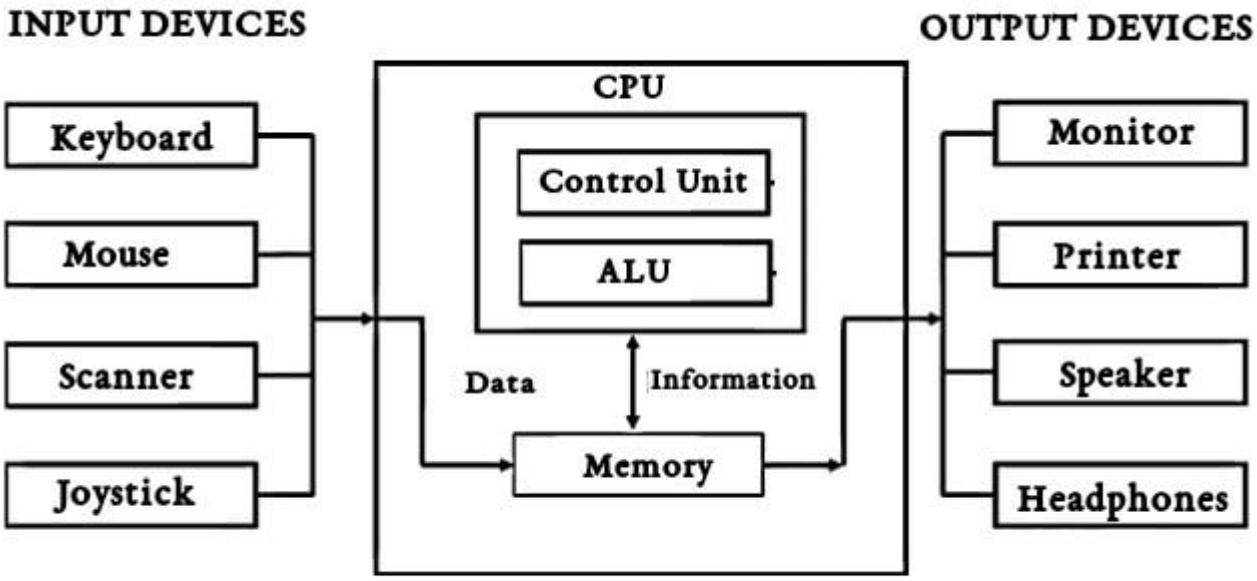
Unit-1

ORGANIZATION AND DESIGN OF DIGITAL COMPUTERS

1. : OVERVIEW OF COMPUTER:

A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption. The different components in the Computer System Architecture are Input Unit, Output Unit, Storage Unit, Arithmetic Logic Unit, Control Unit etc.

A diagram that shows the flow of data between these units is as follows –



Basic Architecture of a Computer

The input data travels from input unit to ALU. Similarly, the computed data travels from ALU to output unit. The data constantly moves from storage unit to ALU and back again. This is because stored data is computed on before being stored again. The control unit controls all the other units as well as their data. Details about all the computer units are −

- **Input Unit**

The input unit provides data to the computer system from the outside. So, basically it links the external environment with the computer. It takes data from the input devices, converts it into machine language and then loads it into the computer system. Keyboard, mouse etc. are the most commonly used input devices.

- **Output Unitsssssss**

The output unit provides the results of computer process to the users i.e it links the computer with the external environment. Most of the output data is the form of audio or video. The different output devices are monitors, printers, speakers, headphones etc.

- **Storage Unit**

Storage unit contains many computer components that are used to store data. It is traditionally divided into primary storage and secondary storage. Primary storage is also known as the main memory and is the memory directly accessible by the CPU. Secondary or external storage is not directly accessible by the CPU. The data from secondary storage needs to be brought into the primary storage before the CPU can use it. Secondary storage contains a large amount of data permanently.

- **Arithmetic Logic Unit**

All the calculations related to the computer system are performed by the arithmetic logic unit. It can perform operations like addition, subtraction, multiplication, division etc. The control unit transfers data from storage unit to arithmetic logic unit when calculations need to be performed. The arithmetic logic unit and the control unit together form the central processing unit.

- **Control Unit**

This unit controls all the other units of the computer system and so is known as its central nervous system. It transfers data throughout the computer as required including from storage unit to central processing unit and vice versa. The control unit also dictates how the memory, input output devices, arithmetic logic unit etc. should behave.

- **Logic gates**

logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels.

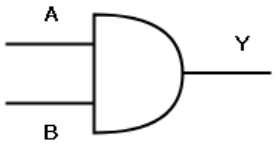
Digital systems are said to be constructed by using logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

AND GATE:

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high.

A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is

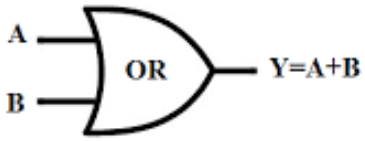
sometimes omitted i.e. AB



Truth Table		
A	B	A . B
1	1	1
1	0	0
0	1	0
0	0	0

OR GATE:

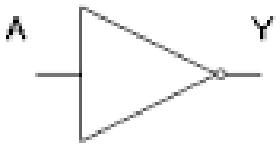
The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs high. A plus (+) is used to show the OR operation.



Inputs		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT GATE:

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs.



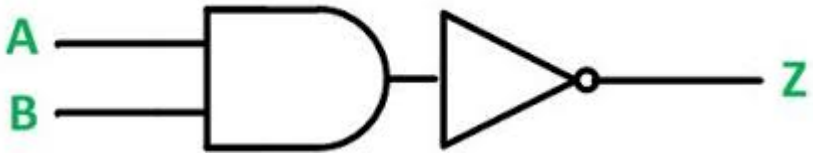
Truth Table	
A	A'
1	0
0	1

NAND GATE:

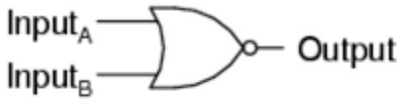
$X = \overline{A.B}$

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

The Logic NAND Gate is a combination of a digital logic AND gate and a NOT gate connected together in series.
The NAND (Not – AND) gate has an output that is normally at logic level "1" and only goes "LOW" to logic level "0" when ALL of its inputs are at logic level "1". The Logic NAND Gate is the reverse or "Complementary" form of the AND gate we have seen previously

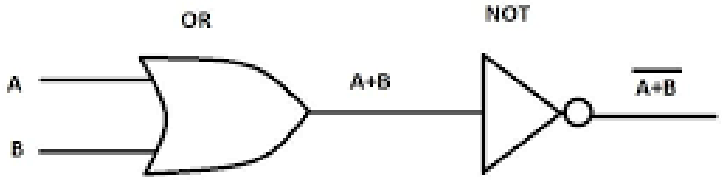


NOR GATE:



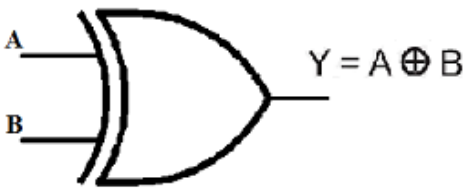
The Logic NOR Gate gate is a combination of the digital logic OR gate and an inverter or NOT gate connected together in series.

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



LOGIC CIRCUIT OF NOR GATE

EX-OR GATE:



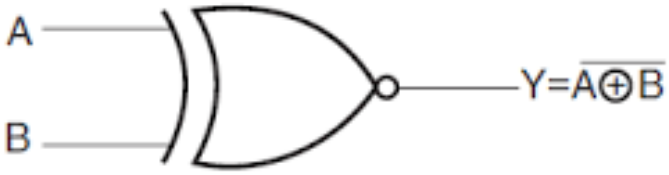
INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-OR gate ONLY goes "HIGH" when both of its two input terminals are at "DIFFERENT" logic levels with respect to each other. If these two inputs, A and B are both at logic level "1" or both at logic level "0" the output is a "0".

$$x = A'B + AB'$$

EX-NOR GATE:

The Exclusive-NOR Gate function is a digital logic gate that is the reverse or complementary form of the Exclusive-OR function.



$$Y = (\overline{A \oplus B}) = (A.B + \overline{A}.\overline{B})$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

- **BASIC OF FLIP FLOPS AND LATCHES:**

A flip-flop is a circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs.

A latch is a device with exactly two stable states and these states are high-output and low-output. A latch has a feedback path, to retain the information. Hence, latches can be memory devices and can store one bit of data. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Both are used as data storage elements. It is the basic storage element in sequential logic.

LATCH	FLIP – FLOP
Latches do not require clock signal.	Flip – flops have clock signals
A latch is an asynchronous device.	A flip – flop is a synchronous device.
Latches are transparent devices i.e. when they are enabled, the output changes immediately if the input changes.	A transition from low to high or high to low of the clock signal will cause the flip – flop to either change its output or retain it depending on the input signal.
A latch is a Level Sensitive device (Level Triggering is involved).	A flip – flop is an edge sensitive device (Edge Triggering is involved).
Latches are simpler to design as there is no clock signal (no careful routing of clock signal is required).	When compare to latches, flip – flops are more complex to design as they have clock signal and it has to be carefully routed. This is because all the flip – flops in a design should have a clock signal and the delay in the clock reaching each flip – flop must be minimum or negligible.
The operation of a latch is faster as they do not have to wait for any clock signal.	Flip - flops are comparatively slower than latches due to clock signal.
The power requirement of a latch is less.	Power requirement of a flip – flop is more.
A latch works based on the enable signal.	A flip – flop works based on the clock signal.

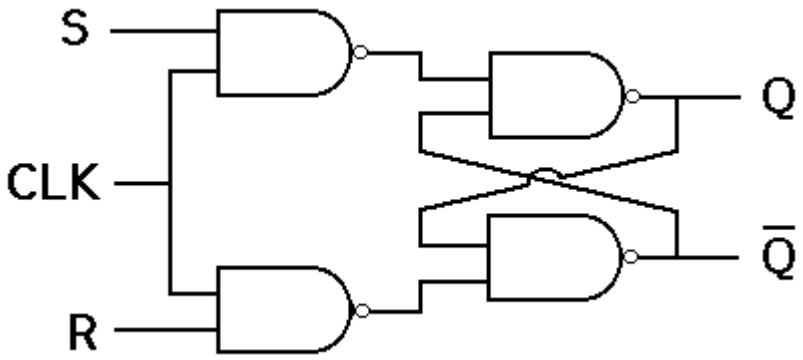
- There are basically 4 types of flip-flops:

  - SR Flip-Flop
  - JK Flip-Flop
  - D Flip-Flop
  - T Flip-Flop

SR FLIP FLOP:

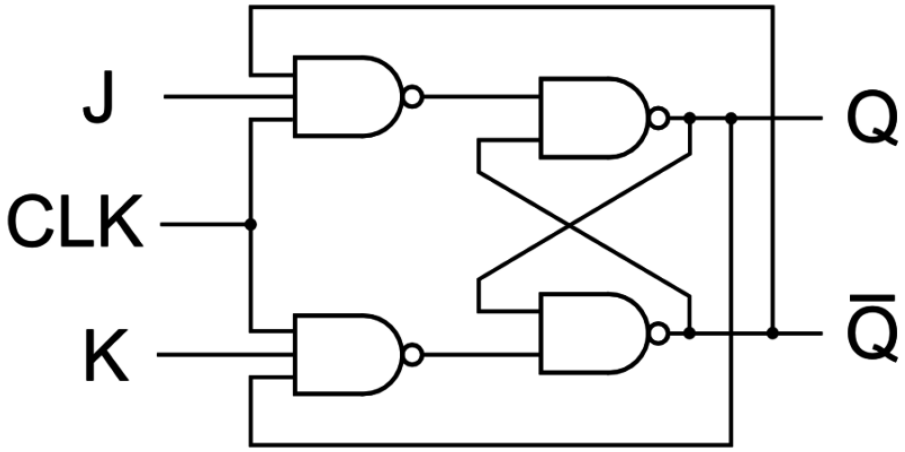
This is the most common flip-flop among all. This simple flip-flop circuit has a set input (S) and a reset input . In this system, when you Set "S" as active, the output "Q" would be high, and "Q'" would be low. Once the outputs are established, the wiring of the circuit is maintained until "S" or "R" go high, or power is turned off.

As shown , it is the simplest and easiest to understand. The two outputs, as shown above, are the inverse of each other. The **truth table of SR Flip-Flop** is highlighted below.



JK FLIPFLOP:

Due to the undefined state in the SR flip-flops, another flip-flop is required in electronics. The JK flip-flop is an improvement on the SR flip-flop where S=R=1 is not a problem.

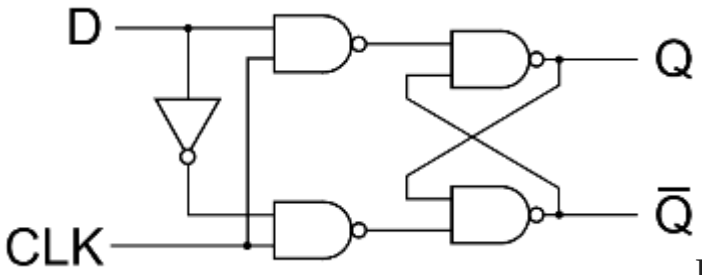


The input condition of J=K=1, gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically.

In simple words, If J and K data input are different (i.e. high and low), then the output Q takes the value of J at the next clock edge. If J and K are both low, then no change occurs. If J and K are both high at the clock edge, then the output will toggle from one state to the other. JK Flip-Flops can function as Set or Reset Flip-flops.

D FLIP-FLOP:

D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift registers and input synchronization.

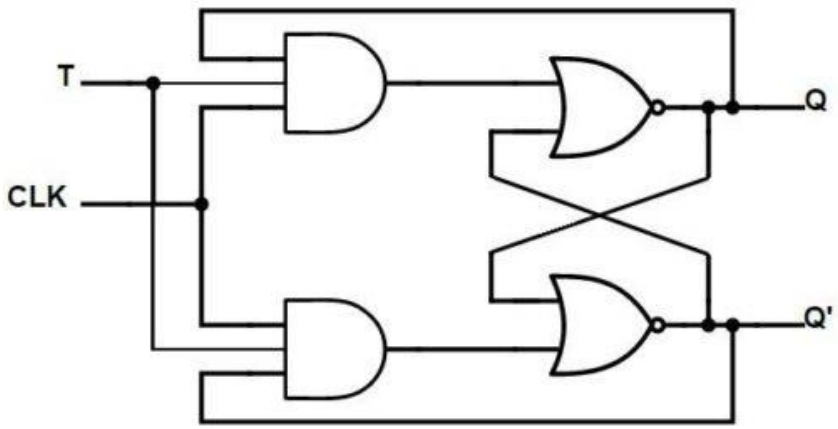


D Flip-Flop

In the D flip-flops, the output can only be changed at the clock edge, and if the input changes at other times, the output will be unaffected. The change of state of the output is dependent on the rising edge of the clock. The output (Q) is the same as the input and can only change at the rising edge of the clock.

T FLIP-FLOP:

A T flip-flop is like a JK flip-flop. These are basically single-input versions of JK flip-flops. This modified form of the JK is obtained by connecting inputs J and K together. It has only one input along with the clock input.



These flip-flops are called T flip-flops because of their ability to complement their state i.e. Toggle, hence they are named **Toggle flip-flops.**

APPLICATIONS:

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers

SHIFT REGISTER:

A group of flip flops which is used to store multiple bits of data and the data is moved from one flip flop to another is known as **Shift Register**. The bits stored in registers shifted when the clock pulse is applied within and inside or outside the registers. To form an n-bit shift register, we have to connect n number of flip flops. So, the number of bits of the binary number is directly proportional to the number of flip flops. The flip flops are connected in such a way that the first flip flop's output becomes the input of the other flip flop.

