input to the memory are in decimal numbers and are converted into binary numbers.

(3) The simulation of associative memory starts from statement numbe encircled ① and ends up at encircled statement number ⑦.

(4) It has been observed that there is a one to one correspondence of R.T.L flow graph.

(5) The simulation program is divided into 8 sections, corresponding to eight different states of state controller in R.T.L. flow chart.

(6) It shows how a bit-by-bit operations are performed by associative memory.

4.4 HARDWARE REALIZATION OF ASSOCIATIVE MEMORY

The hardware realization of associative memory is accomplished from the R.T.L. Chart/graph. Figure 4.3 shows the associative memory circuit and the elementary components of LINC computer. The only change between this circuit and LINC computer circuit is that a comparator and a register C is added to the actual computer. The register C stores the contents of the memory location X, and comparator indicates a matching word in the memory.

The memory address register, MA, gives the current location in the memory. The data can be transferred from memory address register into memory buffer register, MB.

The diagram given in figure 4.3 shows all essential

details, indicating whether or not a desired pattern is stored in the memory. There is no doubt about the usefulness of and desirability of associative memories. And no technical difficulties in building them.

However, disregarding their cost, their properties make them very attractive for general purpose use. Many technical details can be solved only in connection with the proper choice of components which makes associative memories economically feasible.

CHAPTER-5

DIGITAL SIMULATION OF INTEL 8080 MICRO PROCESSOR

5.1 INTRODUCTION

Microprocessors are a remarkable versatile new tool. They lower the cost and increase the flexibility of electronic equipment. Together with the memory and other peripheral circuitry, microprocessor chip form a complete microcomputer.

The INTEL 8080 is taken as a special problem for the simulation purpose. This microprocessor is very commonly used now a days. The simulation of the microprocessor is important where it is not available.

The simultation of 8080 can be done by the following two techniques:

(1) Arithmetic Model Approach

(2) Logical Model Approach.

The 8080 microprocessor is a 40 pin, single chip, N-MOS CPU-containg instruction decoding and hardware on the same chip. The 8080 has a basic word length of 8 bits and a direct addressability upto 65 K bytes. The 8080 transfers data and internal state information via an 8-bit bidirectional 3-state data bus (D_0-D_7). It has six-control outputs, four power inputs, four control inputs and two clock inputs.

The architecture of INTEL 8080 is given in figure 5.1 and is discussed briefly.

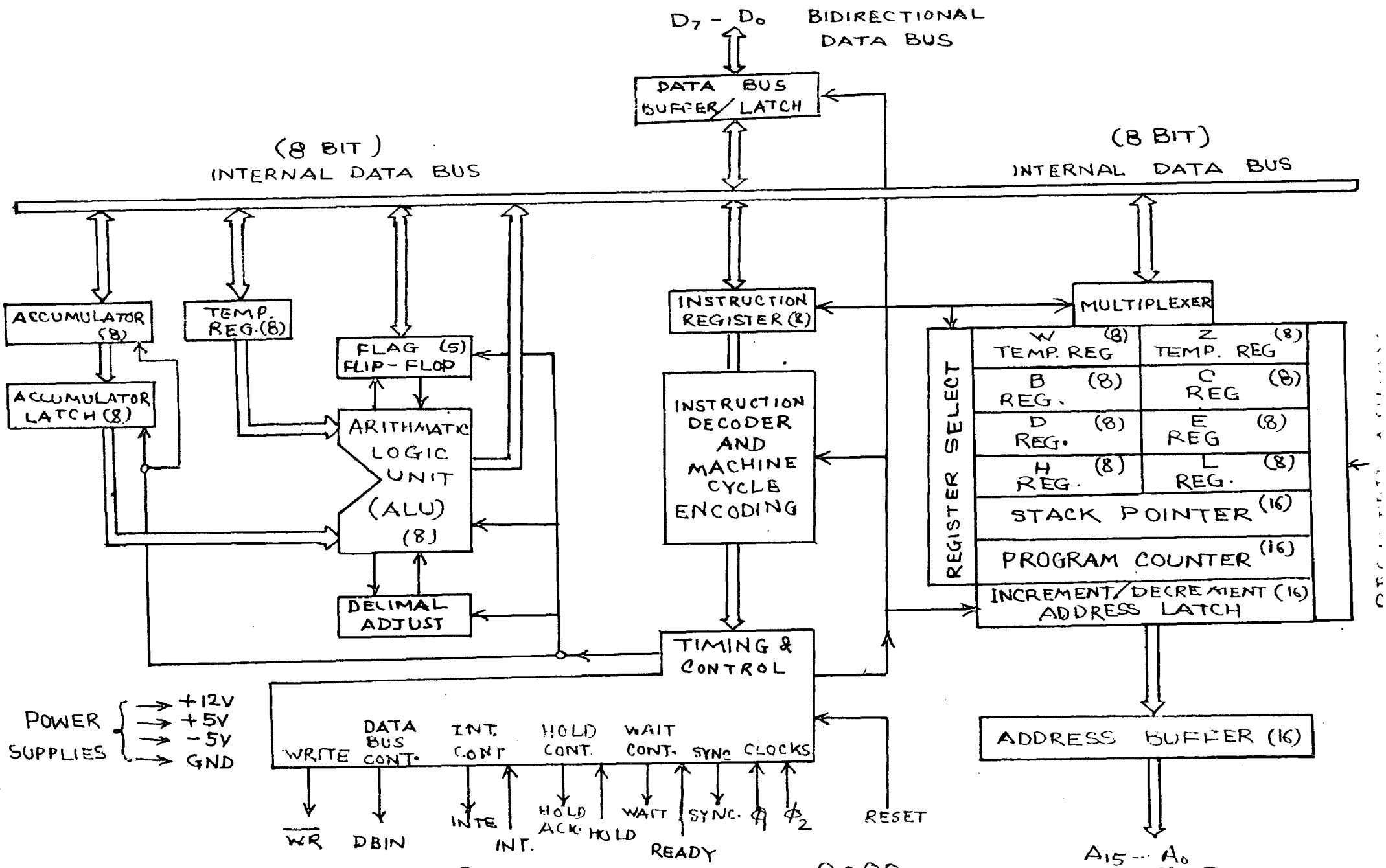


FIG 5.1 FUNCTIONAL BLOCK DIAGRAM OF 8080

5.2.1 Architecture of Intel 8080 Microprocessor

The study of architecture of microprocessor is very useful. The appropriate application of microprocessor is governed by the architecture. Figure 5.1 gives the block diagram of Intel 8080 microprocessor (C.P.U.). It consists of four main sections. These are

i) Register array and address logic,
ii) Arithmetic and logic unit,
iii) Instruction register and control section
iv) Bidirectional, 3-state data bus buffer.

5.2.1 Register Array and Address Logic (6,7,9)

The important part of the CPU is the register section. A 16-bit static RAM array has the following organizational sections:

(i) Six-8 bit general purpose registers may be addressed in pairs named as B,C,D,E and H,L provides single or double precision (16 bit) operations.
(ii) Program counter (PC)
(iii) Stack pointer (SP)
(iv) A temporary register pair called W,Z.

The program counter is a 16 bit register used for the indication of the track of the current instruction and increments upon every instruction fetch. The temporary register pair W-Z is not program addressable and is used only for internal

execution of instructions.

A stack is an area of read/write memory set aside by the programmer in which data or address of the next stack location available in memory are stored and retarived by stack operation. When data is 'pushed onto the stack, the stack pointer is decremented and then 'Popped' off the SP is incremented.

5.2.1.2 Arithmatic and Logic Unit

The ALU performs arithmatical, logical and rotate operations. The ALU contains the following registers:

(i) An 8-bit accumulator (A),

(ii) An 8-bit temporary accumulator (ACT),

(iii) An 8-bit temporary register (TMP), and

(iv) A 5-bit flag register (zero, Carry, Sign, Parity and Auxiliary Carry)

A 8-bit accumulator is a general purpose register. It serves both as source and destination register for operations involving some other register, the ALU or memory. The DAA instruction with auxiliary carry is used for the decimal correction of the accumulator.

5.2.1.3 Instruction Register and Control

The instruction register is an 8-bit register. The first byte of an instruction (containing the op-code) is transferred by it to the decoder and control section for decoding the instruction. The output of the decoder combined with various timing signals and given control signals for the register array,

ALU, and data buffer blocks. It also generates status and timing signals.

5.2.1.4 Tri-state Data Bus Buffer

The data bus buffer is an 8-bit, bidirectional, tristate, buffer. It isolate the internal bus of the processor from the external data bus (D_0-D_7). The internal bus contents are transferred by it to an 8-bit latch during the output mode. The 8-bit latch drives the data bus output buffers. These buffers are switched off during input or nontransfer operations. Data from the external data bus is transferred to the internal bus in the input mode.

The realization of microprocessor, can be bee illustrated by micro-computer (A micro digital computer). Instructions and data are stored in memory. The control unit fetches instructions one at a time from the memory, decodes them, and then execute them. Data needed for execution are fetched from memory, combined with temporary data stored in registers within ALU and restored to the memory.

5.2.2 Pin Configuration of Intel 8060

The study of pin configuration of INTEL 8060 is very important fromthe application point of view. It gives the informations about the terminal connections where the peripheral devices has to be connected.And also given the direction of the flow of informations.

The 8060 is a 40pin dual in line package NMOS chip shown in figure 5.2. The following discussion gives the functions of all of the 8060 input output pins.

