UNIT - I

Digital Computers

Introduction, Block diagram of Digital Computer, Definition of Computer Organization, Computer Design and Computer Architecture.

Register Transfer Language and Micro operations

Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design

Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT - I - Part - I

DIGITAL COMPUTERS

Introduction:-

The digital computer is a digital system that performs various computational tasks. Digital computers use the binary number systems which has two digits: 0 and 1. A binary digit is called a bit. A computer system is sometimes subdivided into two functional entities:

⇒ Hardware
⇒ Software

The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.

Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks. A sequence of instructions for the computer is called a program. The data that are manipulated by the program constitute the data base.

The hardware of the computer is usually divided into three major parts.

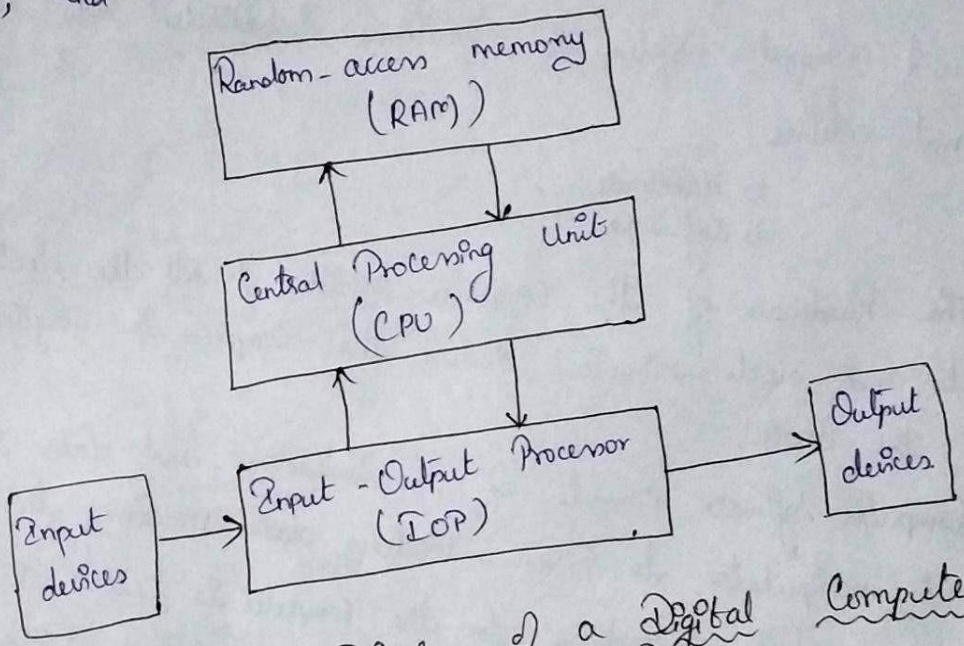
The Central processing unit [CPU] contains an arithmetic and logic unit for manipulating data, a number of registers for storing data and control circuits for fetching and executing instructions.

The memory of a computer contains storage for instructions and data. It is called a random-access memory. [RAM]

RAM because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.

The Input and Output Processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.

The input and output devices connected to the computer includes keyboards, printers, terminals, magnetic disks devices, and other communication devices.



Block Diagram of a Digital Computer.

Functional unit of Digital Computer :-

A Computer consists of five functionally independent main parts :

⇒ Input
⇒ memory
⇒ Arithmetic & Logic
⇒ Output
⇒ Control unit

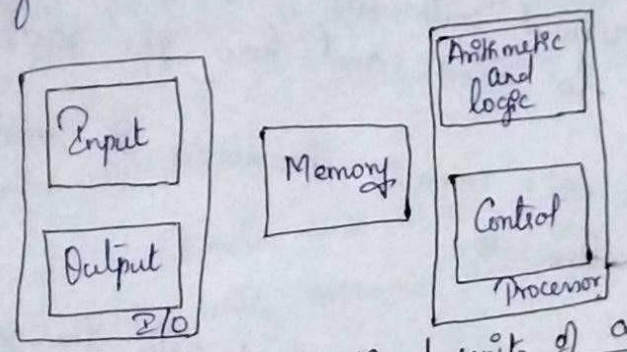
The Input unit accepts coded information from humans operators from an electromechanical device such as keyboard or from other computers over digital communication lines.

The information either stored in computers memory for future reference or immediately used by ALU to perform desired operation.

The processing steps are determined by a program stored in memory.

Finally the results are sent back to the outside world through the output unit.

All of these operations are coordinated by the control unit.



Basic functional units of a computer

Computer Organization:-

Computer Organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organization structure to verify that the computer parts operate as intended.

Computer Organization refers to the operational units and their interconnections that realize the architectural specifications.

Example:

Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals, and the memory technology used.

- Design of the components and functional blocks using which computer systems are built.

Computer Design:-

Computer Design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.

It is concerned with the determination of what hardware should be used and how the parts should be connected.

This aspect of computer hardware is sometimes referred to as computer implementation.

Computer Architecture:-

Computer Architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories and structuring them together into a computer system.

Computer Architecture:- Refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.

Example:

Architectural attributes include the instructions set, the number of bits used to represent various data types (eg. numbers, characters), I/O mechanisms and techniques for addressing memory.

Two basic types of computer architecture are Von-Neumann architecture and Harvard architecture.

Von-Neumann Architecture:-

In a Von-neumann architecture, the same memory and bus are connected used to store both data and instructions that form the program.

Since it cannot access program memory and data memory simultaneously, the Von-neumann architecture is susceptible to bottlenecks and system performance is affected.

The structure of a computer system as being composed of the following components.

1) The central arithmetic unit, which today is called the arithmetic logic unit (ALU). This unit performs the computer's computational and logical functions.

2) Memory; More specifically, the computer's main, or fast, memory such as random access memory (RAM);

3) A control unit that directs other components of the computer to perform certain actions, such as directing the fetching of data or instruction from memory to be the processed by the ALU and

4) man-machine interfaces; (ie) input and output devices, such as keyboard for input and display monitor for output.

Harvard Architecture:-

The Harvard architecture stores machine instructions and data in separate memory units that are connected by different

busses.

In this case, there are at least two memory address spaces to work with. So there is a memory register for machine instructions and another memory register for data.

Computers designed with the Harvard architecture are able to run a program and access data independently and therefore simultaneously.

Harvard architecture has a strict separation between data and code. Thus, Harvard architecture is more complicated but separate pipelines remove the bottleneck that Von-Neumann creates.

UNIT-2 - PART-II

REGISTER TRANSFER LANGUAGE AND MICRO OPERATIONS

Register Transfer Language:

The Operations executed on data stored in registers are microoperations.

A microoperation is an elementary operation performed on the information stored in one or more registers.

The result of the operation may replace the previous binary information of a register or may be transfered to another register.

Examples:

Shift, count, clear and load.

The internal hardware organization of a digital computer is best defined by specifying

1). The set of registers it contains and their functions

2). The sequence of micro-operations performed on the binary information stored.

3) The control that initiates the sequence of micro-operations.

The symbolic notations used to describe the micro-operation transfers among registers is called a Register transfer language.

A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.

Register Transfers:-

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

Four registers are essential to instruction execution:

Program Counter (PC):
Contains the address of an instruction to be fetched.

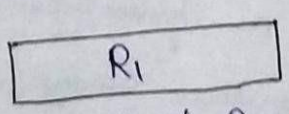
Instruction Register (IR):
Contains the instruction most recently fetched.

Memory Address Register (MAR):
Contains the address of a location in memory.

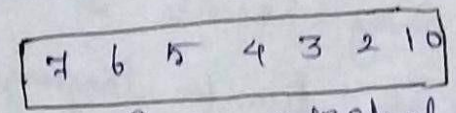
Memory Buffer Register (MBR):
Contains a word of data to be written to memory or the word most recently read.

Example:
The register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR.
Other designations for registers are PC (for Program Counter), IR (for Instruction Register) and R_1 (for Processor Register).

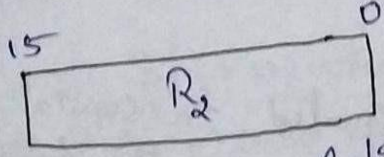
The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.



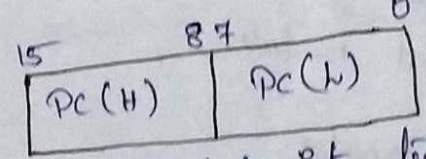
(a) Register R



(b) Showing individual bits



(c) Numbering of bits



(d) Divided into two parts

Block Diagram of Register

Information transfer from one register to another is designed in symbolic form by means of a replacement operator.

The statement

$$R_2 \leftarrow R_1$$

denotes a transfer of the content of register R_1 into register R_2.

Block diagram explanation:-

- In block diagram, the representation of registers in block diagram form. The most common way to represent a register is by a rectangular box with the name of the register inside of diagram (a).
- The individual bits can be distinguished as in diagram (b).
- The numbering of bits in a 16-bit register can be marked on top of the box as shown in diagram (c).
- A 16-bit register is partitioned into two parts in diagram (d).
- Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte).
- The name of the 16-bit register is PC.
- The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.

It designates a replacement of the ~~source~~ content of R_2 by the content of R_1.

By definition, the content of the source register R_1 does not change after the transfer.

Control function:-

If we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if then statement

$$\text{If } (P=1) \text{ then } (R_2 \leftarrow R_1).$$

where P is a control signal generated in the control ~~statement~~ section. It is sometimes convenient to separate the control variables

- Reg Register R_2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

Timing Diagram Explanation:-

- P is activated in the control section by the rising edge of a clock pulse at time t.
- The next positive transition of the clock pulse at time t+1, finds the load input active and the data inputs of R_2 are then loaded into the register in parallel.
- P may go back to 0 at time t+1. otherwise, the transfer will occur with every clock pulse transition while P remains active.
- Note that the clock is not included as a variable in the register transfer statements.
- It is assumed that all transfers occur during a clock edge transition.
- Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t+1.

Basic Symbol for Register Transfer

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R_2
Parentheses ( )	Denotes a part of a register	$R_2(0-7)$, $R_2(L)$
Arrow $\leftarrow$	Denotes transfer of information	$R_2 \leftarrow R_1$
comma,	Separates two microoperations	$R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$

Table Explanation:-

- The basic symbols of the register transfer notation are listed in above table.

- Registers are denoted by capital letters and numerals may follow the letters.
- Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register.
- The arrow denotes a transfer of information and the direction of transfer.
- A comma is used to separate two or more operations that are executed at the same time.
- The statement $T: R_2 \leftarrow R_1, R_1 \leftarrow R_2$ denotes an operation that exchanges the contents of two registers during one common clock pulse provided that $T = 1$.
- This simultaneous operation is possible with registers that have edge triggered flip-flops.

Designate computer ~~flip-flops~~ registers by capital letters. to denote its function.

The register that holds an address for the memory unit is called MAR.

The program counter register is called PC.

IR is the instruction register and R_1 is a processor register.

The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1.

Bus And Memory Transfers:-

Bus Transfer:

Digital computer has many registers and paths must be provided to transfer information from one register to another.

The number of wires will be ~~excessive~~ excessive if separate lines are used between each register and all other registers in the system.

A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system.

