

# COMPUTER ORGANIZATION AND ARCHITECTURE

## MODULE-I

### DIGITAL COMPUTER

A digital system that performs various computational tasks.

**Digital:** The word digital implies that the information in the computer is represented by variables that take a limited number of discrete values. These discrete values are processed internally by components that can maintain a limited number of discrete states. (0,1,...9 provide 10 discrete values)

First electronic digital computer developed in 1940s which was used primarily for numerical computations (using discrete elements i.e., digits). From this application the term digital computer has emerged.

Digital computers function more reliably if only two states are used because of physical restriction of components and because of human logic tends to binary (True/False or YES/NO statements)

Digital components that are constrained to take discrete values are further constrained to take only two values- binary 0/1.

Information in digital computers is represented by a group of bits (binary digit is called a bit).

By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols-decimal digits, letters of alphabets.

By use of binary arrangements and by using various coding techniques, the groups of bits are used to develop computer sets of instructions for performing various types of computations.

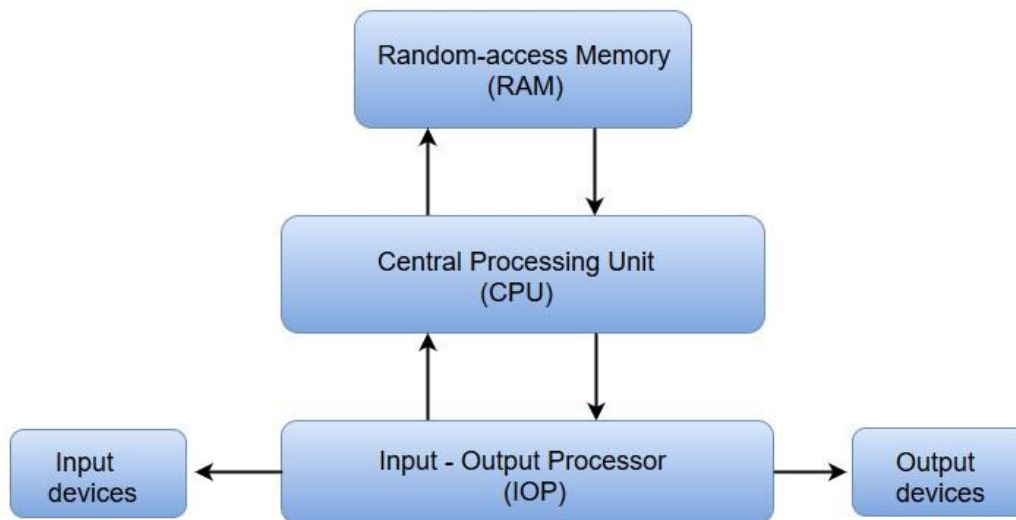
### BLOCK DIAGRAM OF DIGITAL COMPUTER

A computer system is subdivided into two functional entities: hardware, software.

**Hardware**-it consists of all the electronic components and electric mechanical devices that comprise the physical entity of the device.

**Software**-it consists of the instructions and data that the computer manipulates to perform various data processing tasks.

### Block diagram of a digital computer:



**Central Processing Unit (CPU):** Contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and a control circuit for fetching and executing instructions.

**Memory Unit (MU):** Contains storage for instructions and data. ROM(Read-Only Memory) is a part of memory unit. MU is also called as Primary memory/Internal memory/Principal memory.

**Random Access Memory (RAM):** used for real-time processing of the data. It allows your computer to switch between programs and have large files ready to view.

**Input-Output devices:** They are used for generating inputs from the user and displaying the final results to the user.

Eg: keyboard, mouse, terminals, magnetic disk drives, and other communication devices.

## COMPUTER ORGANIZATION

It deals with the internal view of the computer and the roles that the internal components play during the execution of a program. It includes the organization of major parts of a computer such as the processor, memory and peripheral devices.

**Processor organization:** It deals with main components of a processor, how these are interconnected and how these operate execution of an instruction.

**Memory organization:** The memory organization of a memory unit deals with how its different components are structured and interconnected.

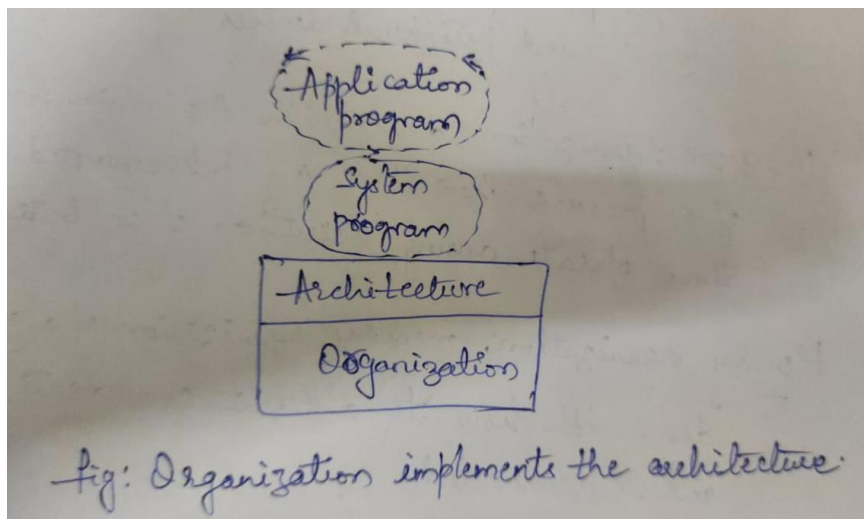
## COMPUTER ARCHITECTURE

It deals with the external view of a computer, that is, it is concerned with the structure and behavior of the computer viewed by a user such as assembly language programmer or machine language programmer.

It can be defined as an interface between hardware and software.

A programmer needs to be aware of:

- ✓ Specific instructions supported by the processor (called as processor instruction set),
- ✓ Instruction formats,
- ✓ Specific registers and their roles,
- ✓ The techniques for accessing the data stored in the memory,
- ✓ The way to perform input-output operations.



System program is directly interacting with the computer hardware. They are written for specific computer architecture.

Eg: Operating Systems, devices, drivers, compilers, etc.

Application programs invoke the services offered by the system programs. They are independent of the architecture and are converted to machine-dependent programs through a system such as a compiler.

## LOGIC GATES USED IN DIGITAL COMPUTER

Binary information is represented in digital computers by physical quantities called signals. Electrical signals such as voltages exist through out the computers in either one of the two recognizable state. The two states represent a binary variable that can be equal to 1 or 0.

Eg: A digital computer utilize a signal of 3volts to represent binary 1 and 0.5volts to represent binary 0. The input terminal of digital circuits will accept binary signals of only 3 and 0.5 volts to represent binary input and output corresponding to 1 or 0 resp.

At the core level, computer communicates in the form of 0 or 1, which is nothing but low and high voltage signals.

**The manipulation of binary** information is done by logic circuits called gates. Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied.



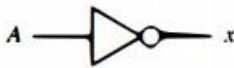
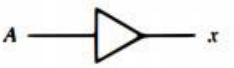




### **LOGIC GATES:**

Binary logic deals with binary variables and with operations that assume a large meaning. It is used to describe in algebraic, or tabular form, the manipulation done by logic circuits called Gates.

Gates are blocks of hardware that produce graphic symbol and it's operation can be described by means of an algebraic expression. The input-output relationship of the binary variables for each gate can be represented by a Truth-Table.

List of Logic Gates:

1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR

Name	Graphic symbol	Algebraic function	Truth table															
AND		$x = A \cdot B$ or $x = AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$x = A + B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$x = A'$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																	
0	1																	
1	0																	
Buffer		$x = A$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	x	0	0	1	1									
A	x																	
0	0																	
1	1																	
NAND		$x = (AB)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$x = (A + B)'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

## REGISTER TRANSFER AND MICROOPERATIONS

- ✓ 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.

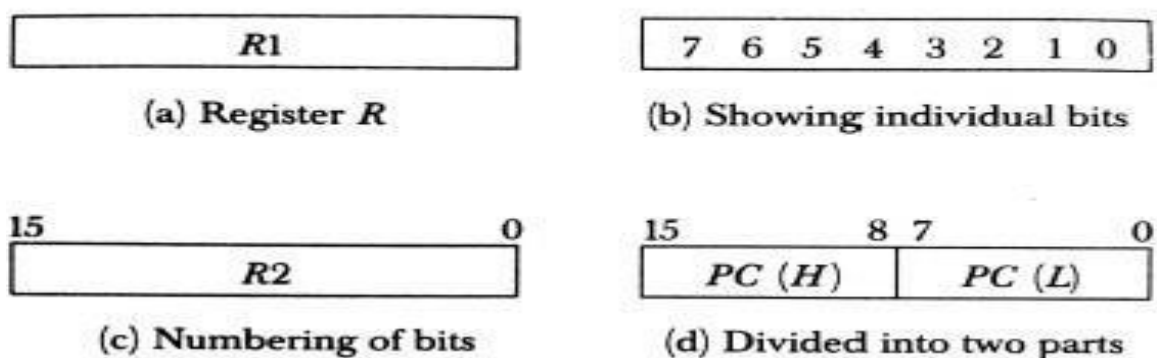
## REGISTER TRANSFER LANGUAGE

- ❑ 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 **RI** (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** Block diagram of register.



- ❑ Figure 4-1 shows 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, 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 byte and  $PC (8-15)$  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  $R2 \leftarrow R1$  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  $R1$  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.

$\text{if } (P=1) \text{ then } R2 \leftarrow R1$

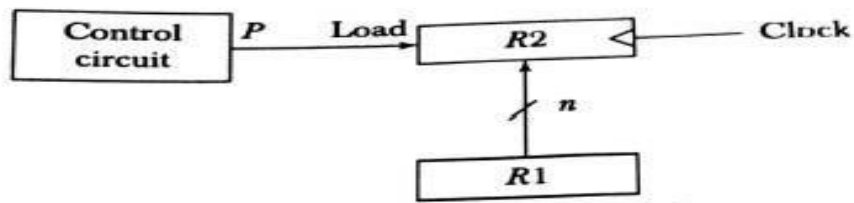
- $P$  is the control signal generated by a control section.
- We can separate the control variables from the register transfer operation by specifying a ***Control Function***.
- Control function is a Boolean variable that is equal to 0 or 1.
- control function is included in the statement as

$P: R2 \leftarrow R1$

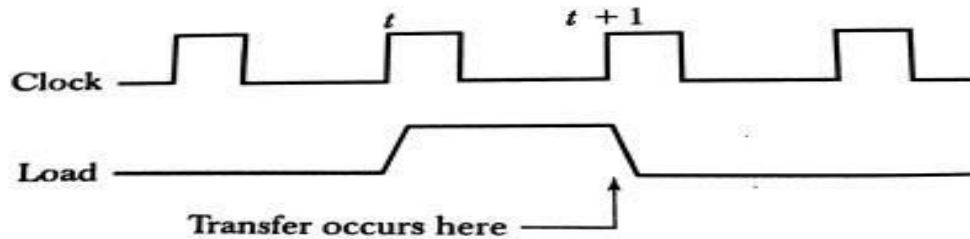
- Control condition is terminated by a colon implies transfer operation be executed by the hardware only if  $P=1$ .
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.
- Figure 4-2 shows the block diagram that depicts the transfer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

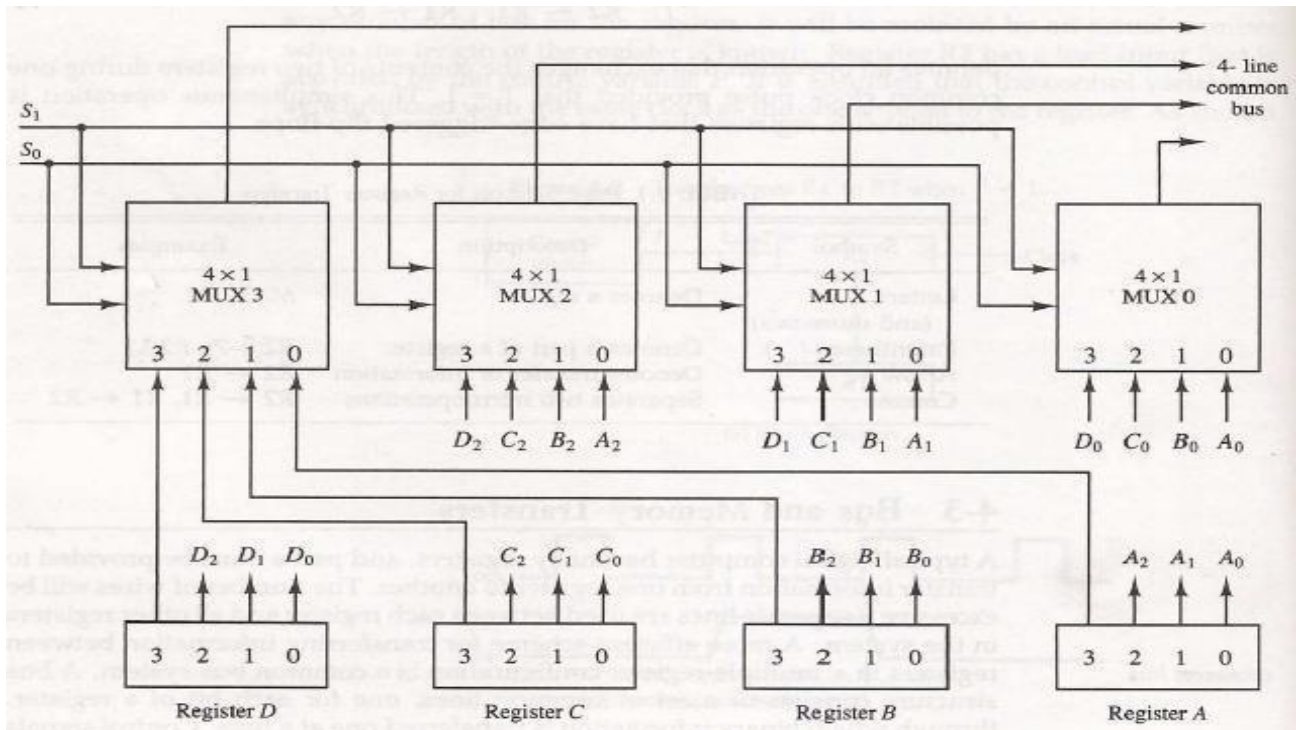
denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T=1.