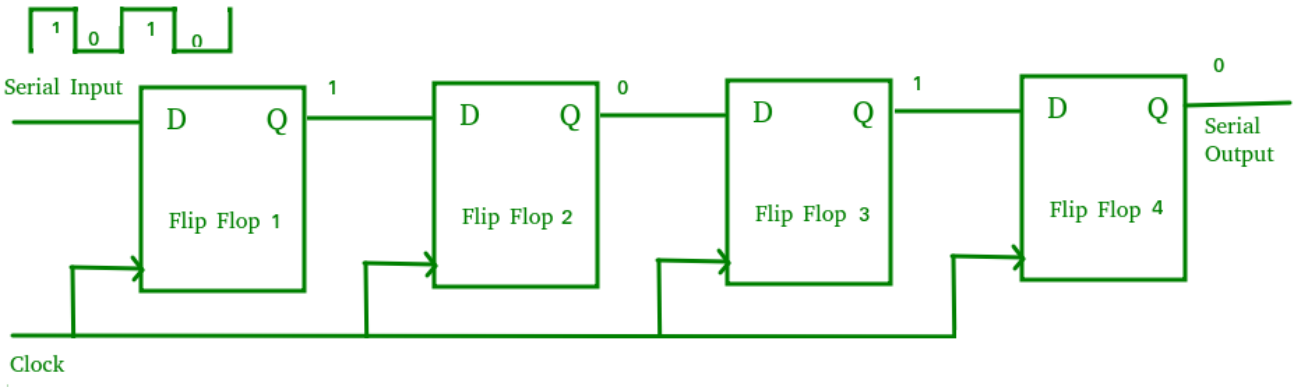
A **Shift Register** can shift the bits either to the left or to the right. A **Shift Register**, which shifts the bit to the left, is known as **"Shift left register"**, and it shifts the bit to the right, known as **"Right left register"**.

The shift register is classified into the following types:

- Serial In Serial Out shift register (SISO)
- Serial In Parallel Out shift register (SIPO)
- Parallel In Serial Out shift register (PISO)
- Parallel In Parallel Out shift register (PIPO

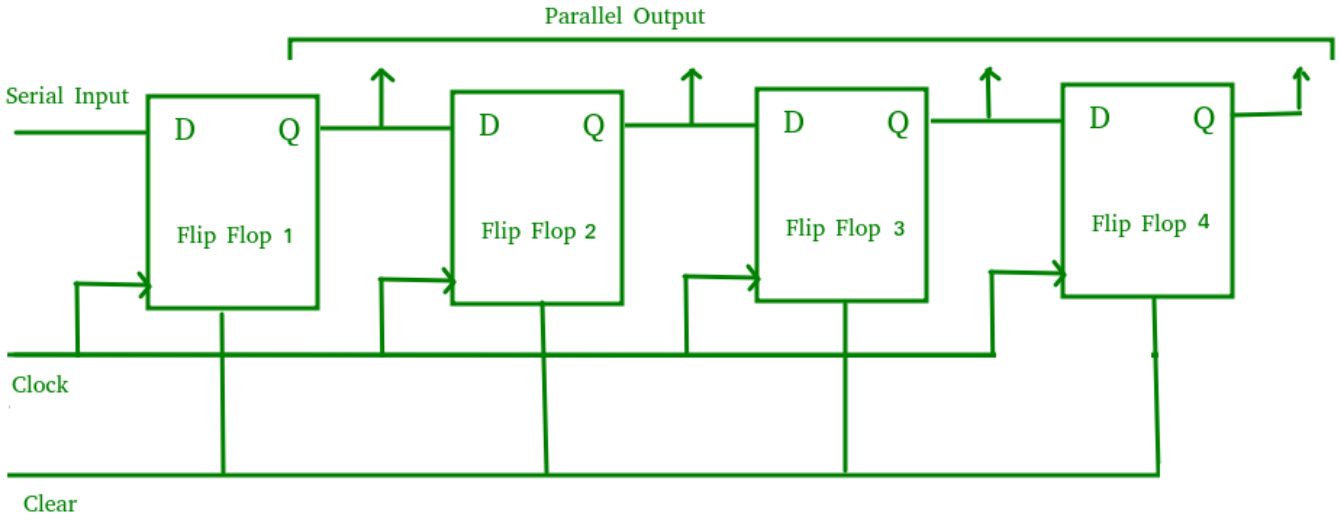
➢ Serial In Serial Out Shift Register:

The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register. The logic circuit given below shows a serial-in serial-out shift register. The circuit consists of four D flip-flops which are connected in a serial manner. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.



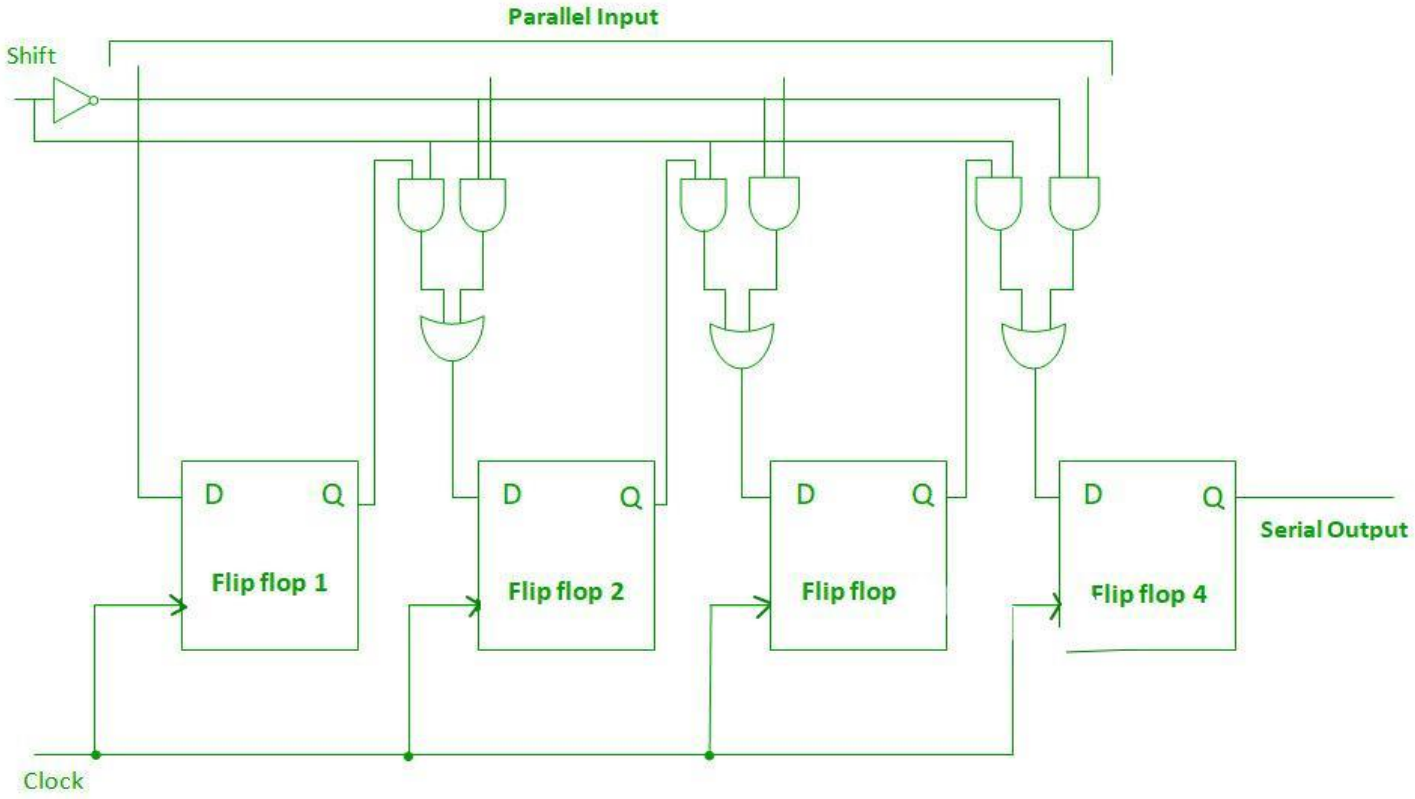
- **Serial-In Parallel-Out shift Register:**

The shift register, which allows serial input (one bit after the other through a single data line)and produces a parallel output is known as Serial-In Parallel-Out shift register.
The logic circuit given below shows a serial-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal is connected in addition to the clock signal to all the 4 flip flops in order to RESET them. The output of the first flip flop is connected to the input of the next flip flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.



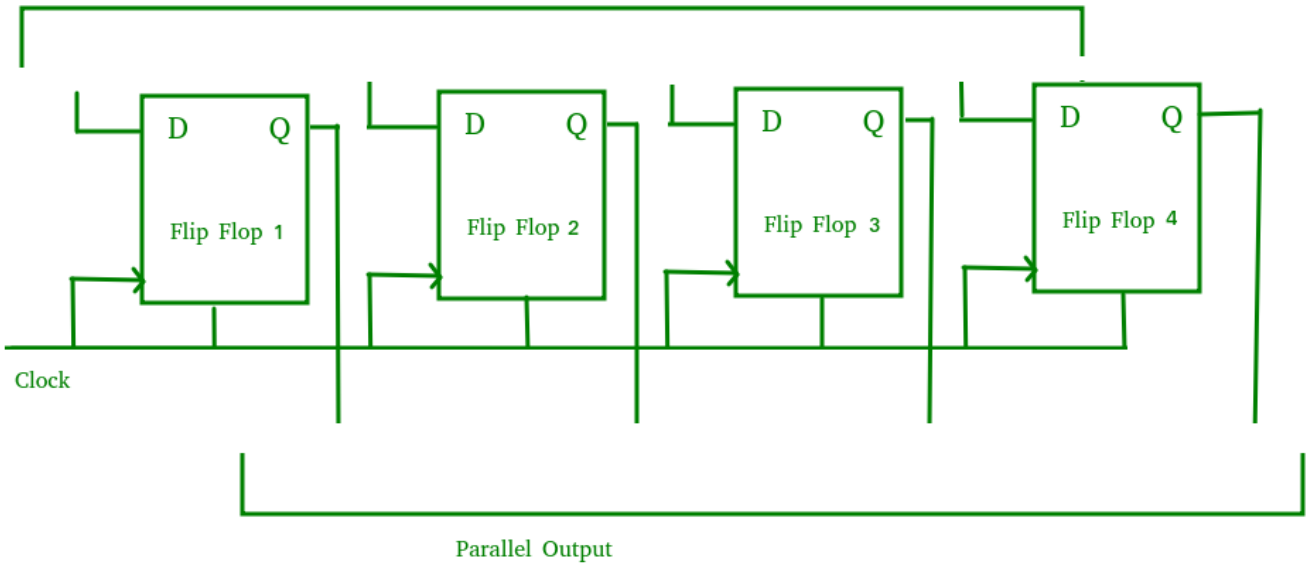
Parallel-In Serial-Out Shift Register:

The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and produces a serial output is known as Parallel-In Serial-Out shift register.
The logic circuit given below shows a parallel-in-serial-out shift register. The circuit consists of four D flip-flops which are connected. The clock input is directly connected to all the flip flops but the input data is connected individually to each flip flop through a multiplexer at the input of every flip flop. The output of the previous flip flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip flop. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.



Parallel-In Parallel-Out Shift Register: The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register.

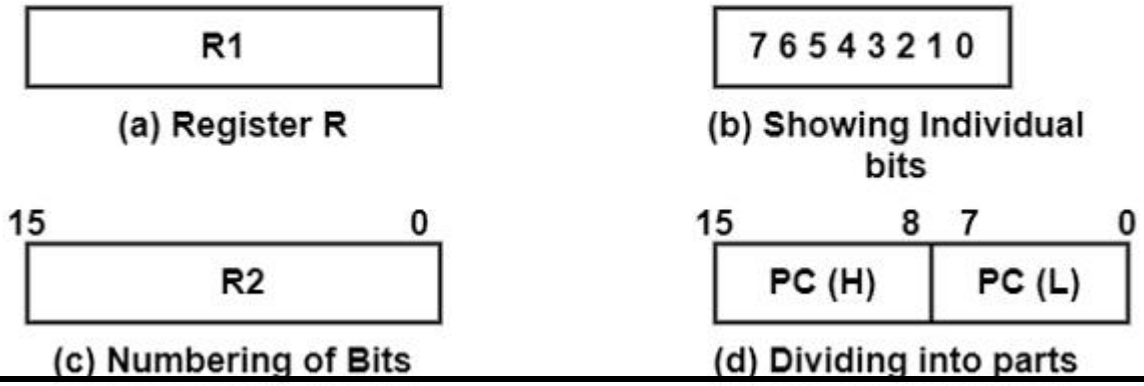
The logic circuit given below shows a parallel-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal and clock signals are connected to all the 4 flip flops. In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required. Data is given as input separately for each flip flop and in the same way, output also collected individually from each flip flop.



- **REGISTERS: -**

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

Block Diagram of Register



- **Register Transfer Operation:**

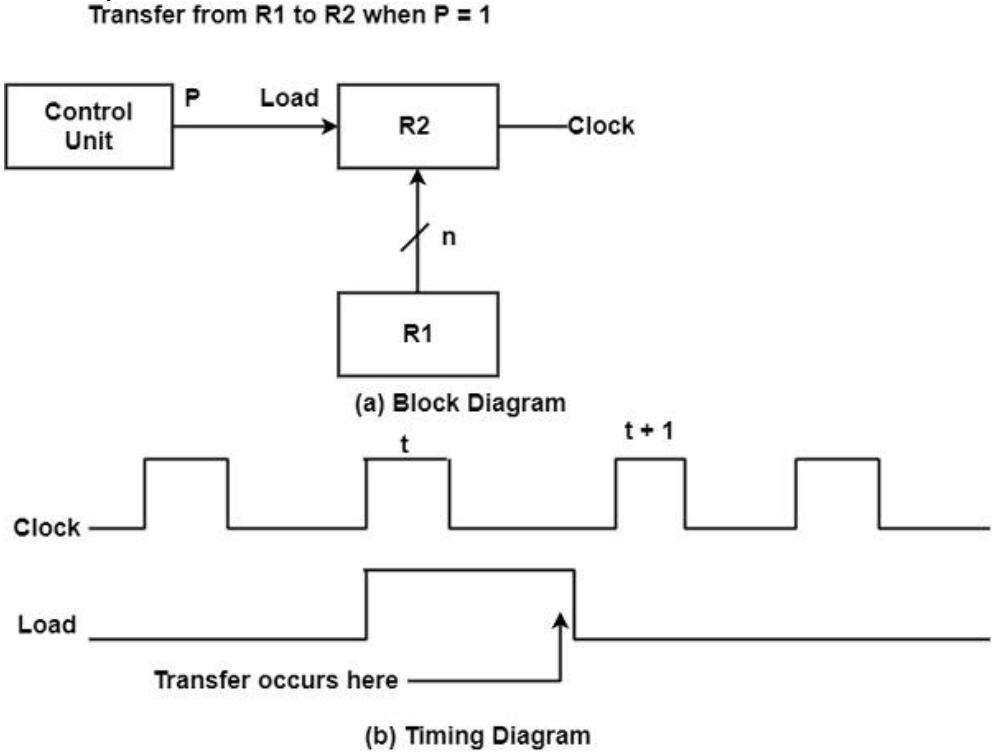
Register Transfer: Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement R2← R1 denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R1. By definition,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

if (P=1) then R2← R1.