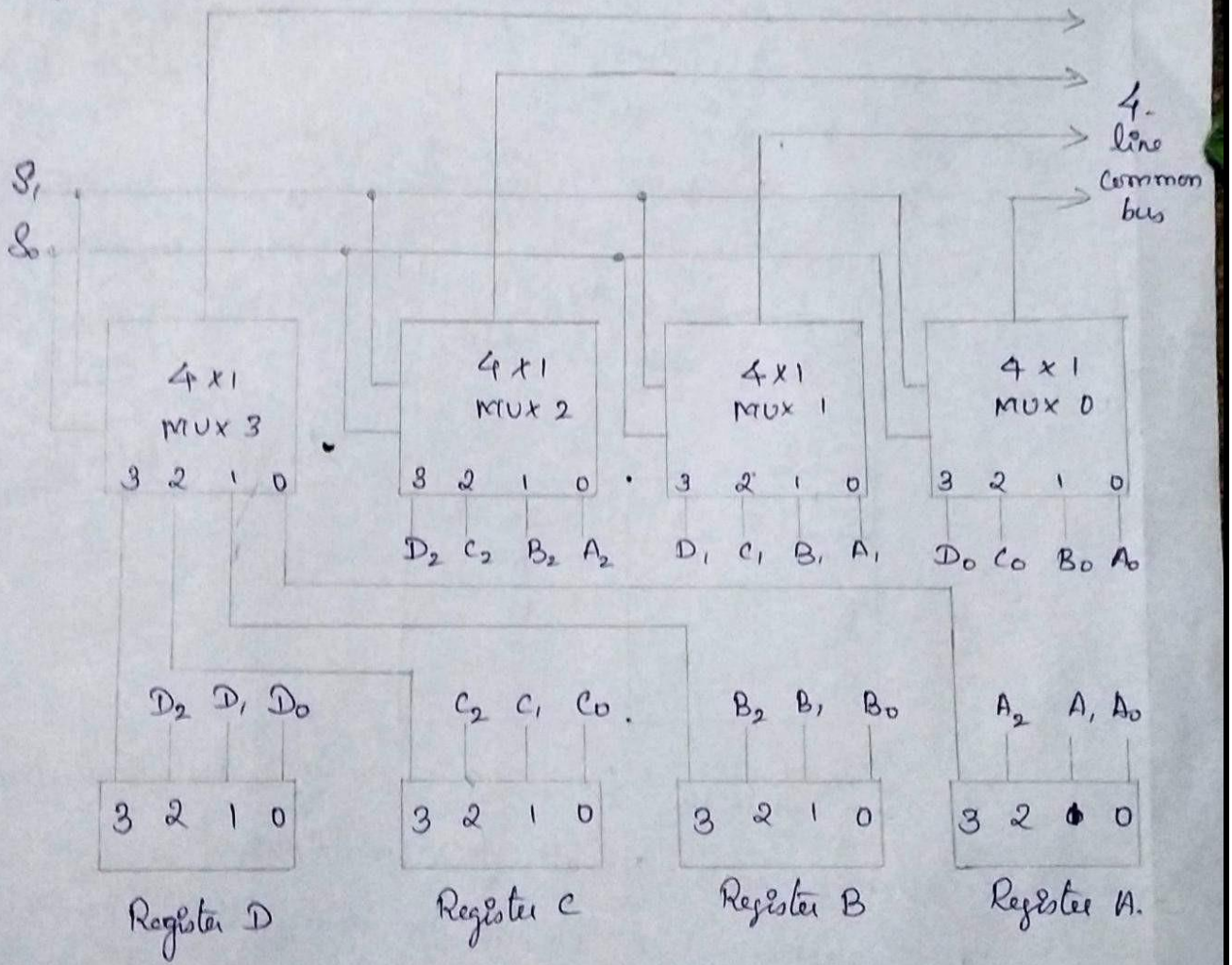
A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.

Control signals determine which register is selected by the bus during each particular register transfer.

Common Bus System Using Multiplexers:

A common bus system is with multiplexers.

The multiplexers select the source register whose binary information is then placed on the bus.



Bus System for four Registers

• The Construction of a bus system for four registers is shown in above figure.

• Each Register has four bits, numbered 0 through 3.

• The bus consists of four 4x1 multiplexers each having four data inputs, 0 through 3 and two selection inputs S_1 and S_0.

In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers for example: Output 1 of Register A is connected to input 0 of MUX 1. because this input is labeled A_1.

The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.

Thus MUX 0 multiplexers the four 0 bits of the registers, MUX 1 multiplexers the four 1 bits of the registers, and similarly for the other two bits.

Function Table for bus

S_1	S_0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers.

The selection lines choose the four bits of one register and transfer them into the four-line common bus.

Unit-1: REGISTER TRANSFER AND MICROOPERATIONS

CONTENTS:

- Register Transfer Language
- Register Transfer
- Bus And Memory Transfers
- Types of Micro-operations
- Arithmetic Micro-operations
- Logic Micro-operations
- Shift Micro-operations
- Arithmetic Logic Shift Unit

BASIC DEFINITIONS:

- A digital system is an interconnection of digital hardware modules.
- The modules are registers, decoders, arithmetic elements, and control logic.
- The various modules are interconnected with common data and control paths to form a digital computer system.
- Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them.
- The operations executed on data stored in registers are called *microoperations*.
- A *microoperation* is an elementary operation performed on the information stored in one or more registers.
- The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- Examples of microoperations are shift, count, clear, and load.
- The internal hardware organization of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.
2. The sequence of microoperations performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

REGISTER TRANSFER LANGUAGE:

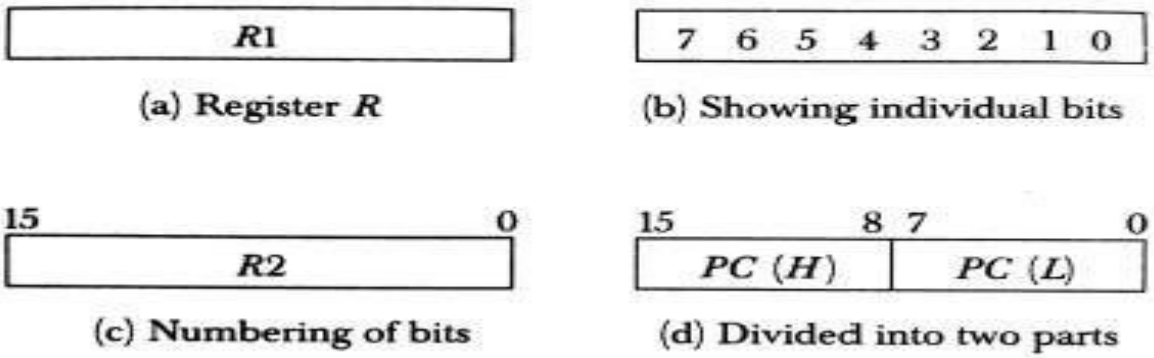
- The symbolic notation used to describe the micro-operation transfer among registers is called RTL (Register Transfer Language).
- The use of ***symbols*** instead of a ***narrative explanation*** provides an organized and concise manner for listing the micro-operation sequences in registers and the control functions that initiate them.

- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- It is a convenient tool for describing the internal organization of digital computers in concise and precise manner.

Registers:

- Computer registers are designated by upper case letters (and optionally followed by digits or letters) to denote the function of the register.
- For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name **MAR**.
- Other designations for registers are ***PC*** (for program counter), ***IR*** (for instruction register, and ***R1*** (for processor register).
- The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.
- Figure 4-1 shows the representation of registers in block diagram form.

Figure 4-1 Block diagram of register.



- The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig. 4-1(a).
- The individual bits can be distinguished as in (b).
- The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c).
- 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol *H* (for high byte).
- The name of the 16-bit register is *PC.* The symbol *PC (0-7)* or *PC (L)* refers to the low-order byteand PC (8-15) or *PC (H)* to the high-order byte**.**

Register Transfer:

- Information transfer from one register to another is designated in symbolic form by means of a *replacement operator*.
- The statement **R2← R1** denotes a transfer of the content of register R1 into register R2.
- It designates a replacement of the content of R2 by the content of R1.
- By definition, the content of the source register R 1 does not change after the transfer.
- If we want the transfer to occur only under a predetermined control condition then it can be shown by an if-then statement.

if (P=1) then R2← R1

- P is the control signal generated by a control section.
- We can separate the control variables from the register transfer operation by specifying a ***Control Function.***
- Control function is a Boolean variable that is equal to 0 or 1.
- control function is included in the statement as

P: R2← R1

- Control condition is terminated by a colon implies transfer operation be executed by the hardware only if P=1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
- Figure 4-2 shows the block diagram that depicts the transfer from R1 to R2.

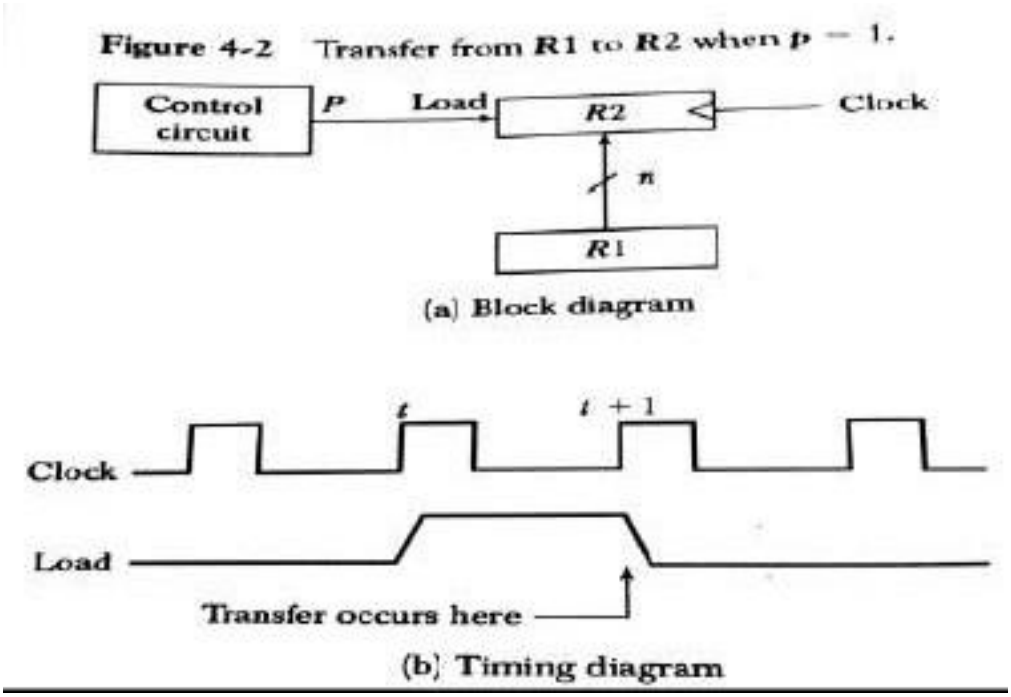


Figure 4-2 Transfer from R1 to R2 when $p = 1$.

(a) Block diagram

(b) Timing diagram

- The n outputs of register R1 are connected to the n inputs of register R2.
- The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
- Register R2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
- As shown in the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time *t.*
- The next positive transition of the clock at time $t + 1$ finds the load input active and the data inputs of R2 are then loaded into the register in parallel.
- P may go back to 0 at time $t+1$; otherwise, the transfer will occur with every clock pulse transition while P remains active.
- Even though the control condition such as P becomes active just after time *t,* the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time

t +1.

- The basic symbols of the register transfer notation are listed in below table

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow <--	Denotes transfer of information	R2 <-- R1
Comma ,	Separates two microoperations	R2 <-- R1, R1 <-- R2

- A comma is used to separate two or more operations that are executed at the same time**.**
- The statement

  T : R2← R1, R1← R2 (exchange operation)

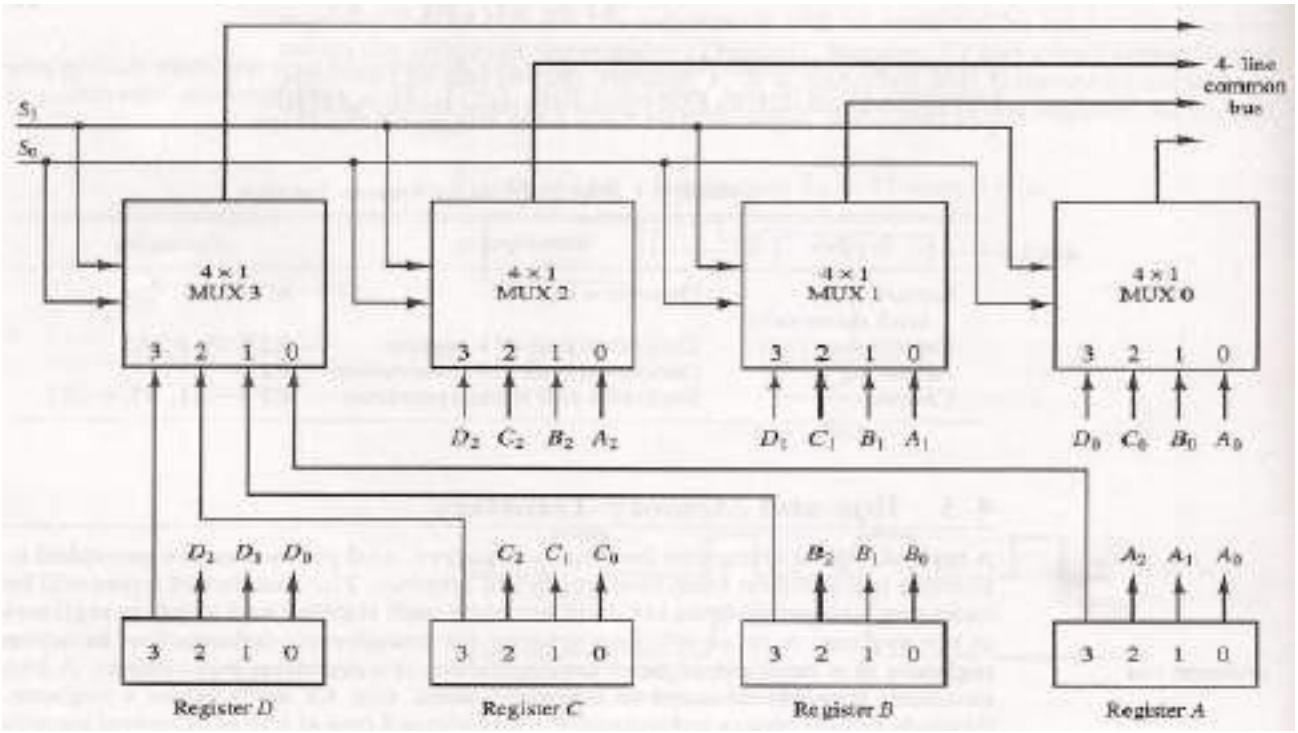
  denotes an operation that exchanges the contents of two rgisters during one common clock pulse provided that T=1.