### 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 the registers, and similarly for the other two bits.
- ❑ The two selection lines  $S_i$  and  $S_o$  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 included in the statement, the register transfer is symbolized as follows:

$BUS \leftarrow C, R1 \leftarrow BUS$

- ☐ 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.

$R1 \leftarrow C$

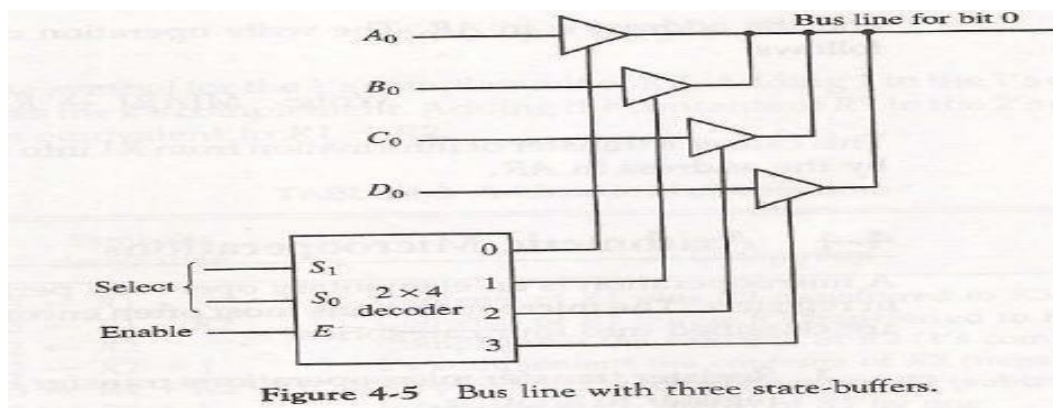
### THREE-STATE BUS BUFFERS:

- ☐ 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.

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



- ☐ The graphic symbol of a three-state buffer gate is shown in Fig. 4-4.
- ☐ 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:  $DR \leftarrow M[AR]$

- ☐ 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:  $M[AR] \leftarrow R1$

## 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-operations move information from source register to destination register.
- Other three types of micro-operations 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

$$R3 \leftarrow R1 + R2 + 1$$

- 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  $R1 - R2$ .

## 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:

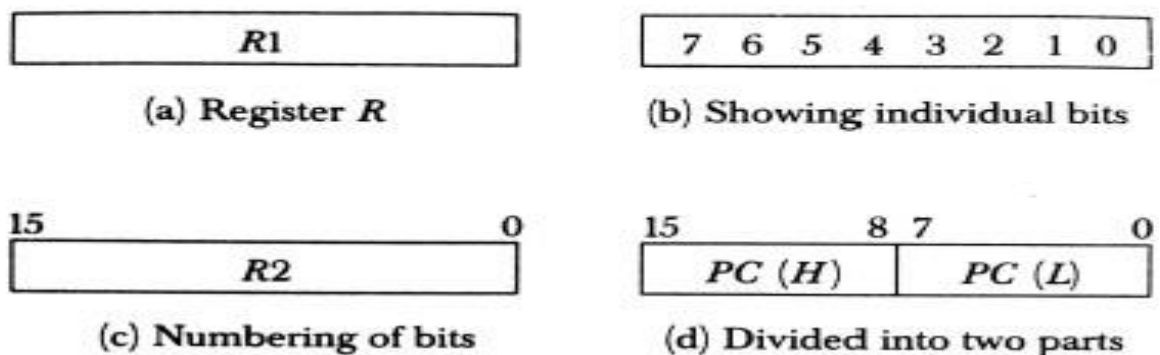
- The symbolic notation used to describe the micro-operation transfer among registers is called RTL (Register Transfer Language).
- The use of *symbols* instead of a *narrative explanation* provides an organized and concise manner for listing the micro-operation sequences in registers and the control 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 **RI** (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 byte and *PC* (8-15) or *PC* (*H*) to the high-order byte.

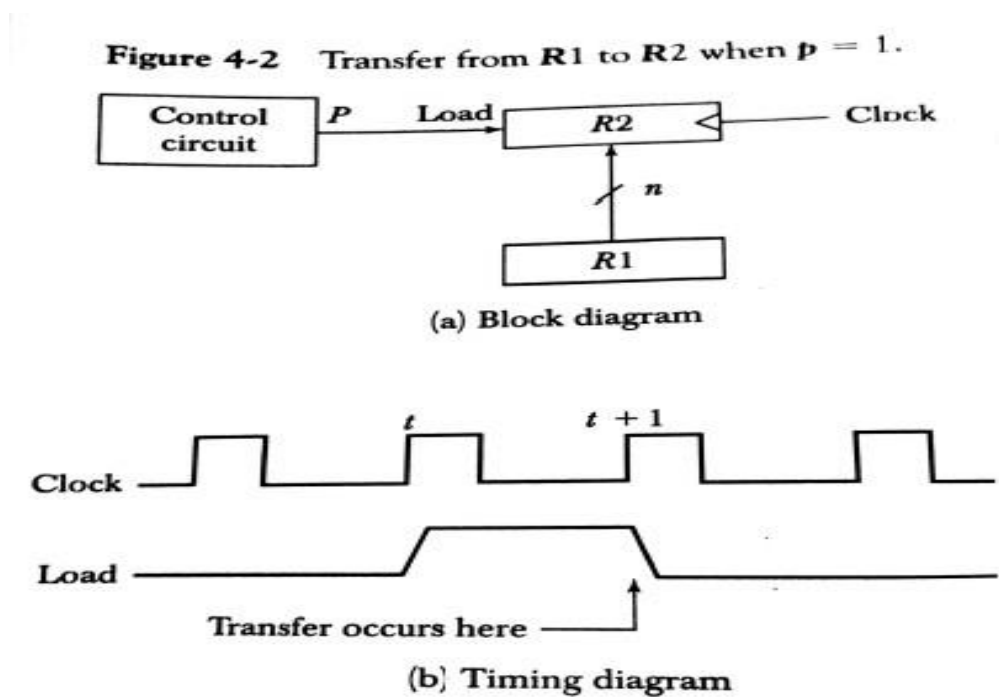
### Register Transfer:

- 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  

$$P: R2 \leftarrow R1$$
- 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.



- 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

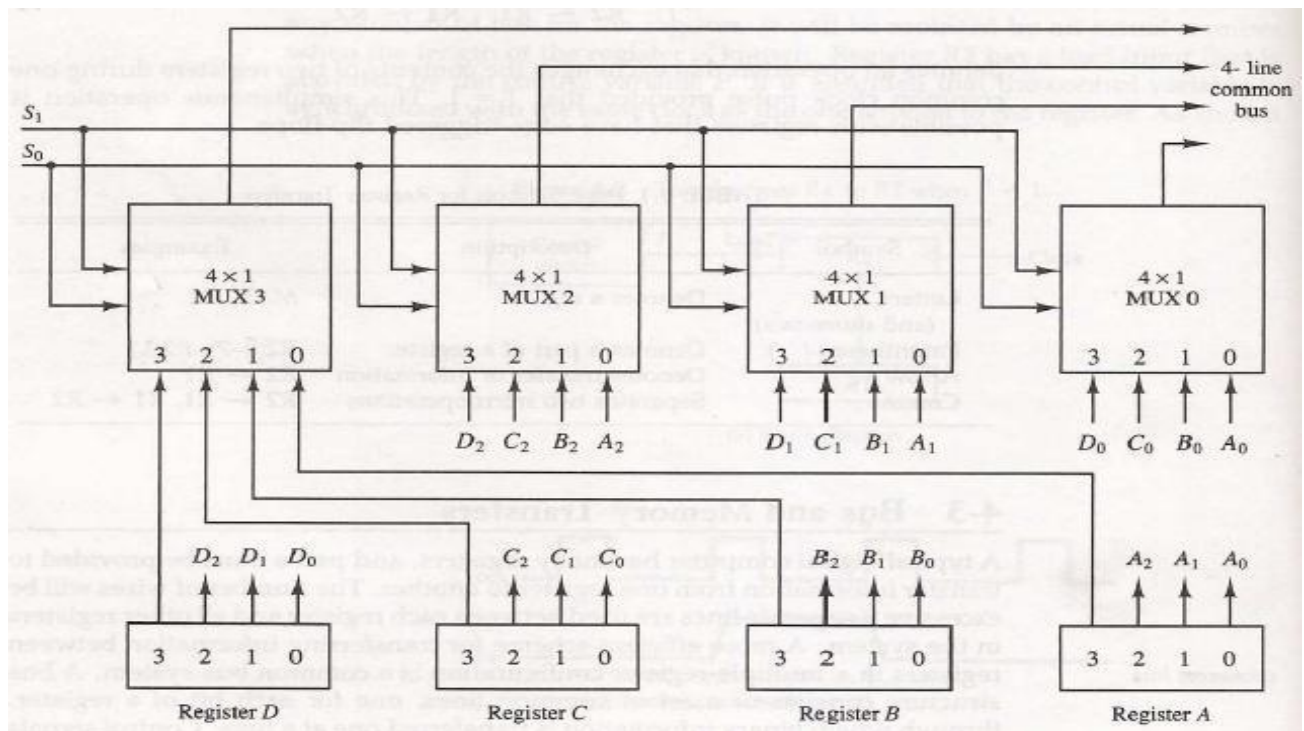
- A comma is used to separate two or more operations that are executed at the same time.
- The statement  
**T : R2 ← R1, R1 ← R2** (exchange operation)  
denotes an operation that exchanges the contents of two registers during one common clock pulse provided that  $T=1$ .