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

P: R2← R1

Control condition is terminated by a colon implies transfer operation be executed by the hardware only if P=1. Every statement written in a register transfer notation implies a hardware construction for implementing the transfer. Figure 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*

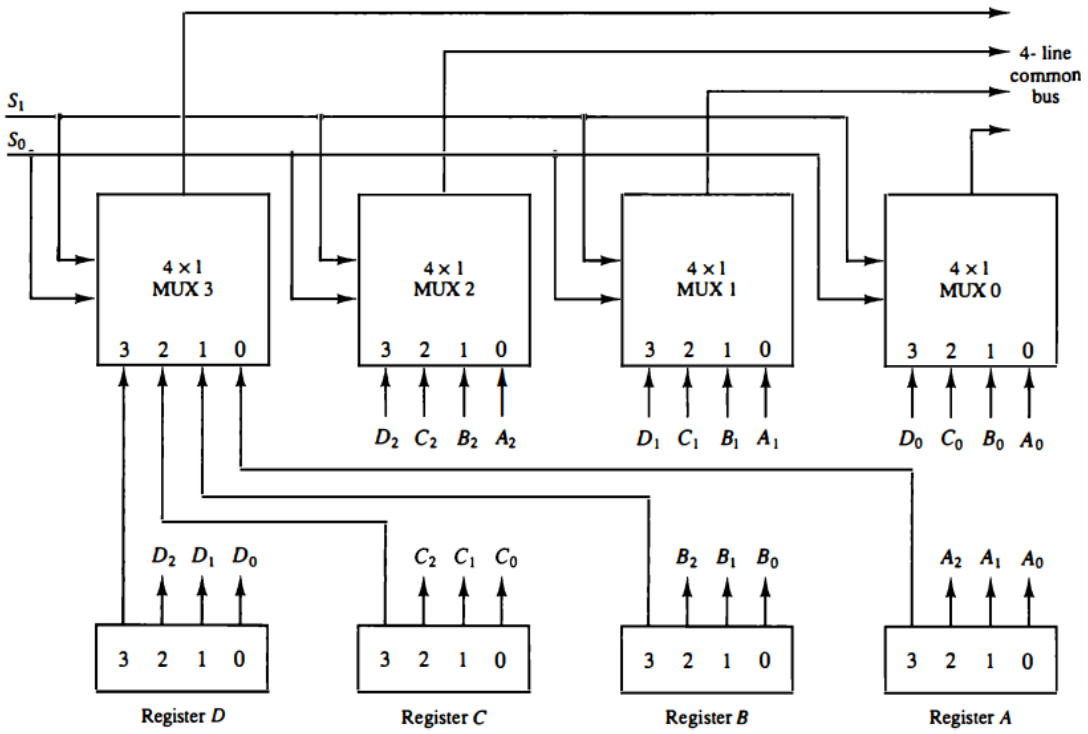
A comma is used to separate two or more operations that are executed at the same time. ¬
The statement T : R2← R1, R1← R2 (exchange operation) denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T=1.

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, S1 and S0. ¬ For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labelled A1The 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 S1 and S0 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 S1S0 = 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 S1S0 = 01, and so on. Table below 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 will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k x 1 since it multiplexes k data lines.

For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected.

The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows: BUS ← C, R1 ← BUS

The content of register C is placed on the bus, and the content of the bus is loaded into register R 1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer. R1 ← C

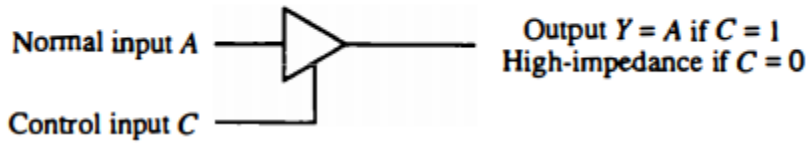
From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

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 a logic significance.
- Three-state gates may perform any conventional logic, such as AND or NAND. However, the one most commonly used in the design of a bus system is the buffer gate.
- The graphic symbol of a three-state buffer gate is shown in Fig. below. 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 high-impedance state of a three-state gate provides a special feature not available in other gates. 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 construction of a bus system with three-state buffers is demonstrated in Fig. below. The outputs of four buffers are connected together to form a single bus line (It must be realized that this type of connection cannot be done with gates that do not have three-state outputs.) 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. Careful investigation will reveal that Fig. below is another way of constructing a 4 x 1 multiplexer since the circuit can replace the multiplexer in Fig. Bus system for four registers.

Figure Graphic symbols for three-state buffer.



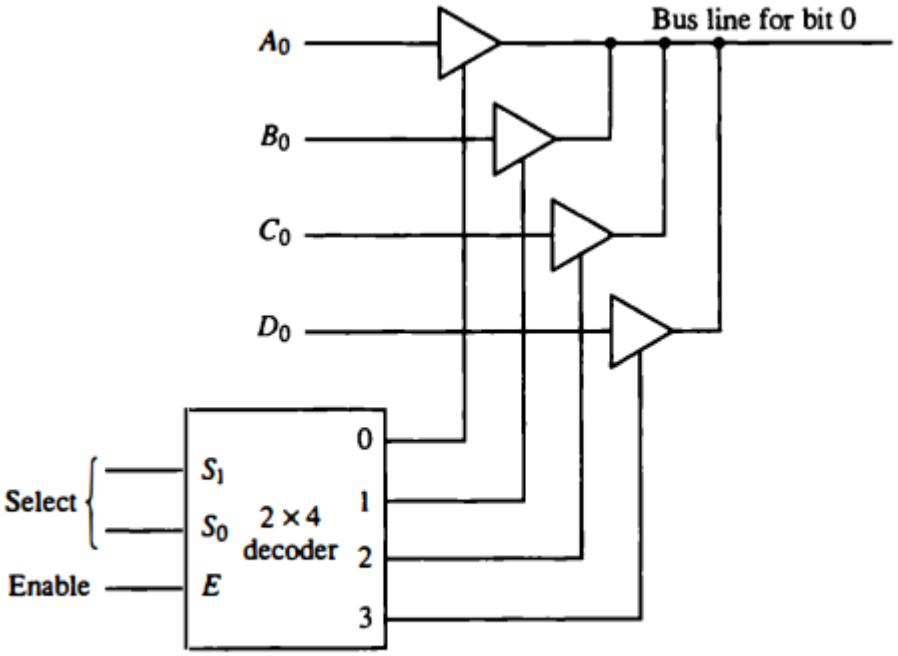


Figure Bus line with three state-buffers.

MEMORY TRANSFER:

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

Read: DR<- M [AR]

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

Write: M [AR] <- R1

TYPES OF MICRO-OPERATIONS:

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

❖ Register Transfer Micro-operation doesn't change the information content when the binary information moves from source register to destination register.
❖ Other three types of micro-operations change the information change the information content during the transfer.

❖ Arithmetic Micro-operations:

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

$$R3 \leftarrow R1 + R2$$

- 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 ¬ R2 is the symbol for the 1's complement of R2

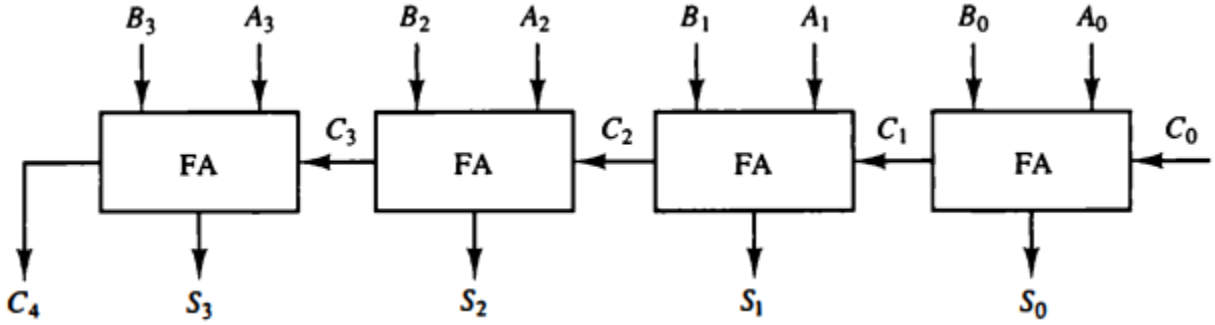
Adding 1 to 1's complement produces 2's complement ¬ Adding the contents of R1 to the 2's complement of R2 is equivalent to R1-R2.

TABLE Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

❖ **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 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.

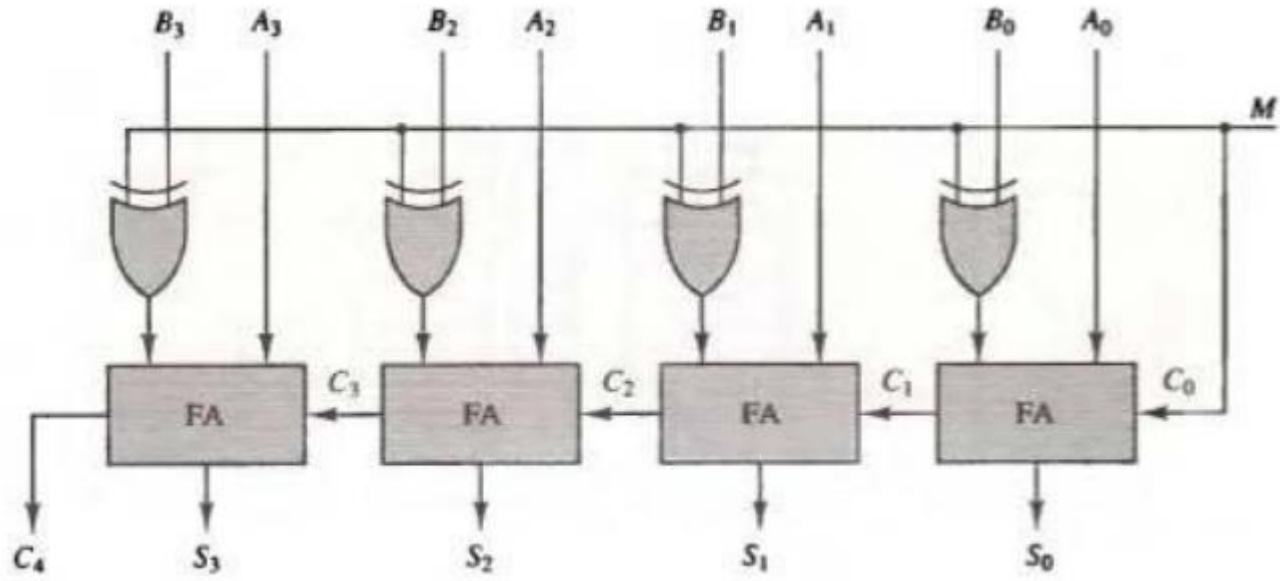


- The augends bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders. The input carry to the binary adder is Co and the output carry is C4. The S outputs of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

❖ **Binary Adder – Subtractor:**

❖

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

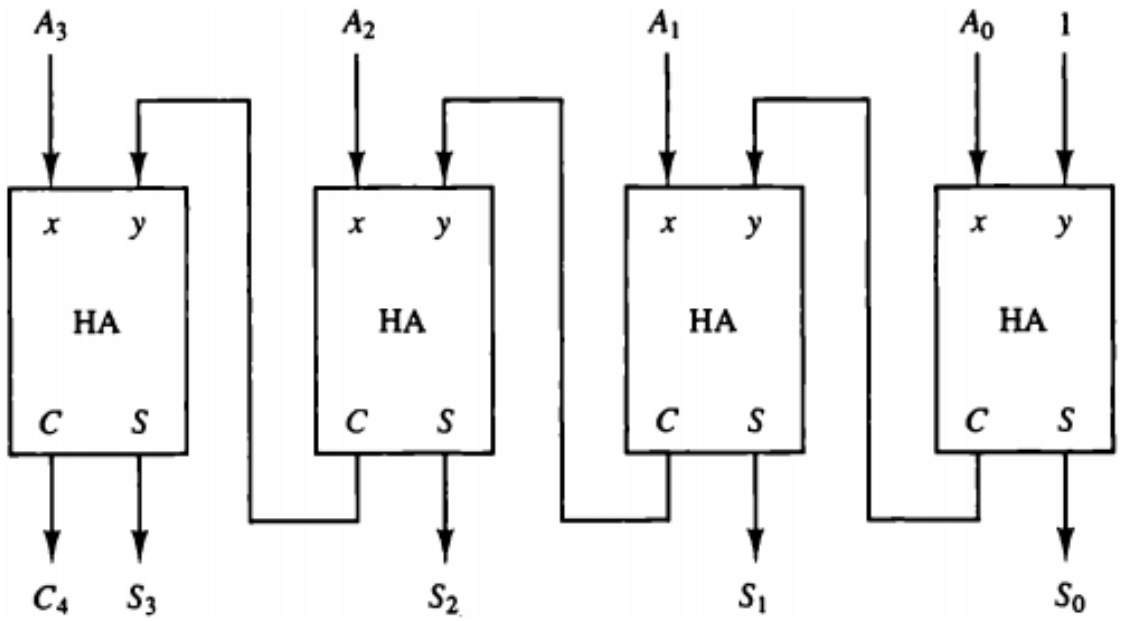


The mode input M controls the operation. When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.

- Each exclusive-OR gate receives input M and one of the inputs of B
- When M = 0, we have B x or 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 x or 1 = B' and Co = 1.
- The B inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of B.

Binary Incrementer :

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



ARITHMETIC CIRCUIT:

The arithmetic microoperations listed in Table below can be implemented in one composite 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. below. 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. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0

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.

The output of the binary adder is calculated from the following arithmetic sum:

$D = A + Y + C_{in}$

where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder.C_{in} is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. 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 below.

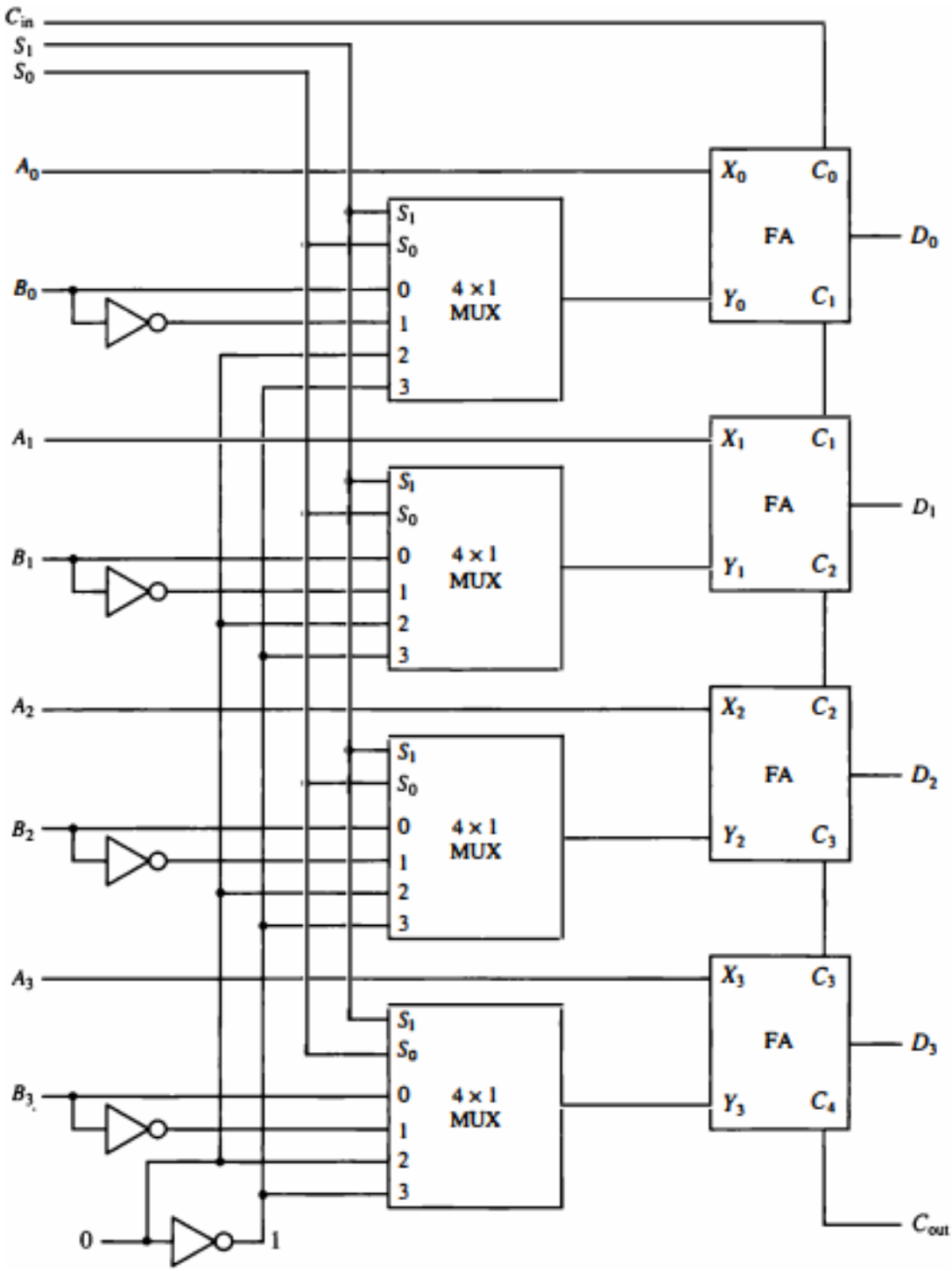

C_{in}
S_1
S_0
A_0
B_0
A_1
B_1
A_2
B_2
A_3
B_3
0
1
S_1
S_0
0
1
2
3
4 × 1
MUX
X_0
C_0
FA
Y_0
C_1
D_0
X_1
C_1
FA
Y_1
C_2
D_1
X_2
C_2
FA
Y_2
C_3
D_2
X_3
C_3
FA
Y_3
C_4
D_3
C_{out}

TABLE Arithmetic Circuit Function Table

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

- 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 + l. Both cases perform the add microoperation with or without adding the input carry.
- 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 + 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 + B. This is equivalent to a subtract with borrow, that is, A - B - 1. When $S_1S_0 = 10$, the inputs from B are neglected, and instead, all O's are inserted into the Y inputs. The output becomes D = A + 0 + Cm路 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. 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.
- Note that the microoperation D = A is generated twice, so there are only seven distinct microoperations in the arithmetic circuit.