Bus and Memory Transfers:

- A more efficient scheme for transferring information between registers in ***a multiple-register configuration*** is a ***Common Bus System.***
- A common bus consists of a set of common lines, one for each bit of a register.
- Control signals determine which register is selected by the bus during each particular register transfer.
- Different ways of constructing a Common Bus System
  - ✓ Using Multiplexers
  - ✓ Using Tri-state Buffers

Common bus system is with multiplexers:

- The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in below Figure.



- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0.
- For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labelled A_1.
- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.
- Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of theregisters, and similarly for the other two bits.
- The two selection lines Si and So are connected to the selection inputs of all four multiplexers.
- The selection lines choose the four bits of one register and transfer them into the four-line common bus.
- When $S_1S_0 = 00$, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
- This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.
- Similarly, register *B* is selected if $S_1S_0 = 01$, and so on.
- Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

S_1	S_0	Register selected
0	0	*A*
0	1	*B*
1	0	*C*
1	1	*D*

- In general a bus system has
  - ✓ multiplex "k" Registers

- ✓ each register of "n" bits
- ✓ to produce "n-line bus"
- ✓ no. of multiplexers required = n
- ✓ size of each multiplexer = k x 1

- When the bus is includes in the statement, the register transfer is symbolized as follows:

BUS← C, R1← BUS

- The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

R1← C

Three-State Bus Buffers:

- A bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third state is a *high-impedance state.*
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.
- The graphic symbol of a three-state buffer gate is shown in Fig. 4-4.

Figure 4-4 Graphic symbols for three-state buffer.

- It is distinguished from a normal buffer by having both a normal input and a control input.
- The control input determines the output state. When the control input is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.
- When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input**.**
- The construction of a bus system with three-state buffers is shown in Fig. 4

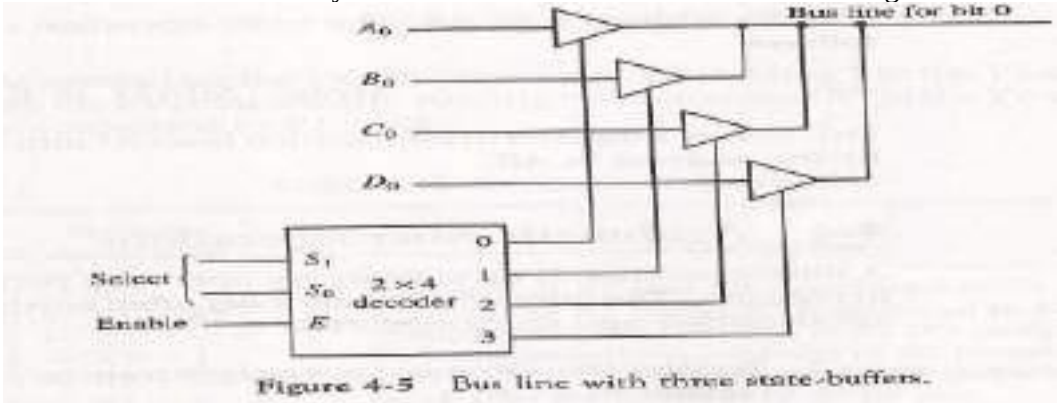


Figure 4-5 Bus line with three state-buffers.

- The outputs of four buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.
- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

Memory Transfer:

- The transfer of information from a memory word to the outside environment is called a *read* operation.
- The transfer of new information to be stored into the memory is called a *write* operation.
- A memory word will be symbolized by the letter M.
- The particular memory word among the many available is selected by the memory address during the transfer.
- It is necessary to specify the address of M when writing memory transfer operations.
- This will be done by enclosing the address in square brackets following the letter M.
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data are transferred to another register, called the data register, symbolized by DR.
- The read operation can be stated as follows:

Read: DR<- M [AR]

- This causes a transfer of information into DR from the memory word M selected by the address in AR.
- The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.
- The write operation can be stated as follows:

Write: M [AR] <- R1

Types of Micro-operations:

- *Register Transfer Micro-operations:* Transfer binary information from one register to another.
- *Arithmetic Micro-operations:* Perform arithmetic operation on numeric data stored in registers.
- *Logical Micro-operations:* Perform bit manipulation operations on data stored in registers.
- *Shift Micro-operations:* Perform shift operations on data stored in registers.

- Register Transfer Micro-operation doesn't change the information content when the binary information moves from source register to destination register.

- Other three types of micro-operations change the information change the information content during the transfer.

Arithmetic Micro-operations:

- The basic arithmetic micro-operations are
  - Addition
  - Subtraction
  - Increment
  - Decrement
  - Shift
- The arithmetic Micro-operation defined by the statement below specifies the add micro-operation.

$$R3 \leftarrow R1 + R2$$

- It states that the contents of R1 are added to contents of R2 and sum is transferred to R3.
- To implement this statement hardware requires 3 registers and digital component that performs addition
- Subtraction is most often implemented through complementation and addition.
- The subtract operation is specified by the following statement

$$R3 \leftarrow R1 + \overline{R2} + 1$$

- instead of minus operator, we can write as
- $\overline{R2}$ is the symbol for the 1's complement of R2
- Adding 1 to 1's complement produces 2's complement
- Adding the contents of *R1* to the 2's complement of R2 is equivalent to R1-R2.

Binary Adder:

- Digital circuit that forms the arithmetic sum of 2 bits and the previous carry is called **FULL ADDER.**
- Digital circuit that generates the arithmetic sum of 2 binary numbers of any lengths is called **BINARY ADDER.**
- Figure 4-6 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.

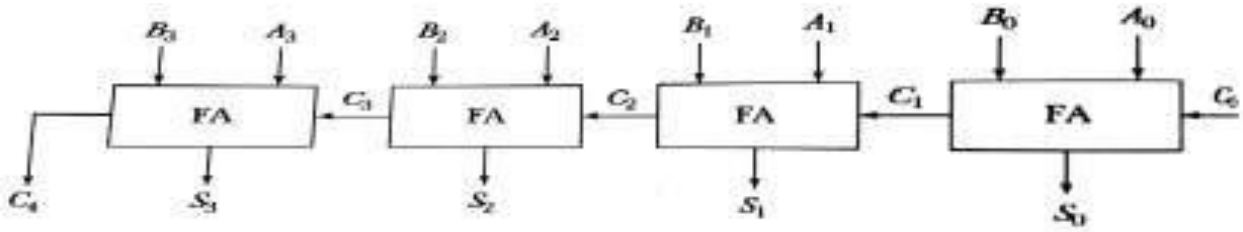
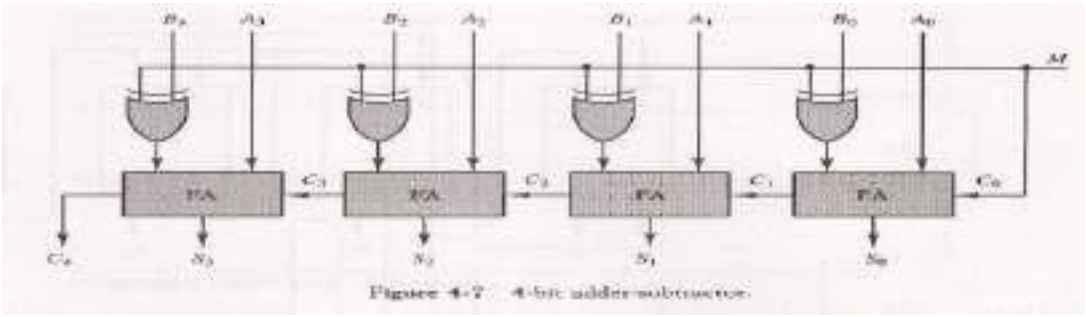


Figure 4-6 4-bit binary adder.

- The augends bits of A and the addend bits of *B* are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders. The input carry to the binary adder is Co and the output carry is C4. The *S* outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.

Binary Adder – Subtractor:

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- A 4-bit adder-subtractor circuit is shown in Fig. 4-7.



- The mode input M controls the operation. When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.
- Each exclusive-OR gate receives input M and one of the inputs of B
- When M = 0, we have *B* xor 0 = B. The full-adders receive the value of *B*, the input carry is 0, andthe circuit performs A plus B.
- When M = 1, we have *B xor 1 = B'* and Co = 1.
- The *B* inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of *B*.

Binary Incrementer:

- The increment microoperation adds one to a number in a register.
- For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.
- This can be accomplished by means of half-adders connected in cascade.
- The diagram of a 4-bit 'combinational circuit incrementer is shown in Fig. 4-8.