### **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 the registers, and similarly for the other two bits.
- 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.
- 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 included in the statement, the register transfer is symbolized as follows:
- $$\text{BUS} \leftarrow C, R1 \leftarrow \text{BUS}$$
- 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.

$$R1 \leftarrow C$$

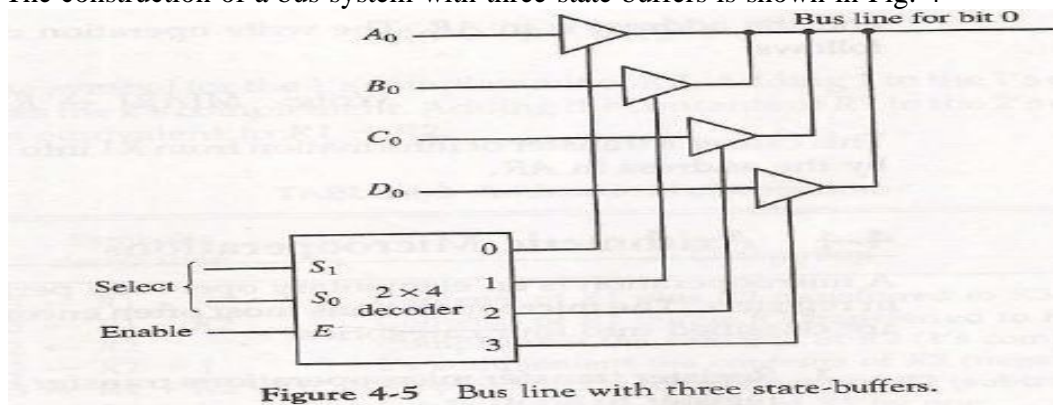
### Three-State Bus Buffers:

- 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



- 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:  $DR \leftarrow M[AR]$

- 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:  $M[AR] \leftarrow R1$

## 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 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.

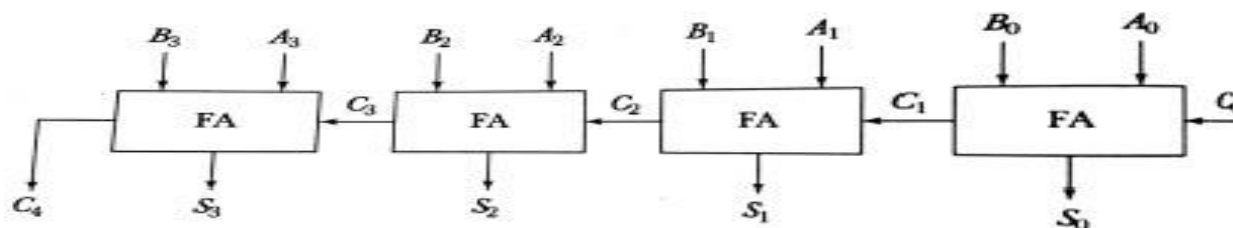


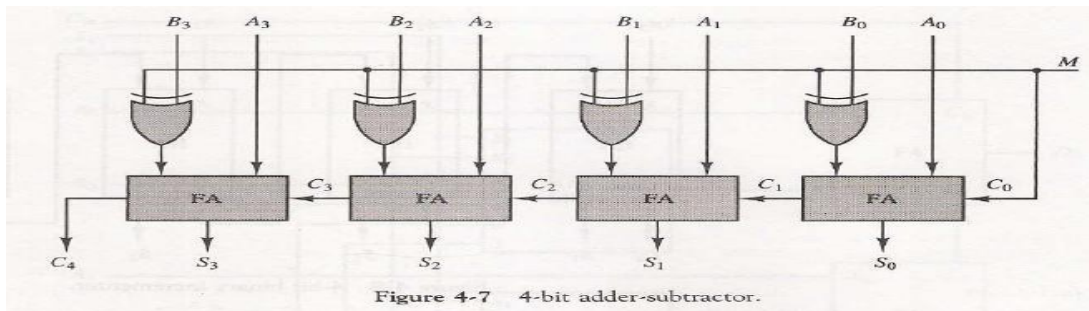
Figure 4-6 4-bit binary adder.

- 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  $C_0$  and the output carry is  $C_4$ . The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.



## Binary Adder – Subtractor:

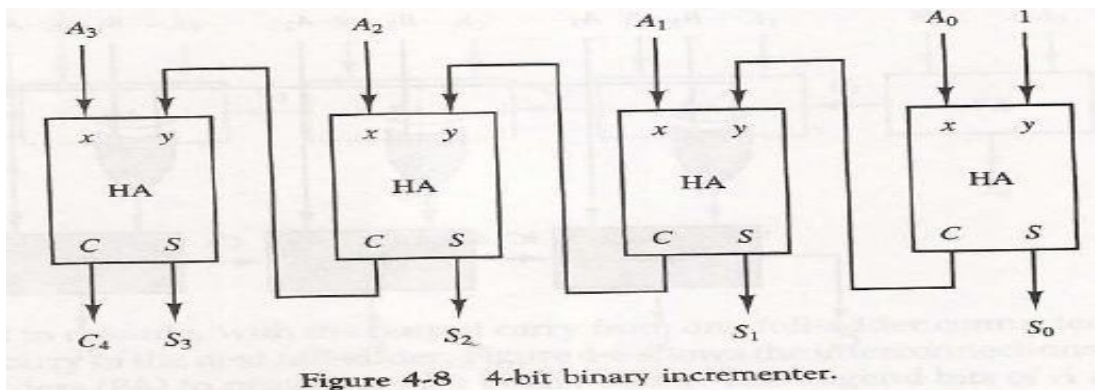
- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- A 4-bit adder-subtractor circuit is shown in Fig. 4-7.



- 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  $B \text{ xor } 0 = B$ . The full-adders receive the value of  $B$ , the input carry is 0, and the circuit performs  $A$  plus  $B$ .
- When  $M = 1$ , we have  $B \text{ xor } 1 = B'$  and  $C_0 = 1$ .
- 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.

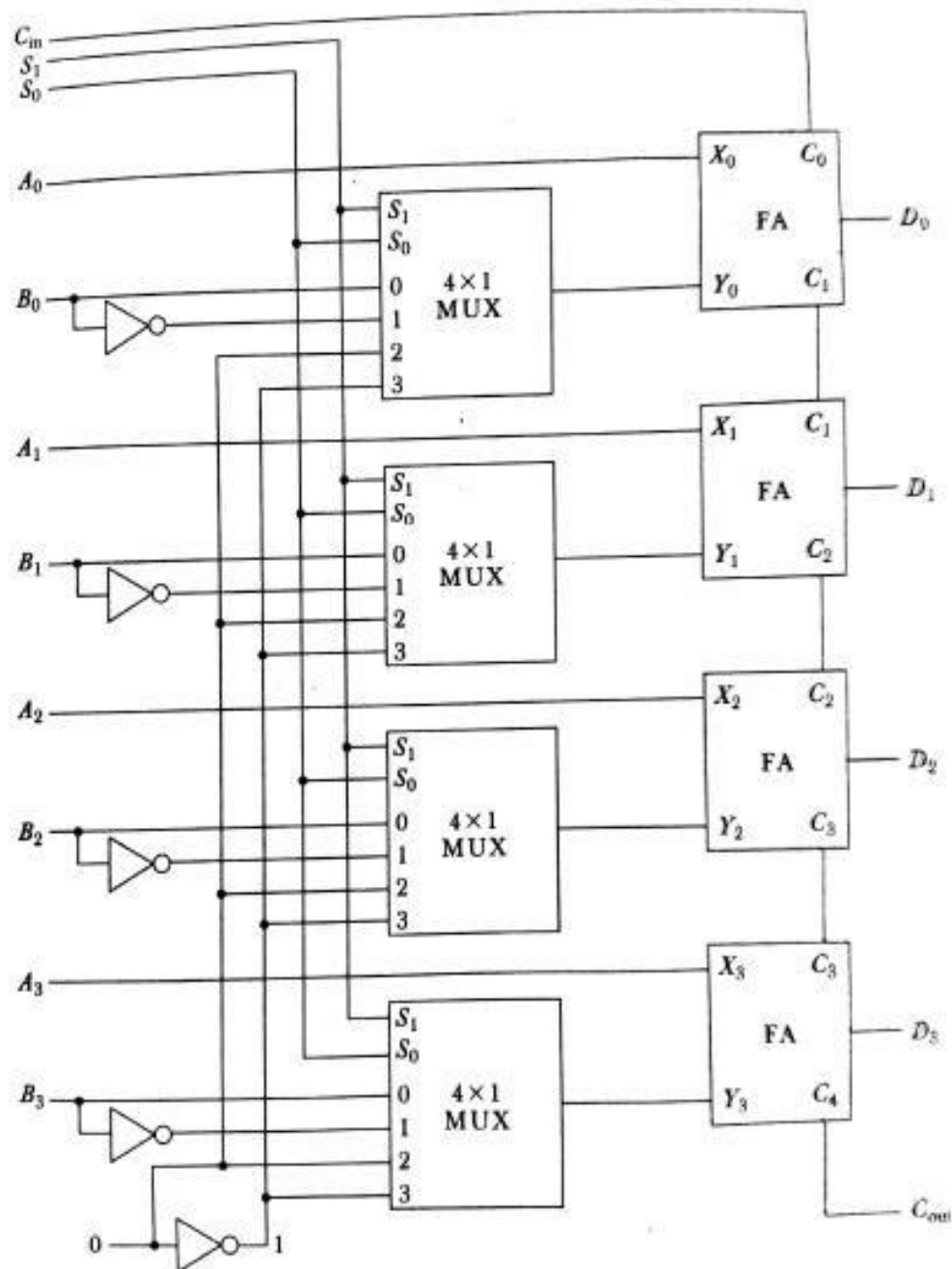


- One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.
- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.
- The circuit 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  $S_0$  through  $S_3$  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 that constitute 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 $Y$	Output $D = A + Y + C_{in}$	Microoperation
$S_1$	$S_0$	$C_{in}$			
0	0	0	$B$	$D = A + B$	Add
0	0	1	$B$	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	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  $C_{in} = 0$ , the output  $D = A + B$ .
  - ✓ If  $C_{in} = 1$ , 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  $D = A + \bar{B} + 1$ . This produces  $A$  plus the 2's complement of  $B$ , which is equivalent to a subtraction of  $A - B$ .
  - ✓ When  $C_{in} = 0$  then  $D = A + \bar{B}$ . 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 1'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 from input 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 R1 and R2 is symbolized by the statement

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

- 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.

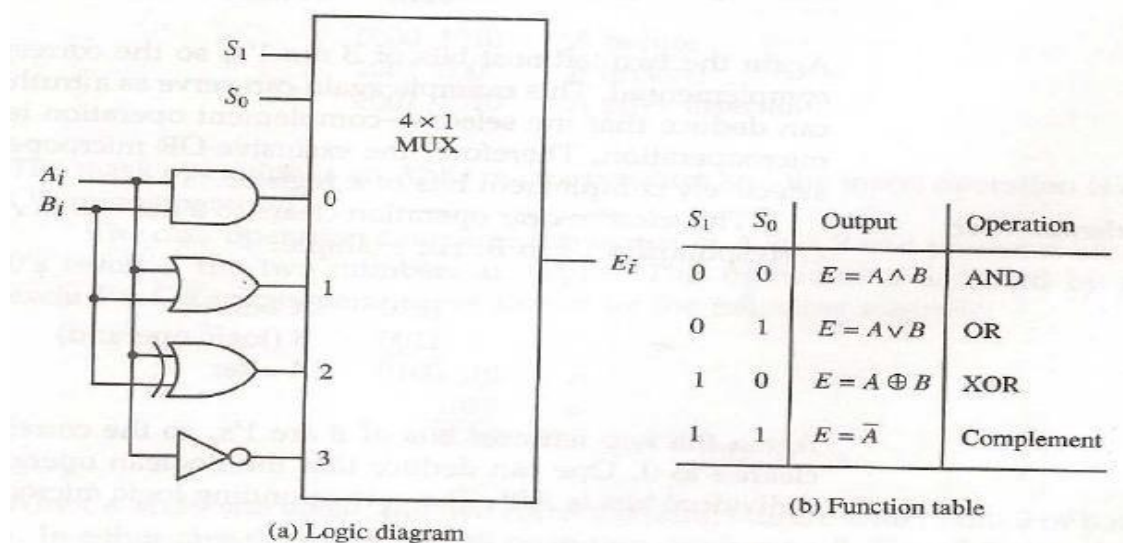
TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer $A$
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	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
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement $B$
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement $A$
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow \text{all 1's}$	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.



## 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 1'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
<u>1100</u>	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
<u>1100</u>	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 1's in B. For example:

1010	A before
<u>1100</u>	B (logic operand)
0010	A after

- ✓ The corresponding logic microoperation is

$$A \leftarrow A \wedge \bar{B}$$

- Mask
  - ✓ The *mask* operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. The mask operation is an AND micro operation as seen from the following numerical example:

0110	1010	A before
<u>0000</u>	<u>1111</u>	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	$A \leftarrow A \oplus B$

## 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.

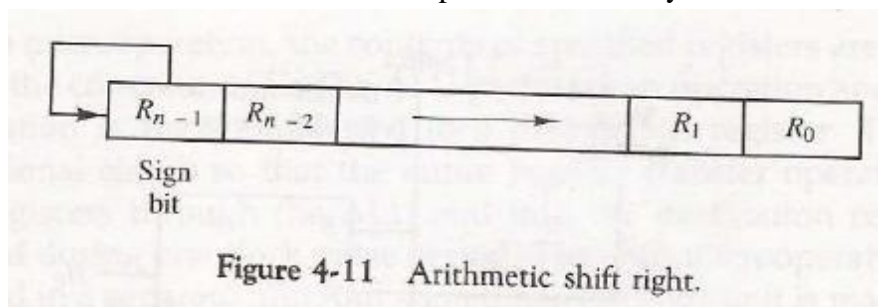
TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow shl R$	Shift-left register R
$R \leftarrow shr R$	Shift-right register R
$R \leftarrow cil R$	Circular shift-left register R
$R \leftarrow cir R$	Circular shift-right register R
$R \leftarrow ashl R$	Arithmetic shift-left R
$R \leftarrow ashr R$	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 during a 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.