❖ **Logic Micro operation :**

- Logic micro operation specify binary operations on the strings of bits in registers.
- Logic micro operations are bit-wise operations, i.e., they work on the individual bits of data.These are useful for bit manipulations on binary data and also useful for making logical decisions based on the bit value.
- There are, in principle, 16 different logic functions that can be defined over two binary inputvariables.
- However, most systems only implement four of these:
- AND (), OR ( ), XOR ( ), Complement/NOT
- The others can be created from combination of these.

TABLE 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

x	0 0 1 1	Boolean Function	Micro-Operations	Name
y	0 1 0 1			
	0 0 0 0	F0 = 0	F ← 0	Clear
	0 0 0 1	F1 = xy	F ← A ∧ B	AND
	0 0 1 0	F2 = xy'	F ← A ∧ B'	
	0 0 1 1	F3 = x	F ← A	Transfer A
	0 1 0 0	F4 = x'y	F ← A'∧ B	
	0 1 0 1	F5 = y	F ← B	Transfer B
	0 1 1 0	F6 = x ⊕ y	F ← A ⊕ B	Exclusive-OR
	0 1 1 1	F7 = x + y	F ← A ∨ B	OR
	1 0 0 0	F8 = (x + y)'	F ← (A ∨ B)'	NOR
	1 0 0 1	F9 = (x ⊕ y)'	F ← (A ⊕ B)'	Exclusive-NOR
	1 0 1 0	F10 = y'	F ← B'	Complement B
	1 0 1 1	F11 = x + y'	F ← A ∨ B	
	1 1 0 0	F12 = x'	F ← A'	Complement A
	1 1 0 1	F13 = x' + y	F ← A'∨ B	
	1 1 1 0	F14 = (xy)'	F ← (A ∧ B)'	NAND
	1 1 1 1	F15 = 1	F ← all 1's	Set to all 1's

- The hardware implementation of logic micro operation requires the insertion of the most important gates like AND, OR, EXOR, and NOT for each bit or pair of bits in the registers.

- one stage of a circuit:

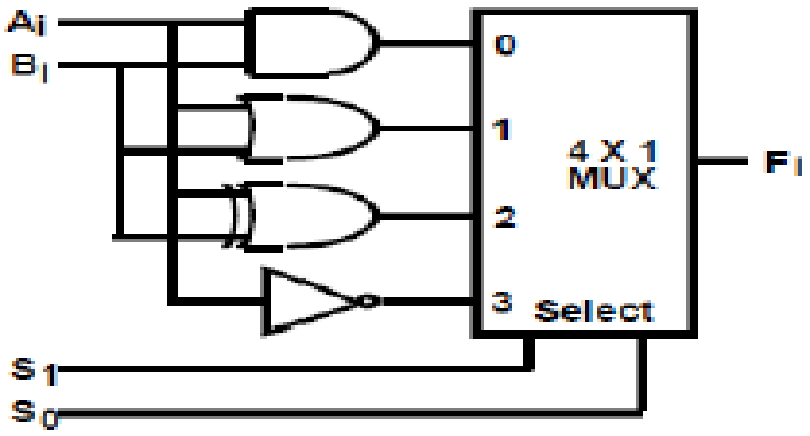
❖ This circuit generates the four basic logic micro operations.

It consists of four gates and a **MULTIPLEXER:**

- The two selection lines of the multiplexer selects one of the four logic operations available atone time.

- The circuit shows one stage for bit “i” but for logic circuit of n bits the circuit should berepeated n times but with one remark; the selection pins will be shared with all stages.

- The circuit shows one stage for bit “i” but for logic circuit of n bits the circuit should berepeated n times but with one remark; the selection pins will be shared with all stages.



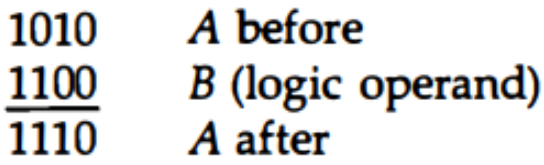
Function table

S_1	S_0	Output	μ-operation
0	0	F = A ∧ B	AND
0	1	F = A ∨ B	OR
1	0	F = A ⊕ B	XOR
1	1	F = A'	Complement

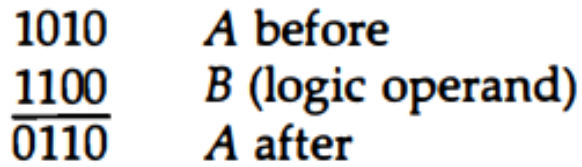
Applications of Logic Microoperations:

- Logic microoperations 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 bit values into a register. The following examples show how the bits of one register (designated by A) are manipulated

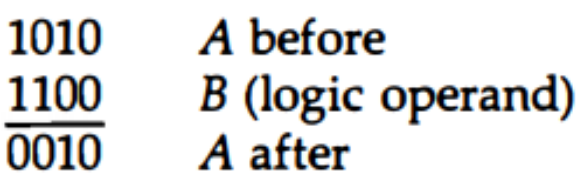
- **by logic microoperations** as a function of the bits of another register (designated by B).
- **In a typical application**, register A is a processor register and the bits of register B constitute a logic operand extracted from memory and placed in register B.
- **The selective-set** operation sets 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:



- **The two leftmost bits** of B are 1' s, so the corresponding bits of A are set to 1.
- **One of these two bits** was already set and the other has been changed from 0 to 1. The two bits of A with corresponding 0' s in B remain unchanged. The example above serves as a truth table since it has all four possible combinations of two binary variables.
- **From the truth table** we note that the bits of A after the operation are obtained from the logic-OR operation of bits in B and previous values of A. Therefore, the OR microoperation can be used to selectively set bits of a register.
- **The selective-complement** operation complements bits in A where there are selective-clear corresponding 1's in B. It does not affect bit positions that have 0's in B. For example:



- **Again the two leftmost bits** of B are 1's, so the corresponding bits of A are complemented.
- **This example** again can serve as a truth table from which one can deduce that the selective-complement operation is just an exclusive-OR microoperation.
- **Therefore**, the exclusive-OR micro operation can be used to selectively complement bits of a register.
- **The selective-clear** operation clears to 0 the bits in A only where there are corresponding 1's in B. For example:



- **Again the two leftmost bits** of B are 1' s, so the corresponding bits of A are cleared to 0.
- **One can deduce** that the Boolean operation performed on the individual bits is AB'. The corresponding logic microoperation is $A \leftarrow A \wedge B$

- **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:

```
1010      A before
1100      B (logic operand)
1000      A after masking
```

- **The two rightmost bits of A** are cleared because the corresponding bits of B are 0' s. The two leftmost bits are left unchanged because the corresponding bits of B are 1's.
- **The mask operation** is more convenient to use than the selective clear operation because most computers provide an AND instruction, and few provide an instruction that executes the microoperation for selective-clear.
- **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.
- **The clear operation** compares the words in A and B and produces an all 0' s result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation as shown by the following example:

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

- **When A and B are equal**, the two corresponding bits are either both 0 or both 1. In either case the exclusive-OR operation produces a 0. The all-0's result is then checked to determine if the two numbers were equal.

❖ **Shift Microoperations**:

- Shift micro-operations are those micro-operations that are used for the serial transfer of information. These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations.
- The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input.
- **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.
- **The information** transferred through the serial input determines the type of shift
- **There are three types of shifts:** logical, circular, and arithmetic.
- **A logical shift** is one that transfers 0 through the serial input.We will adopt the symbols shl and shr for logical shift-left and shift-right microoperations.

TABLE Shift Microoperations

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

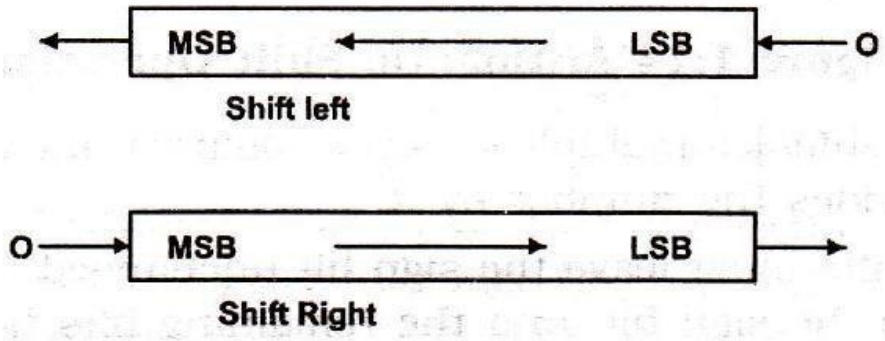
Logical Shift

- A logical shift is one that transfers 0 through the serial input.
- The symbols sh1 for logical shift-left microoperations and the symbol shr for logical shift-right microoperations.

For example :

R1 <— shl R1 R2 <—shl R2

SHL R_1, 1 ; shift left R_1 by 1 bits SHR R_1, 1 ; shift right R_1 by 1 bits.

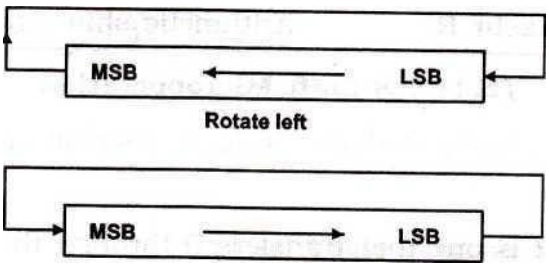


Initial valueof R_1	Value of R_1 after shift operation	Operation
01001101	1001 1010	SHL R_1, 1
01001101	1010 0110	SHR R_1, 1

- There are two microoperations that specify a 1-bit shift to the left of the content
- of register R1 and a 1-bit shift to the right of the content of register R2.
- The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

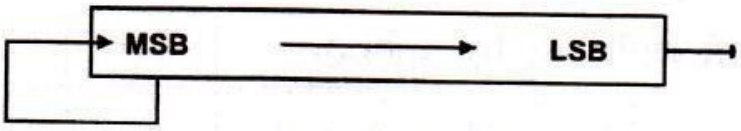
Circular Shift

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- We will use the symbols cil for the circular shift left microoperations and cir for the circular shift right microoperations.



Arithmetic shift

- An arithmetic shift is a microoperation that shifts a signed binary number to the left



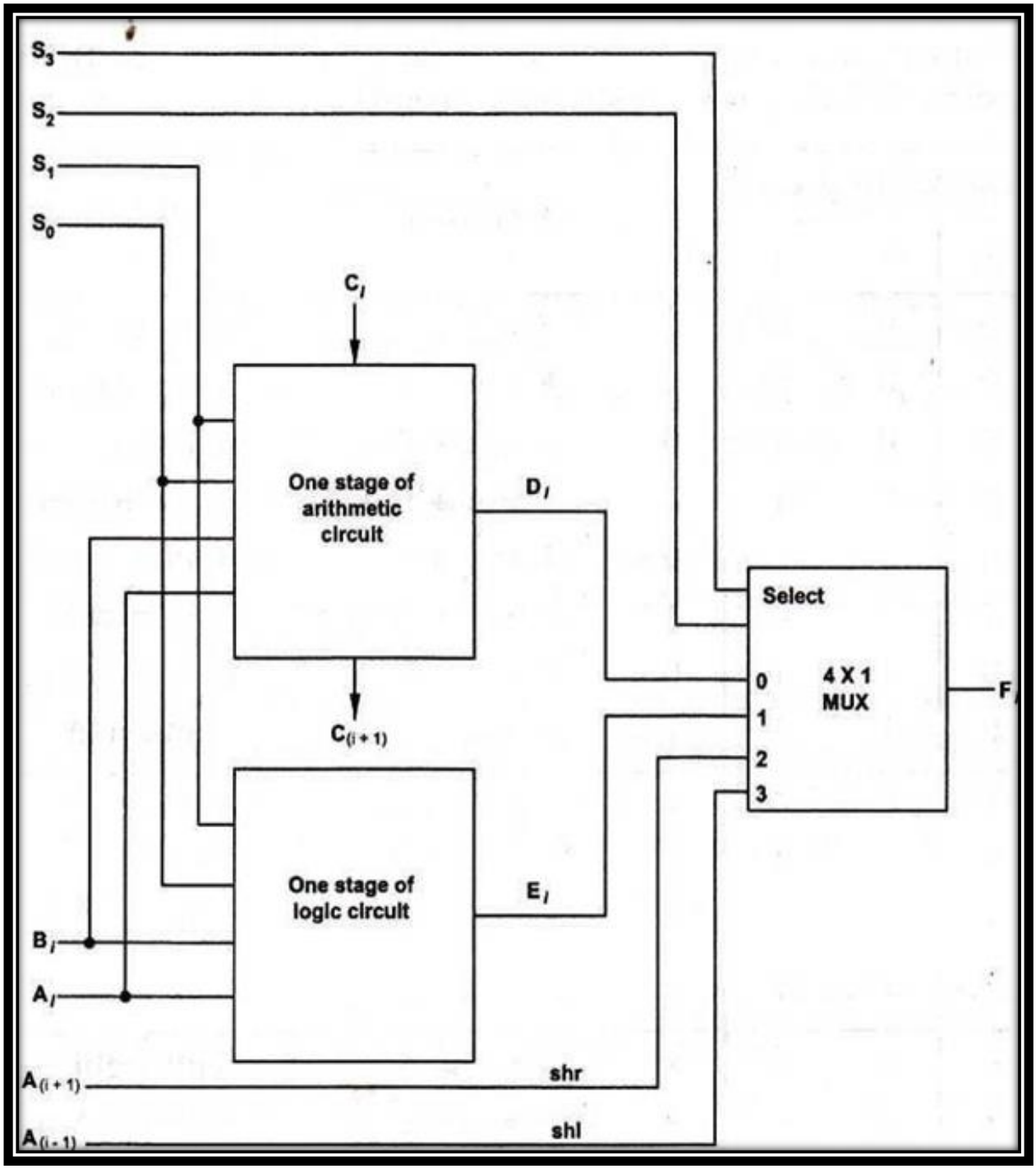
or right.

- 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. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number.
- The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form.

1. Explain in detail Arithmetic Logic Shift Unit.

- Computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, ALU
- To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register.
- 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 subscript; designates a typical stage. Inputs A and B, are applied to both the arithmetic and logic units.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in E_i and a logic output in H_i

- The data in the multiplexer are selected with inputs S_3 and S_2. The other two data input to the multiplexer receive inputs A_{i-1} for the shift right operation and A_{i+1} for the shift left operation.


S3
S2
S1
S0
Ci
One stage of arithmetic circuit
Di
Select
0
1
2
3
4 X 1 MUX
F
C(i + 1)
One stage of logic circuit
Ei
Bi
Ai
A(i + 1)
shr
A(i - 1)
shl

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

UNIT 2

BASIC COMPUTER ORGANIZATION

Instruction Codes:

The organization of the computer is defined by its internal register, the timing and control structure, and the set of instructions that it uses.

The **Internal organization** of a digital system is defined by the sequenceof microoperations it performs on data stored in its registers.

A **Program** is a set of intructions that specify the operations, operands, and the sequence by which processing has to occur.

A **Computer instruction** is a binary code that specifies the micro-operations in a sequence for a computer. They are stored in the memory along with the data. Every computer has its own unique set of instructions.