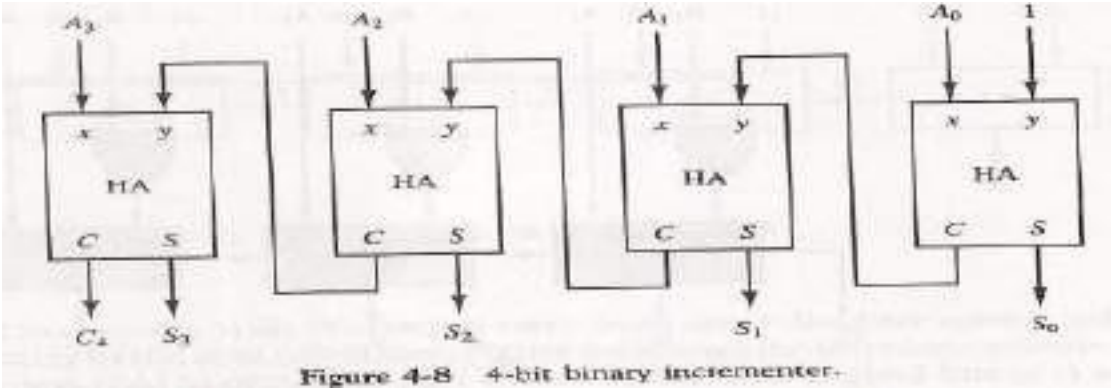


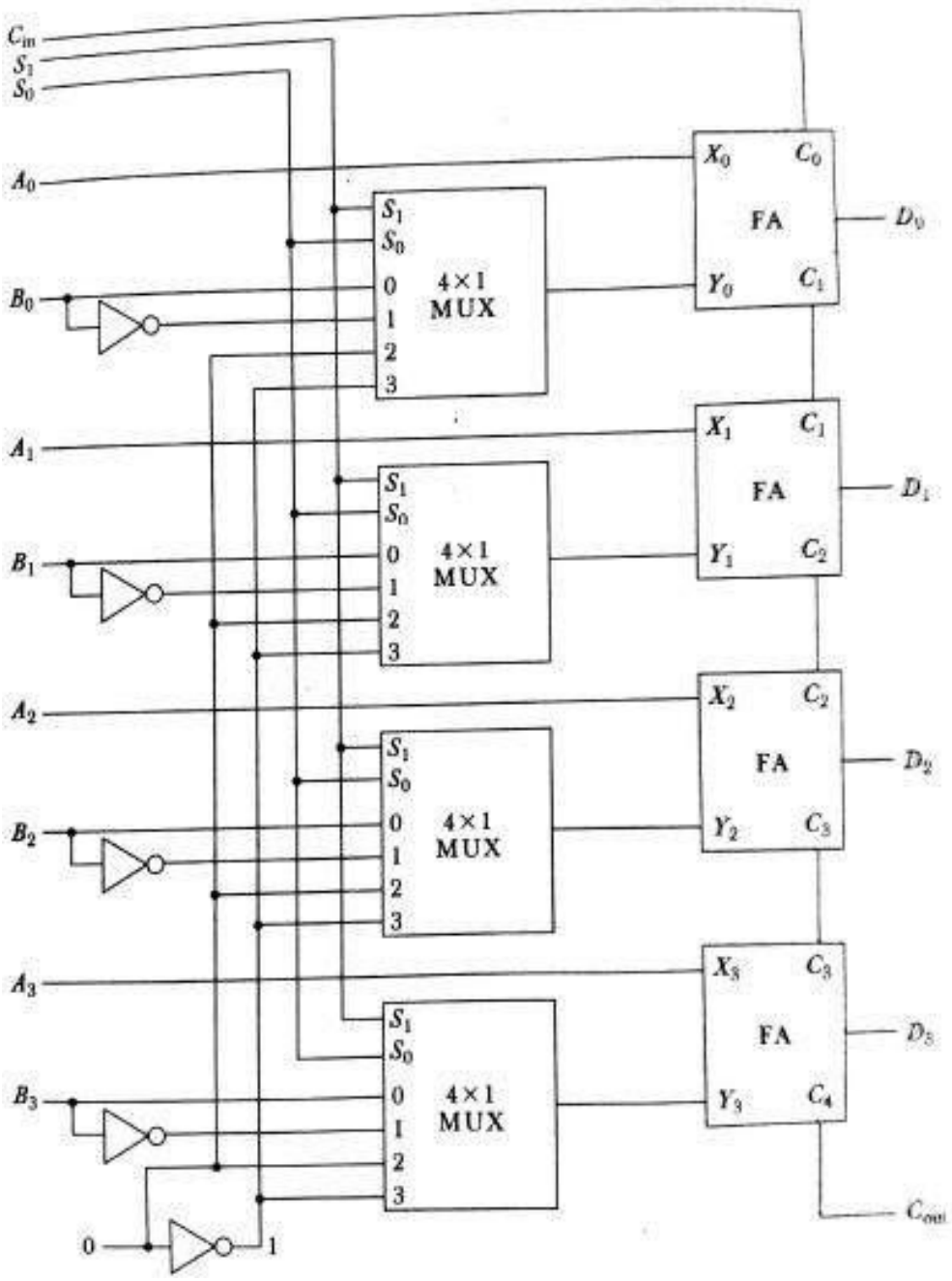
Figure 4-8 4-bit binary incrementer.

- One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.
- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.
- The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3.
- The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0through S_3 to go to 0.

- The circuit of Fig. 4-8 can be extended to an *n* -bit binary incrementer by extending the diagram to include *n* half-adders.
- The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

Arithmetic Circuit:

- The basic component of an arithmetic circuit is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- The diagram of a 4-bit arithmetic circuit is shown in Fig. 4-9. It has four full-adder circuits thatconstitute the 4-bit adder and four multiplexers for choosing different operations.



- There are two 4-bit inputs A and *B* and a 4-bit output *D*.
- The four inputs from A go directly to the X inputs of the binary adder.
- Each of the four inputs from B are connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of B.
- The other two data inputs are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs S_1 and S_0. The input carry C_{in}, goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- By controlling the value of Y with the two selection inputs S_1 and S_0 and making C_{in} equal to 0 or 1,it is possible to generate the eight arithmetic microoperations listed in Table 44.

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Addition:

- When S_1S_0= 00, the value of *B* is applied to the Y inputs of the adder.
  - ✓ If Cir, = 0, the output *D =A+B.*
  - ✓ If Cin = 1, output *D=A+B + 1.*
- Both cases perform the add microoperation with or without adding the input carry.

Subtraction:

- When S_1S_0 = 01, the complement of B is applied to the Y inputs of the adder.
  - ✓ If C_{in} = 1, then D = A $\overline{+}$ B + 1. This produces A plus the 2's complement of B, which isequivalent to a subtraction of A -B.
  - ✓ When C_{in} = 0 then D = A $\overline{+}$ B. This is equivalent to a subtract with borrow, that is, A-B-1.

Increment:

- When S_1S_0 = 10, the inputs from *B* are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 + C_{in}. This gives *D* = A when C_{in} = 0 and D = A + 1 when C_{in} = 1.
- In the first case we have a direct transfer from input A to output D.
- In the second case, the value of A is incremented by 1.

Decrement:

- When S_1S_0= 11, all l's are inserted into the Y inputs of the adder to produce the decrement operation $D = A$ -1 when C_{in} = 0.
- This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A — 1. When C_{in} = 1, then $D = A$ -1 + 1=A, which causes a direct transfer frominput A to output D.

Logic Micro-operations:

- Logic microoperations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- For example, the exclusive-OR microoperation with the contents of two registers RI and R2 is symbolized by the statement

$$P: \quad R1 \leftarrow R1 \oplus R2$$

- It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.

List of Logic Microoperations:

- There are 16 different logic operations that can be performed with two binary variables.
- They can be determined from all possible truth tables obtained with two binary variables as shown in Table 4-5.

TABLE 4-5 Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table 4-6.
- The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.
- The logic micro-operations listed in the second column represent a relationship between the binary content of two registers A and B.

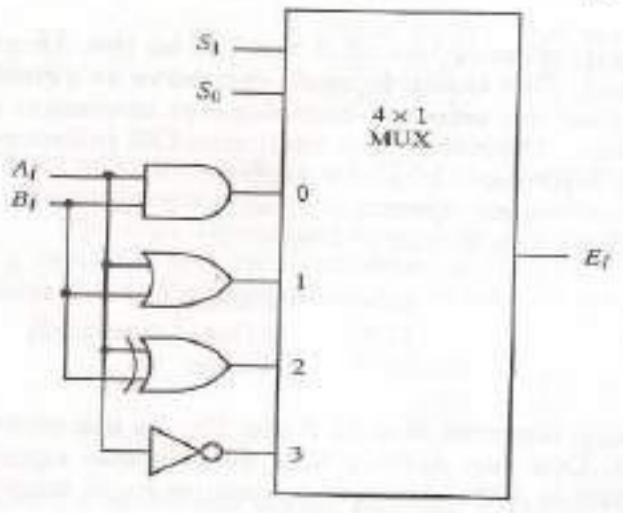
TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation:

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Although there are 16 logic microoperations, most computers use only four--AND, OR, XOR(exclusive-OR), and complement from which all others can be derived.
- Figure 4-10 shows one stage of a circuit that generates the four basic logic microoperations.
- It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic.
- The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs S_1 and S_0 choose one of the data inputs of the multiplexer and direct its value to the output.

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

Some Applications:

- Logic micro-operations are very useful for manipulating individual bits or a portion of a word stored in a register.
- They can be used to change bit values, delete a group of bits or insert new bits values into a register.
- The following example shows how the bits of one register (designated by A) are manipulated by logic microoperations as a function of the bits of another register (designated by B).
- Selective set
  - The *selective-set* operation sets to 1 the bits in register A where there are corresponding l's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation:

```
1010     A before
1100     B (logic operand)
1110     A after
```

  - The OR microoperation can be used to selectively set bits of a register.

- Selective complement
  - ✓ The *selective-complement* operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in *B.* For example:

```
1010     A before
1100     B (logic operand)
0110     A after
```

  - ✓ The exclusive-OR microoperation can be used to selectively complement bits of a register.
- Selective clear
  - ✓ The *selective-clear* operation clears to 0 the bits in A only where there are corresponding l's in B. For example:

```
1010     A before
1100     B (logic operand)
0010     A after
```

  - ✓ The corresponding logic microoperation is $A \leftarrow A \wedge \overline{B}$
- Mask
  - ✓ The *mask* operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding O's in B . The mask operation is an AND micro operation as seen from the following numerical example:

```
0110 1010     A before
0000 1111     B (mask)
0000 1010     A after masking
```

- Insert
  - ✓ The *insert* operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.

- ✓ For example, suppose that an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001 we first mask the four unwanted bits:

```
0110 1010      A before
0000 1111      B (mask)
0000 1010      A after masking
```

and then insert the new value:

```
0000 1010      A before
1001 0000      B (insert)
1001 1010      A after insertion
```

- ✓ The mask operation is an AND microoperation and the insert operation is an OR microoperation.

- Clear
  - ✓ The *clear* operation compares the words in *A* and *B* and produces an all 0's result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation as shown by the following example

```
1010     A
1010     B
0000     A ← A ⊕ B
```

Shift Microoperations:

- Shift microoperations are used for serial transfer of data.
- The contents of a register can be shifted to the left or the right.
- During a shift-left operation the serial input transfers a bit into the rightmost position.
- During a shift-right operation the serial input transfers a bit into the leftmost position.
- There are three types of shifts: logical, circular, and arithmetic.
- The symbolic notation for the shift microoperations is shown in Table 4-7.

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

- **Logical Shift:**
  - A *logical* shift is one that transfers 0 through the serial input.
  - The symbols *shl* and shr for logical shift-left and shift-right microoperations.

- The microoperations that specify a 1-bit shift to the left of the content of register R and a 1-bit shift to the right of the content of register R shown in table 4.7.
- The bit transferred to the end position through the serial input is assumed to be 0 duringa logical shift.

- **Circular Shift:**
  - The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information.
  - This is accomplished by connecting the serial output of the shift register to its serial input.
  - We will use the symbols *cil* and *cir* for the circular shift left and right, respectively.
- **Arithmetic Shift:**
  - An *arithmetic shift* is a microoperation that shifts a signed binary number to the left or right.
  - An arithmetic shift-left multiplies a signed binary number by 2.
  - An arithmetic shift-right divides the number by 2.
  - Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

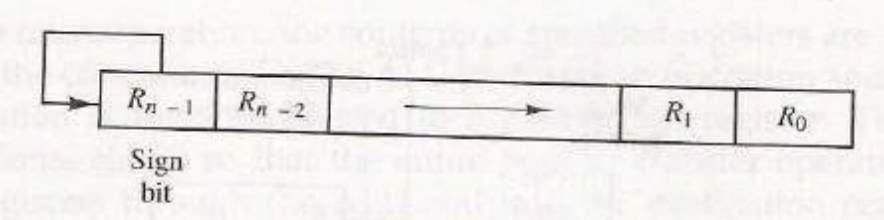
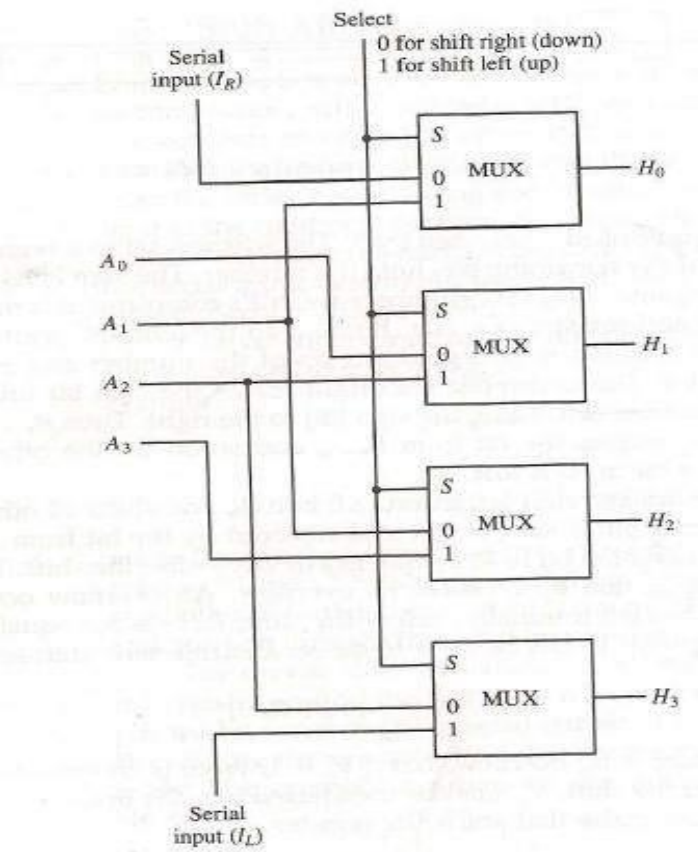


Figure 4-11 Arithmetic shift right.