#### 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 types of shifts.



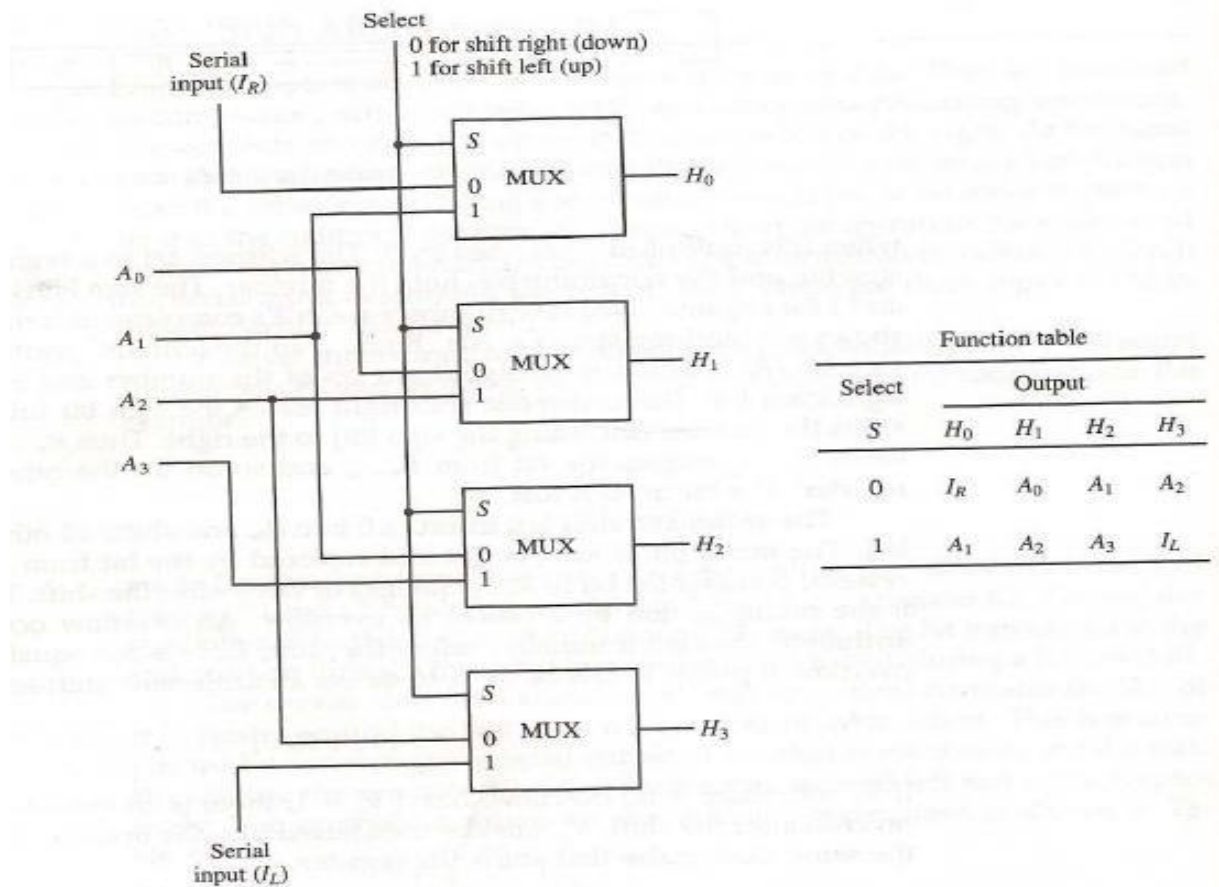
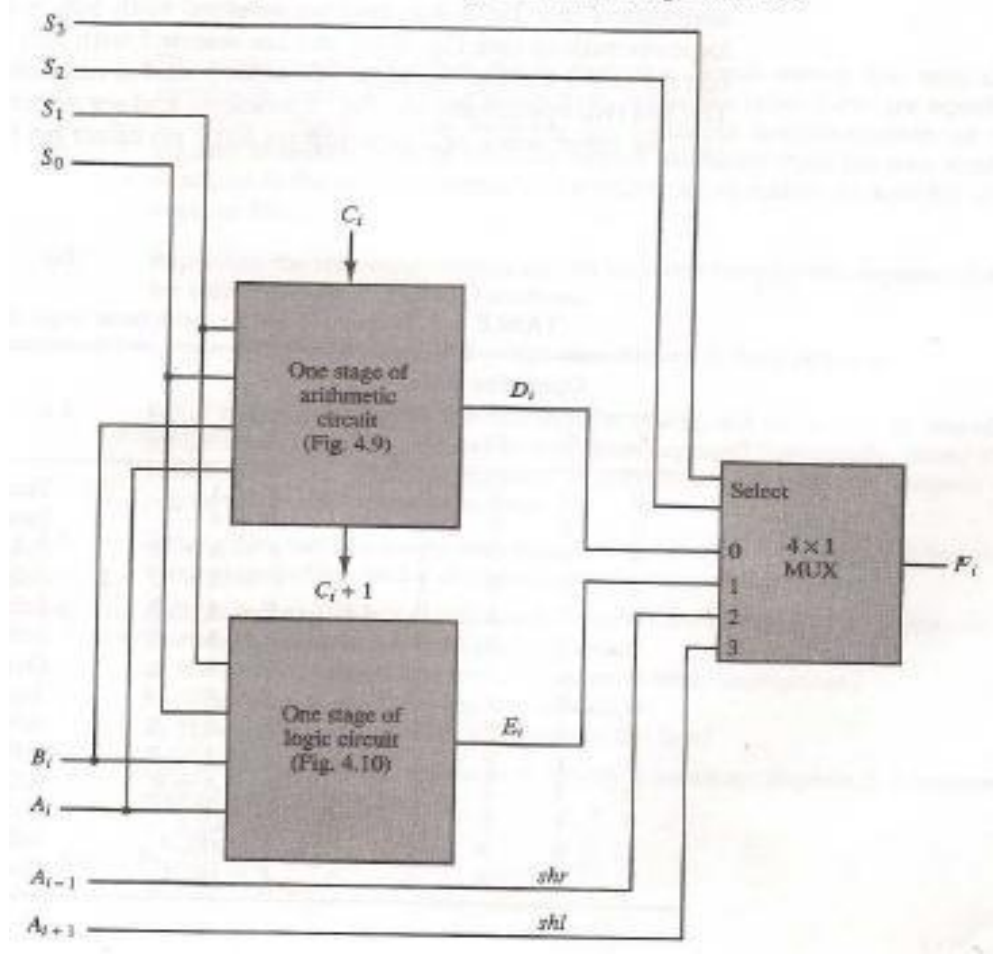


Figure 4-12 4-bit combinational circuit shifter.

### 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				$C_{in}$	Operation	Function
$S_3$	$S_2$	$S_1$	$S_0$			
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	x	$F = A \wedge B$	AND
0	1	0	1	x	$F = A \vee B$	OR
0	1	1	0	x	$F = A \oplus B$	XOR
0	1	1	1	x	$F = \overline{A}$	Complement A
1	0	x	x	x	$F = shr A$	Shift right A into F
1	1	x	x	x	$F = shl A$	Shift left A into F

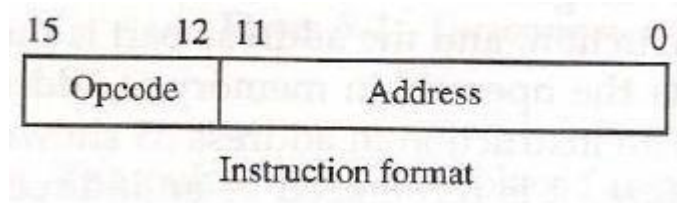
# BASIC COMPUTER ORGANIZATION AND DESIGN

## CONTENTS:

- ✓ Instruction Codes
- ✓ Computer Registers
- ✓ Computer Instructions
- ✓ Timing And Control
- ✓ Instruction Cycle
- ✓ Register – Reference Instructions
- ✓ Memory – Reference Instructions
- ✓ Input – Output And Interrupt

## 1. Instruction Codes:

- The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
- Internal organization of a computer is defined by the sequence of micro-operations it performs on data stored in its registers.
- Computer can be instructed about the specific sequence of operations it must perform.
- User controls this process by means of a Program.
- **Program:** set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- **Instruction:** a binary code that specifies a sequence of micro-operations for the computer.
- The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. – *Instruction Cycle*
- **Instruction Code:** group of bits that instruct the computer to perform specific operation.
- Instruction code is usually divided into two parts: Opcode and address(operand)

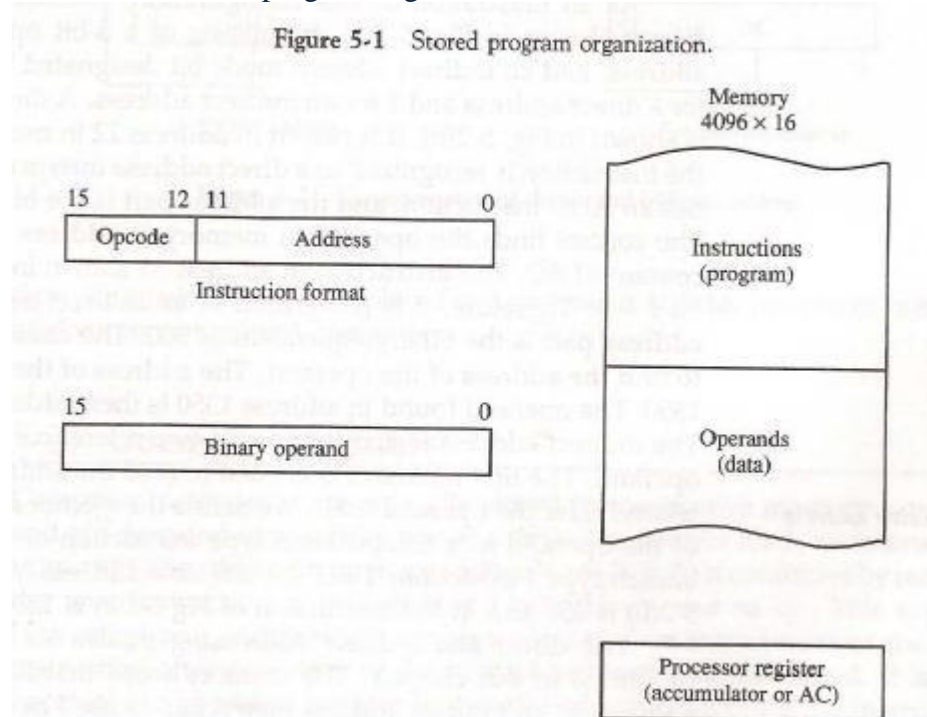


- Operation Code (opcode):
  - ✓ group of bits that define the operation
  - ✓ Eg: add, subtract, multiply, shift, complement.
  - ✓ No. of bits required for opcode depends on no. of operations available in computer.
  - ✓  $n$  bit opcode  $\geq 2^n$  (or less) operations
- Address (operand):
  - ✓ specifies the location of operands (registers or memory words)
  - ✓ Memory words are specified by their address
  - ✓ Registers are specified by their  $k$ -bit binary code

- ✓  $k\text{-bit address} \geq 2^k \text{ registers}$

## Stored Program Organization:

- The ability to store and execute instructions is the most important property of a general-purpose computer. That type of stored program concept is called stored program organization.
- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.
- The below figure shows the 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 the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- **Accumulator (AC):**
  - ✓ Computers that have a single-processor register usually assign to it the name accumulator and label it AC.
  - ✓ The operation is performed with the memory operand and the content of AC.

## Addressing of Operand:

- Sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an **immediate operand**.
- When the second part specifies the address of an operand, the instruction is said to have a **direct address**.
- When second part of the instruction designate an address of a memory word in which the address of the operand is found such instruction have **indirect address**.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- The instruction code format shown in Fig. 5-2(a). 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.

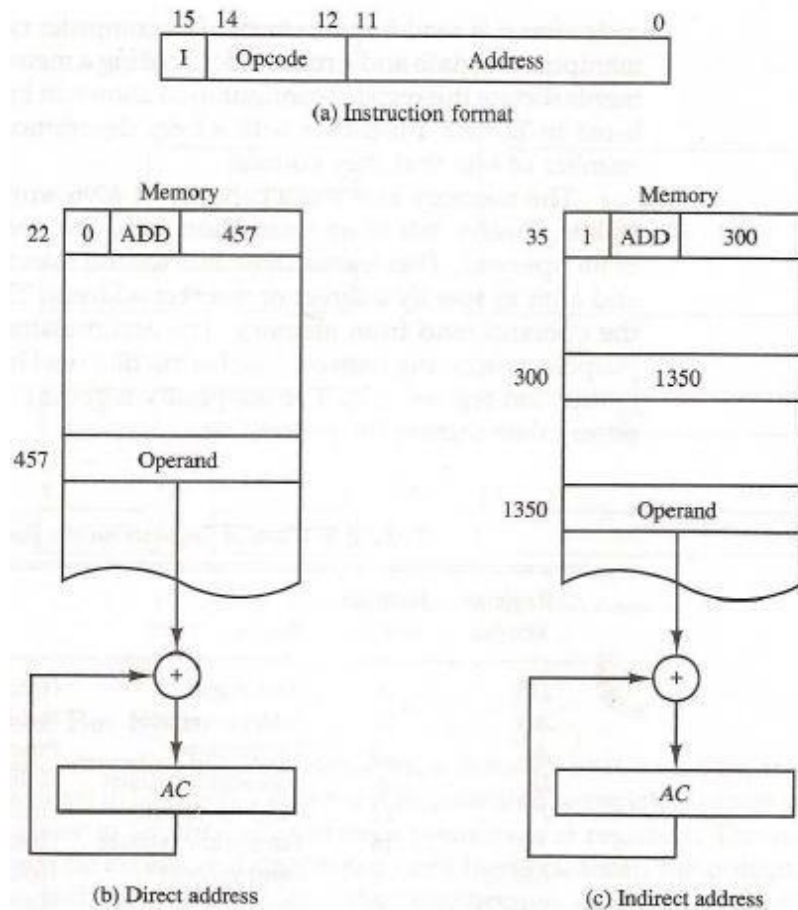


Figure 5-2 Demonstration of direct and indirect address.