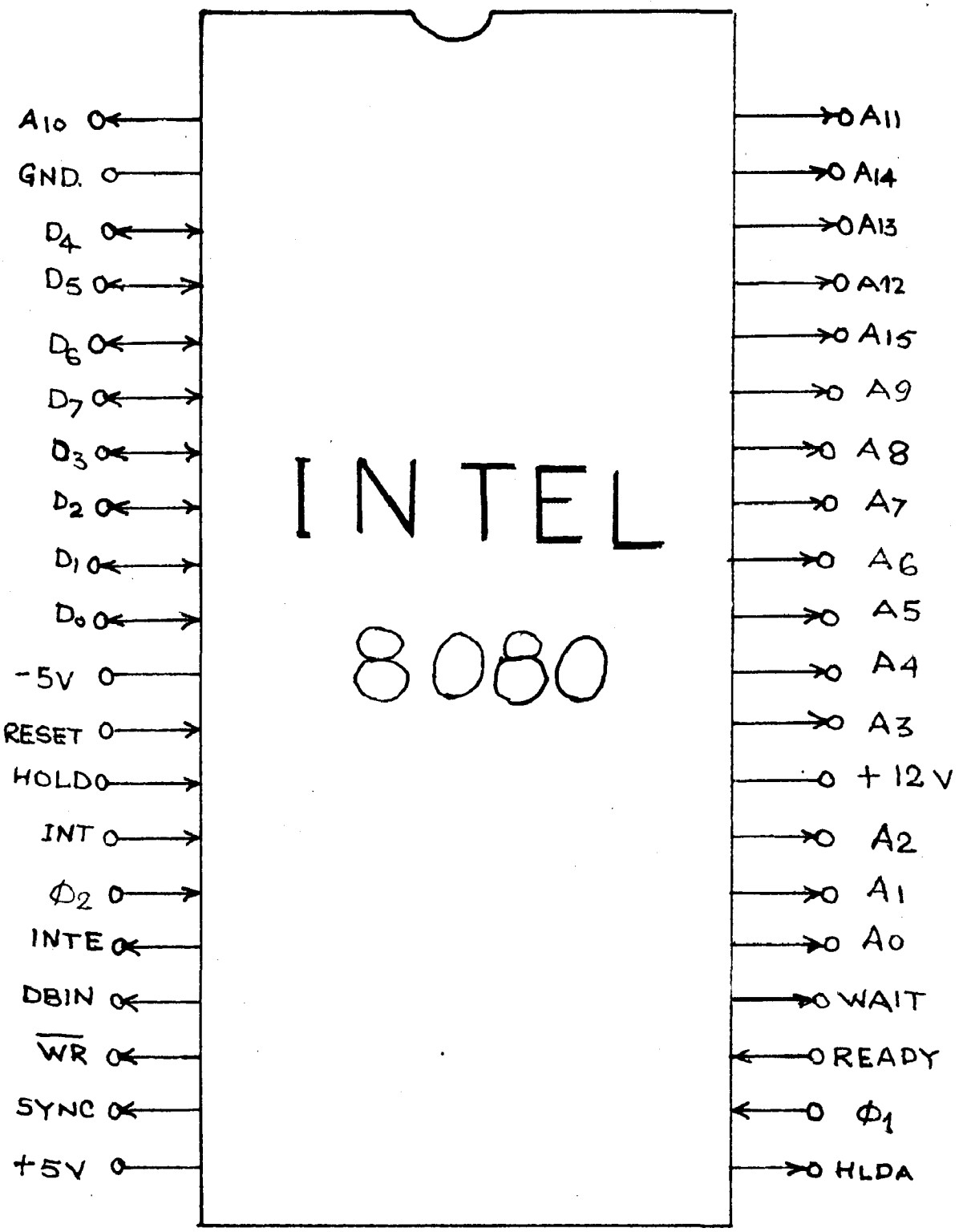


FIG. 5.2. 8080 CPU PIN CONFIGURATION

(1) ADDRESS BUS

The pins A_0 to A_{15} acts as an address bus. These are 3 state output pins, which gives the address to memory upto 64K, 8-bit word or denotes the input-output device number for upto 256 input devices and 256 output devices. The least significant address bit is denoted by A_0 and most significant bit by A_{15} reading from left to right sequence. (9)

(2) Data Bus- The three state bidirectional 8 pin lines acts as a data bus. The function of this bus is to provide bidirectional communication between the CPU, memory and input/ output devices for instruction and data transfer.(9)

(3) SYNC- It gives an output signal to indicate the start of each machine cycle.

(4) DBIN- It is referred as data bus in, gives an output signal to external circuits indicating that the data bus is in the input mode i.e. data bus can accept data if is empty.

(5) READY- The ready signal indicates to the 8080 that valid memory or input data is available on the 8080 data bus. The signal is used to synchronize the CPU with slower memory or input/output devices. The 8080 will be in wait state for as long as the READY line is low after sending an address out.

(6) WAIT- The wait signal is an output signal. It gives the information that the CPU is in a wait state.

(7) $\overline{WR}$ (write)- When the informations are to be written into memory or output function, this signal actuates low ($\overline{WR}$=0)

showing that this particular machine cycle is MEMORY WRITE OR OUTPUTfunction cycle. The data on the data bus is stable while WR signal is active low ($\overline{WR}$ = 0).

(8) HOLD- This hold signal requests the CPU to enter the HOLD state. The hold state allows an external device to gain control of the 8080 address and data bus as soon as the 8080 has complo ed its use of these buses for the current machine cycle. It is recognized under the following conditions:

(1) The CPU is in the HALT state.

(2) The CPU is in the T2 or wait state and the ready signal is active. As a result of entering the HOLD state the CPU address. BUS(A_{15}-A_0) and data bus (D_7-D_0) will be in their high impedance state. The CPUacknowledges its state with the HOLD ACKNOWLEDGE pin.

(9) HLDA (Hold Acknowledgement) - HLDA gives an outgoing signal. It appears with the hold signal. At this time data and Address bus are at high impedance state. The HLDA signal starts at

(1) T3 state for READY MEMORY or INPUT,and

(2) The clock period following T3 for WRITE MEMORY OR OUTPUT operation.

(10) INTERRUPT ENABLE (WRITE output) - An interrupt enable line is high acts in coincidence with the ϕ_2 clock to set the internal interrupt latch. This gives assurance that any interrupt instruction in progress is completed before the INT can be processed.

(11) INTERRUPT (INT input)- CPU recognizes an interrupt request on this line at the end of the current instruction or while

halted. If the CPU is in the HOLD state or if the interrupt enable flip-flop is reset it will not honour the request.

12. RESET (input): The reset is an input signal when given to the CPU it, clears the contents of program counter. After RESET , the program starts at location zero in memory. INTE and HLDA flip-flops are also reset and rest remain unchanged.

15. V_{ss} - the V_{ss} pin acts as a ground reference pin.

14. V_{cc} - +5 ± 5% volts
19. V_{bb} - -3 ± 5% volts
] used for the peripheral devices or substate bias

16. V_{dd} - +12 ± 5% volt - used for the oscillator circuit

17. ϕ_1, ϕ_2 - These are two clock pulses used by 8080. Sync. signal is related to the rising edge of ϕ_2 clock.

5.3 <u>GENERAL FLOW OF MICRO CODE MICROPROCESSOR</u>

The simulation is carried out in three parts.

In the first section RTL productions are developed for different instructions. In the second section arithmetic models are developed for each instruction, simply giving the operation performed by each instruction.

The third section describe the logical model for different instructions corresponding to the RTLproductions. This part gives bit-by-bit operations performed by an instruction.

The RTL productions, arithmetic models and logical models for CDC6 instruction set is given in tabular form.

The execution of an instruction is performed in the following sub-cycles.

(1) Fetch cycle

(2) Execute cycle

9.3.1 FETCH Cycle

(a) In the fetch cycle the CPU provides the address of an instruction in a memory location via the address bus.

(b) The address is decoded and the instruction is read from memory into the memory data register of the CPU.

The RTL flowgraph and the arithmetic and logical model for the fetch cycle are given below:

RTL FLOW GRAPH

ADDR ← PC

DBR ← M(ADDR)

IR ← DBR

PC ← PC+1

where ADDR denotes address bus of 16 bits long

DBR denotes Data bus of 8 bit length

PC is program counter of 16 bit value

R is a 16 bit register

The Arithmetic Model

ADDD ← PC

DDD ← M(ADDD)

IR ← DDD

PC ← PC+1

Logical Model

CALL RD (PC, ADDD, 16)

CALL LODA(U, ADDD, DDD, L^{M}, M)

CALL RD(DDD, IR, M)

CALL TADD (PC, P, PC, 0, M)

9.3.2 EXECUTE CYCLE- In the execution cycle, the instruction is decoded and the required operation is performed corresponding to the decoded instructions.

The following notations are used in the simulation:

SR is SOURCE REGISTER

DR is DATA REGISTER

Z is TEMP. REGISTER

U is TEMP. REGISTER

WZ is TEMP. REGISTER PAIR

HL is REGISTER PAIR

TMP is TEMPORARY REGISTER

ACT is TEMPORARY ACCUMULATOR

M denotes MEMORY

T is 8 bit REGISTER

L^{M} denotes MEMORY LOCATION

N is 8 bits

M is 16 bits

D is CARRY BIT

B2 is BYTE 2

B3 is BYTE 3

RP denotes REGISTER PAIR

CC is 8 bit REGISTER

The RTL flow graphs, the Arithmatic models and the logical models for each instructions has been described and are given in a tabular form.