Hardware Implementation:

- A combinational circuit shifter can be constructed with multiplexers as shown in Fig. 4-12.
- The 4-bit shifter has four data inputs, A_0 through A_3, and four data outputs, H_0 through H_3.
- There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).
- When the selection input *S=0* the input data are shifted right (down in the diagram).
- When *S* = 1, the input data are shifted left (up in the diagram).
- The function table in Fig. 4-12 shows which input goes to each output after the shift.
- A shifter with n data inputs and outputs requires n multiplexers.
- The two serial inputs can be controlled by another multiplexer to provide the three possible typesof shifts.



Function table

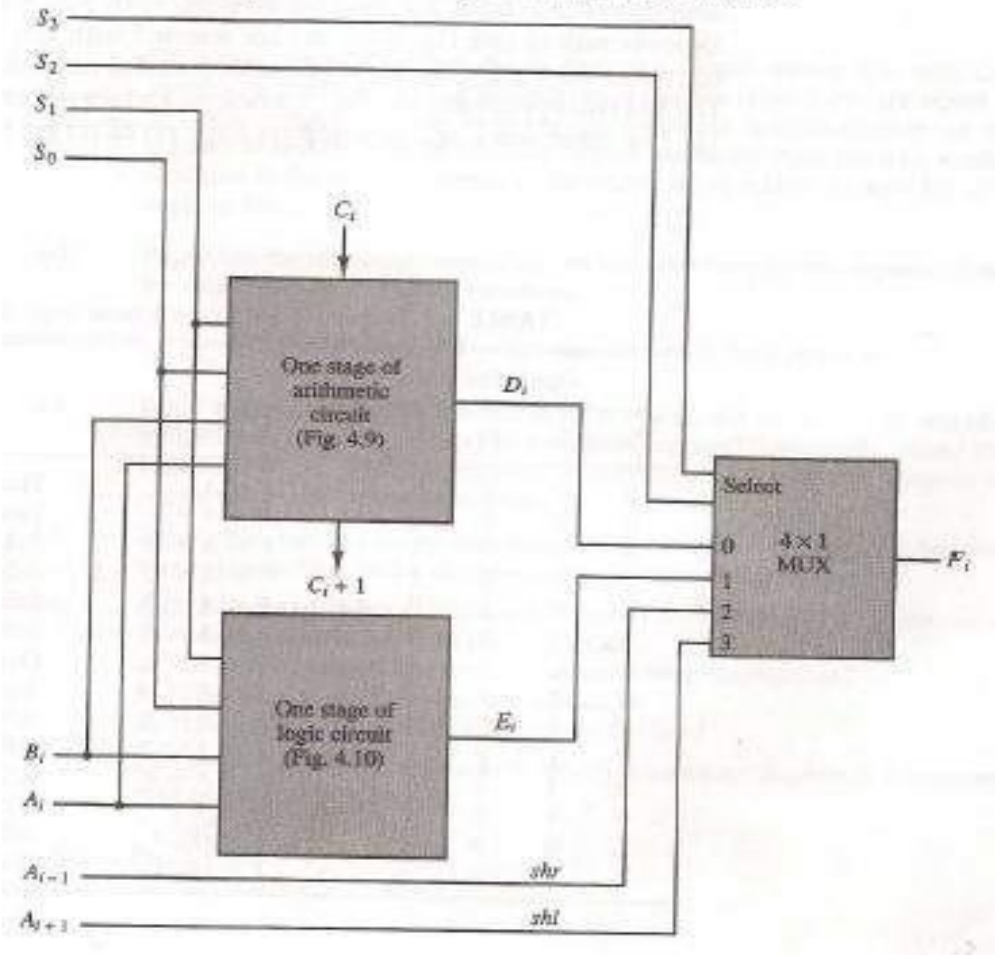
Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 4-12 4-bit combinational circuit shifter.

Arithmetic Logic Shift Unit:

- Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
- The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.
- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in Fig. 4-13.
- Particular microoperation is selected with inputs S_1 and S_0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in D_i and a logic output in E_i.
- The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.
- The circuit whose one stage is specified in Fig. 4-13 provides eight arithmetic operation, four logic operations, and two shift operations.
- Each operation is selected with the five variables S_3, S_2, S_1, S_0 and C_{in}.
- The input carry C_{in} is used for selecting an arithmetic operation only.

Figure 4-13 One stage of arithmetic logic shift unit.



- Table 4-8 lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with $S_3S_2 = 00$.
- The next four are logic and are selected with $S_3S_2 = 01$.
- The input carry has no effect during the logic operations and is marked with don't-care x's.
- The last two operations are shift operations and are selected with $S_3S_2 = 10$ and 11.
- The other three selection inputs have no effect on the shift.

TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

UNIT 1 – BASIC COMPUTER ORGANIZATION AND DESIGN

CONTENTS:

- Instruction codes
- Computer Registers Computer instructions
- Timing and Control
- Instruction cycle
- Memory Reference Instructions
- Input – Output and Interrupt.

INSTRUCTION CODE

An instruction code is a group of bits that instruct the computer to perform a specific operation.

Operation Code

The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Accumulator (AC)

Computers that have a single-processor register usually assign to it the name accumulator (AC) accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

Stored Program Organization

- The simplest way to organize a computer is to have one processor register and aninstruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies anaddress.
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together withthedata stored in the processor register.
- The following figure 2.1 shows this type of organization.

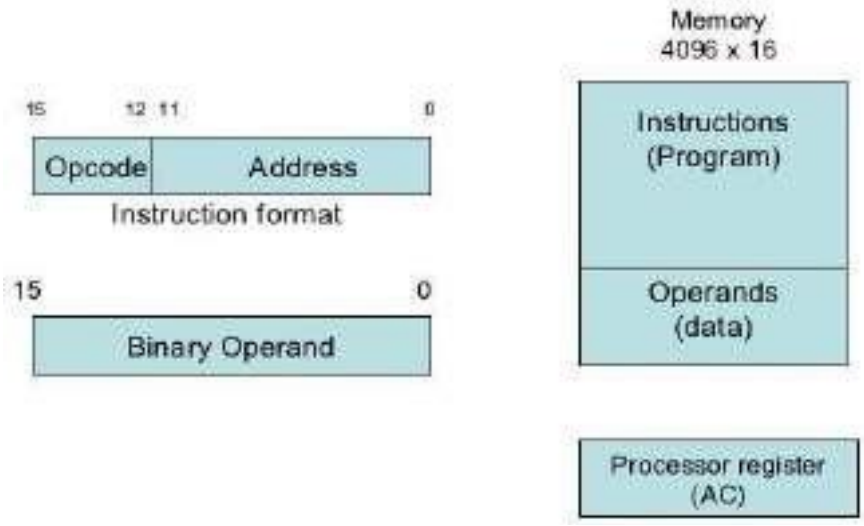


Figure 2.1: Stored Program Organization

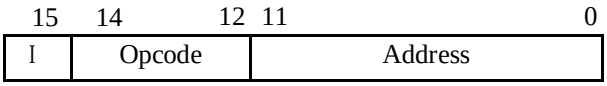
- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words, we need 12 bits to specify an address since 2^{12} = 4096.

- If we store each instruction code in one 16-bit memory word, we have available four bits for operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- The control reads a 16-bit instruction from the program portion of memory.
- It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
- It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name accumulator and label it AC.
- If an operation in an instruction code does not need an operand from memory, the restof the bits in the instruction can be used for other purposes.
- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory. For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

Direct and Indirect addressing of basic computer.

- The second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, the bits in the second part of the instruction designate an address ofa memory word in which the address of the operand is found.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The mode bit is 0 for a direct address and 1 for an indirect address.

- A direct address instruction is shown in Figure 2.2. It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction.
- The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Figure 2.3 has a mode bit I = 1, recognized as an indirect address instruction.
- The address part is the binary equivalent of 300.
- The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC.
- The indirect address instruction needs two references to memory to fetch an operand.
    1. The first reference is needed to read the address of the operand
    2. Second reference is for the operand itself.
- The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data.



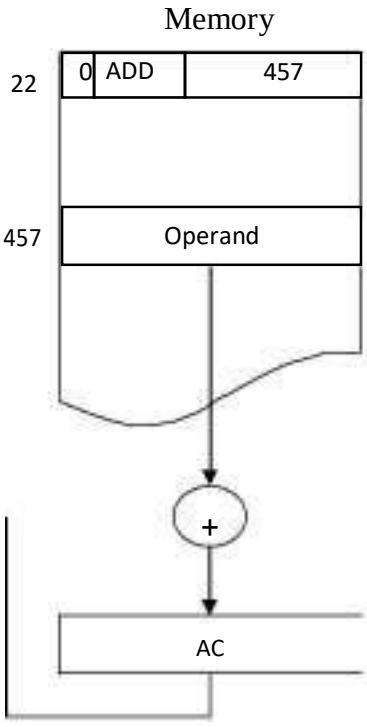


Figure 2.2: Direct Address

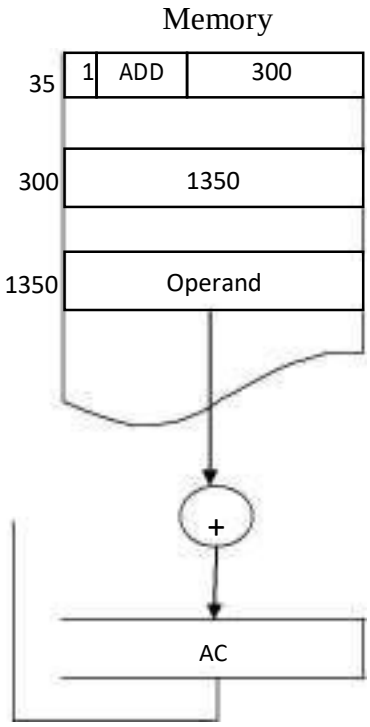


Figure 2.3: Indirect Address

Direct Address	Indirect Address
When the second part of an instruction code specifies the addressof an operand, the instruction is said to have a direct address.	When the second part of an instruction code specifies the address of a memory word in which the address of the operand,the instruction is said to have a direct address.
For instance the instruction MOV R0 00H. R0, when converted to machinelanguage is the physical address of register R0. The instruction moves 0 to R0	For instance the instruction MOV @R0 00H,when converted to machine language, @R0becomes whatever is stored in R0, and thatis the address used to move 0 to. It can be whatever is stored in R0.

Registers of basic computer

- It is necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holdinga memory address.
- These requirements dictate the register configuration shown in Figure 2.4.