- A direct address instruction is shown in Fig. 5-2(b).
- 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 Fig. 5-2(c) has a mode bit I = 1.
- Therefore, it is 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 **effective address** to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction.
- Thus the effective address in the instruction of Fig. 5-2(b) is 457 and in the instruction of Fig 5-2(c) is 1350.

## 2. Computer Registers:

- What is the need for computer registers?
  - ✓ The need of the registers in computer for
    - Instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed (**PC**).
    - Necessary to provide a register in the control unit for storing the instruction code after it is read from memory (**IR**).
    - Needs processor registers for manipulating data (AC and TR) and a register for holding a memory address (**AR**).
- The above requirements dictate the register configuration shown in Fig. 5-3.



- The registers are also listed in Table 5.1 together with a brief description of their function and the number of bits that they contain.

**TABLE 5-1** List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
<i>DR</i>	16	Data register	Holds memory operand
<i>AR</i>	12	Address register	Holds address for memory
<i>AC</i>	16	Accumulator	Processor register
<i>IR</i>	16	Instruction register	Holds instruction code
<i>PC</i>	12	Program counter	Holds address of instruction
<i>TR</i>	16	Temporary register	Holds temporary data
<i>INPR</i>	8	Input register	Holds input character
<i>OUTR</i>	8	Output register	Holds output character

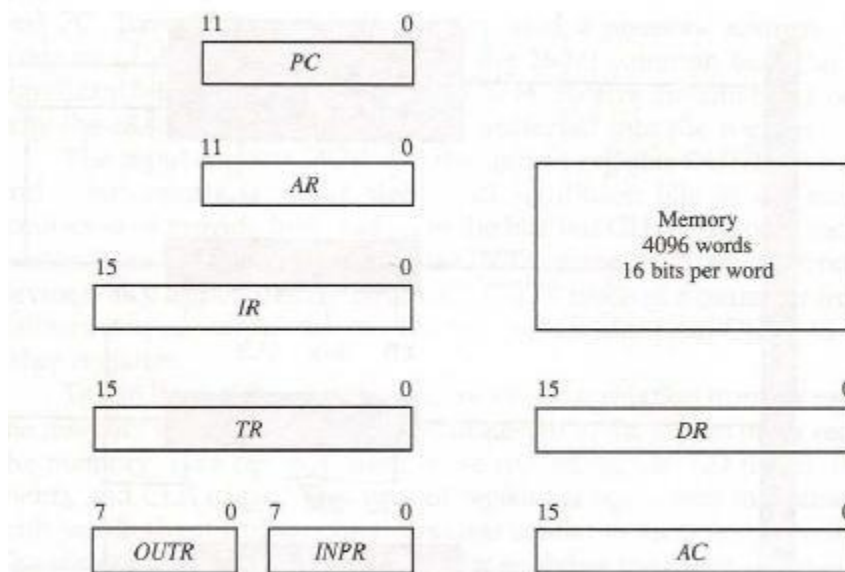


Figure 5-3 Basic computer registers and memory.

- 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 since this is the width of a memory address.
- 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.
- 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.

### 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 registers.
- A more efficient scheme for transferring information in a system with many registers 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 Fig. 5-4.
- 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  $S_2$ ,  $S_1$ , and  $S_0$ .
- The number along each output shows the decimal equivalent of the required binary selection.
- For example, the number along the output of *DR* is 3. The 16-bit outputs of *DR* are placed on the bus lines when  $S_2S_1S_0 = 011$ .

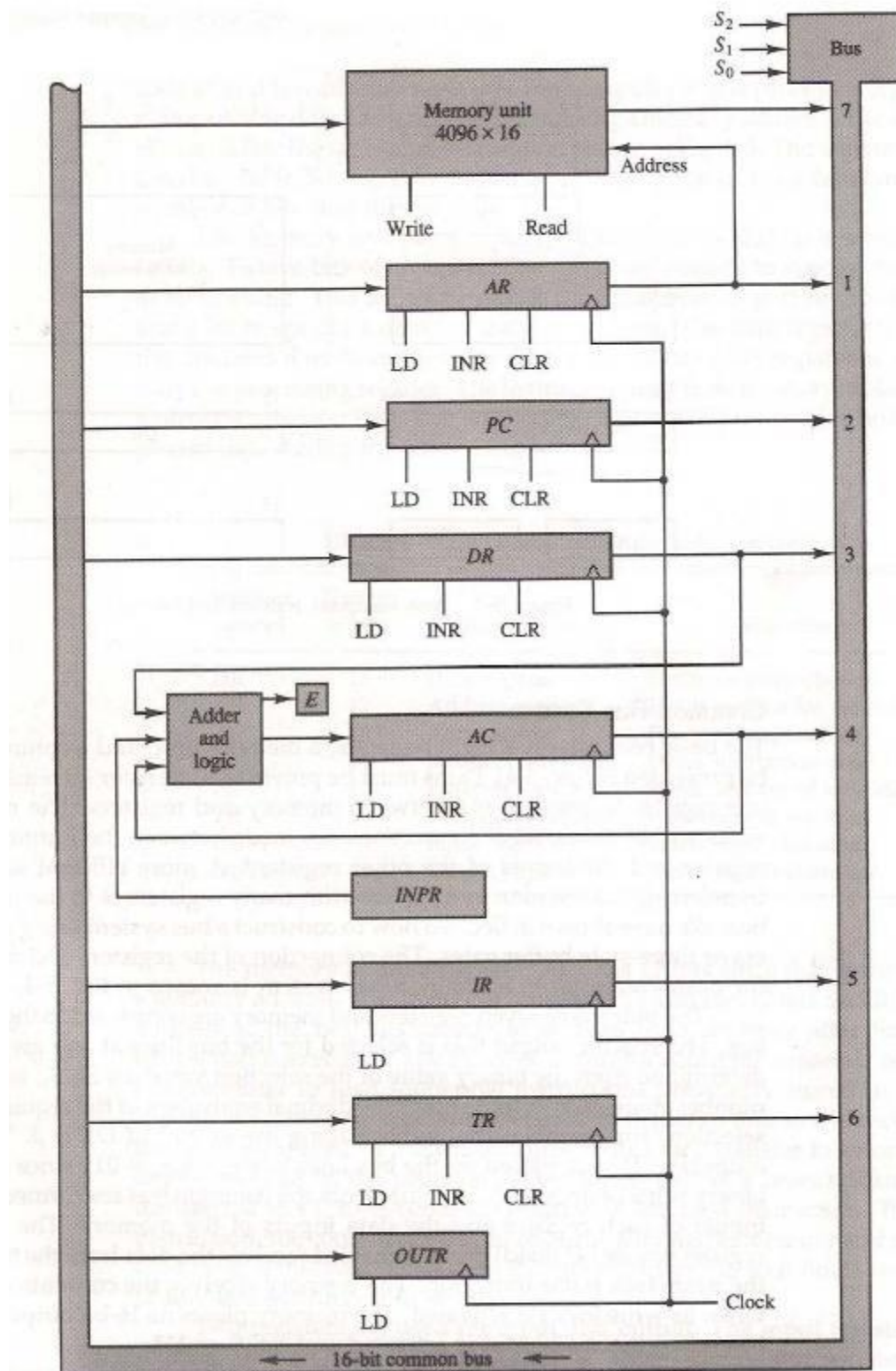


Figure 5-4 Basic computer registers connected to a common bus.

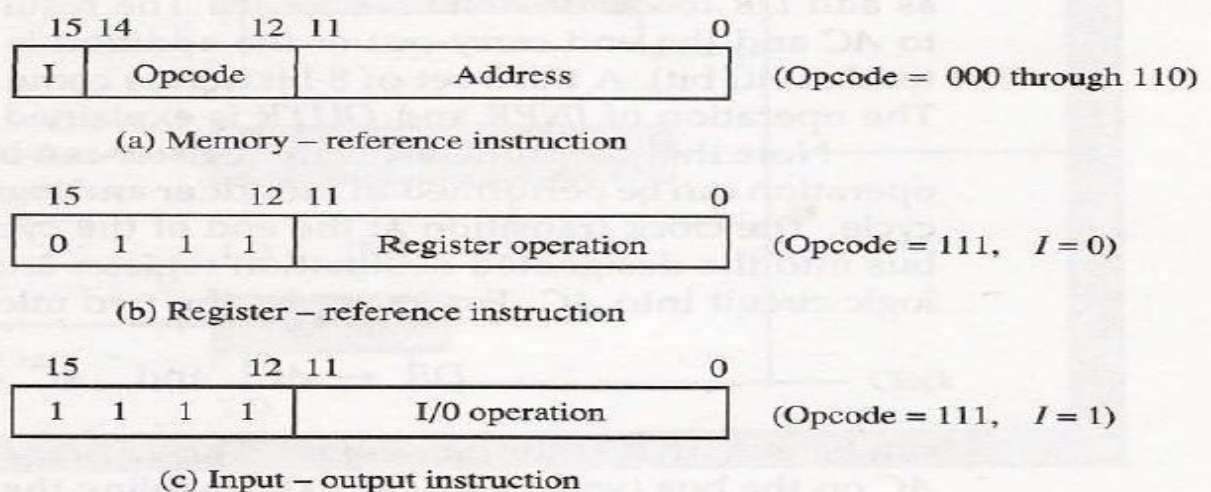
- The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.

- 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 when the read input is activated and  $S_2S_1S_0 = 111$ .
- 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* or *PC* receives 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.
- They 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.
- This is because *INPR* receives a character from an input device which is then transferred to *AC*.
- *OUTR* receives a character from *AC* and delivers it to an output device.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).
- This type of register is equivalent to a binary counter with parallel load and synchronous clear.
- Two registers have only a LD input.
- The input data and output data of the memory are connected to the common bus, but the memory address is connected to *AR*.
- Therefore, *AR* must always be used to specify a memory address.
- The 16 inputs of *AC* come from an adder and logic circuit. This circuit has three sets of inputs.
  - One set of 16-bit inputs come from the outputs of *AC*.
  - Another set of 16-bit inputs come from the data register *DR*.
  - 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).
  - A third set of 8-bit inputs come from the input register *INPR*.
- The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.
- For example, the two microoperations  $DR \leftarrow AC$  and  $AC \leftarrow DR$  can be executed at the same time.
- This can be done by placing the content of *AC* on the bus (with  $S_2S_1S_0 = 100$ ), enabling the LD (load) input of *DR*, transferring the content of *DR* through the adder and logic circuit into *AC*, and enabling the LD (load) input of *AC*, all during the same clock cycle.

### 3. Computer Instructions:

- The basic computer has three instruction code formats, as shown in Fig. 5-5. Each format has 16 bits.

**Figure 5-5 Basic computer instruction formats.**





- The operation code (opcode) part of the instruction contains three bits and the meaning of 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 1.11 with a 0 in the leftmost bit (bit 15) of the instruction.
- A register-reference instruction specifies an operation on the AC register. So an operand from memory is not needed. Therefore, the other 12 bits are used to specify the operation 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.
- The instructions for the computer are listed in Table 5-2.

**TABLE 5-2 Basic Computer Instructions**

Symbol	Hexadecimal code		Description
	<i>I</i> = 0	<i>I</i> = 1	
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear <i>E</i>
CMA	7200		Complement AC
CME	7100		Complement <i>E</i>
CIR	7080		Circulate right AC and <i>E</i>
CIL	7040		Circulate left AC and <i>E</i>
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if <i>E</i> is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

- The symbol designation is a three-letter word and represents an abbreviation intended for programmers and users.
- The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

### Instruction Set Completeness:

- A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
  - Arithmetic, logical, and shift instructions
  - Data Instructions (for moving information to and from memory and processor registers)
  - Program control or Branch
  - Input and output instructions
- There is one arithmetic instruction, ADD, and two related instructions, complement AC(CMA) and increment AC(INC). With these three instructions we can add and subtract binary numbers when negative numbers are in signed-2's complement representation.
- The circulate instructions, CIR and CIL; can be used for arithmetic shifts as well as any other type of shifts desired.
- There are three logic operations: AND, complement AC (CMA), and clear AC(CLA). The AND and complement provide a NAND operation.
- Moving information from memory to AC is accomplished with the load AC (LDA) instruction. Storing information from AC into memory is done with the store AC (STA) instruction.
- The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions.
- The input (INP) and output (OUT) instructions cause information to be transferred between the computer and external devices.