MOV TRANSFER GROUP

S.No.	MNEMONIC	OP-CODE	RTL FLOW CHART	ARITHMETIC MODEL	LOGICAL MODEL
1.	MOVB r1,r2	01DDDSSS	Z ← (SR) DR ← Z	r1 = r2	CALL BR (SR,Z,M) CALL BW (Z,DR,M)
2.	MOV r,M	01DDD110,10	ADDR ← (HL) DDD ← M(ADDR) DR ← DDD	r = M(HL)	CALL BR(HL,ADDR,M) CALL LOCA(M,ADDR,DDD,L^{M},M) CALL BW(DDD,DR,M)
3.	MOV M,r	01110SSS	Z ← r ADDR ← HL DDD ← Z M(ADDR)←DDD	M(HL)=r	CALL BR(r,Z,M) CALL BR(HL,ADDR,M) CALL BW(Z,DDD,M) CALL LOCW(DDD,M,ADDR,LM)
4.	MVI r,data	00DDD110	DDD ← B2 DR ← DDD	r=B(2)	CALL EQUAL (B2,9,16,ADDR,1,8) CALL EQUAL(DDD,1,8,DR,1,8)
5.	MVI M,data	00110110	DDD ←⟨B2⟩ Z ← (DDD) ADDR ← HL M(ADDR)← Z		CALL EQUAL (B2,9,1y, DDD,1,0) CALL BW (DDD,Z,M) CALL BR(HL,ADDR,M,16) CALL LOCW (Z,M,ADDR,L^{M},M)

Table continued

S.No.	MNEMONIC	OP-CODE	RTL FLOW GRAPH	ARITHMETIC MODEL	LOGICAL MODEL
6.	LXI rp, data	00RP0001	DDB←⟨B2⟩ Z ← DDB rl ← Z DDB ←⟨B3⟩ W ← DDB rh ← W	rh = B(3) rl = B(2)	CALL EQUAL (B2,9,16,DDB,1,8) CALL EQUAL (DDB,1,8,Z,1,8) CALL EQ (Z,rl,N) CALL EQUAL (B3,17,24,DDB,1,8) CALL EQ (DDB,W,N) CALL EQ (W,rh,N)
7.	LDA addr	00111010	ADDB ←⟨B(2)⟩ Z ← M(ADDB) ADDB ←⟨B(3)⟩ W ← M(ADDB) PC ← PC+1 ADDB ← WZ A ← M(ADDB)	A = B(3)B(2)	CALL EQUAL (B2,9,16,ADDB,1,8) CALL LOAD(M,ADDB,Z,L*,N) CALL EQUAL (B3,17,24,ADDB,9,16) CALL LOAD(M,ADDB,W,L*,N) CALL TADD(PC,P,PC,D,N) CALL EQ (WZ,ADDB,N) CALL LOAD(M,ADDB,A,L*,N)
8.	STA addr	00110010	Z ← ⟨B(2)⟩ PC ← PC+1 W ←⟨B(3)⟩ PC ← PC+1 WZ ← W,Z ADDB ← WZ M(ADDB) ← A	M(B(3)B(2))=A	CALL EQUAL(B2,9,16,AZ ,1,8) CALL TADD(PC,P,PC,D,N) CALL EQUAL(B3,17,24,W ,1,8) CALL TADD(PC,P,PC,D,N) CALL EQ ((W,Z),WZ,16) CALL EQ(WZ,ADDB,N) CALL LOAD(A,M,ADDB,L*,N)

S.No.	MNEMONIC	OP CODE	RTL DESCRIPTION	MICRO-PROGRAM MODEL	LOGICAL-MODEL
9.	LHLD addr	00101010	Z ← M(PC) PC ← PC+1 W ← M(PC) PC ← PC+1 WZ ← W,Z ADDR ← WZ L ← M(ADDR) WZ ← WZ+1 ADDR ← WZ H ← M(ADDR)	MA ← M(PC)PC ← PC+1	CALL MA(PC,Z,M) CALL TADD(PC,P,PC,D,M*) CALL MA(PC,W,M) CALL TADD(PC,P,PC,D,M*) CALL MOVAL(W,Z,WZ,16) CALL MA(WZ,ADDR,16) CALL LODA(M,ADDR,L,L^M,M) CALL TADD(WZ,P,WZ,D,M*) CALL MA(WZ,ADDR,16) CALL LODA(M,ADDR,H,L^M,M)
10.	SHLD addr	00100010	WZ ← M(PC)M(PC) ADDR ← WZ DDD ← L M(ADDR) ← DDD WZ ← WZ+1 PC ← PC+1 ADDR ← WZ DDD ← H M(ADDR) ← DDD	M(M(PC)M(PC))←L M(M(PC)M(PC)+1)←H	CALL MA(PC,W,16) CALL MA(WZ,ADDR,16) CALL MO(L,DDD,8) CALL LOAM(DDD,M,ADDR,L^M,M) CALL TADD(WZ,P,WZ,D,16) CALL TADD(PC,P,PC,D,16) CALL MA(WZ,ADDR,16) CALL MA(H,DDD,8) CALL LOAM(DDD,M,ADDR,L^M,M)
11.	LDAX rp	00RP1010	ADDR ← RP A ← M(ADDR)	A←M(RP)	CALL MA(RP,ADDR,16) CALL LODA(M,ADDR,A,L^M,M)

No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL-MODEL
12.	STAX DP	00000010	ADDR ← DP M(ADDR) ← A	M(DP) ← A	CALL BUS(DP,ADDR,16) CALL LOGIC(A,M,ADDR,MEMW,M)
13.	XCHG	11101011	DSAV ← D Z ← DSAV DSAV ← D W ← DSAV DSAV ← L E ← DSAV DSAV ← H D ← DSAV DSAV ← W H ← DSAV DSAV ← Z L ← DSAV	DE ← HL HL ← DE	CALL BUS(D,DSAV,8) CALL BUS(DSAV,Z,8) CALL BUS(D,DSAV,8) CALL BUS(DSAV,W,8) CALL BUS(L,DSAV,8) CALL BUS(DSAV,E,8) CALL BUS(H,DSAV,8) CALL BUS(DSAV,D,8) CALL BUS(W,H,8) CALL BUS(Z,L,8)

ARITHMETIC GROUP

S.No.	MNEMONIC	OP-CODE	REG FLAG CHANGE	ARITHMETIC MODEL	LOGICAL MODEL
1.	ADD r	10CCOSSS	TMP ← SR ACT ← A A ← ACT+TMP	A←A+r	CALL MOV(SR,TMP,D) CALL MOV(A,ACT,D) CALL MADD (ACT,TMP,A,B,D)
2.	ADD M	10CCOOLO	ADDR ← HL TMP ← M(ADDR) ACT ← (A) A ← ACT+TMP	A←A+M(HL)	CALL MOV(HL,ADDR,P) CALL LOAD(M,ADDR,TMP,C,D) CALL MOV(A,ACT,D) CALL MADD(ACT,TMP,A,B,D)
3.	ADI data	11CC0110	ACT ← A TMP ← ⟨D2⟩ A ← ACT+TMP	A←A+D(2)	CALL MOV (A,ACT,D) CALL MVI(D(2),TMP,N) CALL MADD(ACT,TMP,A,B,D)
4.	ADC r	10001SSS	TMP ← SR ACT ← A A ← ACT+TMP+CC	A←A+r+CC	CALL MOV(SR,TMP,D) CALL MOV(A,ACT,D) CALL MADD(TMP,CC,TMP,D,D) CALL MADD (TMP,ACT,A,D,D)
5.	ACI data	11001110	ACT ← A TMP ←⟨D(2)⟩ A←ACT+TMP+CC	A←A+D(2)+CC	CALL MOV(A,ACT,D) CALL MVI(D2,TMP,D) CALL ADD(ACT,TMP,A,D)
6.	ADC M	10001110	ADDR ← HL TMP ←M(ADDR) ACT ←(A) A ← T+TMP+CC	A ← A+M(HL)+CC	CALL MOV(HL,ADDR,D) CALL LOAD(M,ADDR,TMP,LOC,D) CALL MOV(A,ACT,D) CALL ADD(ACT,TMP,A,D)

S.No	MNEMONIC	OP-CODE	RTL FLOW CHART	ARITHMETIC MODEL	LOGICAL MODEL
7.	SUB r	10010SSS	TMP ← SR ACT ← A A ← ACT-TMP	A = A-r	CALL MV(SR,TMP,D) CALL MV(A,ACT,D) CALL TCOM(TMP,TMP,D) CALL TADD(ACT,TMP,A,D,D)
8.	SUB M	10010110	ADDR←HL TMP ← M(ADDR) ACT ← (A) A ← ACT-TMP	A = A-M(HL)	CALL MV(HL,ADDR,D) CALL MOVM(M,ADDR,TMP,1^{M},D) CALL MV(A,ACT,D) CALL COM(TMP,TMP,D) CALL TADD(ACT,TMP,A,D,D)
9.	SUI data	11010110	ACT ← A TMP ←(B2) A ← ACT-TMP	A = A-B(2)	CALL MV(A,ACT,D) CALL MVI(B2,TMP,D) CALL COM(TMP,TMP,D) CALL TADD(ACT,TMP,A,D,D)
10.	SBB	10011SSS	ACT ← A TMP ← SR A ← ACT-TMP-CC	A=A-r-CC	CALL MV(A,ACT,D) CALL MV (SR,TMP,D) CALL CCM(CC,CC,D) CALL ADD(TMP,CC,TMP,D,D) CALL COM(TMP,TMP,D) CALL TADD(ACT,TMP,A,D,D)