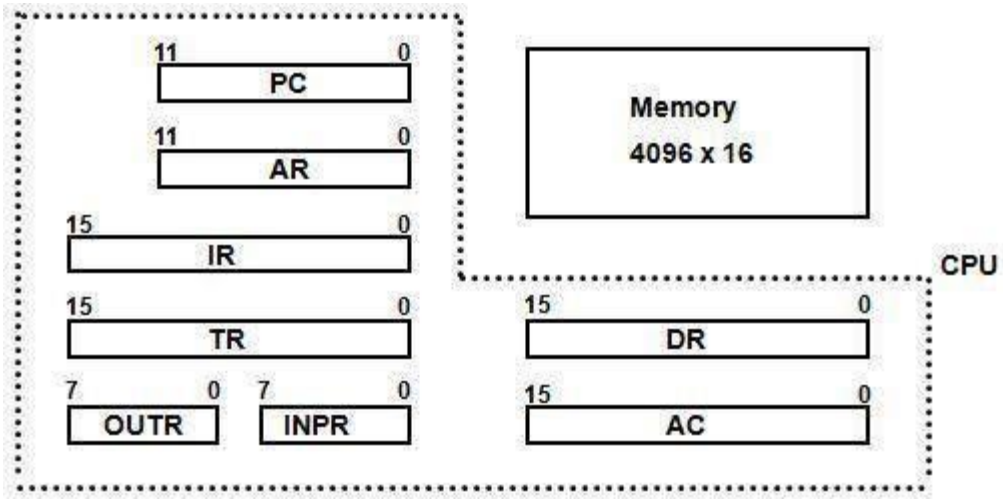


Figure 2.4: Basic Computer Register and Memory

- The data register (DR) holds the operand read from memory.
- The accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the instruction register (IR).
- The temporary register (TR) is used for holding temporary data during the processing.
- The memory address register (AR) has 12 bits.
- The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program.
- Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

Register Symbol	Bits	Register Name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Table 2.1: List of Registers for Basic Computer

Common Bus System for basic computer register.

What is the requirement of common bus System?

- The basic computer has eight registers, a memory unit and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and register.
- The number of wires will be excessive if connections are between the outputs of each register and the inputs of the other registers. An efficient scheme for transferring information in a system with many register is to use a common bus.
- The connection of the registers and memory of the basic computer to a common bus system is shown in figure 2.5.
- The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S2, S1, and S0.
- The number along each output shows the decimal equivalent of the required binary selection.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus whenthe read input is activated and S2 S1 S0 = 1 1 1.
- Four registers, DR, AC, IR, and TR have 16 bits each.
- Two registers, AR and PC, have 12 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's. When AR and PC receive information from the bus, only the 12 least significant bits are transferred into the register.
- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus. INPR is connected to provide information to the bus but OUTR can only receive information from the bus.

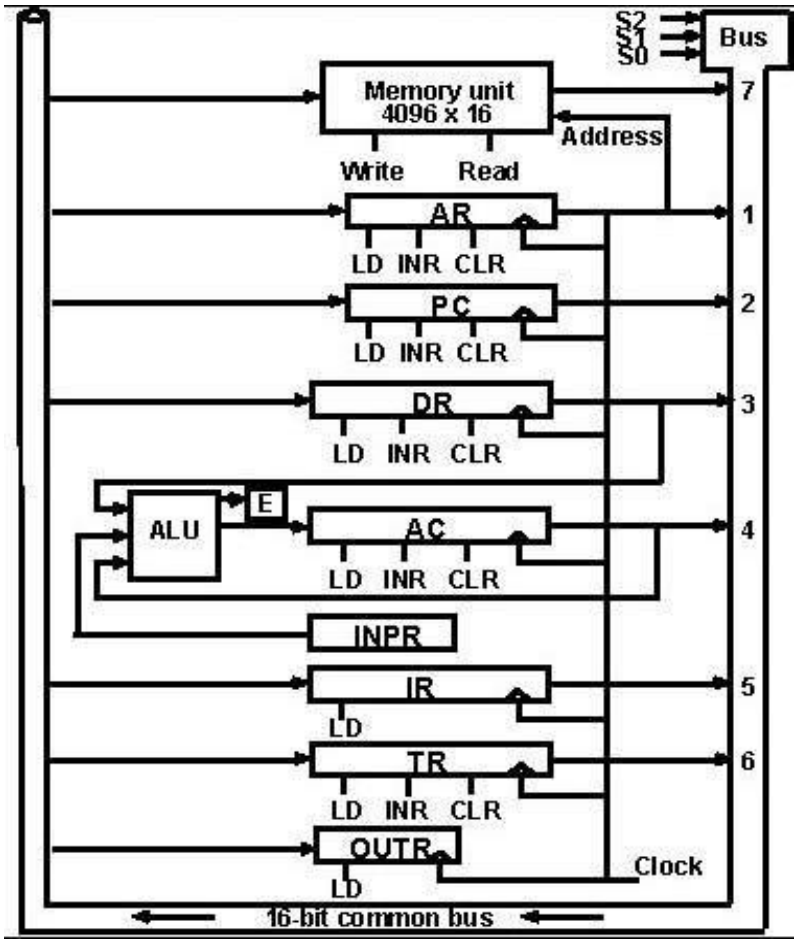


Figure 2.5: Basic computer registers connected to a common bus

- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). Tworegisters have only a LD input.
- AR must always be used to specify a memory address; therefore memory address is connected to AR.
- The 16 inputs of AC come from an adder and logic circuit. This circuit has three sets ofinputs.
  1. Set of 16-bit inputs come from the outputs of AC.
  2. Set of 16-bits come from the data register DR.
  3. Set of 8-bit inputs come from the input register INPR.
- The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit).
- The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

Instruction Format with its types.

- The basic computer has three instruction code formats, as shown in figure 2.6.

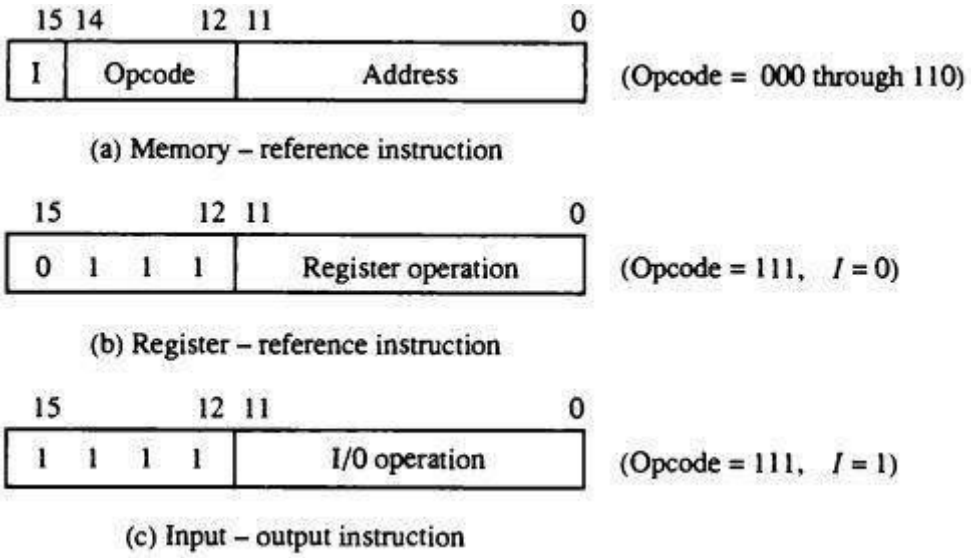


Figure 2.6: Basic computer instruction format

- Each format has 16 bits.
- The operation code (opcode) part of the instruction contains three bits and the meaningof the remaining 13 bits depends on the operation code encountered.
- A **memory-reference instruction** uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
- The **register reference instructions** are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
- An **input-output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.

Control Unit with timing diagram.

- The block diagram of the control unit is shown in figure 2.7.
- Components of Control unit are
  1. Two decoders
  2. A sequence counter
  3. Control logic gates
- An instruction read from memory is placed in the instruction register (IR). In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 X 8 decoder.

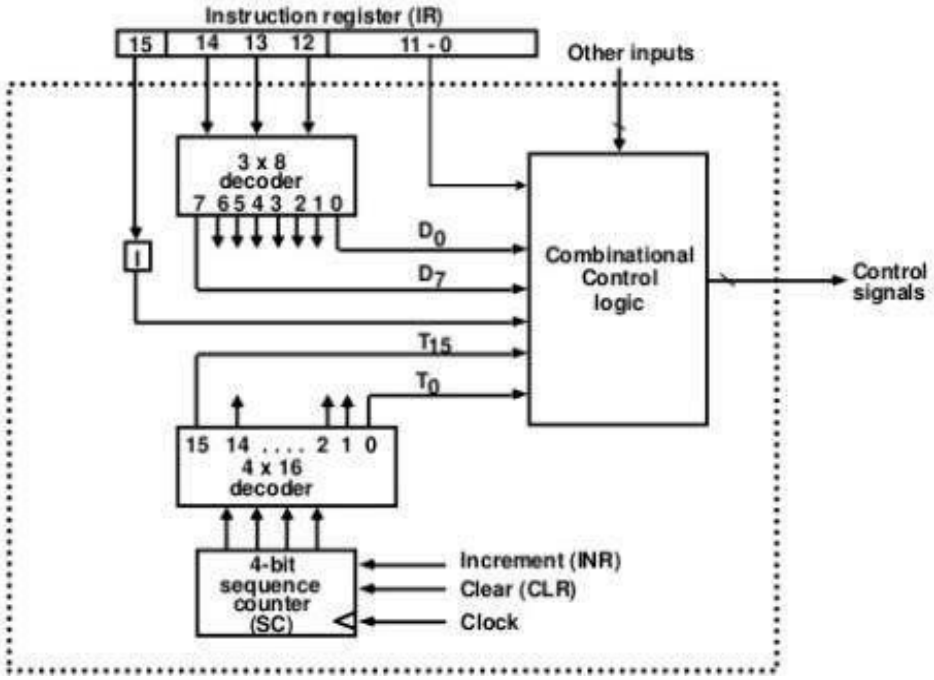


Figure: Control unit of basic computer

- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.
- The eight outputs of the decoder are designated by the symbols D0 through D7. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can countin binary from 0 through 15.The outputs of counter are decoded into 16 timing signals T0 through T15.
- The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of 4 X 16 decoder. Once in awhile, the counter is cleared to 0, causing the next timing signal to be T0.
- As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3 and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement

$$D3T4: SC \leftarrow 0$$

Timing Diagram:

- The timing diagram figure2.8 shows the time relationship of the control signals.
- The sequence counter SC responds to the positive transition of the clock.
- Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to 0, which in turn activates the timingT0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labeled T0 in the diagram will trigger only those registers whose control inputs areconnected to timing signal T0.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This procedures the sequence of timing signals T0, T1, T2, T3 and T4, and so on. If SCis notcleared, the timing signals will continue with T5, T6, up to T15 and back to T0.

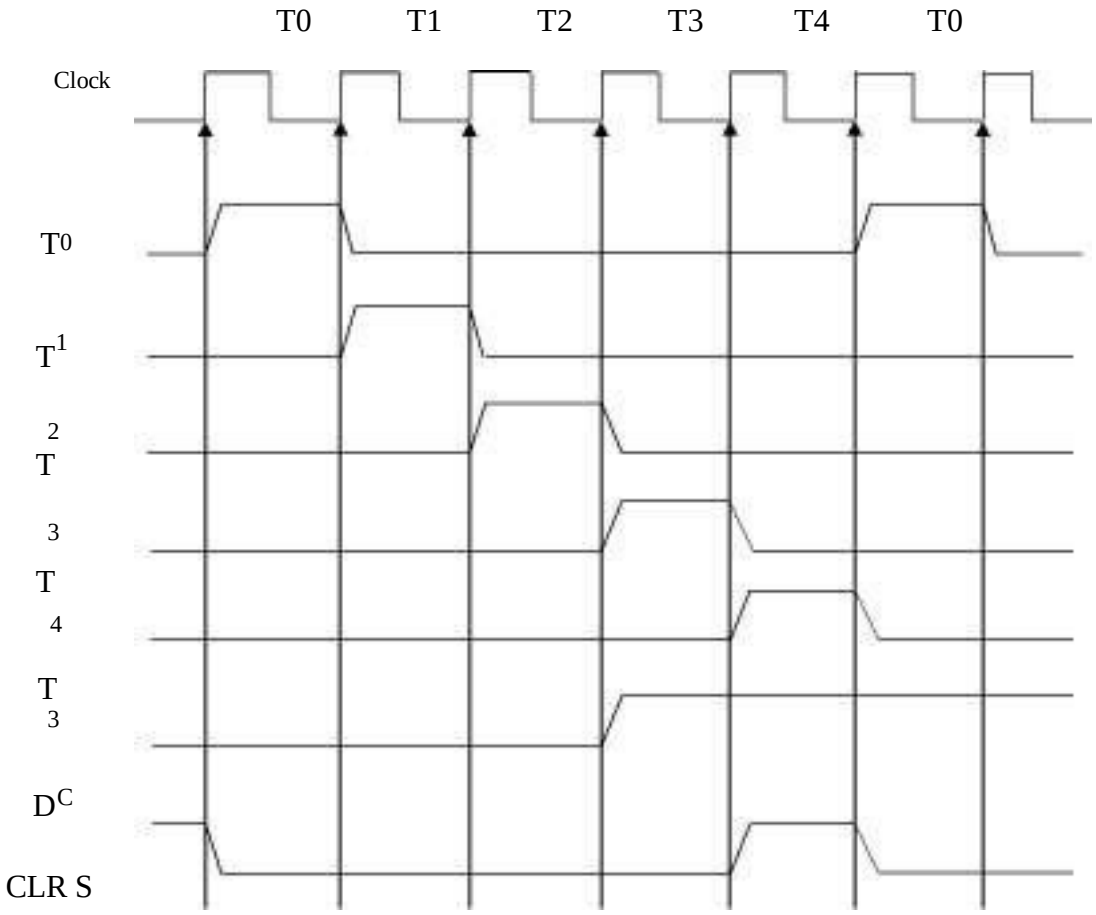


Figure: Example of control timing signals

- The last three waveforms shows how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active.
- This signal is applied to the CLR input of SC. On the next positive clock transition the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.