### 4. Timing and Control:

- The timing for all registers in the basic computer is controlled by a master clock generator.
- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.
- There are two major types of control organization:
  - *Hardwired control*
  - *Microprogrammed control*
- The differences between hardwired and microprogrammed control are

Hardwired control	Microprogrammed control
✓ The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.	✓ The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
✓ The advantage that it can be optimized to produce a fast mode of operation.	✓ Compared with the hardwired control operation is slow.
✓ Requires changes in the wiring among the various components if the design has to be modified or changed.	✓ Required changes or modifications can be done by updating the microprogram in control memory.



- The block diagram of the hardwired control unit is shown in Fig. 5-6.
- It consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register (IR). It is divided into three parts: The I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols  $D_0$  through  $D_7$ .
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit sequence counter can count in binary from 0 through 15.

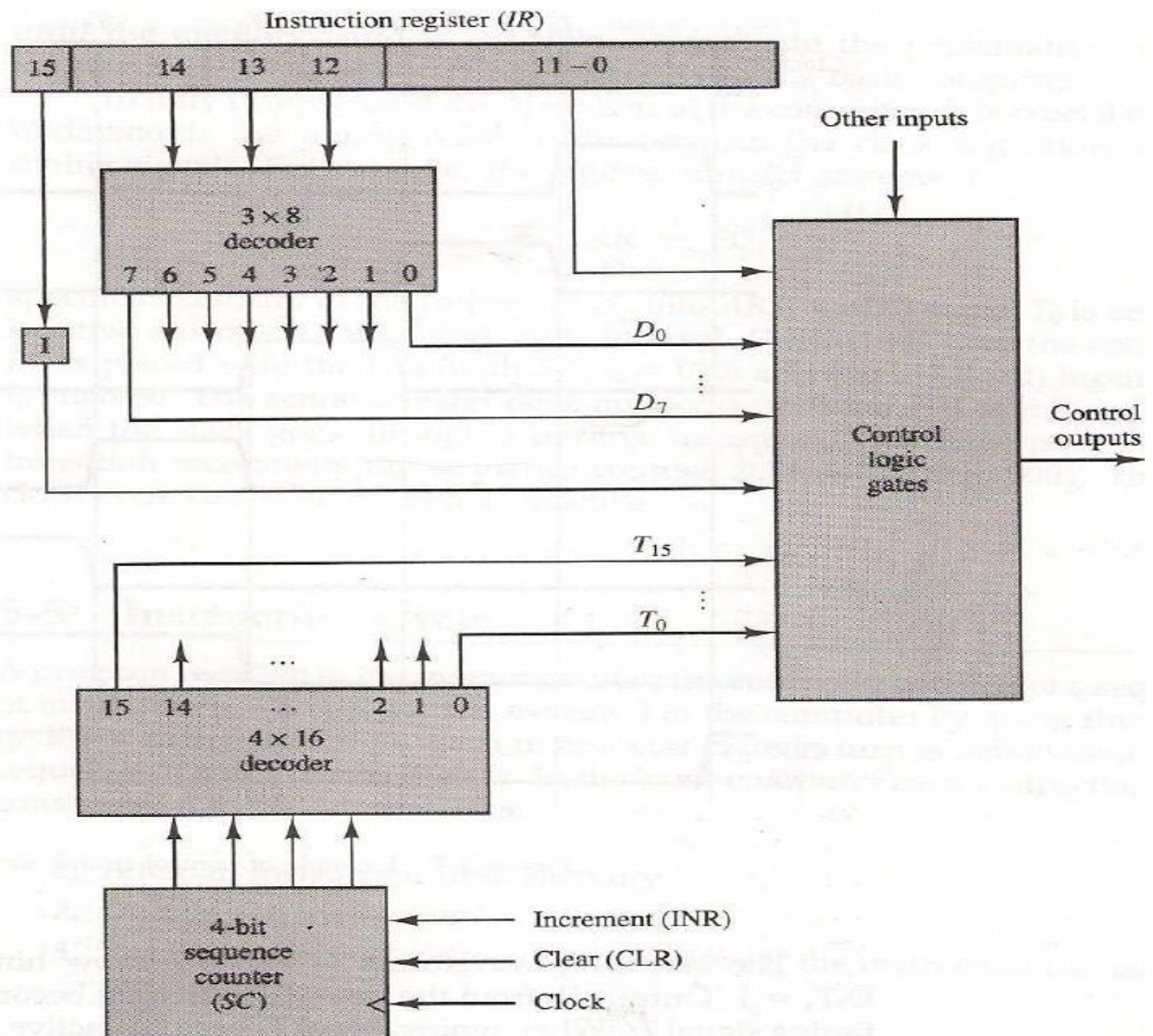


Figure 5-6 Control unit of basic computer.

- The outputs of the counter are decoded into 16 timing signals  $T_0$  through  $T_{15}$ .
- The sequence counter SC can be incremented or cleared synchronously.
- The counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.
- As an example, consider the case where SC is incremented to provide timing signals  $T_0$ ,  $T_1$ ,  $T_2$ ,  $T_3$  and  $T_4$  in sequence. At time  $T_4$ , SC is cleared to 0 if decoder output  $D_3$  is active.
- This is expressed symbolically by the statement

**$D_3T_4: SC \leftarrow 0$**

- The timing diagram of Fig. 5-7 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 timing signal  $T_0$  out of the decoder.  $T_0$  is active during one clock cycle.
- $SC$  is incremented with every positive clock transition, unless its CLR input is active.
- This produces the sequence of timing signals  $T_0, T_1, T_2, T_3, T_4$  and so on, as shown in the diagram.
- The last three waveforms in Fig.5-7 show how  $SC$  is cleared when  $D_3T_4 = 1$ .
- Output  $D_3$  from the operation decoder becomes active at the end of timing signal  $T_2$ .
- When timing signal  $T_4$  becomes active, the output of the AND gate that implements the control function  $D_3T_4$  becomes active.
- This signal is applied to the CLR input of  $SC$ . On the next positive clock transition (the one marked  $T_4$  in the diagram) the counter is cleared to 0.
- This causes the timing signal  $T_0$  to become active instead of  $T_5$  that would have been active if  $SC$  were incremented instead of cleared.

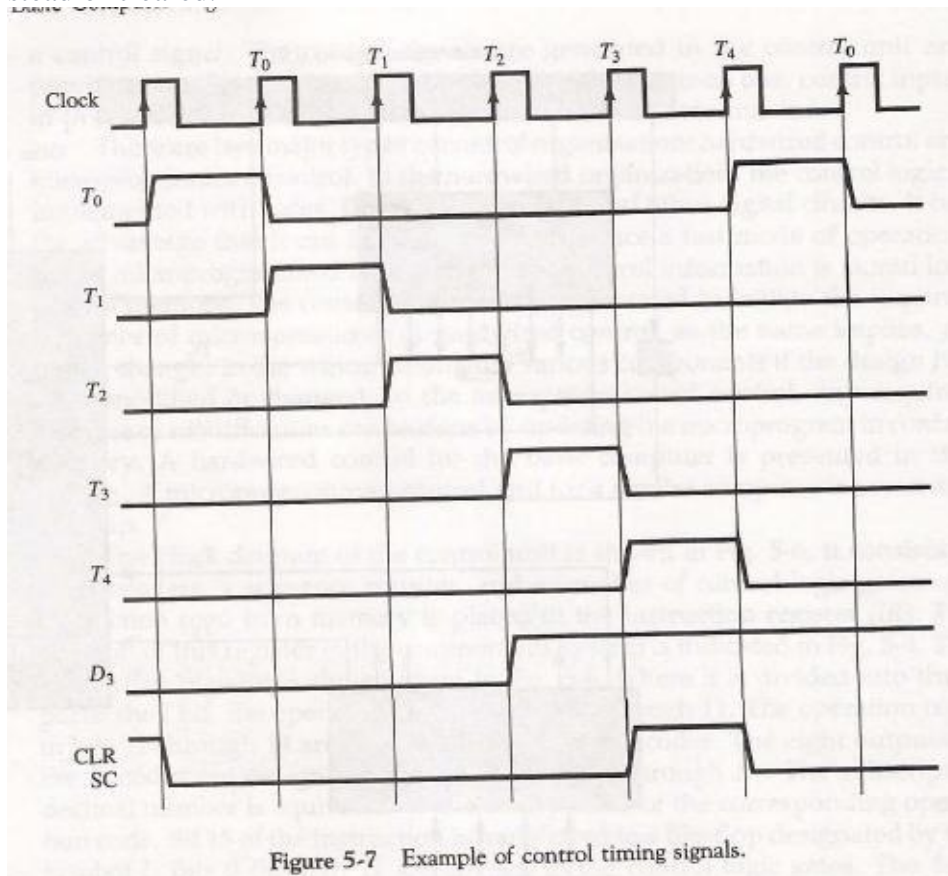


Figure 5-7 Example of control timing signals.

## **5. Instruction Cycle:**

- A program residing in the memory unit of the computer consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.
- 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.
- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.

### **Fetch and Decode:**

- Initially, the program counter PC is loaded with the address of the first instruction in the program.

- The sequence counter  $SC$  is cleared to 0, providing a decoded timing signal  $T_0$ .
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

$T_0: AR \leftarrow PC$   
 $T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$   
 $T_2: D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$

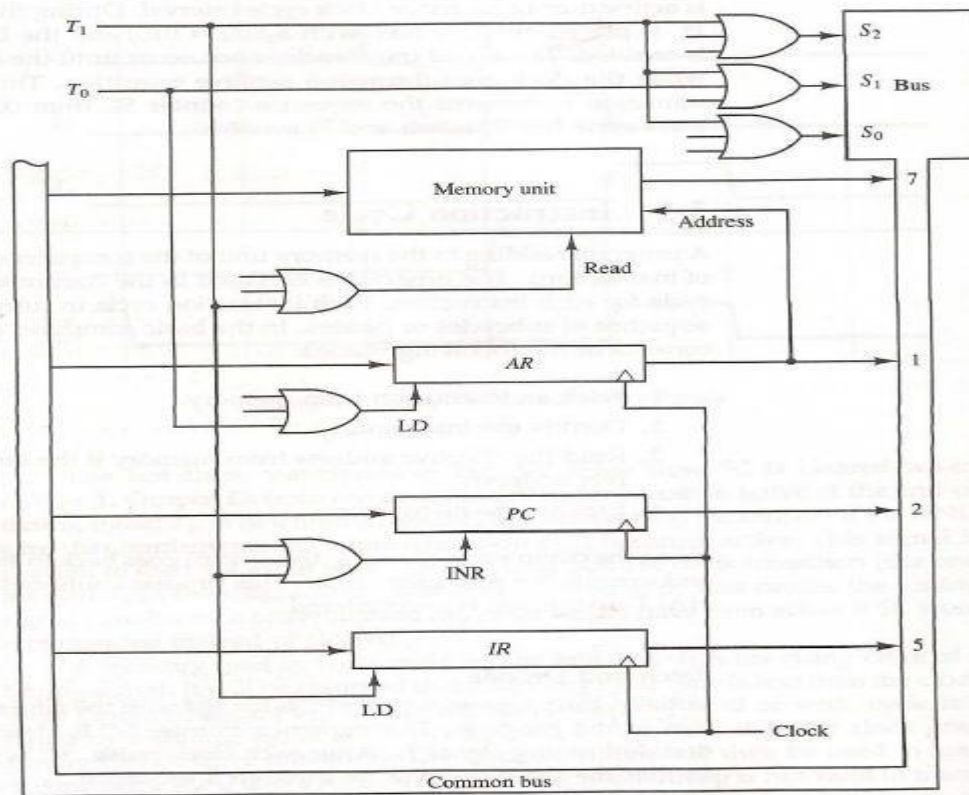


Figure 5-8 Register transfers for the fetch phase.

- Figure 5-8 shows how the first two register transfer statements are implemented in the bus system.
- To provide the data path for the transfer of  $PC$  to  $AR$  we must apply timing signal  $T_0$  to achieve the following connection:
  - Place the content of  $PC$  onto the bus by making the bus selection inputs  $S_2, S_1, S_0$  equal to 010.
  - Transfer the content of the bus to  $AR$  by enabling the  $LD$  input of  $AR$ .
- In order to implement the second statement it is necessary to use timing signal  $T_1$  to provide the following connections in the bus system.
  - Enable the read input of memory.
  - Place the content of memory onto the bus by making  $S_2S_1S_0=111$ .
  - Transfer the content of the bus to  $IR$  by enabling the  $LD$  input of  $IR$ .
  - Increment  $PC$  by enabling the  $INR$  input of  $PC$ .
- Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

### Determine the Type of Instruction:

- The timing signal that is active after the decoding is  $T_3$ .
- During time  $T_3$ , the control unit determine the type of instruction that was read from the memory.
- The flowchart of fig.5-9 shows the initial configurations for the instruction cycle and also how the control determines the instruction cycle type after the decoding.
- Decoder output  $D_7$  is equal to 1 if the operation code is equal to binary 111.
- If  $D_7=1$ , the instruction must be a register-reference or input-output type.
- If  $D_7 = 0$ , the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.



- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I.
- If  $D_7 = 0$  and  $I = 1$ , indicates a memory-reference instruction with an indirect address. So it is then necessary to read the effective address from memory.
- If  $D_7 = 0$  and  $I = 0$ , indicates a memory-reference instruction with a direct address.
- If  $D_7 = 1$  and  $I = 0$ , indicates a register-reference instruction.
- If  $D_7 = 0$  and  $I = 1$ , indicates an input-output instruction.
- The three instruction types are subdivided into four separate paths.
- The selected operation is activated with the clock transition associated with timing signal  $T_3$ .
- This can be symbolized as follows:

$D_7 I T_3$ :  $AR \leftarrow M[AR]$   
 $D_7 I' T_3$ : Nothing  
 $D_7 I' T_3$ : Execute a register-reference instruction  
 $D_7 I T_3$ : Execute an input-output instruction

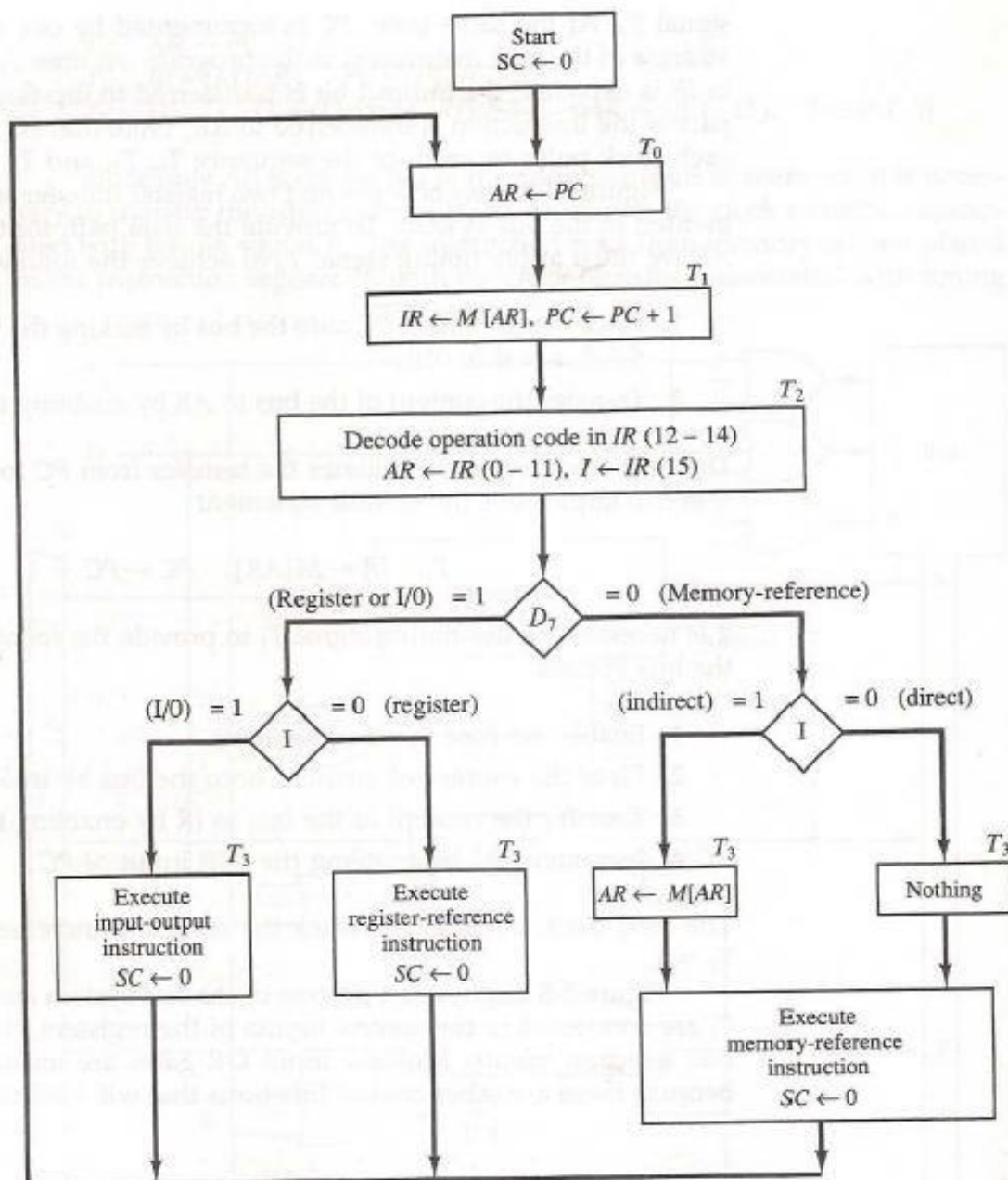


Figure 5-9 Flowchart for instruction cycle (initial configuration).

## Register-Reference Instructions:

- Register-reference instructions are recognized by the control when  $D_7 = 1$  and  $I=0$ .
- These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions.
- These 12 bits are available in IR (0-11).
- The control functions and microoperations for the register-reference instructions are listed in Table 5-3.
- These instructions are executed with the clock transition associated with timing variable  $T_3$ .
- Control function needs the Boolean relation  $D_7I'T_3$ , which we designate for convenience by the symbol  $r$ .
- By assigning the symbol  $B_i$  to bit  $i$  of IR, all control functions can be simply denoted by  $rB_i$ .

TABLE 5-3 Execution of Register-Reference Instructions			
$D_7I'T_3 = r$ (common to all register-reference instructions)			
$IR(i) = B_i$ [bit in $IR(0-11)$ that specifies the operation]			
	$r$ :	$SC \leftarrow 0$	Clear $SC$
CLA	$rB_{11}$ :	$AC \leftarrow 0$	Clear $AC$
CLE	$rB_{10}$ :	$E \leftarrow 0$	Clear $E$
CMA	$rB_9$ :	$AC \leftarrow \overline{AC}$	Complement $AC$
CME	$rB_8$ :	$E \leftarrow \overline{E}$	Complement $E$
CIR	$rB_7$ :	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	$rB_6$ :	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	$rB_5$ :	$AC \leftarrow AC + 1$	Increment $AC$
SPA	$rB_4$ :	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_3$ :	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	$rB_2$ :	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if $AC$ zero
SZE	$rB_1$ :	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if $E$ zero
HLT	$rB_0$ :	$S \leftarrow 0$ ( $S$ is a start-stop flip-flop)	Halt computer

- For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent to  $I'$ .
- The next three bits constitute the operation code and are recognized from decoder output  $D_7$ .
- Bit 11 in IR is 1 and is recognized from  $B_{11}$ . The control function that initiates the microoperation for this instruction is  $D_7I'T_3 B_{11} = rB_{11}$ .
- The execution of a register-reference instruction is completed at time  $T_3$ .
- The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal  $T_0$ .
- 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 a stated condition is satisfied. The skipping of the instruction is achieved by incrementing  $PC$  once again.
- 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.

## 6. Memory-Reference Instructions:

- Table 5-4 lists the seven memory-reference instructions.
- The decoded output  $D_i$  for  $i = 0, 1, 2, 3, 4, 5$ , and 6 from the operation decoder that belongs to each instruction is included in the table.
- The effective address of the instruction is in the address register AR and was placed there during timing signal  $T_2$  when  $I = 0$ , or during timing signal  $T_3$  when  $I = 1$ .
- The execution of the memory-reference instructions starts with timing signal  $T_4$ .

- The symbolic description of each instruction is specified in the table in terms of register transfer notation.

TABLE 5-4 Memory-Reference Instructions		
Symbol	Operation decoder	Symbolic description
AND	$D_0$	$AC \leftarrow AC \wedge M[AR]$
ADD	$D_1$	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	$D_2$	$AC \leftarrow M[AR]$
STA	$D_3$	$M[AR] \leftarrow AC$
BUN	$D_4$	$PC \leftarrow AR$
BSA	$D_5$	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	$D_6$	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

### AND to AC:

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memoryword specified by the effective address.
- The result of the operation is transferred to AC.
- The microoperations that execute this instruction are:

$D_0T_4: DR \leftarrow M[AR]$   
 $D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$

### 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  $C_{out}$  is transferred to the E (extended accumulator) flip-flop.
- The microoperations needed to execute this instruction are

$D_1T_4: DR \leftarrow M[AR]$   
 $D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$

### LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC.
- The microoperations needed to execute this instruction are

$D_2T_4: DR \leftarrow M[AR]$   
 $D_2T_5: AC \leftarrow DR, SC \leftarrow 0$

### STA: Store AC

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



- Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation.

$$D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0$$

#### **BUN:** Branch Unconditionally

- This instruction transfers the program to the instruction specified by the effective address.
- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.
- The instruction is executed with one microoperation:

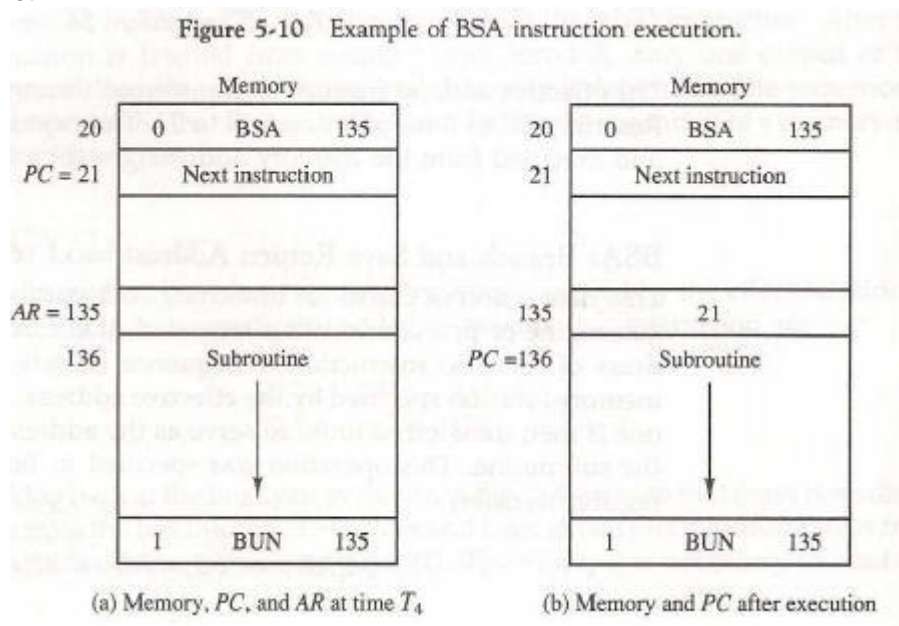
$$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$$

#### **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.
- The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.
- This operation was specified with the following register transfer:

$$M[AR] \leftarrow PC, PC \leftarrow AR + 1$$

- A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10.



- The BSA instruction is assumed to be in memory at address 20.
- The I bit is 0 and the address part of the instruction has the binary equivalent of 135.
- After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address). AR holds the effective address 135.
- This is shown in part (a) of the figure.
- The BSA instruction performs the following numerical operation:  

$$M[135] \leftarrow 21, PC \leftarrow 135 + 1 = 136$$
- The result of this operation is shown in part (b) of the figure.
- The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.
- The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

- When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.
- When the BUN instruction is executed, the effective address 21 is transferred to PC.
- The next instruction cycle finds *PC* with the value 21, so control continues to execute the instruction at the return address.
- The BSA instruction must be executed with a sequence of two microoperations:

$D_5T_4: M[AR] \leftarrow PC, AR \leftarrow AR + 1$   
 $D_5T_5: PC \leftarrow AR, SC \leftarrow 0$

### ISZ: Increment and Skip if Zero

- This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1 to skip the next instruction in the program.
- Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.
- This is done with the following sequence of microoperations:

$D_6T_4: DR \leftarrow M[AR]$   
 $D_6T_5: DR \leftarrow DR + 1$   
 $D_6T_6: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$

### Control Flowchart:

- A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. 5.11.