They can be divided into two parts namely, Operation codes (Opcodes) and Address.

An address specifies the registers or the locations that can be used for that operation.

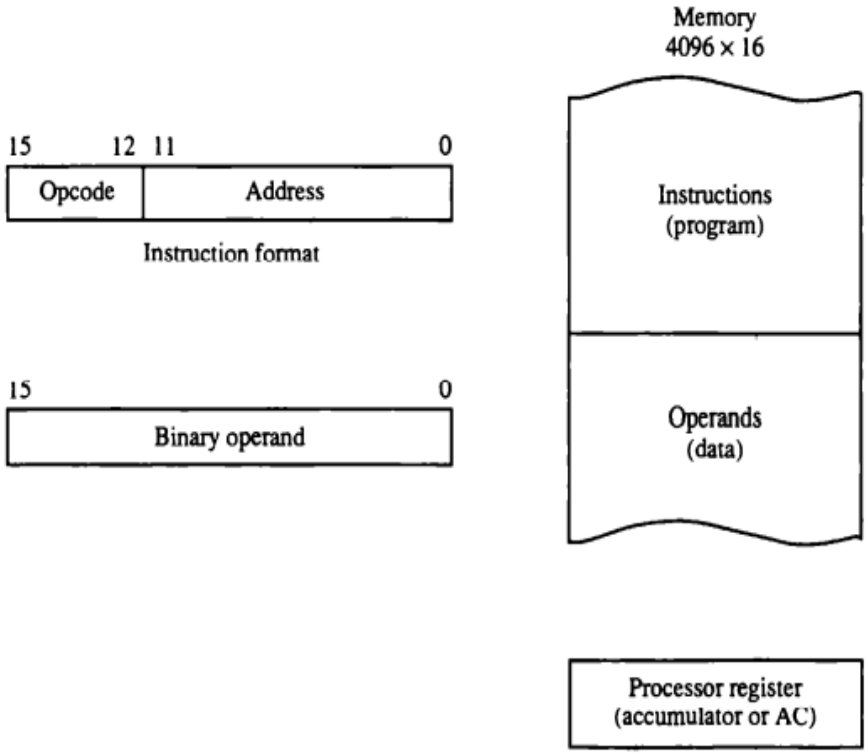
Operands are specific parts of computer instruction that depict what data is to be operated on.

Instruction Code is a group of bits that instruct the computer to perform a specific operation.

Operation Code is a group of bits in an instruction code that define operation to be performed. E.g. Add, Subtract, Shift. Opcodes specifies the operation for a particular instruction. An operation code is sometimes called a Macro-operation because it specifies a set of micro operations.

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 of instruction specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.



- As par the diagram, 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.

- The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.

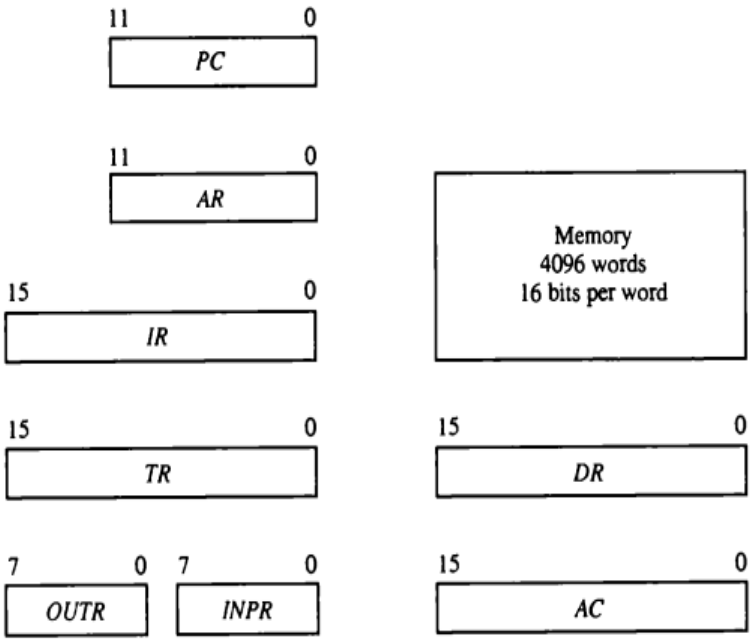
- It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name accumulator and label it AC.

- The operation is performed with the memory operand and the content of AC.

- If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.

- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory.

- For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.



Computer Registers

- Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.
- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address.

Register Symbol	No. of bits	Register Name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of next instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds character from input device
OUTR	8	Output Register	Holds character for output device

List of Registers for the Basic Computer

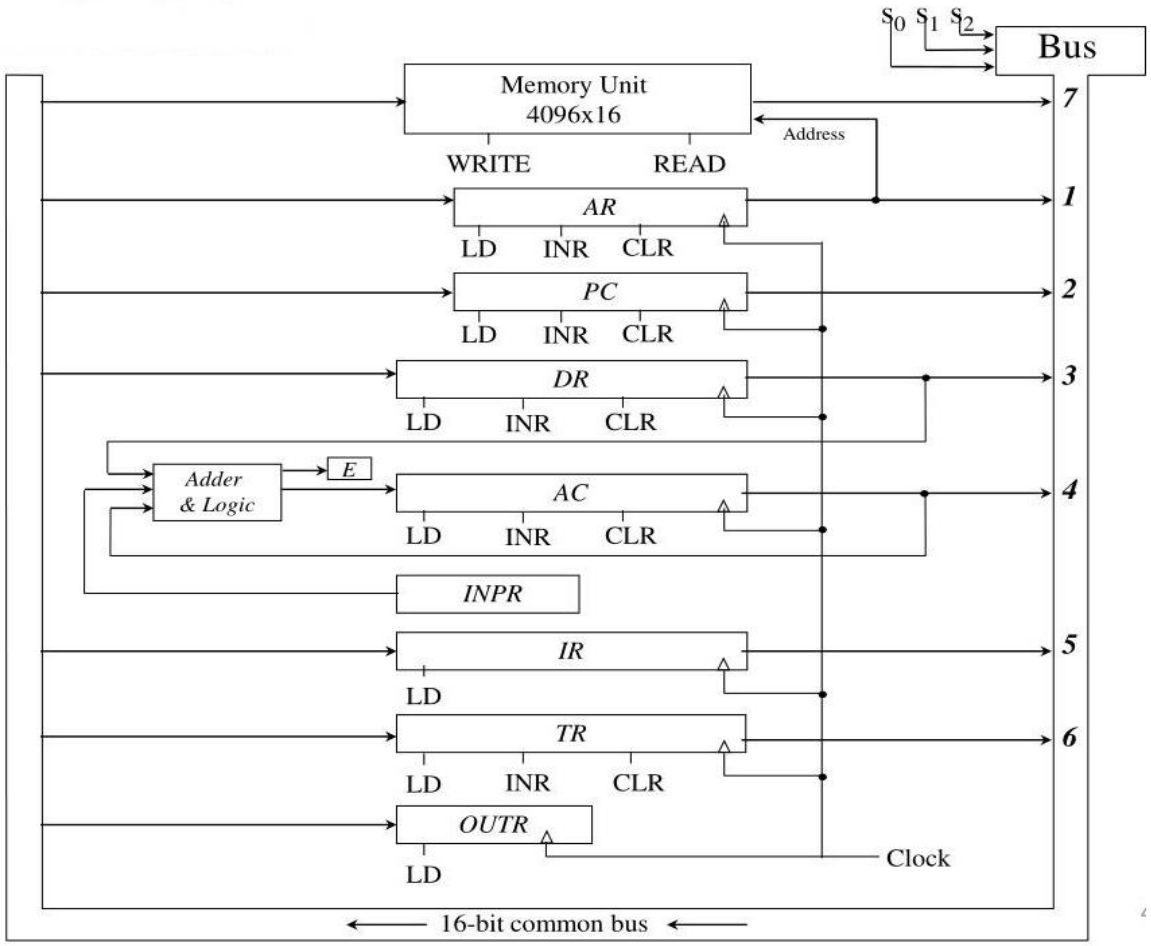
- The memory unit has a capacity of 4096 words and each word contains 16-bits of Data.
- Each word requires 12-bit memory address.
- Hence, the registers that deal with data (operand) and instruction are of 16-bit size. And registers that holds address are of 12-bit size.
- Registers that require 16-bit are:
  - Data register (DR) – holds data or operands fetched from memory.
  - Accumulator (AC) – also called processor register.
  - Instruction register (IR) – holds 16-bit instruction.
  - Temporary Register (TR) – holds intermediate or temporary data.
- Registers that require 12-bit are:
  - Address register (AR) – holds address for memory.
  - Program Counter (PC) – holds address of next instruction in memory.
- While Input register (IPNR) and Output register (OUTR) of 8-bits.
  - Input register holds 8-bit character from input device.
  - Output register holds 8-bit character for output device.

Common bus system

The basic computer has eight registers, a memory unit, hence, paths must be provided to transfer information from one register to another and between memory and registers.

An 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 diagram.



Basic Computer Registers Connected to a Common Bus.

- 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 $S_2S_1S_0 = 011$.
- The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.

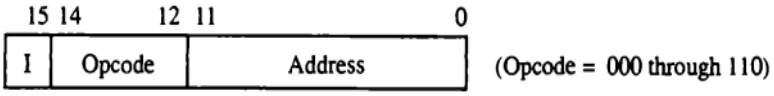
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$.
- 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 receive information from the bus, only the 12 least significant bits are transferred into the register.
- INPR receives an 8-bit character from an input device which is then transferred to AC.
- OUTR receives an 8-bit character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).
- The increment operation is achieved by enabling the count input of the counter. 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 content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC.
- The 16-bits input 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. They are used to implement register microoperations such as complement AC and shift AC.
    - Another set of 16-bit inputs come from the data register DR.
    - A third set of 8-bit inputs come from the input register INPR.
- The flip-flop E is used to hold carry while an addition microoperation is performed and result is transferred to AC.
- The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

- **For Example,**
  DR ← AC and AC ← DR
  The two micro – operations can be executed at the same time. This can be done by placing the content of AC on the bus (S2S1S0 = 100), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC.

Computer instruction [Types and format]

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

- **Memory-reference instruction**



(a) Memory – reference instruction

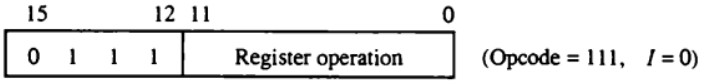
  - A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.
  - I is equal to 0 for direct address and to 1 for indirect address.

  - **Example –**

    IR register contains = 0001XXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.

```
Hence, DR ← M [AR]
AC ← AC + DR, SC ← 0
```

- **Register reference instruction**



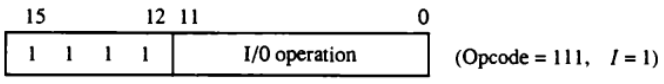
(b) Register – reference instruction

  - The register reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.
  - A register-reference instruction specifies an operation on or a test of the AC register.
  - An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.
  - The left most four bits are always 0111, equivalent to hexadecimal 7.
  - **Example –**

    IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

    Hence, AC ← ~AC

- **Input-output instruction**



(c) Input – output instruction

- An input-output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction.
- The remaining 12 bits are used to specify the type of input-output operation or test performed.
- The left most four bits are always 1111, equivalent to hexadecimal F.
- If the three opcode bits in positions 12 to14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I.
- If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type.
- Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.
- **Example –**

  IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.
- The instructions for the computer are listed in Table below.
- By using the hexadecimal equivalent, the 16 bits of an instruction code is reduced to four digits with each hexadecimal digit being equivalent to four bits.

Symbol	**Hexadecimal code**		**Description**
	I = 0	**I = 1**	
Memory reference instruction			
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 unconditional
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
Register reference instruction			
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
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 E is 0
HLT	7001	Halt computer
Input-output instruction		
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

Basic Computer Instructions

Instruction Set Completeness

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical and shift instructions
- A set of instructions for moving information to and from memory and processor registers.
- Instructions which controls the program together with instructions that check status conditions.
- Input and Output instructions

Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the capability of moving information between these two units. Program control instructions such as branch instructions are used change the sequence in which the program is executed.

Input and Output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results of computations must be transferred back to the user.

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.
- The control signals 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:
    1. Hardwired Control
    2. Microprogrammed Control

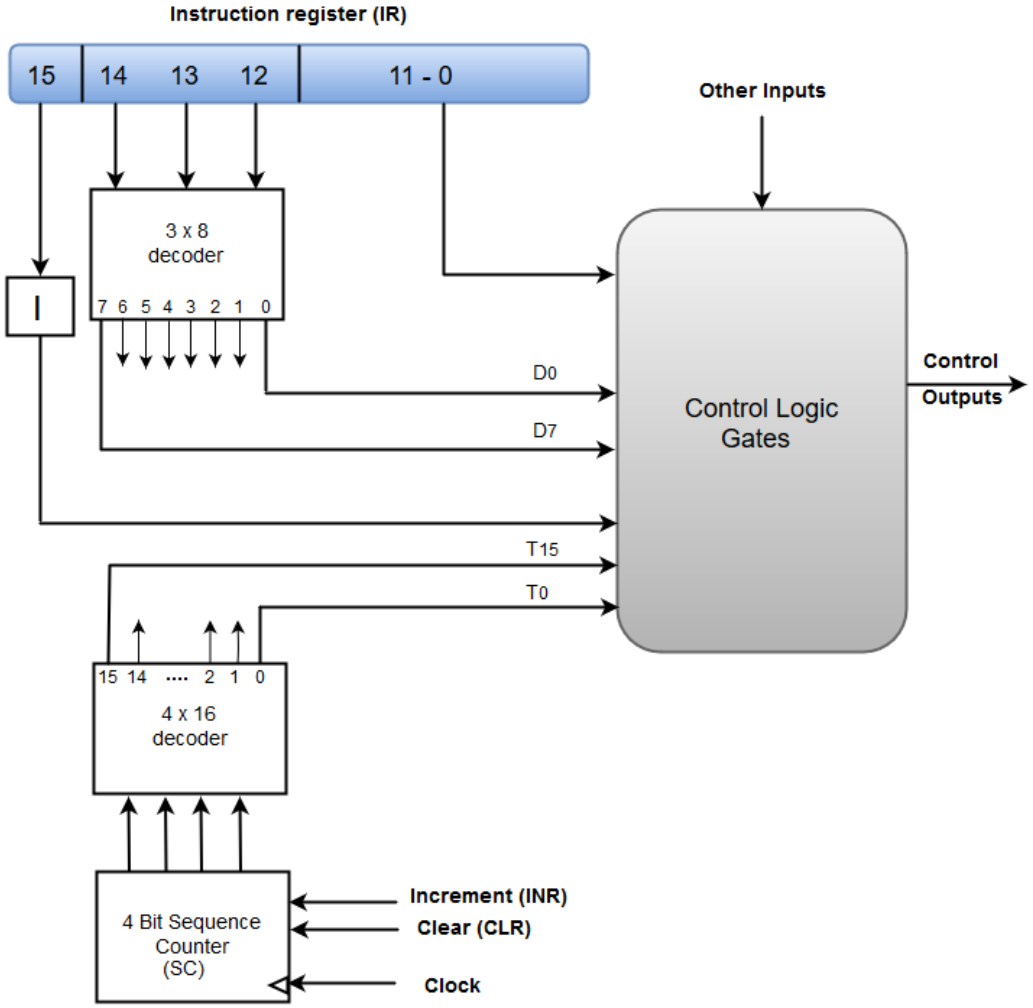
Hardwired Control

- The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

Micro-programmed Control

- The Microprogrammed Control organization is implemented by using the programming approach.
- In Microprogrammed Control, the micro-operations are performed by executing a program consisting of micro-instructions.