Instruction cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
- After step 4, the control goes back to step 1 to fetch, decode and execute the nex instruction.

- This process continues unless a HALT instruction is encountered.

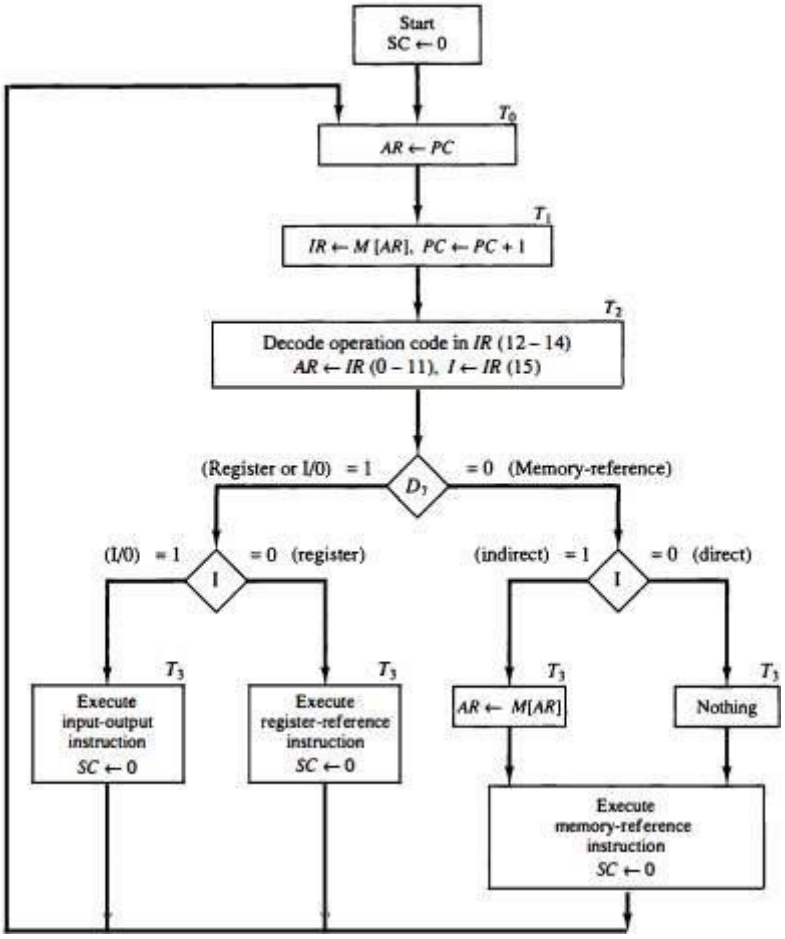


Figure 2.9: Flowchart for instruction cycle (initial configuration)

- The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- If D7 = 1, the instruction must be register-reference or input-output type. If D7 = 0, the operation code must be one of the other seven values 110, specifying a memory- reference instruction. Control then inspects the value of the first bit of the instruction, which now available in flip-flop I.
- If D7 = 0 and I = 1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory.

- The three instruction types are subdivided into four separate paths. The selected ration is activated with the clock transition associated with timing signal T3.This canbe symbolized as follows:

D'7 I T3: AR M [AR]
D'7 I' T3: Nothing
D7 I' T3: Execute a register-reference instruction
D7 I T3: Execute an input-output instruction

- When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.
- However, the sequence counter SC must be incremented when D'7 I T3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T4.
- A register-reference or input-output instruction can be executed with the click associated with timing signal T3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T0 =1. SC is either incremented or cleared to 0 with every positive clock transition.

Register reference instruction.

- When the register-reference instruction is decoded, D7 bit is set to 1.
- Each control function needs the Boolean relation D7 I' T3

15 12	11 0
0 1 1 1	Register Operation

There are 12 register-reference instructions listed below:

	r:	SC← 0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_9:	AC ← AC'	Complement AC
CME	rB_8:	E ← E'	Complement E
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ← AC(0)	Circular Right
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15)	Circular Left
INC	rB_5:	AC ← AC + 1	Increment AC
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)	Skip if positive
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1	Skip if negative
SZA	rB_2:	if (AC = 0) then (PC ← PC+1)	Skip if AC is zero
SZE	rB_1:	if (E = 0) then (PC ← PC+1)	Skip if E is zero
HLT	rB_0:	S← 0 (S is a start-stop flip-flop)	Halt computer

- These 12 bits are available in IR (0-11). They were also transferred to AR during time T2.
- These instructions are executed at timing cycle T3.
- The first seven register-reference instructions perform clear, complement, circular shift,and increment microoperations on the AC or E registers.
- The next four instructions cause a skip of the next instruction in sequence when condition is satisfied. The skipping of the instruction is achieved by incrementing PC.
- The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if all the flip-flops of the register are zero.
- The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

Memory reference instructions

- When the memory-reference instruction is decoded, D7 bit is set to 0.

15	14 12	11 0
I	000~110	Address

- The following table lists seven memory-reference instructions.

Symbol	**Operation Decoder**	**Symbolic Description**
AND	D_0	AC← AC + M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC← M[AR]

STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

- The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T4.

AND to AC

This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC.

D0T4: DR ← M[AR]
D0T5: AC ← AC ∧ DR, SC ← 0

ADD to AC

This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop.

D1T4: DR ← M[AR]
D1T5: AC ← AC + DR, E ← Cout, SC ← 0

DA: Load to AC

This instruction transfers the memory word specified by the effective address to AC.

D2T4: DR ← M[AR]
D2T5: AC ← DR, SC ← 0

STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address.

D3T4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

This instruction transfers the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.

D4T4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

M[AR] PC, PC AR + 1 M[135]
21, PC 135 + 1 = 136

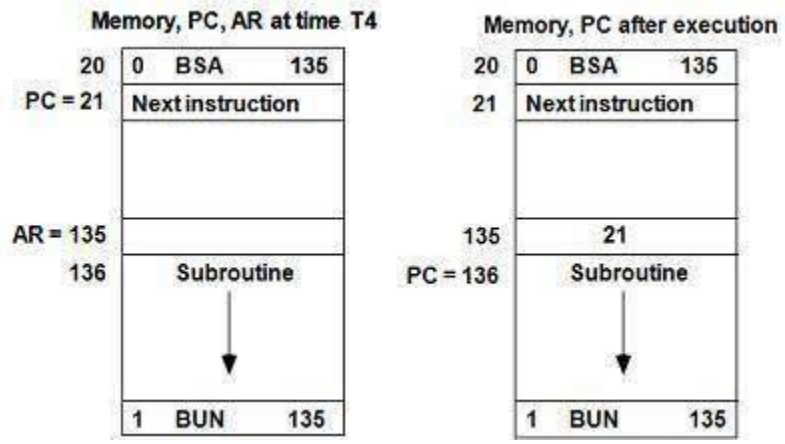


Figure2.10: Example of BSA instruction execution

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

D5T4: M[AR] PC, AR AR
+ 1D5T5: PC AR, SC 0

ISZ: Increment and Skip if Zero

These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. Since it is not possible to

increment a word inside the memory, it is necessary to read the word into DR, incrementDR, and store the word back into memory.

D6T
4: DR M[AR]D6T5: DR DR + 1
D6T4: M[AR] DR, if (DR = 0) then (PC PC + 1),
SC 0

Control Flowchart

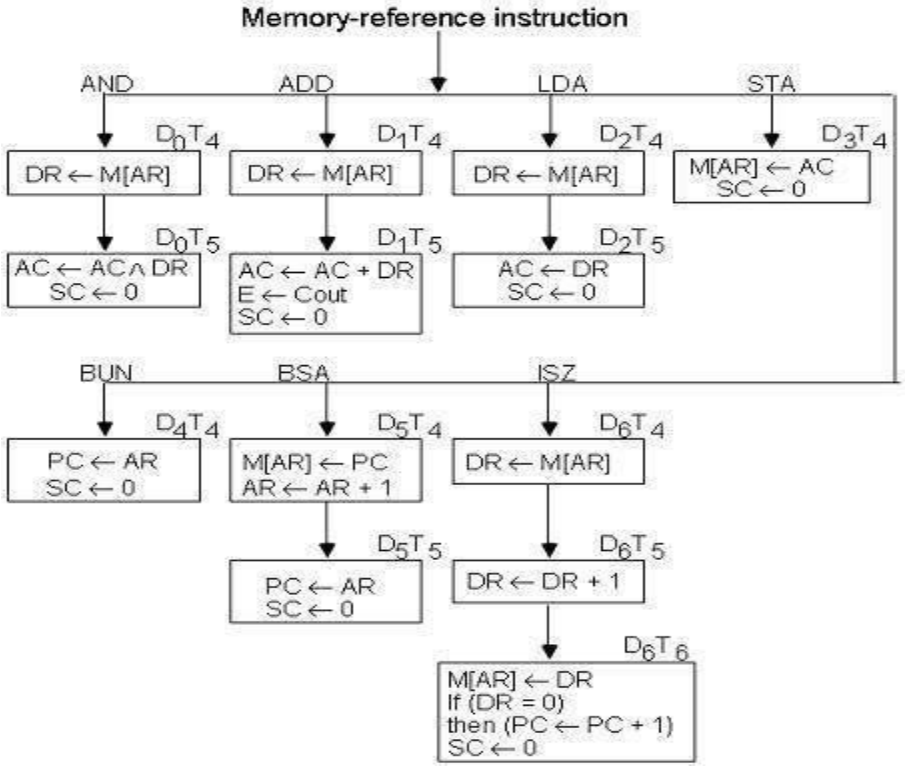


Figure 2.11: Flowchart for memory-reference instructions

Input-output configuration of basic computer

- A computer can serve no useful purpose unless it communicates with the external environment.
- To exhibit the most basic requirements for input and output communication, we will use a terminal unit with a keyboard and printer.

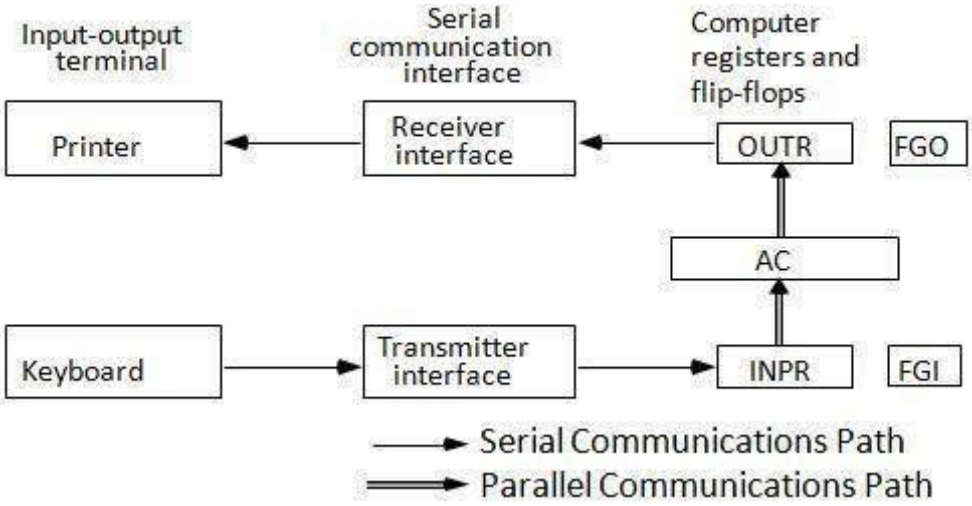


Figure: Input-output configuration

- The terminal sends and receives serial information and each quantity of information haseight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with theAC in parallel.
- The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.
- The 1-bit input flag FGI is a control flip-flop. It is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The flag is needed to synchronize the timing rate difference between the input deviceand the computer.
- The process of information transfer is as follows:

The process of input information transfer:

- Initially, the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.
- As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.
- Once the flag is cleared, new information can be shifted into INPR by striking anotherkey.