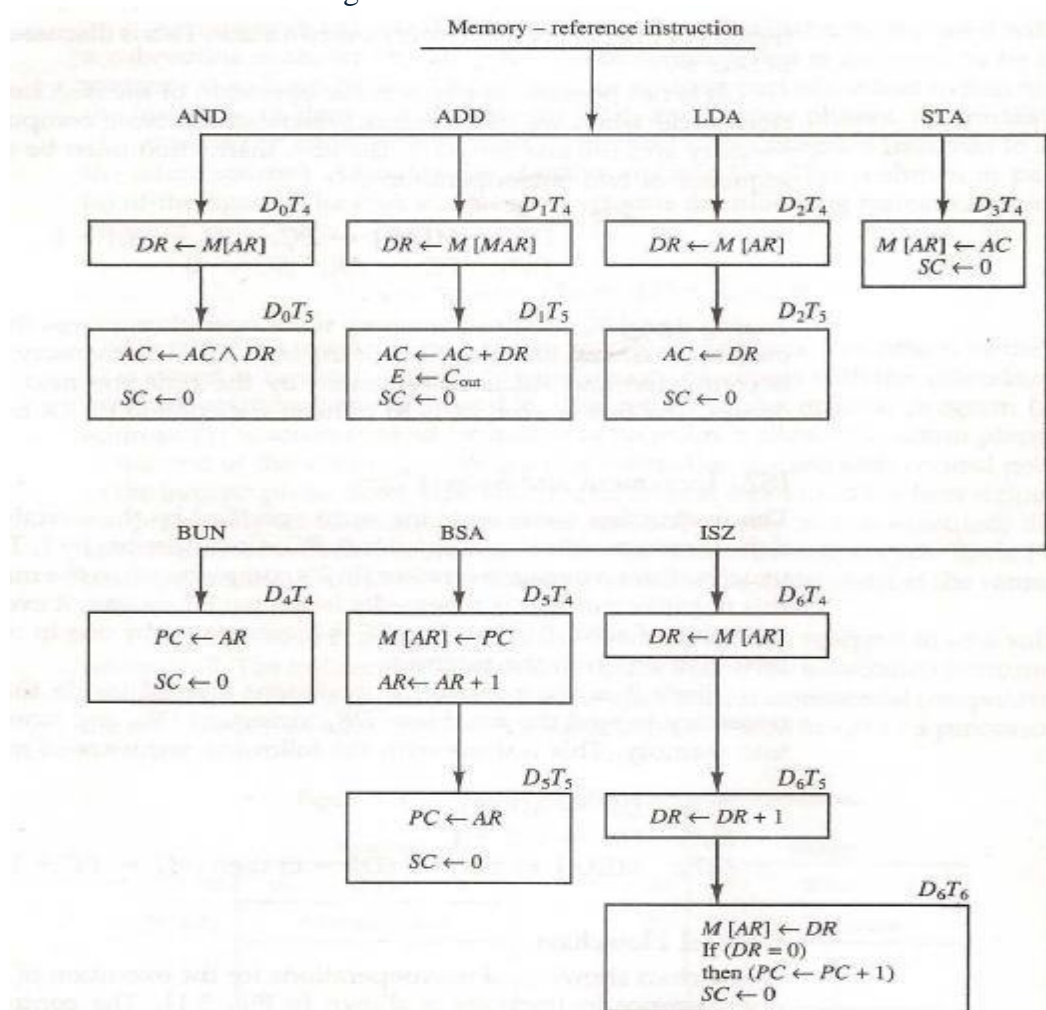


Figure 5-11 Flowchart for memory-reference instructions.

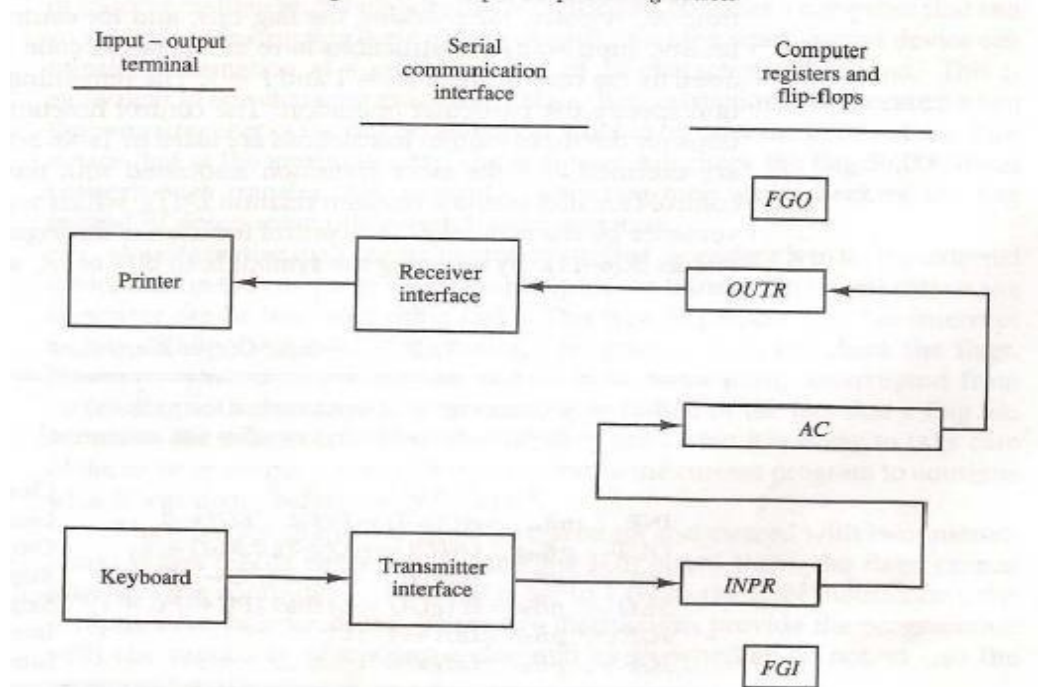
## 7. Input-Output and Interrupt:

- Instructions and data stored in memory must come from some input device.
- Computational results must be transmitted to the user through some output device.
- To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

### **Input-Output Configuration:**

- The terminal sends and receives serial information.
- Each quantity of information has eight 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 the AC in parallel.
- The input—output configuration is shown in Fig. 5-12.

Figure 5-12 Input-output configuration.



- The input register *INPR* consists of eight bits and holds alphanumeric input information.
- The 1-bit input flag *FGI* is a control flip-flop.
- The flag bit 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 output register *OUTR* works similarly but the direction of information flow is reversed.
- 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.

### **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 the interrupt 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 in Table 5-5.

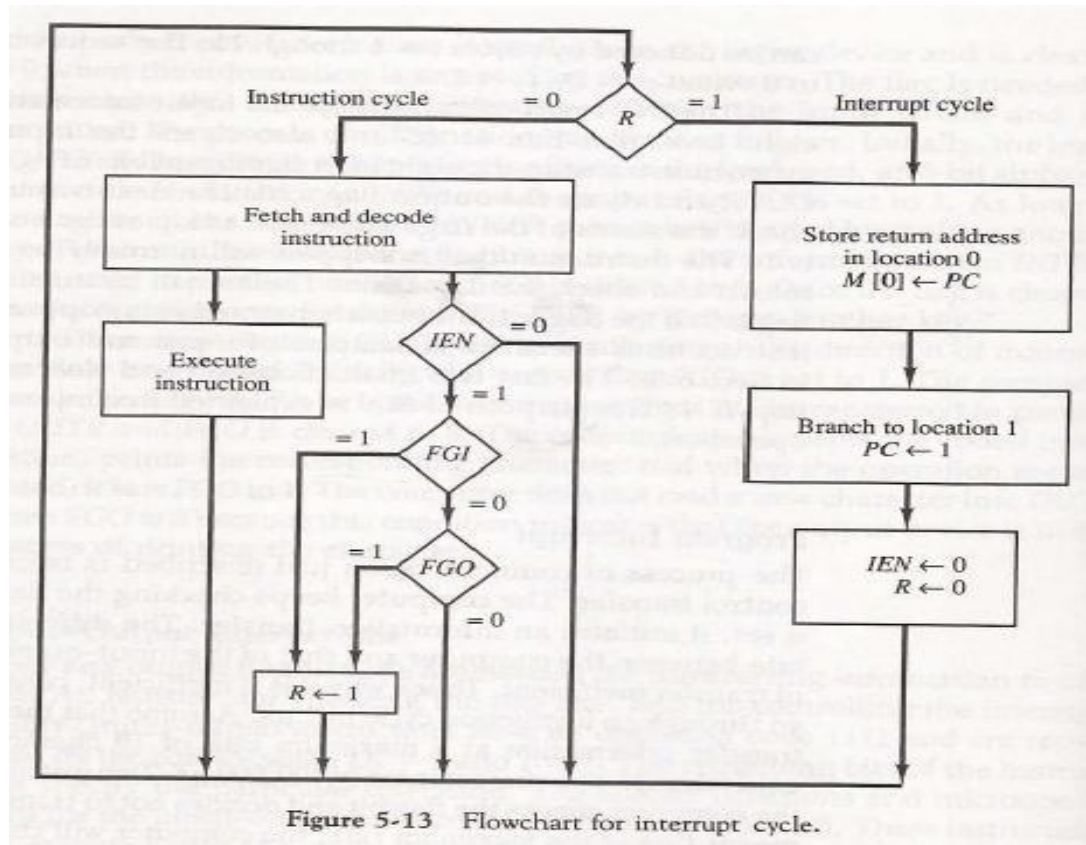
TABLE 5-5 Input-Output Instructions		
$D_7IT_3 = p$ (common to all input-output instructions)		
$IR(i) = B_i$ [bit in $IR(6-11)$ that specifies the instruction]		
	$p$ :	$SC \leftarrow 0$
INP	$pB_{11}$ :	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$
OUT	$pB_{10}$ :	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$
SKI	$pB_9$ :	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	$pB_8$ :	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
IEN	$pB_7$ :	$IEN \leftarrow 1$
IOF	$pB_6$ :	$IEN \leftarrow 0$
		Clear SC
		Input character
		Output character
		Skip on input flag
		Skip on output flag
		Interrupt enable on
		Interrupt enable off

- These instructions are executed with the clock transition associated with timing signal  $T_3$ .
- Each control function needs a Boolean relation  $D_7IT_3$ , which we designate for convenience by the symbol  $p$ .
- The control function is distinguished by one of the bits in  $IR$  (6-11).
- By assigning the symbol  $B_i$  to bit  $i$  of  $IR$ , all control functions can be denoted by  $pB_i$  for  $i = 6$  through 11.
- The sequence counter  $SC$  is cleared to 0 when  $p = D_7IT_3 = 1$ .
- The last two instructions set and clear an interrupt enable flip-flop  $IEN$ .

### Program Interrupt:

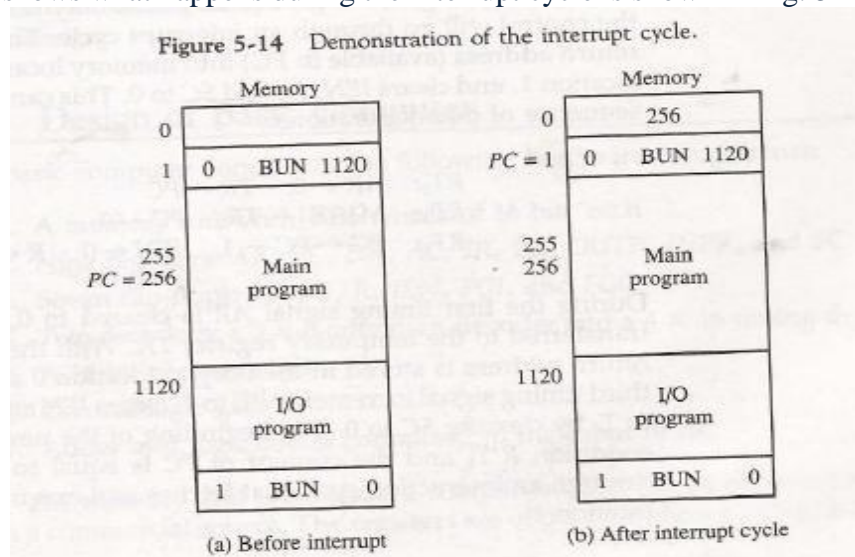
- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- The difference of information flow rate between the computer and that of the input—output device makes this type of transfer inefficient.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check the flags.
- When a flag is set, the computer is momentarily interrupted from the current program.
- The computer deviates momentarily from what it is doing to perform of the input or output transfer.
- It then returns to the current program to continue what it was doing before the interrupt.
- The interrupt enable flip-flop  $IEN$  can be set and cleared with two instructions.
  - When  $IEN$  is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer.
  - When  $IEN$  is set to (with the ION instruction), the computer can be interrupted.
- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. 5-13.
- 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 the control.
- 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 1 while  $IEN = 1$ , 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.





### Interrupt cycle:

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location.
- This location may be a processor register, a memory stack, or a specific memory location.
- An example that shows what happens during the interrupt cycle is shown in Fig. 5-14.



- When an interrupt occurs and R is set to 1 while the control is executing the instruction 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. This is shown in Fig. 5.14(a).
- When control reaches timing signal T<sub>0</sub> and finds that R = 1, it proceeds with the interrupt cycle.
- The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.
- 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 returns to the location where it was interrupted.
- This is shown in Fig. 5-14(b).