Control Unit of basic computer

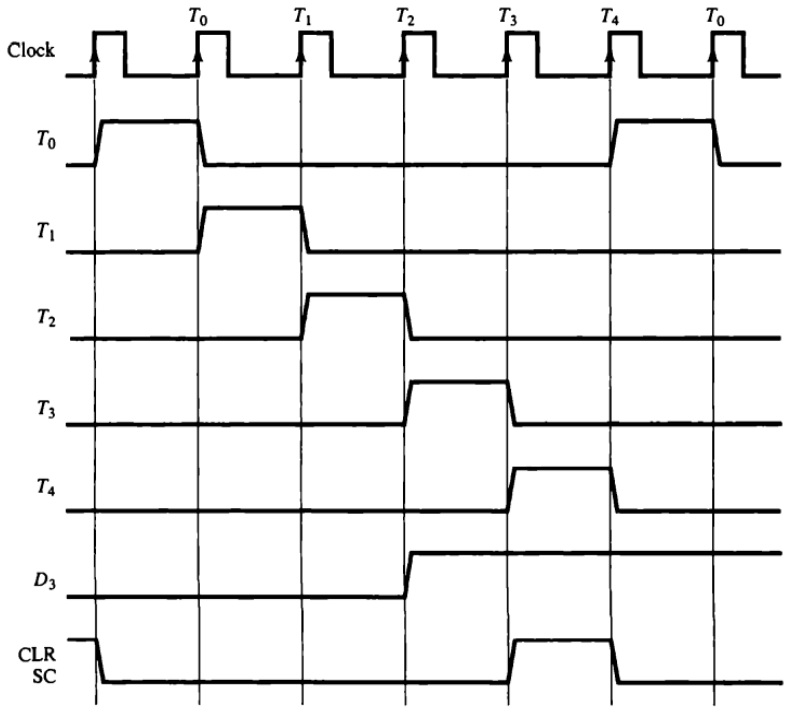


- The block diagram of the control unit is shown in Fig. It consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register (IR).
- The instruction register is divided into three parts: the I bit, the operation 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.
- The subscripted decimal number is equivalent to the binary value of the corresponding operation code.
- Bit 15 of the instruction is transferred to a flip-flop 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. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

Timing diagram

- 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 is active.
- This is expressed symbolically by the statement.

D_3T_4: SC ← 0



- The timing diagram 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 actives the timing signal T_0.
- 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 and T_4 and so on.
- If SC is not cleared, the timing signals will continue with signals T_5, T_6, up to T_{15} and back to signals T_0.

Instruction Cycle:

Terms related to execute any instructions

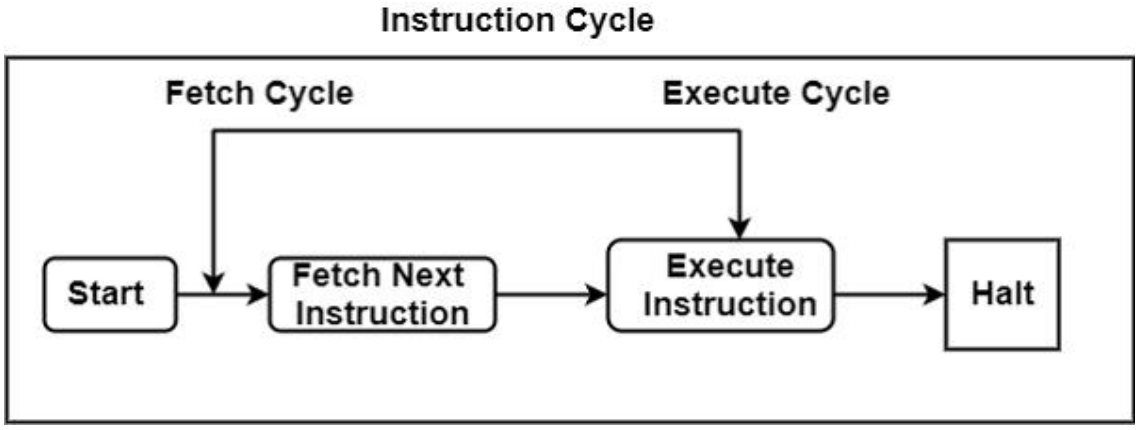
- **Instruction Fetch** − The instruction is read from its specific memory location to the processor.
- **Instruction Operation Decoding** − The instruction is interpreted and the type of operation to be implemented and the operand(s) to be used are decided.
- **Operand Address Calculation** − The address of the operand is evaluated if it has a reference to an operand in memory or is applicable through the Input/Output.
- **Operand Fetch** − The operand is read from the memory or the I/O.
- **Data Operation** − The actual operation that the instruction contains is executed.
- **Store Operands** − It can store the result acquired in the memory or transfer it to the I/O.

The main execution process is done by the processor. The processing of instruction involves two steps, instruction fetch and instruction execution. Each instruction is fetched from the memory separately and executed. Depending on the nature of the instruction its execution may deal with a number of operations.

An instruction cycle refers to the processing of a particular instruction. Each instruction cycle goes through the following phases during its processing:

1. Fetching instruction from memory.
2. Decoding the instruction.
3. Reading the effective address from memory in case of indirect address.
4. Executing the instruction.

After the following four procedures are done, the control switches back to the first step and repeats the similar process for the next instruction. Therefore, the cycle continues until a **Halt** condition is met. The figure shows the phases contained in the instruction cycle.



As display in the figure, the halt condition appears when the device receive turned off, on the circumstance of unrecoverable errors, etc.

Fetch Cycle

The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.

Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.

Execute Cycle

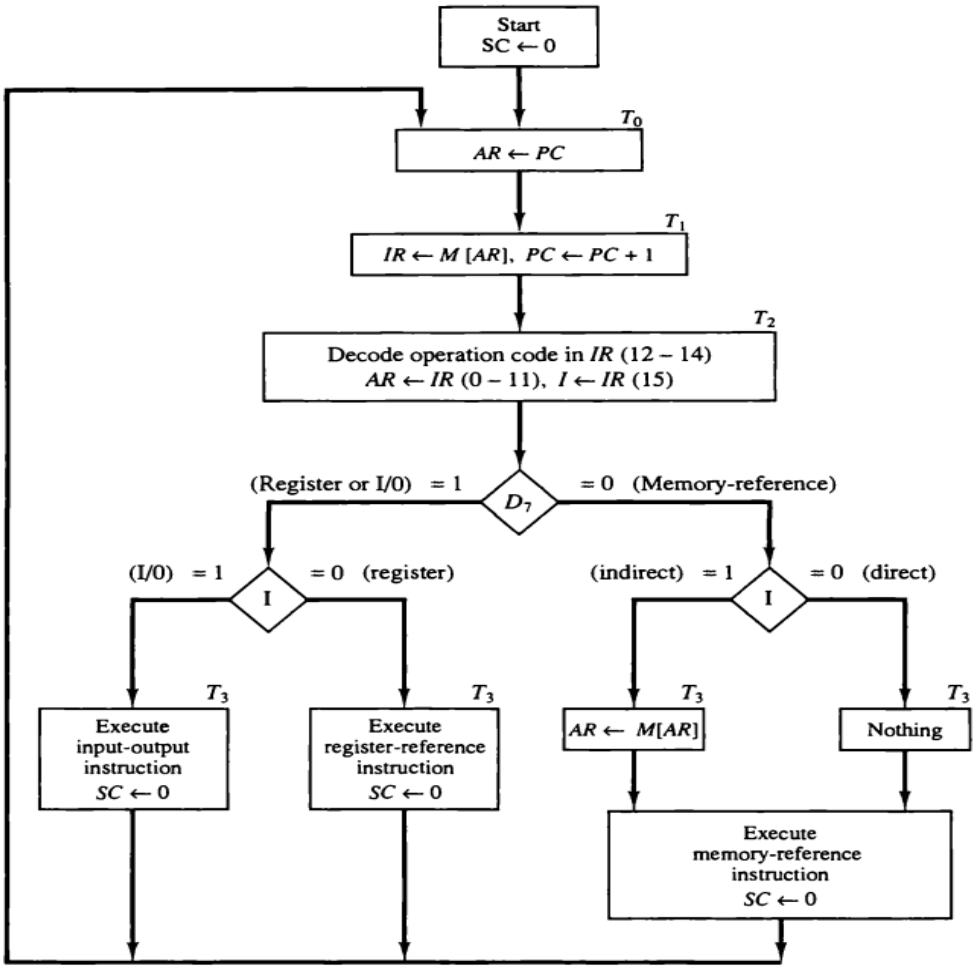
The data transfer for implementation takes place in two methods are as follows −

Processor-memory − The data sent from the processor to memory or from memory to processor.

Processor-Input/Output − The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

In the execute cycle, the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.

Flowchart for instruction cycle



- The sequence counter is set to 0 initially.
- At initial stage the PC holds address of first instruction of the program. Hence at timing signal T_0 the address of first instruction is transferred to AR.
- At time signal T_1 the instruction is fetched from memory and stored to IR, and PC is incremented by 1.
- At time signal T_2 the instruction stored in IR is decoded. IR [0-11] is stored in AR, and opcode in IR [12-14] is decoded, left most significant bit IR [15] is stored to I.
- If IR[12-14] is 111, also known as D_7:
    - If $D_7 = 0$, the instruction must be memory reference instruction.
        - If I = 0 while $D_7 = 0$, the instruction is Direct memory reference instruction.
        - If I = 1 while $D_7 = 0$, the instruction is Indirect memory reference instruction.
        - In indirect memory reference instruction, the reference address is replaced with effective address in AR at time signal T_3.
        - he execution of memory reference instruction begins from time signal T_4.
        - Direct memory reference instruction always has effective address, hence, time signal T3 is considered as 'Nothing'
    - If $D_7 = 1$, the instruction can be either register reference instruction or input-output instruction.
        - If I = 0 while $D_7 = 1$, the instruction is register reference instruction.
        - If I = 1 while $D_7 = 1$, the instruction is input-output instruction.
        - Register reference instruction and input-output instruction do not hold any memory address and contains opcode in IR [0-11] – Hence at time signal T_3 register reference and input-output instruction get executed.
- After execution of each instruction, the SC is set to Zero.

Memory Reference Instruction:

Usually programmers write codes in assembly language which has a very close reference with machine language. These instruction formats are known as memory reference instructions.

The different memory reference instructions are:

Type of instruction	Symbol	Description
Memory Reference Instruction	ADD	Add memory word to AC
	STA	Store content to AC in memory
	BUN	Branch unconditionally
	BSA	Branch and save return address
	ISZ	Increment and skip if zero

Flowchart components of memory reference instruction

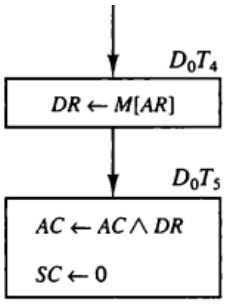
The exection of Memory reference instruction starts form time signal T_4.

AND [D_0] : AND memory word to AC

This operation performes AND logical operation between bits of AC and DR.

At D_0T_4 data of M[AR] is fetched and stored to DR.

At D_0T_5 AND operation is performed between bits of AC and DR, and resulting bits are stored to AC.



D_0T_4: DR← M[AR]

D_0T_5: AC ← AC ∧ DR , SC ← 0

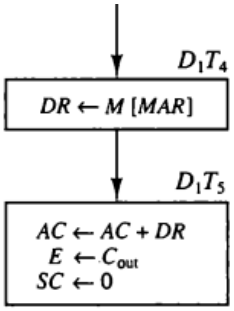
ADD [D_1] : Add memory word to AC

This operation performes ADD arithmetic operation between bits of AC and DR.

At D_1T_4 data of M[AR] is fetched and stored to DR.

At D_1T_5 ADD operation is performed between bits of AC and DR, and resulting bits are stored to AC.

Final carry C_{out} is stored in extended accumulator E.



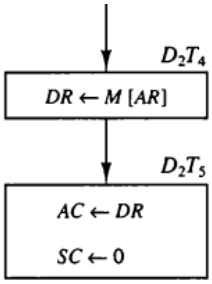
D_1T_4: DR← M[AR]

D_1T_5: AC ← AC + DR , E← C_{out} , SC ← 0

LDA [D_2] : Load memory word to AC

At D_2T_4 data of M[AR] is fetched and stored to DR as input of AC is not directly connected with memory unit.

At D_2T_5 data of DR is transferred to AC.

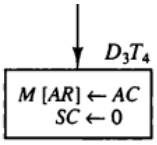


D_2T_4: DR← M[AR]

D_2T_5: AC ← DR , SC ← 0

STA [D_3] : Store content of AC in memory word

At D_3T_4 data of AC is stored directly at M[AR] as input of AC is directly connected with memory unit.



D_3T_4: M[AR]←AC , SC←0

BUN [D_4] : Branch unconditionally

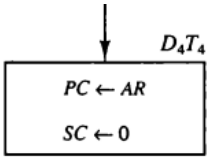
This instruction transfers the execution of program to the address specified by effective address.

PC holds the address of the instruction to be read from memory in the next instruction cycle.

The BUN instruction specify an instruction out of sequence so that the program branches (or jumps) unconditionally.

The effective address frpom AR is transferred to PC.

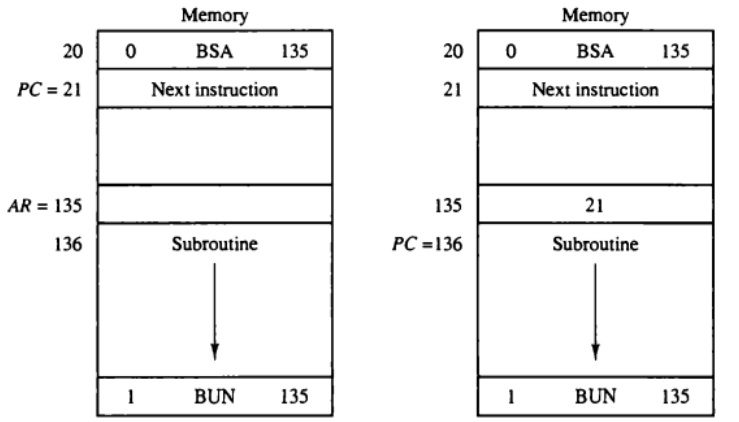
The next instruction is then fetched and executed from the memory address given by the new value in PC .



D_4T_4: PC←AR , SC←0

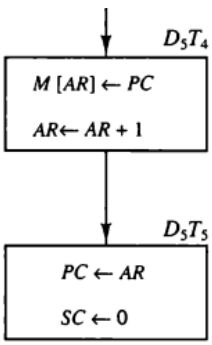
BSA [D_5] : Branch and save return address

- 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 (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.
- The diagram demonstrates how this instruction is used with a subroutine.



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

- 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] ← 21, PC ← 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.

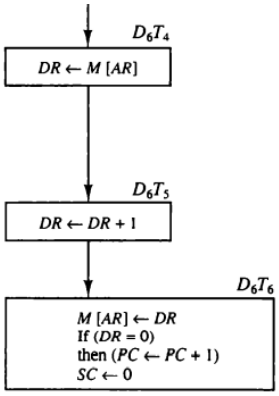


D_5T_4: M[AR]←PC , AR ← AR + 1

D_5T_5: PC←AR , SC←0

ISZ [D_6] : Increment and skip if zero

- This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- The programmer usually stores a negative number (in 2's complement) in the memory word.
- As this negative number is repeatedly incremented by one, it eventually reaches the value of zero.
- At that time PC is incremented by one in order to skip the next instruction in the program. M[AR] is transfered into DR, then DR is incremented by 1, and incremented value of DR is stored back into M[AR].