The process of outputting information:

- The output register OUTR works similarly but the direction of information flow isreversed.
- Initially, the output flag FGO is set to 1. The computer checks the flag bit;

if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

- The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in theprocess of printing the character.

Input-Output instructions

- Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling theinterrupt facility.
- Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1.
- The remaining bits of the instruction specify the particular operation.
- The control functions and microoperations for the input-output instructions are listed below.

INP	AC(0-7) □ INPR, FGI □ 0	Input char. to AC
OUT	OUTR □ AC(0-7), FGO □ 0	Output char. from AC
SKI	if(FGI = 1) then (PC □ PC + 1)	Skip on input flag
SKO	if(FGO = 1) then (PC □ PC + 1)	Skip on output flag
ION	IEN □ 1	Interrupt enable on
IOF	IEN □ 0	Interrupt enable off

Table 2.2: Input Output Instructions

- The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0.
- The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0.
- The next two instructions in Table 2.2 check the status of the flags and cause a skip of the next instruction if the flag is 1.
- The instruction that is skipped will normally be a branch instruction to return and check the flag again.
- The branch instruction is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed.
- The last two instructions set and clear an interrupt enable flip-flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

Interrupt Cycle

The way that the interrupt is handled by the computer can be explained bymeans of the flowchart shown in figure 2.13.

- An interrupt flip-flop R is included in the computer.
- When R = 0, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by thecontrol.
- If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits.
- If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information.
- In this case, control continues with the next instruction cycle. If either

flag is set to 1while IEN = 1, flip-flop R is set to 1.

- At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

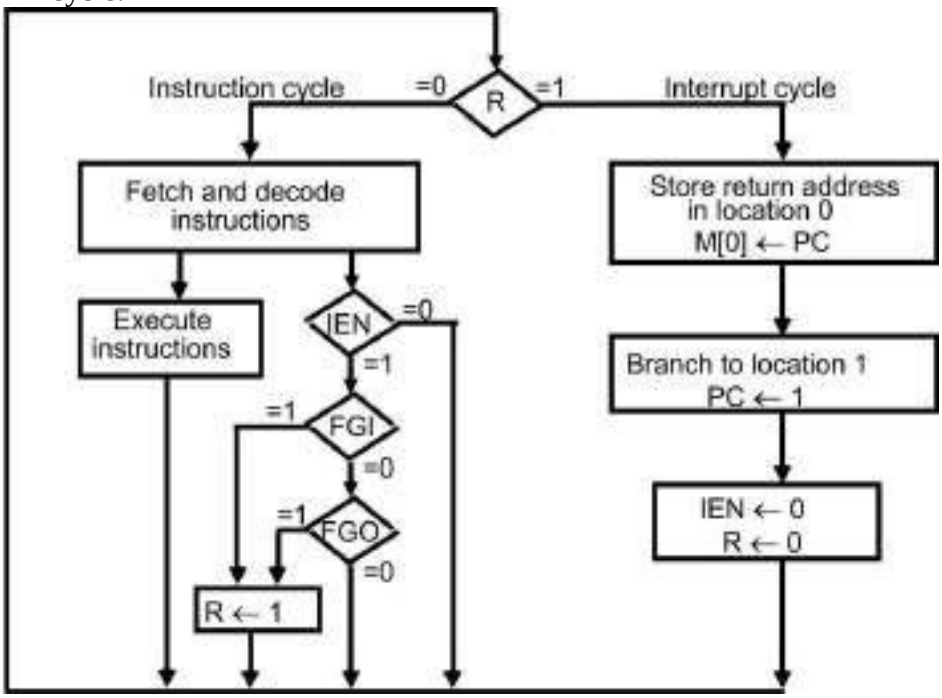


Figure 2.13: Flowchart for interrupt cycle

Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return addressoperation.
- The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location.
- Here we choose the memory location at address 0 as the place for storing the return address.
- Control then inserts address 1 into PC and clears IEN and R so that no more interruptionscan occur until the interrupt request from the flag has been serviced.
- An example that shows what happens during the interrupt cycle is shown in Figure 2.14:

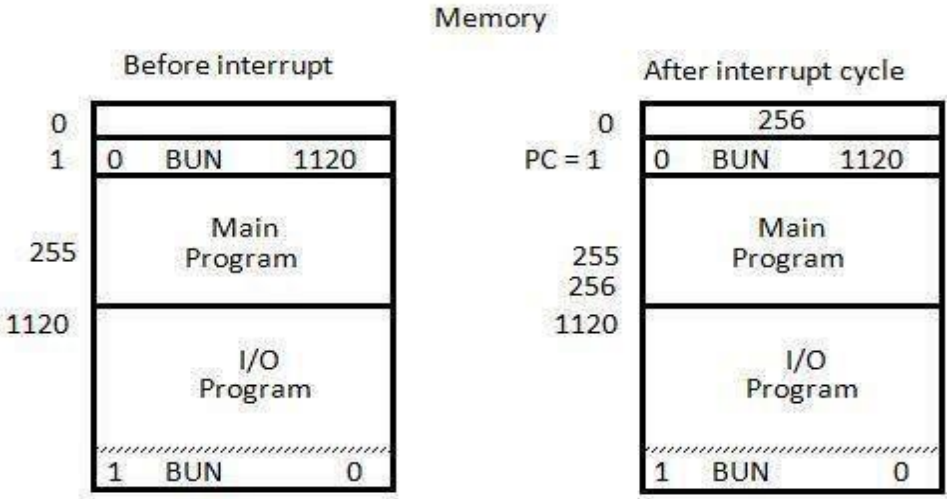


Figure 2.14: Demonstration of the interrupt cycle

- Suppose that an interrupt occurs and R = 1, while the control is executing theinstruction at address 255. At this time, the return address 256 is in PC.
- The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.
- The content of PC (256) is stored in memory location 0, PC is set to 1, and R iscleared to0.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returnsto the location where it was interrupted.
- The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O service program.
- The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Register transfer statements for the interrupt cycle

- The flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signalsT0, T1 or T2 are active.
- The condition for setting flip-flop R= 1 can be expressed with thefollowing register transfer statement:

  T0 T1 T2 (IEN) (FGI + FGO): R1
- The symbol + between FGI and FGO in the control functiondesignates a logic OR operation. This is AND with IEN and T0 T1 T2 .
- The fetch and decode phases of the instruction cycle must be modifiedand Replace T0,T1, T2 with R'T0, R'T1, R'T2
- Therefore the interrupt cycle statements are

  RT0: AR 0,TR PC

  RT1: M[AR] TR, PC 0

  RT2: PC PC + 1, IEN 0, R 0, SC 0
- During the first timing signal AR is cleared to 0, and the content of PCis transferred tothe temporary register TR.
- With the second timing signal, the return address is stored in memory atlocation 0 andPC is cleared to 0.
- The third timing signal increments PC to 1, clears IEN and R, andcontrol goes back to T0 by clearing SC to 0.
- The beginning of the next instruction cycle has the condition RT0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

Flow chart for computer operation.

- The final flowchart of the instruction cycle, including the interruptcycle for the basiccomputer, is shown in Figure

2.15.

- The interrupt flip-flop R may be set at any time during the indirect or executephases.
- The control returns to timing signal T0 after SC is cleared to 0.
- If R = 1, the computer goes through an interrupt cycle. If R = 0, the computer goes through an instruction cycle.
- If the instruction is one of the memory-reference instructions, the computer first checks if there is an indirect address and then continues to execute the decoded instruction according to the flowchart.
- If the instruction is one of the register-reference instructions, it is executed with one of the microoperations register reference.
- If it is an input-output instruction, it is executed with one of the microoperation's input-output reference.

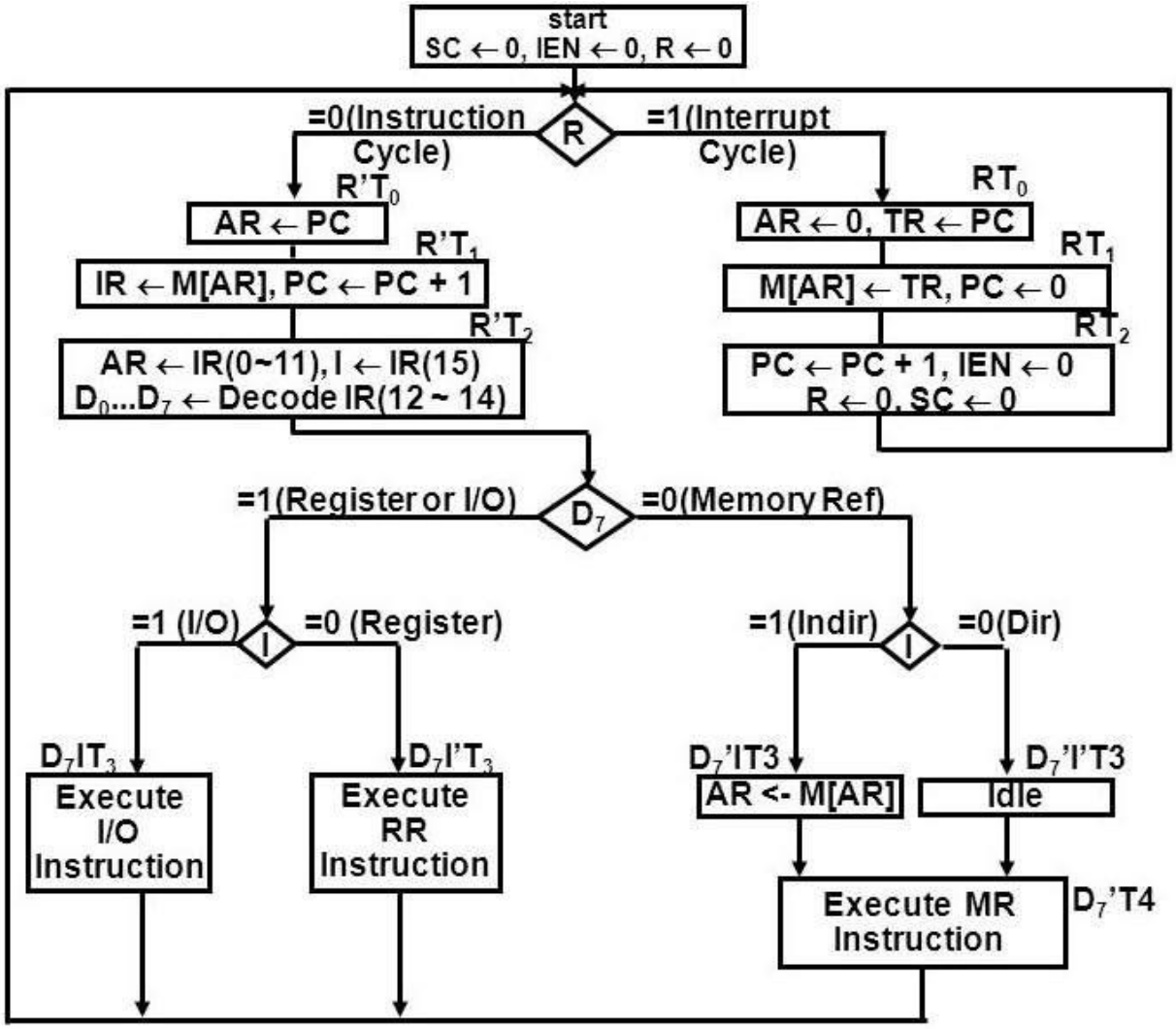


Figure: Flowchart for computer operation