S.No.	MNEMONIC	OP-CODE	REG FLOW CHART	ARITHMETIC TRACE	LOGICAL MODEL
11.	SBB M	10011110	ACT ← A ADDR ← HL TMP ← M(ADDR) A ← ACT-TMP-CC	A=A-M(HL)-CC	CALL RQ(A,ACT,D) CALL RQ(HL,ADDR,D) CALL LOD1(D,ADDR,TMP,L,D) CALL CC1(CC,CC,D) CALL CC1(TMP,TMP,D) CALL ADD(ACT,TMP,A,D)
12.	SBI data	11011110	ACT ← A TMP ← (B2) A ← ACT-TMP-CC	A=A-D(2)-CC	CALL RQ(A,ACT,D) CALL RQ(B2,TMP,D) CALL CC1(CC,CC,D) CALL ADD(TMP,CC,TMP,D,D) CALL CC1(TMP,TMP,D) CALL TADD(ACT,TMP,A,D,D)
13.	INR r	00DDD100	TMP ← SR A ← TMP+1 SR ← A	r = r+1	CALL RQ(SR,TMP D) CALL TADD(TMP,T, A,D,D) CALL RQ(A,ACT,D)
14.	INR M	00110100	ADDR ← HL TMP ← M(ADDR) A ← TMP+1 PC ← PC+1 ADDR ← HL M(ADDR) ← A	M(HL)=M(HL)+1	CALL RQ(HL,ADDR,D) CALL LOD1(D,ADDR,TMP,L,D) CALL TADD(TMP,T, A,D,N) CALL TADD (PC,P,CC,D,D) CALL RQ(HL,ADDR,D) CALL LOD1(A,D,ADDR,L,D)

3. LOGICAL GROUP

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ALGEBRAIC EQUATION	LOGICAL MODEL
1.	ANA r	10100[illegible]	TMP ← SR ACT ← A A ← ACT ∧ TMP	Does not exist	CALL MOV(SR,TMP,□) CALL MOV(A,ACT,□) CALL AND (ACT,TMP,A,□)
2.	ANA M	10100110	ADDR ← (HL) TMP ← M(ADDR) ACT ← (A) A ← ACT ∧ TMP	Does not exist	CALL MOV(HL,ADDR,□) CALL LOAD(M,ADDR,TMP,□) CALL MOV(A,ACT,□) CALL AND(ACT,TMP,A,□)
3.	ANI data	11100110	ACT ← A DR ← (B2) TMP ← DR A ← (ACT) ∧ (TMP)	Does not exist	CALL MOV(A,ACT,□) CALL MOV(B2,DR,□) CALL MOV (DR,TMP,□) CALL AND (ACT,TMP,A,□)
4.	XRA r	10101[illegible]	ACT ← A TMP ← SR A ← ACT ⊕ TMP	Does not exist	CALL MOV(A,ACT,□) CALL MOV(SR,TMP,□) CALL EXOR(ACT,TMP,A,□)
5.	XRA M	10101110	ACT ← A ADDR ← H L TMP ← M (ADDR) A ← (ACT) ⊕ (TMP)	Does not exist	CALL MOV(A,ACT,□) CALL MOV(HL,ADDR,□) CALL LOAD (M,ADDR,TMP,□) CALL EXOR(ACT,TMP,A,□)

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
6.	XRI data	11101110	ACT ← A DR ← ⟨PC⟩ TMP ← DR PC ← PC + 1 A ← ACT ⊕ TMP	Does not exist	CALL DR(A,ACT,R) CALL DR(PC,DR,R) CALL DR(DR,TMP,R) CALL ADD(PC,P,PC,D,R) CALL EXOR(ACT,TMP,A,R)
7.	ORA r	10110SSS	ACT ← A TMP ← SR A ← ACT ∨ TMP	Does not exist	CALL DR(A,ACT,R) CALL DR(SR,TMP,R) CALL OR(ACT,TMP,A,R)
8.	ORA M	10110110	ADR ← HL TMP ← M(ADR) ACT ← A A ← ACT ∨ TMP	Does not exist	CALL DR(L,ADR,R) CALL LOAD(M,ADR,TMP,L ,R) CALL DR(A,ACT,R) CALL OR(ACT,TMP,R)
9.	ORI data	11110110	ACT ← A ADR ← ⟨PC⟩ TMP ← DR PC ← PC+1 A ← ACT∨TMP	Does not exist	CALL DR(A,ACT,R) CALL DRAL(PC,0,16,DR,1,R) CALL LOAL(ADR,2,0,TMP,1,0) CALL ADD(PC,P,PC,D,R) CALL OR (ACT,TMP,A,R)

10.	CMP r	10111SSS	ACT ← A
			TMP ← SR
			ACT←TMP
			IF(ACT.EQ.TMP)Z=1
			Z=0
			IF(ACT.LT.TMP)CY=1
			CY = 0
				IF(A.EQ.r)Z=1.0
				Z=0.0
				IF(A.LT.r)C 1.0
				CY=0.0
					CALL DR(A,ACT,R)
					CALL DR(SR,TMP,R)
					CALL TEST DR(ACT,TMP,Z,R)
					IF(ACT.LT.TMP)CY=TRUE
					CY=FALSE

No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL

11.	CMP D	10111110	ADDR ← HL
			TMP ← D(ADDR)
			ACT ← A
			ACT ← SUB
			IF(ACT.D.TMP)Z←1
			Z←0
			IF(ACT.LT.TMP)CY←1
			CY ← 0
				IF(ACT.D.TMP)Z←1.0
				Z←0.0
				IF(ACT.LT.TMP)CY←1.0
				CY←0.0
					CALL D(HL,ADDR,D)
					CALL MOV(D,ADDR,TMP,L,D)
					CALL D(A,ACT,D)
					CALL TEST B.(ACT,TMP,Z,D)
					IF(ACT.LT.TMP)CY←TRUE
					CY←FALSE.

12.	CPI data	11111110	ACT ← A
			DATA ←(PC)
			TMP ← DATA
			IF(ACT.D.TMP)Z←1
			Z←0
			IF(ACT.LT.TMP)CY←1
			CY←0
				IF(ACT.D.TMP)Z←1.0
				Z←0.0
				IF(ACT.LT.TMP)(Y←1.0)
				CY←0.0
					CALL D(A,ACT,D)
					CALL D(D2,DATA,D)
					CALL D(DATA,TMP,D)
					CALL TEST B(ACT,TMP,Z)
					IF(ACT.LT.TMP)CY←TRUE
					CY←FALSE

13.	RLC	00000111	ACT ← A A ← rl ACT A(0)←A(7) CY ← A(7)	Does not exist	CALL D(A,ACT,D) CALL RL(ACT,A,D) (A(1).EQ.A(0) CY.D.A(0)
14.	RRC	00001111	ACT ← A A ← RR ACT A(7)←A(0) ACY ← A(0)	Does not exist	CALL D(A,ACT,D) CALL RR(ACT,A,D) A(7).D.A(0) CY.D.A(0)

S.No.	MNEMONIC	OP-CODE	REG. PLAN CHART	ARITHMETIC MODEL	LOGICAL MODEL
15.	DCR r	00DDD101	TMP ← SR A ← TMP-1 SR ← A	r = r-1	CALL DR(SR,TMP,B) CALL CON(T,F,B) CALL ADD(TMP,F,A,B) CALL DR(A,SR,B)
16.	DCR M	00110101	ADDR ← HL TMP ← M(ADDR) A ← TMP-1 PC ← PC+1 ADDR ← HL M(ADDR)←A	M(HL)=M(HL)-1	CALL DR(HL,ADDR,B) CALL LOAD(B,ADDR,TMP,L,B) CALL CON(T,F,B) CALL ADD(TMP,F,A,B) CALL ER(HL,ADDR,B) CALL LOAD(A,B,ADDR,L,B)
17.	INX RP	00RP1011	RP ← PC RP ← RP+1	RP=RP+1	CALL ER(PC,RP,B) CALL SADD(RP,P,RP,B,B)
18.	DAD RP	00RP1001	(rh)(rl)←RP ACT ← rl TMP ← L A ← TMP+ACT L ← A CC ← (CY)CARRY ACT ← rh TMP ← H A ← TMP+ACT H ← A CC ← CARRY	HL=HL+RP	CALL ERVAL(RP,1,0,rl,1,0) CALL ER(rl,ACT,B) CALL ER(L,TMP,B) CALL SADD(TMP,ACT,A,B,B) CALL ER(A,L,B) CC(B)=CY CALL ER(rh,ACT,B) CALL ER(H,TMP,B) CALL SADD(TMP,ACT,A,B,B) CALL ER(A,H,B) CC(B)=CY

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
15.	RAL	00010111	ACT ← A CY ← A(7) A ← RLACT A(0) ← CY	Does not exist	CALL U. (A,ACT,N) CY=A(7) CALL RL(ACT,A,N) A(0)=CY
16.	RAR	00011111	ACT ← A CY ← A(0) A ← RRACT A(7) ← CY	Does not exist	CALL D3(A,ACT,N) CY=A(0) CALL RR(ACT,A,N) A(7)=CY
17.	CMA	00101111	A ← $\bar{A}$	A = -A	CALL COM(A,A,N)
18.	C C	00111111	CY ← $C\bar{Y}$	CY = -CY	CALL COM (CY,CY,1)
19.	STC	00110111	CY ← 1	CY = 1.0	CY =.TRUE.