D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR + 1

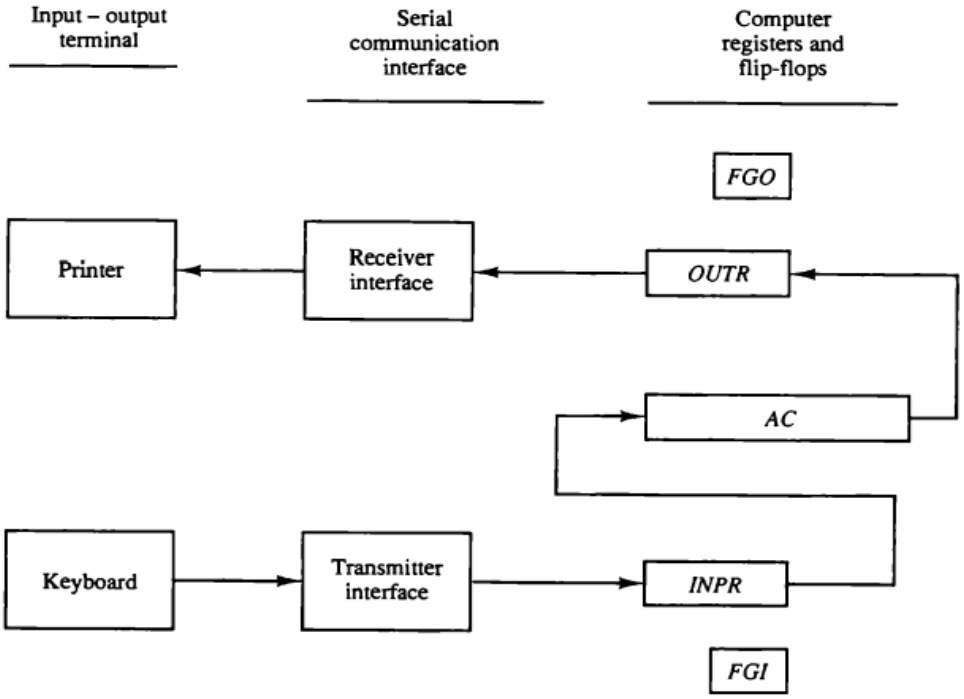
D_6T_6: M[AR] ← DR, If (DR = 0) then (PC←PC + 1) , SC←0

Input/Output Instructions:

The I/O instructions are similar to the register reference instructions. They also use a 16-bit format to specify an operation. The difference is that the leftmost bits here are always 1111, which is the binary equivalent of hexadecimal F.

Input-Output Configuration

The input-output configuration is shown in diagram below.



- The transmitter interface receives serial information from the keyboard and transmits it to INPR.
- The receiver interface receives information from OUTR and sends it to the printer serially.
- The input register INPR consists of eight bits and holds an 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.
- **Input Process:**
- Initially, the input flag FGI is cleared to 0.
    - When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.
    - The computer checks the flag bit; if it is 1, the information from INPR is transferred into AC and FGI is cleared to 0.
    - Once the flag is cleared, new information can be shifted into INPR by striking another key.
- **Out Process:**
- 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 into OUTR and FGO is cleared to 0.

- The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

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

Input/Output Instructions:

The I/O devices are given specific addresses. The processor similarly views the I/O operations as memory operations. It concerns commands that include the address for the device.

The I/O instructions are needed for the following objectives −

It is used to analyzing flag bits.

It can transfer data to or from the AC register.

It can controlling interrupts.

The I/O instructions have the opcode as 1111. They are recognized through the control when D7 = 1 and I=1. The operation to be executed is determined by the different remaining bits.

There are various I/O Instructions which are shown in the table −

Symbol	Description
INP	The INP instruction address the information from the INPR to AC which has 8 low order bits. It also clears the input flag to 0.
OUT	It can send the 8 low order bits from AC into output register OUTPR. It also clears the output flag to 0.
SKI	These are the status flags. They skip the next instructions when flag = 1, They are primarily branching instructions.
SKO	It is similar to SKI.
ION	Enables (set) interrupt.
IOF	Disables (clear) interrupt.

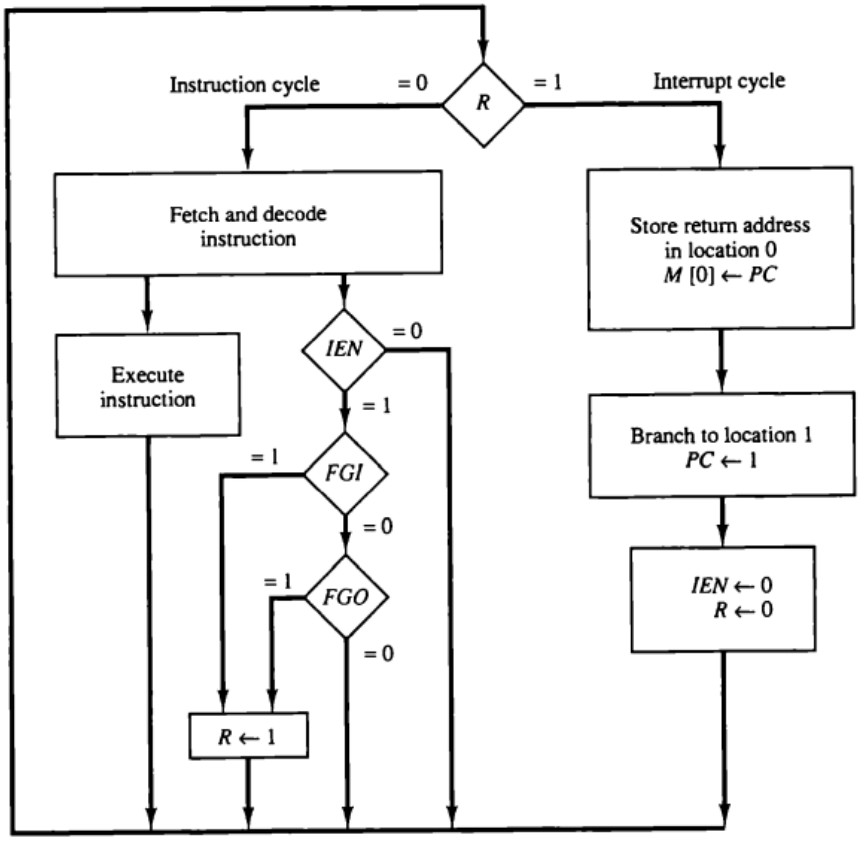
Interrupt Cycle:

"The interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current working process."

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 1 (with the ION instruction), the computer can be interrupted.

These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.

The **Interrupt cycle flowchart** shows interrupt handling by computer.



An interrupt flip-flop R is included in the computer.

The instruction cycle

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, control continues with the next instruction cycle.

If IEN is 1, control checks the flag bits FGI and FGO. 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.

The interrupt cycle

The interrupt cycle is a hardware implementation of a branch and save return address (BSA) operation.

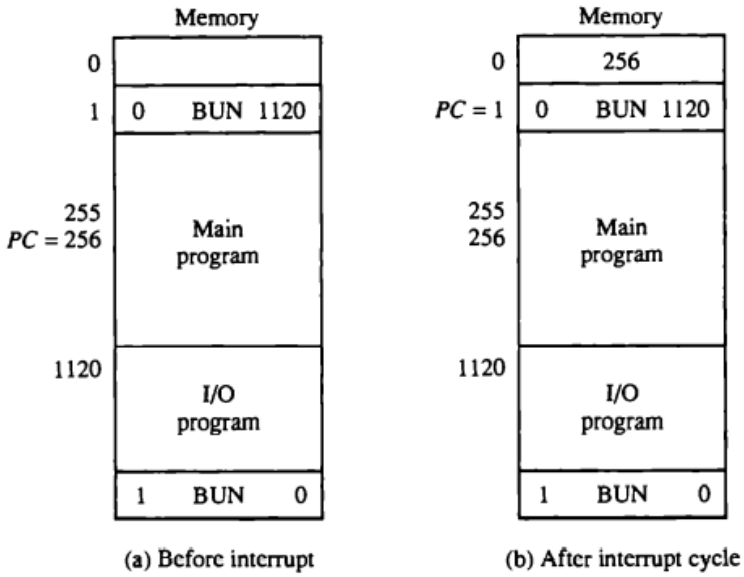
The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.

The memory location at address 0 is the place for storing the return address.

Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.

Interrupt Handling

An example that shows what happens **during the interrupt cycle** is shown in diagram below.



Suppose that 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.

An input-output service program is stored in memory starting from address 1120.

An instruction BUN 1120 is stored at address 1 in memory.

Fig. (a).

When control reaches timing signal T_0 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.

At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC.

The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.

This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.

Fig. (b).

The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0.

This instruction is placed at the end of the I/O service program.

After this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address.

The effective address is in location 0 and is the return address that was stored there during the previous interrupt cycle.

The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

The complete computer description:

The complete computer description includes both, the instruction cycle and the interrupt cycle.

Interrupt cycle because there may arise a case of I/O operation anytime during normal operation.

Instruction cycle because in absence of interrupts, the CPU is always busy with stored program instructions.

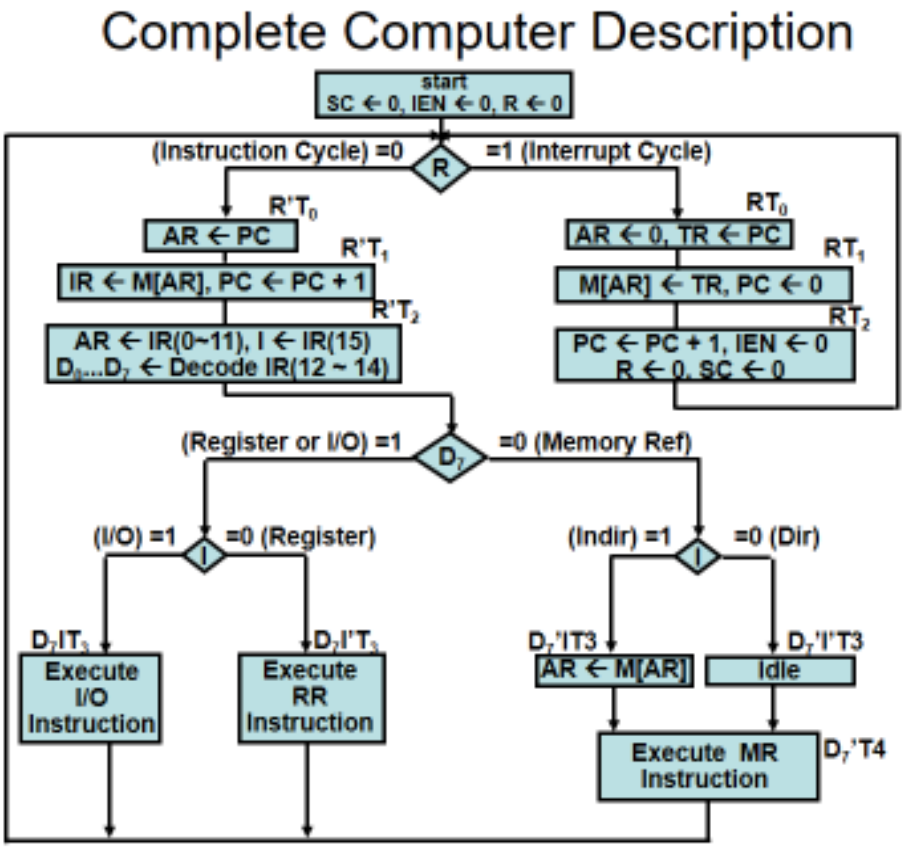
A flip flop R is used as a condition to determine the type of operation.

When R=0, the instruction cycle continues.

Similarly when R=1, the computer goes through an interrupt cycle.

Here it must be noted that an interrupt is signaled by IEN and R is just to make it sure that another interrupt does not get entertained while processing of an interrupt.

Consider the following diagram:



A basic computer consist of the following hardware components:

A memory unit of 4096 X 16 bits

Nine Registers- PC, AR, IR, DR, AC, TR, INPR, OUTR and SC

Five Flip-Flops- I, R, IEN, FGI and FGO

Two decoders- a 3 to 8 operation decoder and a 4 to 16 timing decoder

A 16-bit common bus

Control Logic Gates

Arithmetic and Logic Unit connected to AC

Control functions and microoperations for a basic computer are as follows:

Instruction Cycle

Fetch Phase:

$R'T_0$: $AR \leftarrow PC$
$R'T_1$: $IR \leftarrow M[AR], PC \leftarrow PC + 1$

Decode Phase:

$R'T_2$: $D_0 \ldots D_7 \leftarrow$ Decode IR[12-14],
$AR \leftarrow IR[0\text{-}11]$,
$I \leftarrow IR[15]$

Indirect address in Memory Reference Instructions:

$R'T_0$: $AR \leftarrow PC$
$R'T_1$: $IR \leftarrow M[AR], PC \leftarrow PC + 1$

Interrupt Cycle

$T_0'T_1'T_2'(IEN)(FGI + FGO)$: $R \leftarrow 1$
RT_0 : $AR \leftarrow 0, TR \leftarrow PC$
RT_1 : $M[AR] \leftarrow TR, PC \leftarrow 1$
RT_2 : $IEN \leftarrow 0, R \leftarrow 0$

UNIT-3

CENTRAL PROCESSING UNIT, PIPELINING

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.

Registers are the most important components of CPU. Each register performs a specific function. A brief description of most important CPU's registers and their functions are given below:

1. **Memory Address Register (MAR):** This register holds the address of memory where CPU wants to read or write data. When CPU wants to store some data in the memory or reads the data from the memory, it places the address of the required memory location in the MAR.

2. **Memory Buffer Register (MBR):** This register holds the contents of data or instruction read from, or written in memory. The contents of instruction placed in this register are transferred to the Instruction Register, while the contents of data are transferred to the accumulator or I/O register. In other words you can say that this register is used to store data/instruction coming from the memory or going to the memory.

3. **Program Counter (PC):** Program Counter register is also known as Instruction Pointer Register. This register is used to store the address of the next instruction to be fetched for execution. When the instruction is fetched, the value of IP is incremented. Thus this register always points or holds the address of next instruction to be fetched.

4. **Instruction Register (IR):** Once an instruction is fetched from main memory, it is stored in the Instruction Register. The control unit takes instruction from this register, decodes and executes it by sending signals to the appropriate component of computer to carry out the task.

5. **Accumulator Register:** The accumulator register is located inside the ALU; it is used during arithmetic & logical operations of ALU. The control unit stores data values fetched from main memory in the accumulator for arithmetic or logical operation. This register holds the initial data to be operated upon, the intermediate results, and the final result of operation. The final result is transferred to main memory through MBR.

6. **Stack Control Register:** A stack represents a set of memory blocks; the data is stored in and retrieved from these blocks in an order, i.e. First In and Last Out (FILO). The Stack Control Register is used to manage the stacks in memory. The size of this register is 2 or 4 bytes.

7. **Flag Register:** The Flag register is used to indicate occurrence of a certain condition during an operation of the CPU. It is a special purpose register with size one byte or two bytes. Each bit of the flag register constitutes a flag (or alarm), such that the bit value indicates if a specified condition was encountered while executing an instruction.

For example, if zero value is put into an arithmetic register (accumulator) as a result of an arithmetic operation or a comparison, then the zero flag will be raised by the CPU. Thus, the subsequent instruction can check this flag and when a zero flag is "ON" it can take an appropriate route in the algorithm.

Memory

This unit can store instructions, data, and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer. Functions of the memory unit are −

1. It stores all the data and the instructions required for processing.
2. It stores intermediate results of processing.
3. It stores the final results of processing before these results are released to an output device.
4. All inputs and outputs are transmitted through the main memory.

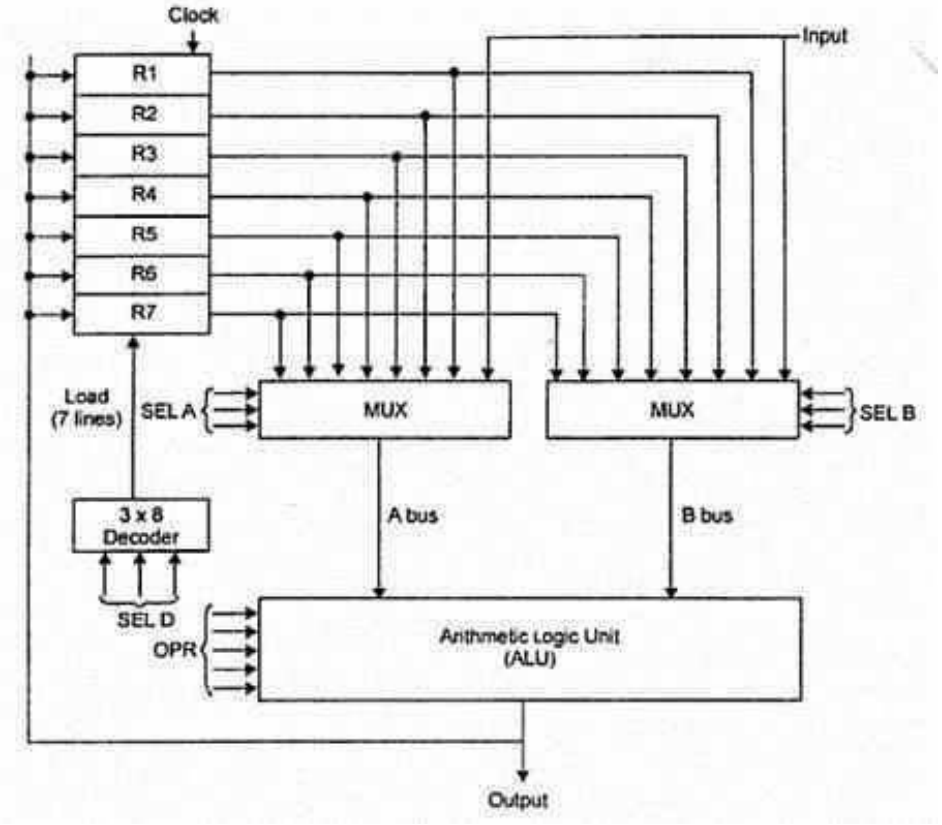
❖ **General Register organization :**

Generally CPU has seven general registers. Register organization show how registers are selected and how data flow between register and ALU. A decoder is used to select a

particular register. The output of each register is connected to two multiplexers to form the two buses A and B. The selection lines in each multiplexer select the input data for the particular bus.