4 BRANCH GROUP

O.Do.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
1.	JMP addr	11000011	ADDR ← ⟨B2⟩ ⟨B3⟩ Z ← M(ADDR) PC ← PC+1 ADDR ← ⟨B3⟩ W ← M(ADDR) PC ← B2	PC=(B(3)B(2))	CALL EQUAL(B2,9,16,ADDR),1,0) CALL LOAD(M,ADDR,Z,L*,M) CALL ZADD(PC,P,PC,D,M*) CALL EQ(B6,9,A ,ADDR),1,0) CALL LOAD(M,ADDR,W,L*,M) CALL EQUAL(WZ,9,24,PC,1,16)
2.	J CONDITION addr.	11CCC010	IF(CCC)GO TO 10 GO TO FETCH CYCLE 10 ADDR ⟨B2⟩ Z ← M(ADDR) PC ← PC+1 ADDR ⟨B3⟩ W ← M(ADDR) PC ← WZ	IF(CCC.EQ.1) GO TO 10 GO TO FETCH CYCLE 10 PC= B3 B2	IF(CCC, GO TO 10 GO TO FETCH CYCLE 10 CALL EQUAL(B2,9,16,ADDR),1,0) CALL LOAD(M,ADDR,Z,L*, M) CALL ZADD(PC,P,PC,D,M*) CALL EQUAL(B3,17,24,ADDR) 9,16 CALL LOAD(M,ADDR,W,L*,M) CALL E (WZ,PC,WM*)
3.	CALL addr	11C01101	ADDR ← PC SP ← M(ADDR) SP ← SP-1 M(SP) ← PCH SP ← SP-1 M(SP) ← PCL SP ← SP-2 (CONTINUE)	M(SP-1) = PCH M(SP-2)= PCL SP = SP-2 PC = B3 B2	CALL EQ(PC,ADDR,M*) CALL LOAD(M,ADDR,SP,L* ,M) CALL TCON(P,P,M*) CALL ZADD(SP,P,SP,D,M*) CALL LOAD(PCH,M,SP,L*,M) CALL TCON(P,P,M) CALL ZADD(SP,P,SP,D,M*)

S.No.	MNEMONIC	OP-CODE	REG. TRANSFER	ARITHMETIC MODEL	LOGICAL MODEL
			ADDR ← PC Z ← M(ADDR) PC ← PC+1 ADDR ← PC W ← M(ADDR) PC ← WZ		CALL LOAD(PCL,M,SP,L*,M) CALL INR(PC,ADDR,M*) CALL LOAD(M,ADDR,A,L*,M) CALL SADD(PC,P,PC,D,M*) CALL INR(PC,ADDR,M*) CALL LOAD(M,ADDR,W,L*,M) CALL BR(WZ,PC,M*)
4.	CONDITIONAL CALL addr.	11CCC100	IF(CCC) GO TO 10 GO TO FETCH CYCLE 10 ADDR ← PC SP ← M(ADDR) SP ← SP-1 M(SP) ← PCH SP ← SP-1 M(SP) ← PCL SP ← SP-2 ADDR ← PC Z ← M(ADDR) PC ← PC+1 ADDR ← PC W ← M(ADDR) PC ← WZ	IF (CCC=④) GO TO 10 GO TO FETCH CYCLE 10 M(SP-1) = PCH M(SP-2) = PCL SP = SP-2 PC = (B3)(B2)	IF(CCC) GO TO 10 GO TO FETCH CYCLE 10 CALL BR(PC,ADDR,M*) CALL LOAD(M,ADDR,SP,L*,M) CALL LOAD(PCH,M,SP,L*,M) CALL TCAI(P,P,M*) CALL SADD(SP,P,SP,D,M*) CALL LOAD(PCL,M,SP,L,N*) CALL TCAI(P,P,M*) CALL SADD(SP,P,SP,D,M*) CALL INR(PC,ADDR,M*) CALL LOAD(M,ADDR,Z,L*,M) CALL SADD(PC,P,PC,D,M*) CALL INR(PC,ADDR,M*) CALL LOAD(M,ADDR,W,L*,M) CALL BR(WZ,PC,M*)

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ALGORITHMIC MODEL	LOGICAL MODEL
5.	RSP	11601001	ADDR ← SP PCL ← M(ADDR) SP ← SP+1 ADDR ← SP PCH ← M(ADDR) SP ← SP+2	PCL = M(SP) PCH = M((SP)+1) SP = SP+2	CALL ER(SP,ADDR,16) CALL LOAD(M,ADDR,PCL,L*,M) CALL SADD(SP,P,SP,D,M) CALL ER(SP,ADDR,16) CALL LOAD(M,ADDR,PCH,L*,M) CALL SADD(SP,P,SP,D,M)

6.	R Condition	11CCC000	IF(CCC) GO TO 10
			GO TO FETCH CYCLE
			10 ADDR ← SP
			PCL ← M(ADDR)
			SP ← SP+1
			ADDR ← SP
			PCH ← M(ADDR)
			SP ← SP+2
				IF(CCC.EQ.1.0) GO TO 10
				ELSE
				GO TO FETCH CYCLE
				10 PCL = M(SP)
				P(Clo(SP+1))
				SP=SP+2
					IF(CCC) GO TO 10
					GO TO FETCH CYCLE
					10 CALL ER(SP,ADDR,16)
					CALL LOAD(M,ADDR,PCL,L*,M)
					CALL SADD(SP,P,SP,D,M)
					CALL ER(SP,ADDR,16)
					CALL LOAD(M,ADDR,PCH,L*,M)
					CALL SADD(SP,P,SP,D,M)

7.	PCHL	11101001	PCH ← H PCL ← L	PCH = H PCL = L	CALL EQUAL(H,1,8,PCH,9,16) CALL EQUAL(L,1,8,PCL,1,8)

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
3.	POP r²	11rr0001	ADDR ← SP r1 ← M(ADDR) SP ← SP+1 ADDR ← SP rh ← M(ADDR) SP ← SP+2 PC ← PC+1	r1 = M(SP) rh = M(SP+1) SP = SP+2	CALL DR(SP ADDR MV) CALL LODA (M,ADDR,r1,LV,M) CALL TADD(SP,P,SP,D,M1) CALL DR (SP,ADDR,MV) CALL LODA(M,ADDR,rh,LV,M) CALL TADD(SP,P,SP,D,MV) CALL TADD(PC,P,PC,D,MV)
4.	POP PSW	11110001	ADDR ← SP SR ← M(ADDR) SP ← SP+1 ADDR ← SP A ← M(ADDR) PC ← PC+1	CY = M(SP)$_0$ P = M(SP)$_1$ AC = M(SP)$_4$ Z = M(SP)$_6$ S = M(SP)$_7$ A = M(SP+1) SP = SP+2	CALL DR(SP,ADDR,MV) CALL LODA(M,ADDR,SR,LV,M) CALL TADD(SP,P,SP,D,MV) CALL DR(SP,ADDR,MV) CALL LODA(M,ADDR,A,LV,M) CALL TADD(PC,P,PC,D,MV)
5.	XTHL	11100011	ADDR ← SP C ← M(ADDR) SP ← SP+1 ADDR ← SP B ← M(ADDR) SP ← SP+1 DDR ← L M(SP) ← DDR SP ← SP+1 DDR ← H M(SP) ← DDR DDR ← C	L M(SP) H M(SP+1)	CALL DR(SP,ADDR,MV) CALL LODA(M,ADDR,C,LV,M) CALL TADD(SP,P,SP,D,16) CALL DR(SP,ADDR,16) CALL LODA(M,ADDR,B,LV,M) CALL TADD(SP,P,SP,D,16) CALL DR(L,DDR,M) CALL LODA(DDR,M,SP,LV,M) CALL TADD(SP,P,SP,D,16) CALL DR(H,DDR,M)

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARCHITECTURAL MODEL	LOGICAL MODEL
5.	RET	11c01c01	ADDR ← SP PCL ← M(ADDR) SP ← SP+1 ADDR ← SP PCH ← M(ADDR) SP ← SP+2	PCL = M(SP) PCH = M((SP)+1) SP = SP+2	CALL RD(SP,ADDR,16) CALL MOVE(M,ADDR,PCL,L*,M) CALL SADD(SP,P,SP,D,M*) CALL RD(SP,ADDR,16) CALL MOVE(M,ADDR,PCH,L*,M) CALL SADD(SP,P,SP,D,M*)
6.	R Condition	11cccc00	IF(CCC) GO TO 10 IF(CCC,EQ,1,0) GO TO FETCH CYCLE GO TO 10 10 ADDR ← SP PCL ← M(ADDR) SP ← SP+1 ADDR ← SP PCH ← M(ADDR) SP ← SP+2	ELSE GO TO FETCH CYCLE 10 PCL = M(SP) P(M=(SP+1)) SP=SP+2	IF(CCC) GO TO 10 GO TO FETCH CYCLE 10 CALL RD(SP,ADDR,16) CALL MOVE(M,ADDR,PCL,L*,M) CALL SADD(SP,P,SP,D,M*) CALL RD(SP,ADDR,16) CALL MOVE(M,ADDR,PCH,L*,M) CALL SADD(SP,P,SP,D,M*)
7.	PCHL	11101001	PCH ← H PCL ← L	PCH = H PCL = L	CALL EQUAL(H,1,8,PCH,9,16) CALL EQUAL(L,1,8,PCL,1,8)

5 STACK, I/O, AND MACHINE CONTROL GROUP

S.No.	MNEMONIC	OP-CODE	REG. PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
1.	PUSH rP	11rr0101	SP ← SP-1 ADDR ← SP M(ADDR) ← rh SP ← SP-1 ADDR ← SP M(ADDR) ← rl PC ← PC+1	M(SP-1)=rh M(SP-2)=rl SP=SP-2	CALL CON(P,P,1*) CALL SADD(SP,P,SP,D,1*) CALL DI(SP,ADDR,16) CALL LOAD(rh,1,ADDR,L*,D) CALL CON(P,P,1*) CALL SADD(SP,P,SP,L,1*) CALL DI(SP,ADDR,16) CALL LOAD(rl,1,ADDR,L*,D) CALL SADD(PC,P,PC,D,1*)
2.	PUSH PSW	11110101	SP ← SP-1 ADDR ← SP M(ADDR) ← A SP ← SP-1 ADDR ← SP M(ADDR) ← FLAG BITS (FB) PC ← PC+1	M(SP-1)=A M(SP-2) =CY M(SP-2)1=1 M(SP-2)2=P M(SP-2)3=0 M(SP-2)4=AC M(SP-2)5=0 M(SP-2)6=Z M(SP-2)7=S SP=SP-2	CALL CON(P,P,1*) CALL SADD(SP,P,SP,D,1*) CALL DI(SP,ADDR,16) CALL LOAD(A,1,ADDR,L*,D) CALL SADD(PC,P,PC,D,1*) CALL CON(P,P,1*) CALL SADD(SP,P,SP,D,1*) CALL DI(SP,ADDR,16) CALL LOAD(FB,1,ADDR,L*,D)

FLAG BITS

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
3.	POP r1	11rr0001	ADDR ← SP r1 ← M(ADDR) SP ← SP+1 ADDR ← SP rh ← M(ADDR) SP ← SP+2 PC ← PC+1	r1 = M(SP) rh = M(SP+1) SP = SP+2	CALL RD(SP,ADDR,MY) CALL LODA (M,ADDR,r1,LY,M) CALL ZADD(SP,P,SP,D,MY) CALL RD(SP,ADDR,MY) CALL LODA(M,ADDR,rh,LY,M) CALL ZADD(SP,P,SP,D,MY) CALL ZADD(PC,P,PC,D,MY)
4.	POP PSW	11110001	ADDR ← SP SB ← M(ADDR) SP ← SP+1 ADDR ← SP A ← M(ADDR) PC ← PC+1	CY = M(SP)$_0$ P = M(SP)$_1$ AC = M(SP)$_4$ Z = M(SP)$_6$ S = M(SP)$_7$ A = M(SP+1) SP = SP+2	CALL RD(SP,ADDR,MY) CALL LODA(M,ADDR,SB,LY,M) CALL ZADD(SP,P,SP,D,MY) CALL RD(SP,ADDR,MY) CALL LODA(M,ADDR,A,LY,M) CALL ZADD(PC,P,PC,D,MY)
5.	XTHL	11100011	ADDR ← SP C ← M(ADDR) SP ← SP+1 ADDR ← SP B ← M(ADDR) SP ← SP+1 DDR ← L M(SP) ← DDR SP ← SP+1 DDR ← H M(SP) ← DDR DDR ← C	L M(SP) H M(SP+1)	CALL RD(SP,ADDR,MY) CALL LODA(M,ADDR,C,LY,M) CALL ZADD(SP,P,SP,D,16) CALL RD(SP,ADDR,16) CALL LODA(M,ADDR,B,LY,M) CALL ZADD(SP,P,SP,D,16) CALL RD(L,DDR,M) CALL LODA(DDR,M,SP,LY,M) CALL ZADD(SP,P,SP,D,16) CALL RD(H,DDR,M)

S.No.	MNEMONIC	OP-CODE	RTL PRODUCTION	ARITHMETIC MODEL	LOGICAL MODEL
			DDD ← B E ← DDD PC ← PC+1	L ⇌ M(SP) H ⇌ M(SP+1)	CALL LOAD (DDD,M,SP,L,N) CALL EQ(C,DDD,M) CALL EQ(DDD,L,N) CALL TADD(SP,P,SP,L,N) CALL EQ(B,DDD,M) CALLDD(DDD,M,N) CALL TADD(PC,P,PC,D,16)
6.	SPHL	11111001	HL ← H,L SP ← HL	SP=H,L	CALL EQUAL(H,L),HL,16) CALL EQ(HL,SP,16)
7.	IN Port	11011011	DDD ← ⟨B2⟩ B ← DDD A ← B	A= DATA	CALL EQUAL (B2,9,16,DDD,1,8) CALL EQ(DDD,A,N)
8.	OUT Port	11010011	M(ADD) ← ⟨B2⟩ DDD ← A	DATA=A	CALL LOAD (B2,M,ADD,L,N) CALL EQ(A,DDD,M)
9.	EI	11111011	INTE ← 1	INTE = 1.0	-
10.	DI	11110011	INTE ← 0	INTE = 0.0	-
11.	HLT	01110110	-	-	-
12.	NOP	00000000	-	-	-

The simulation of INTEL 8080 is best illustrated by the following example.

EXAMPLE - WRITE THE SIMULATION PROGRAM FOR THE ADDITION OF TWO 16 DIGIT-BCD NUMBERS NAMED AS NA AND NB.

It is assumed that the locations for NA and NB are known and the sum of NA and NB is stored in the memory locations previously assigned to NA.

The following allocations are made for the memory.

MEMORY	$200-207_{16}$	NA
	$208-20F_{16}$	NB
8-BIT REGISTERS WITHIN MICRO-PROCESSOR	D and E	LOCA
	H and L	LOCB
		IC
	C	

Once the locations of the variables have been specified, the program in machine language can be written down.

STEPS USED IN WRITING PROGRAM

The following sequence is to be followed for the program written by using 8080 instructions.

(1) Load D and E immediately with 100_{16}, the address of NA_1

(2) Load H and L immediately with 108_{16}, the address of NB_1.

(3) Load C with 8.

(4) Clear the carry flag by the exclusive-OR of A with A.

(5) Load A with contents of the address specified by D and E.

(6) Load A with carry to contents of the address specified by H and L.

(7) Decimal adjust accumulator.

(8) Store A in the location addressed by D and E.

(9) Increment the address stored in H and L.

(10) Increment the address stored in D and E.

(11) Decrement

(12) Jump to step 5 if zero flag is not set by step 11.

If the program is stored in ROM with the starting location 500_{16}, then the ROM will contain the binary information.

S.No.	Mnemonic	Machine Code	Location	Contents (Binary)	Instruction	Operation
1.	NOP	00010001	500	00010001	No-operation	Clear counter Reset the system
2.	LXID	00010001	501	00000000	E ← ⟨B2⟩	E ← 0
			502	00000001	D ← ⟨B3⟩	D ← 1
3.	LXIH	00100001	503	00100001	Load Immediate	-
			504	00001000	H ← ⟨B2⟩	L←08
			505	00000001	H ← ⟨B3⟩	H←01
4.	MVI C	00001110	506	00001110	C ← ⟨B2⟩	Load counter C ← 08
	-	-	507	00001000		
5.	XRA A	10101111	508	10101111	A ← A ⊕ A	Carry Flag ← 0
6.	LDAX D	00011010	509	00011010	A ← [(D)(E)]	A← [(D)(E)]
7.	ADC H	10001110	50A	10001110	A ← A+[(H)(L)]+ carry	A←A+[(H)(L)]+ carry

8.	DAA	00100111	507	00100111	Decimal Adjust accumulator	Decimal Adjust accumulator
9.	STAX D	00010010	50C	00010010	[(D)(E)]←A	[(D)(E)]←A
10.	INX H	00100011	50D	00100011	H,L←H,L+1	H,L←H, L+1
11.	INX D	00010011	50E	00010011	D,E←D,E+1	D,E←D,E+1
12.	DCRC	00001101	50F	00001101	C←C-1	C← C-1
13.	JNZ	11000010	510	11000010	IF(ZERO), PC←<B3><B2>	IF(ZERO)
14.			511	00001001	—	PC ← 509
15.	"	"	512	00000101	—	JUMP TO STEP3

Nineteen memory locations are required for the program. The location of each instruction must be catalogued in order to implement the JUMP instruction. The byte B2 and B3 at location 510, for example, refer to the address of a previous instruction.

The logical simulation program for the above machine language program is given below:

```
C     C   SIMULATION PROGRAMER FOR THE ADDITION OF
C         TWO SIXTEEN DIGIT NUMBERS
C         N = NO. OF BITS/WORD, M=NO. OF BITS/INSTRUCTION
C         NOS.ARE IN HEXADECIMAL CODE; INPUT IN BINARY
C         B2  = BYTE TWO;  B3  = BYTE THREE
C         CI= COUNTER; H,L ARE TWO 8-BIT REGISTERS
C         D,E ARE TWO 8-BIT REGISTER, H = REGISTER
C         PAIR OF 16-BIT; D = REGISTER PAIR OF 16-BIT
C         P= TEMP REGISTER OF 16-BIT
C         T= TEMP.REGISTER OF 8-BIT
```

```
      CALL EQ(WZ,DE,16)
      PRINT 60,PC,A,DE
      GO TO 50
C     LXI H
C     LOAD REGISTER PAIR HL IMMEDIATELY
      CALL EQUAL(B2,9,16,Z,1,8)
      CALL TADD(PC,P,PC,D,MM1)
      CALL EQUAL(B3,17,24,W,1,8)
      CALL EQUAL (W,1,8,WZ,1,8)
      CALL EQUAL(Z,1,8,WZ,9,16)
      CALL EQ(WZ,HL,16)
      PRINT 60,PC,A,HL
C     GO TO 50
C     MVI C
C     MOVE IMMEDIATE COUNTER
      CALL EQUAL(B2,9,16,DDB,1,8)
      CALL EQUAL(DDB,1,8,DR,1,8)
      PRINT 70,PC,A,C
      GO TO 50
C     XRA  A
C     EXCLUSIVE OR WITH A
      CALL EQ(A,ACT,N)
      CALL EQ(SR,TMP,N)
      CALL EXOR(ACT,TMP,A,N)
      PRINT 75, PC,A
      GO TO 50
C     LDAX D
```

8.	DAA	00100111	90D	00100111	Decimal Adjust accumulator	Decimal Adjust accumulator
9.	STAX D	00010010	90C	00010010	[(D)(E)]←A	[(D)(E)]←A
10.	INX H	00100011	90D	00100011	H,L←H,L+1	H,L←H, L+1
11.	INX D	00010011	90E	00010011	D,E←D,E+1	D,E←D,E+1
12.	DCRC	00001101	90F	00001101	C←C-1	C← C-1
13.	JNZ	11000010	910	11000010	IF(ZERO), PC←<D3><B2>	IF(ZERO)
14.			911	00001001	—	PC ← 909
15.	-	-	912	00000101	—	JUMP TO STEP5

Nineteen memory locations are required for the program. The location of each instruction must be catalogued in order to implement the JUMP instruction. The byte B2 and B3 at location 910, for example, refer to the address of a previous instruction.

The logical simulation program for the above machine language program in given below:

```
C     C   SIMULATION PROGRAMER FOR THE ADDITION OF
C         TWO SIXTEEN DIGIT NUMBERS
C         N = NO. OF BITS/WORD, M=NO. OF BITS/INSTRUCTION
C         NOS.ARE IN HEXADECIMAL CODE, INPUT IN BINARY
C         B2  = BYTE TWO,  B3  = BYTE THREE
C         CI= COUNTER, H,L ARE TWO 8-BIT REGISTERS
C         D,E ARE TWO 8-BIT REGISTER, H = REGISTER
C         PAIR OF 16-BIT, D = REGISTER PAIR OF 16-BIT
C         P= TEMP REGISTER OF 16-BIT
C         T= TEMP.REGISTER OF 8-BIT
```

```
      INTEGER NA,NB
      LOGICAL*1 A(8),ACT(8),B(8),E(8),H(8),L(8),DE(16),F(8),
     $          HL(16),ADDB(16),PC(16),P(16),TMP(8),DDB(8)
     $          D,Z(4)
      READ(5,20)L,NT1,N,Z
      WRITE(6,11)
      CALL FILMEM(M,L,N)
      CALL PRTMEM(M,L,N)
      CALL COM(IR,IRC,N)
      CALL FALSE(P,P,NT1)
      P(NT1)=.TRUE.
      CALL FALSE(T,T,N)
      Z(N)= TRUE
C     FETCH CYCLE
   50 CALL EQ(PC,ADDB,16)
      CALL LODA(M,ADDB,IR,L**,N)
      CALL TADD(PC,P,PC,D,NM)
C     DECODING PART OF FETCHED INSTRUCTION
C     DECODE ALL TYPES OF INSTRUCTIONS
C     BIT-BY-BIT DECODING IS MADE,NOT GIVEN HERE.
C     EXECUTION OF DECODED INSTRUCTIONS
C     LXID
C     LOAD REGISTER PAIR D AND E IMMEDIATE
      CALL EQUAL(B2,9,16,Z,1,0)
      CALL TADD(PC,P,PC,D,NM)
      CALL EQUAL(B3,17,24,M,1,8)
      CALL EQUAL(M,1,8,MZ,1,8)
      CALL EQUAL(Z,1,8,MZ,9,16)
```

```
      CALL EQ(WZ,DE,16)
      PRINT 60,PC,A,DE
      GO TO 50
C     LXI H
C     LOAD REGISTER PAIR HL IMMEDIATELY
      CALL EQUAL(B2,9,16,Z,1,8)
      CALL TADD(PC,P,PC,D,MM)
      CALL EQUAL(B3,17,24,W,1,8)
      CALL EQUAL (W,1,8,WZ,1,8)
      CALL EQUAL(Z,1,8,WZ,9,16)
      CALL EQ(WZ,HL,16)
      PRINT 60,PC,A,HL
C     GO TO 50
C     MVI C
C     MOVE IMMEDIATE COUNTER
      CALL EQUAL(B2,9,16,DDB,1,8)
      CALL EQUAL(DDB,1,8,DR,1,8)
      PRINT 70,PC,A,C
      GO TO 50
C     XRA  A
C     EXCLUSIVE OR WITH A
      CALL EQ(A,ACT,N)
      CALL EQ(SR,TMP,N)
      CALL EXOR(ACT,TMP,A,N)
      PRINT 75, PC,A
      GO TO 50
C     LDAX D
```

```
C           LOAD ACCUMULATOR WITH THE REGISTER
C           PAIR DIRECTLY
            CALL EQ (DE,RP,MM)
            CALL EQ(RP,ADDR,MM)
            CALL LODA(M,ADDR,A,LE,N)
            PRINT 75, PC,A
            GO TO 50
C           ADC M
C           ADD MEMORY WITH CARRY
            CALL EQ(HL,ADDR,MM)
            CALL LODA(M,ADDR,TMP,LE,N)
            CALL EQ(A,ACT,N)
            CALL TADD(TMP,T,TMP,D,N)
            CALL TADD(ACT,TMP,A,D,N)
            PRINT 80,PC,HL,A
            GO TO 50
C           DAA
C           Z=0110
C           DECIMAL ADJUST ACCUMULATOR
            IF(S.GT.9) GO TO 9
          8 GO TO 55
          9 CALL TADD(Z,S,A,D,4)
         55 PRINT 80,PC,ML,S
            GO TO 50
C           STAX D
C           STORE ACCUMULATOR IN REGISTER PAIR D DIRECTLY
            CALL EQ(RP,ADDR,MM)
```

```
      CALL LODM(A,H,ADDD,LM,M)
      CALL PRIMEM(H,LM,M)
      GO TO 50
C     INX H
C     INCREASE REGISTER PAIR H
      CALL ED(HL,RP,MM)
      CALL TADD(RP,P,RP,D,MM)
      PRINT 60,PC,A,HL
      GO TO 50
C     INX D
C     INCREASE REGISTER PAIR D
      CALL ED(DE,RP,MM)
      CALL TADD(RP,P,RP,D,MM)
      PRINT 60,PC,A,DE
      GO TO 50
C     DCR C
C     DECREMENT COUNTER C BY ONE
      CALL ED(DR,TMP,M)
      CALL COM(T,T,M)
      CALL TADD(TMP,T,A,M)
      CALL ED(A,SR,M)
      PRINT 70,PC,A,C
      GO TO 50
C     JNZ
C     JUMP IF NOT ZERO
      IF(.NOT.(ZERO))GO TO 10
      ELSE GO TO 50
   10 CALL EQUAL(D2,9,16,ADDD,1,8)
```

```
 10   CALLLODA (M,ADDB,Z,L[B],N)
      CALL EQUAL(B3,17,24,ADDB,9,16)
      CALL LODA (M,ADDB,W,L[B],N)
      CALLE EQUAL (W,1,8,WZ,9,16)
      CALL EQUAL(Z,1,8,WZ,1,8)
      CALL EQ(WZ,PC,16)
      PRINT 60,PC,A,HL
      PRINT 110
      STOP
 20   FORMAT(3I4)
 11   FORMAT(10X 'L=  ',I4,10X 'M=  ',I4,10X'N= ',I4)
 60   FORMAT(10X'PC= ',16L2,10X, 'A= ',8L2,10X'DE= ',16L2)
 70   FORMAT(10X 'PC = ', 16L2, 5X'A= ',8L2,5X'C= ',8L2)
 75   FORMAT(10X 'PC=', 16L2, 10X'A= ',8L2)
 80   FORMAT(10X 'PC= ',16L2,10X' HL= ',16L2,5X,'A= ',8L2)
110   FORMAT(10X 'FINAL MEMORY CONTENTS')
      END
```

From the above simulation program, it is clear that in the absence of the microprocessor one can get the tested desired output of his designed digital system on an existing digital computer before its hardware implementation.

The advantage of software is that it is a nonrecurring cost item.

C O N C L U S I O N

In this dissertation a theoretical study has been made to simulate a digital systems on an existing computer. R.T.L. language is developed to permit description of the internal operation of digital system. R.T.L. is still an informal method that is in the formative stages, but is very useful in the development of complex digital systems. The R.T.L. modules reflect the hardware of the system in a sequential manner.

R.T.L. flow charts relieves the designer of many tedious details and repetitive operations of and eliminate logical inconsistencies, timing conflicts and logical errors. Knowing R.T.L. flow chart one can proceed to either simulate the system on a digital computer or to the hardware implementation side of the system. The simulation of a digital system is carried out either by Arithmetic model approach or by Logic model approach.

The Arithmetic model approach is simple and very concise. It takes less time to run a program. It does not give the details of the system. Wherever the details are not of much importance this approach is very useful.

The Logic model approach is not concise and it takes more time to run a program. There is one to one correspondence between the RTL flow chart and logical model of a given system. The logical model gives bit by bit details of the system.

The simulation methods becomes very useful when microprocessors are not available as a physical component. Nevertheless one is interested in studying the application of microprocessors in different fields of process controlled systems, in such a case once a machine language program has been developed for a particular applications, this machine language programme can be tested, if the microprocessor simulation program is available, does even in the absence of even in the case of non-availability of the components for testing prototypes on bread board research can progress in a field of application of microprocessors.

Further advancement is possible in simulation models and development of sub-routines. The two approaches, the Arithmetic model approach and the Logic model approach can be combined together to produce a third h HYBRID approach. We can call them ARITHMATICO-LOGIC MODEL. This model can take care off and this model shall have advantages of both systems. Subroutines then can be developed to suit this model.

REFERENCES

1. BREUER, MELVIN(ed.), 'Digital System Design Automation: Languages, Simulation and Data base', Woodland Hills, California:Computer Science Press, Inc., 1975.

2. CHU, YAOHAN, 'Computer Organization and Microprogramming' Englewood Cliffs, N.J.: Prentice-Hall,Inc., 1972.

3. DAVID, M.KLINE, 'Digital Computer Design' Franklin F.Kuo, Editor, Englewood Cliffs, N.J. Prentice-Hall, Inc., 1977.

4. DARDACCI, M.R., 'A Comparison of Register Transfer Languages for Describing Computers and Digital Systems', IEEE Trans.Computer, Vol.C-24, 1975, pp.133-50.

5. HAYNE, W.THOMAS, 'Fundamentals of Digital Systems Design', Englewood Cliffs, N.J., Prentice-Hall, Inc., 1975.

6. SOUZEK, BRANKO, 'Microprocessors and Microcomputers', New York, John Wiley and Sons, 1976.

7. HILBURN, L.JOHN and JULICH, M.PAUL, 'Microcomputers, Microprocessors: Hardware, Software, and Applications, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1976.

8. RAJARAMAN, V. and RADHAKRISHNAN, J. 'An Introduction to Digital Computer Design', Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1978

9. INTEL 8080 MANUAL, 1979

10. BELL,C.G., NEWELL, J.L., CARLOW, J., and WILLIAMS, P., 'The Description and Use of (RMI'S) register transfer modules', IEEE Trans.Computers C22, p.493-500, 1972.

```
      SUBROUTINE FILMEM(M,L,N,O)
C***  FILL FIRST L LOCATIONS IN MEMORY,M, FROM DATA CARDS
C     5 OCTAL NUMBERS/CARD, FORMAT 110
C     N=BITS /WORD
      IMPLICIT INTEGER(O)
      LOGICAL*1 M(L,N)
      WRITE(6,89)
      DO 20 J=1,L
 20   PRINT 93,J,(M(J,I),I=1,N)
      WRITE(6,85)
      RETURN
 85   FORMAT (000/000)
 89   FORMAT(20X,'MEMORY CONTENTS'/20X,'0----------------0'/)
 93   FORMAT(19X,I3,'  'OL2)
      END
      SUBROUTINE LOTO(L,O,N)
C***  L=LOGICAL, O=OCTAL, N=BITS/L
      INTEGER O
      LOGICAL*1 L(N)
C     ACCOUNT FOR SIGN
      J=0
      IF (.NOT.L(1)) GO TO 5
      CALL TCOM (L,N)
      J=1
C     CONVERSION PROPER
 5    O=0
      DO 2 I=1,N,3
      O=O*10
      IF(L(I+2))O=O+1
      IF(L(I+1)) O=O+2
 2    IF(L(I)) O=O+4
      IF(J.EQ.1) O=-O
      RETURN
      END
      SUBROUTINE TESTZE(A,B,N)
C***  IF N BIT NUMBER A=0, SINGLE BIT B=1.
      LOGICAL*1 A(N),B
      B=.TRUE.
      DO 10 I=1,N
 10   IF(A(I)) B=.FALSE.
      RETURN
      END
      SUBROUTINE TADD(X,Y,S,C,N)
      LOGICAL*1 C,CC,S(N),X(N),Y(N),P,Q,D
      C=.FALSE.
      DO 1 J=1,N
      I=N+1-J
      P=.NOT.X(I)
      Q=.NOT.Y(I)
      D=.NOT.C
      CC=(X(I).AND.C).OR.(X(I).AND.Y(I)).OR.(Y(I).AND.C)
      S(I)=(P.AND.Q.AND.C).OR.(P.AND.Y(I).AND.D).OR.(X(I).AND.Q.AND.D)
     $.OR.(X(I).AND.Y(I).AND.C)
 1    C=CC
      RETURN
      END
```

```
      SUBROUTINE ADD(X,Y,S,N)
C***  CREB COMP. ADD
      LOGICAL*1 X(N),Y(N),S(N),CC(12),C
      CALL TAND(X,Y,S,C,N)
      CALL FALSE(CC,N)
      CC(N)=C
      CALL TAND(S,CC,S,C,N)
      RETURN
      END
      SUBROUTINE EQUAL(D,L,N,A,J,K)
C***  EQUATE SUBSCRIPT TERMS
C     A(JTHRU K)=D(L THRU N) RESPECTIVELY
C     REQUIRED=K-J=N-L
      LOGICAL*1 A(K),D(N)
      N=N-L+1
      DO 10 I=1,N
      J1=J-1+I
      L1=L-1+I
   10 A(J1)=D(L1)
      RETURN
      END
      SUBROUTINE LODA(N,D,A,L,N)
C***  MEM.. N, LOCATION D IS LOADED IN REGISTER A
C     L=WORDS/ MEM.    N=BITS/ WORD
      LOGICAL*1 A(N),B(N), M(L,N)
      CALL LOTD (D,J,N)
      DO 10 I=1,N
   10 A(I)=M(J,I)
      RETURN
      END
      SUBROUTINE LODM(A,M,D,L,N)
C***  REGISTER A IS LOADED IN MEM.. M, LOCATION D
C     L=WORDS/ MEM.    N=BITS/WORD
      LOGICAL*1 A(N), B(N), M(L,N)
      CALL LOTD (D,J,N)
      DO 10 I=1,N
   10 M(J,I)=A(I)
      RETURN
      END
      SUBROUTINE TESTEQ(A,B,C,N)
C***  IF A=B, SINGLE BIT C=1
      COMMON I(900),E(900)
      LOGICAL*1 A(N),B(N),C,D,E
      CALL COM(A,D,N)
      CALL COM(B,E,N)
      CALL AND(D,E,E,N)
      CALL AND(A,B,D,N)
      CALL OR(D,E,D,N)
      CALL COM(D,D,N)
      CALL TESTZE(D,C,N)
      RETURN
      END
```

```
      SUBROUTINE PRTMEM (M,L,N)
C*** PRINT FIRST L LOCATION IN MEMORY
      LOGICAL*1 L1(50),M(L,N)
      DIMENSION O(50)
      READ(5,70) (O(I),I=1,L)
      DO 20 J=1,L
      CALL OTBL(O(J),L1,N)
      DO 20 I=1,N
   20 M(J,I)=L1(I)
      RETURN
   70 FORMAT(5I10)
      END
      SUBROUTINE OTOL(OO,L,N)
C**  OO=OCTAL, L=LOGICAL-T,F,
C     N=BITS/L, BIT1=SIGN
      INTEGER O, OH, OO
      LOGICAL*1 L(N)
      O=OO
C     CHECK FOR NEGATIVE
      J=0
      IF (O.GT. 0) GO TO 10
      O=-O
      J=1
C CONVERSION PROPER
   10 CALL FALSE (L,N)
      K=N-2
      DO 0 I=3,K,3
    2 OH=O-(O/10)*10
      O=O/10
      IF (OH.EQ.0) GO TO 0
      GO TO (0,4,3,4,3,4,3), OH
    3 L(N+1-I)=.TRUE.
    4 GO TO (0,5,5,6,6,6,5), OH
    5 L(N-1) =.TRUE.
    6 GO TO (0,8,8,7,7,7,7), OH
    7 L(N-1-I) =.TRUE.
    0 CONTINUE
      IF(J.NE.1) GO TO 11
      CALL TCOM (L,N)
   11 RETURN
      END
      SUBROUTINE SL1A,B,N)
C**** REGISTER A IS SHIFTED LEFT 1BIT,RESULT IN B,
C     N=BITS,F TO LOD,BIT N,
      LOGICAL*1 A(N),B(N)
      M=N-1
      DO 20 I=1,M
   20 B(I)=A(I+1)
      B(N)=.FALSE.
      RETURN
      END
      SUBROUTINE PRTMEM(M,L,N)
C**  PRINT FIST L LOCATION IN MEMORY
```

```
      SUBROUTINE FALSE(A,N)
      LOGICAL*1 A(N)
      DO 10 I=1,N
 10   A(I)=.FALSE.
       RETURN
      END
      SUBROUTINE COM(A,B,N)
C***  B=AO,N=BITS. A IS UNCHANGED
      LOGICAL*1 A(N), B(N)
      DO 10 I=1,N
  10  B(I)=.NOT. A(I)
      RETURN
      END
      SUBROUTINE SR(A,B,N)
C***  REGISTER A IS SHIFTED RIGHT 1BIT,RESULT IN B,
C     N=BITS,F TO MSB, DITI.
      LOGICAL*1 A(N),B(N)
      M=N-1
      DO 20 I=1,M
 20   B(N-I+1)=A(N-I)
      B(1)=.FALSE.
      RETURN
      END
      SUBROUTINE EQ(B,A,N)
C***  A(I)=B(I) FOR ALL I. B IS UNCHANGED
      LOGICAL*1 A(N),B(N)
      DO 10 I=1,N
  10  A(I)=B(I)
      RETURN
      END
      SUBROUTINE FILMD(M,L,N)
C  C  FILL FIRSTL LOCATIONS IN MEMORY,M,FROM DATA CARDS
C     DDECIMAL NO,FROJ EACH CARD,FORMAT I10
C     N=DITS/CARD
      IMPLICIT INTEGER(D)
      LOGICAL*1 L1(90),M(L,N)
      DIMENSION D(20)
      READ(5,60)(D(I),I=1,L)
      DO 20 J=1,L
      CALL DTOL(D(J),L1,N)
      DO 20 I=1,N
 20   M(J,I)=L1(I)
 60   FORMAT(5I10)
      RETURN
      END
```

```
      SUBROUTINE TCOM(A,N)
C***  TWO'S COMPLEMENT, A=A'+1. N=BITS
      LOGICAL*1 A(N),J(12)
      CALL COM (A,A,N)
      CALL FALSE (J,N)
      J(N)=.TRUE.
      CALL TADD(A,J,A,C,N)
      RETURN
      END
      SUBROUTINE LOTO(L,O,N)
C***  L=LOGICAL, O=DECIMAL, N=BITS/WORD
      INTEGER O
      LOGICAL*1 L(N)
C     ACCOUNT FOR SIGN
      J=0
      IF (.NOT.L(1))GO TO 5
      CALL TCOM (L,N)
      J=1
C     CONVERSION PROPER
 5    O=0
      DO 2 I=1,N,3
      O=O*8
      IF(L(I+2) ) O=O+1
      IF(L(I+1)) O=O+2
 2    IF(L(I) ) O=O+4
      IF(J.EQ.1) O=-O
      RETURN
      END
      SUBROUTINE OR(B,C,A,N)
C***  C=A OR B. N=BITS
      LOGICAL*1 A(N),B(N),C(N)
      DO 10 I=1,N
 10   C(I)=B(I).OR.A(I)
      RETURN
      END
      SUBROUTINE AND(A,B,C,N)
C***  A AND B ARE ADDED WITH RESULT STORED IN C
      LOGICAL*1 A(N),B(N),C(N)
      DO 10 I=1,N
  10   C(I)=A(I) .AND. B(I)
      RETURN
      END
```