The A and B buses form the two inputs of an ALU. The operation select lines decide the micro operation to be performed by ALU. The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

A bus organization for seven CPU registers



EXAMPLE:

- To perform the operation ***R3 = R1+R2*** **we** have to provide following binary selection variable to the selectinputs.

1. **SEL A**: **001** -To place the contents of R1 into bus A.
2. **SEL B**: **010** - to place the contents of R2 into bus B
3. **SEL OPR**: **10010** – to perform the arithmetic addition A+B
4. **SEL REG or SEL D**: **011** – to place the result available on output bus in R3.

❖ **Register and multiplexer input selection code:**

Binary code	SELA	SELB	SELD or SELREG
000	Input	Input	---
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

❖ **Operation with symbol:**

Operation selection code	Operation	symbol
0000	Transfer A	TSFA
0001	Increment A	INC A
0010	A+B	ADD
0011	A-B	SUB
0100	Decrement A	DEC
0101	A AND B	AND
0110	A OR B	OR
0111	A XOR B	XOR
1000	Complement A	COMA
1001	Shift right A	SHR
1010	Shift left A	SHL

❖ **What is CONTROL WORD?**

- The combined value of a binary selection inputs specifies the control word.
- It consists of four fields SELA, SELB, and SELD or SELREG contains three bit each and SELOPR field contains four bits thus the total bits in the control word are 13-bits.

SEL A	**SELB**	**SELREG OR SELD**	**SELOPR**

FORMATE OF CONTROL WORD

1. The three bit of SELA select a source registers of the input of the ALU.
2. The three bits of SELB select a source registers of the b input of the ALU.
3. The three bits of SELED or SELREG select a destination register using the decoder.
4. The four bits of SELOPR select the operation to be performed by ALU.

CONTROL WORD FOR OPERATION R2 = R1+R3

SEL A	**SEL B**	**SEL D OR SELREG**	**SELOPR**
001	011	010	0010

Note: Control words for all micro operation are stored in the control memory

Example:

MICROOPERATION	**SEL A**	**SEL B**	**SEL D OR SELREG**	**SELOPR**	**CONTROL WORD**			
R2 = R1+R3	R1	R3	R2	ADD	00 1	01 1	01 0	001 0

❖ STACK ORGANIZATION:

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called ***Stack Pointer***.

- **Stack Operations**

The two operations of a stack are:

1. **Push:** Inserts an item on top of stack.
2. **Pop:** Deletes an item from top of stack.

- **Implementation of Stack**

In digital computers, stack can be implemented in two ways:

1. Register Stack
2. Memory Stack

1. Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

2. Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as *stack pointer*.

❖ INSTRUCTION FORMATS:

The physical and logical structure of computers is normally described in reference manuals provided with the system. Such manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities. They list all

hardware-implemented instructions, specify their binary code format, and provide a precise definition of each instruction. A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor registers.
3. A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

1 Single accumulator organization.
2 General register organization.
3 Stack organization.

- **THREE-ADDRESS INSTRUCTIONS:**

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) ∗ (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

ADD R1, A, B	R1 ← M [A] + M [B]
ADD R2, C, D	R2 ← M [C] + M [D]
MUL X, R1, R2	M [X] ← R1 ∗ R2

It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions

require too many bits to specify three addresses. An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

TWO-ADDRESS INSTRUCTIONS

Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) ∗ (C + D) is as follows:

MOV R1, A	R1 ← M [A]
ADD R1, B	R1 ← R1 + M [B]
MOV R2, C	R2 ← M [C]
ADD R2, D	R2 ← R2 + M [D]
MUL R1, R2	R1 ← R1∗ R2
MOV X, R1	M [X] ← R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

- **ONE-ADDRESS INSTRUCTIONS**

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of tall operations. The program to evaluate X = (A + B) ∗ (C + D) is

LOAD A	AC ← M [A]
ADD B	AC ← A [C] + M [B]
STORE T	M [T] ← AC
LOAD C	AC ← M [C]
ADD D	AC ← AC + M [D]
MUL T	AC ← AC ∗ M [T]
STORE X	M [X] ← AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

❖ ZERO-ADDRESS INSTRUCTIONS:

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) ∗ (C + D) will be written for a stack organized computer. (TOS stands for top of stack)

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← (A + B)
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← (C + D)
MUL	TOS ← (C + D) ∗ (A + B)
POP X	M [X] ← TOS

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

❖ ADDRESSING MODES

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution in dependent on the addressing mode of the instruction. The addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

1 Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions. Instruction format with mode field Zero-address instructions in a stack-

organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2 Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value. It was mentioned that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in the register mode.

3 Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2k registers.

4 Register Indirect Mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the Op code Mode Address operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address for the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

5 Auto increment or Auto decrement Mode: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access. The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it

is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational-type instruction. It is the address where control branches in response to a branch-type instruction.

6 Direct Address Mode: In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.

7 Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

8 Relative Address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

9 Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory

relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stores in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation. Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

10 Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of the base register requires updating to reflect the beginning of a new memory segment.

CONCEPT OF CPU DESIGN, RISC & CISC

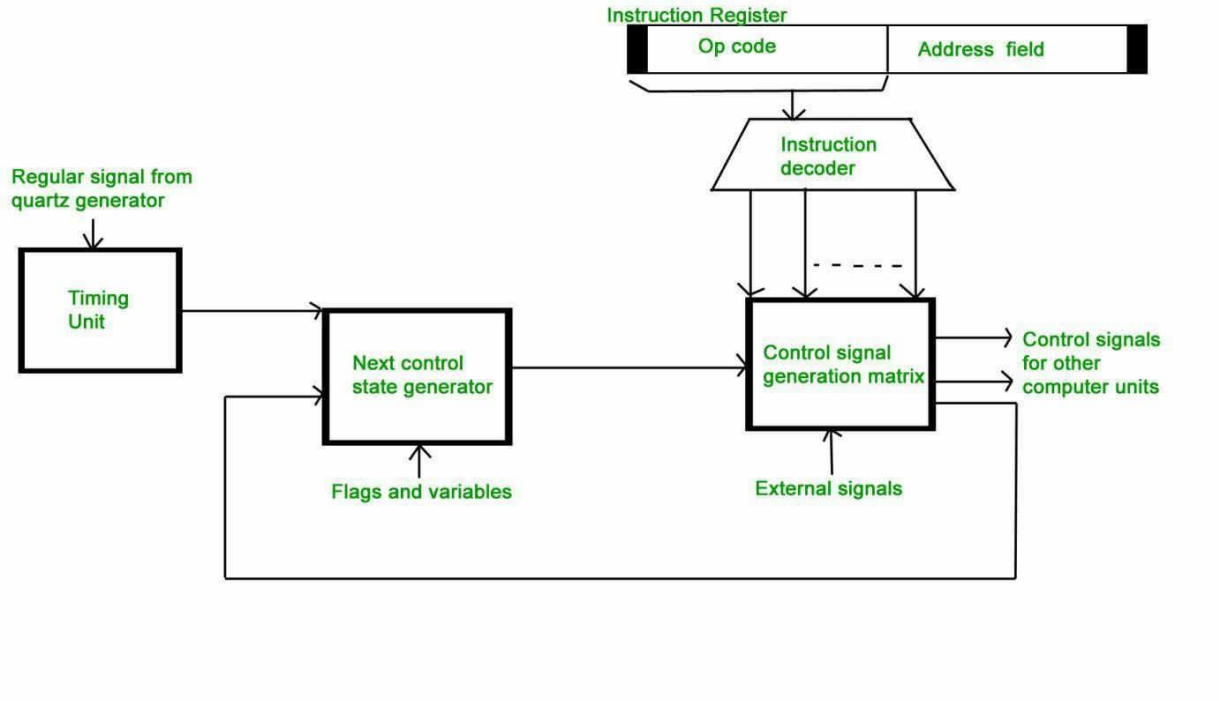
Hardwired v/s Micro-programmed Control Unit

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit –

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

1. Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
2. Hardwired control is faster than micro-programmed control.
3. A controller that uses this approach can operate at high speed.

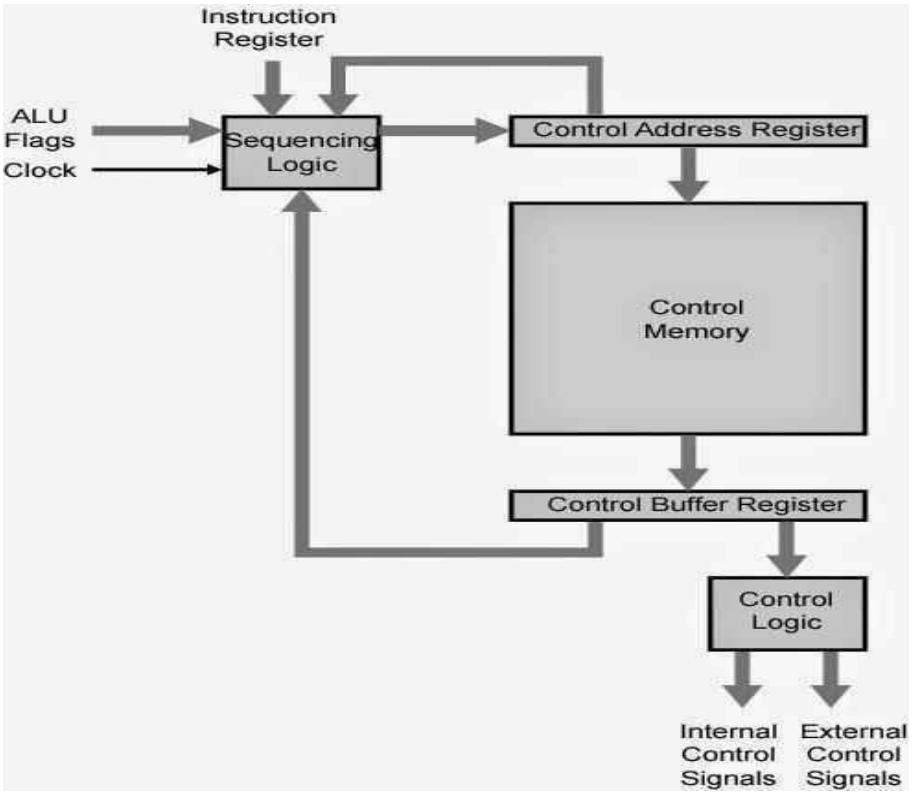


Characteristics:

1. It uses flags, decoder, logic gates and other digital circuits.
2. As name implies it is a hardware control unit.
3. On the basis of input Signal output is generated.
4. Difficult to design, test and implement.
6. Inflexible to modify.
7. Faster mode of operation.
8. Expensive and high error.
9. Used in RISC processor.

Micro-programmed Control Unit –

1. The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
2. Control signals are generated by a program are similar to machine language programs.
3. Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.



Characteristics

1. It uses sequence of micro-instruction in micro programming language.
2. It is mid-way between Hardware and Software.
3. It generates a set of control signal on the basis of control line.
4. Easy to design, test and implement.
5. Flexible to modify.
6. Slower mode of operation.
7. Cheaper and less error.
8. Used in CISC processor.

Reduced Instruction Set Computer

A reduced instruction set computer (RISC) is a computer that uses a central processing unit (CPU) that implements the processor design principle of simplified instructions. To date, RISC is the most efficient CPU architecture technology.

This architecture is an evolution and alternative to complex instruction set computing (CISC). With RISC, the basic concept is to have simple instructions that do less but execute very quickly to provide better performance.

The most basic RISC feature is a processor with a small core logic that allows engineers to increase the register set and increase internal parallelism by using the following:

1. Thread level parallelism: Increases the number of parallel threads executed by the CPU
2. Instruction level parallelism: Increases the speed of the CPU's executing instructions

The words "reduced instruction set" are often misinterpreted to refer to a reduced number of instructions. However, this is not the case, as several RISC processors, like the PowerPC, have numerous instructions. At the opposite end of the spectrum, the DEC PDP-8, a CISC CPU, has only eight basic instructions. Reduced instruction actually means that the amount of work done by each instruction is reduced in terms of number of cycles - at most only a single data memory cycle - compared to CISC CPUs, in which dozens of cycles are required prior to completing the entire instruction. This results in faster processing.

CISC

The main intend of the CISC processor architecture is to complete task by using less number of assembly lines. For this purpose, the processor is built to execute a series of operations. Complex instruction is also termed as MULT, which operates memory banks of a computer directly without making the compiler to perform storing and loading functions.

Features of CISC Architecture

1. To simplify the computer architecture, CISC supports microprogramming.
2. CISC have more number of predefined instructions which makes high level languages easy to design and implement.
3. CISC consists of less number of registers and more number of addressing modes, generally 5 to 20.
4. CISC processor takes varying cycle time for execution of instructions – multi-clock cycles.
5. Because of the complex instruction set of the CISC, the pipelining technique is very difficult.

6. CISC consists of more number of instructions, generally from 100 to 250.
7. Special instructions are used very rarely.
8. Operands in memory are manipulated by instructions.

Advantages of CISC architecture

1. Each machine language instruction is grouped into a microcode instruction and executed accordingly, and then are stored inbuilt in the memory of the main processor, termed as microcode implementation.
2. As the microcode memory is faster than the main memory, the microcode instruction set can be implemented without considerable speed reduction over hard wired implementation.
3. Entire new instruction set can be handled by modifying the micro program design.
4. CISC, the number of instructions required to implement a program can be reduced by building rich instruction sets and can also be made to use slow main memory more efficiently.
5. Because of the superset of instructions that consists of all earlier instructions, this makes micro coding easy.

Drawbacks of CISC

1. The amount of clock time taken by different instructions will be different – due to this – the performance of the machine slows down.
2. The instruction set complexity and the chip hardware increases as every new version of the processor consists of a subset of earlier generations.
3. Only 20% of the existing instructions are used in a typical programming event, even though there are many specialized instructions in existence which are not even used frequently.
4. The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the condition code bits – so, the compiler has to examine the condition code bits before this happens.

Difference between CISC and RISC

Architectural Characteristics	**Complex Instruction Set Computer(CISC)**	**Reduced Instruction Set Computer(RISC)**
Instruction size and format	Large set of instructions with variable formats (16-64 bits per instruction).	Small set of instructions with fixed format (32 bit).
Data transfer	Memory to memory.	Register to register.
CPU control	Most micro coded using control memory (ROM) but modern CISC use hardwired control.	Mostly hardwired without control memory.
Instruction type	Not register based instructions.	Register based instructions.
Memory access	More memory access.	Less memory access.
Clocks	Includes multi-clocks.	Includes single clock.
Instruction nature	Instructions are complex.	Instructions are simple.

Introduction:

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called peripheral devices. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called peripherals.

For example: Keyboards, display units and printers are common peripheral devices.

There are three types of peripherals:

1.Input peripherals : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.

2. Output peripherals: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc

3. Input-Output peripherals: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

Interfaces

Interface is a shared boundary btween two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Inteface

2. I/O Interface

Let's understand the I/O Interface in details

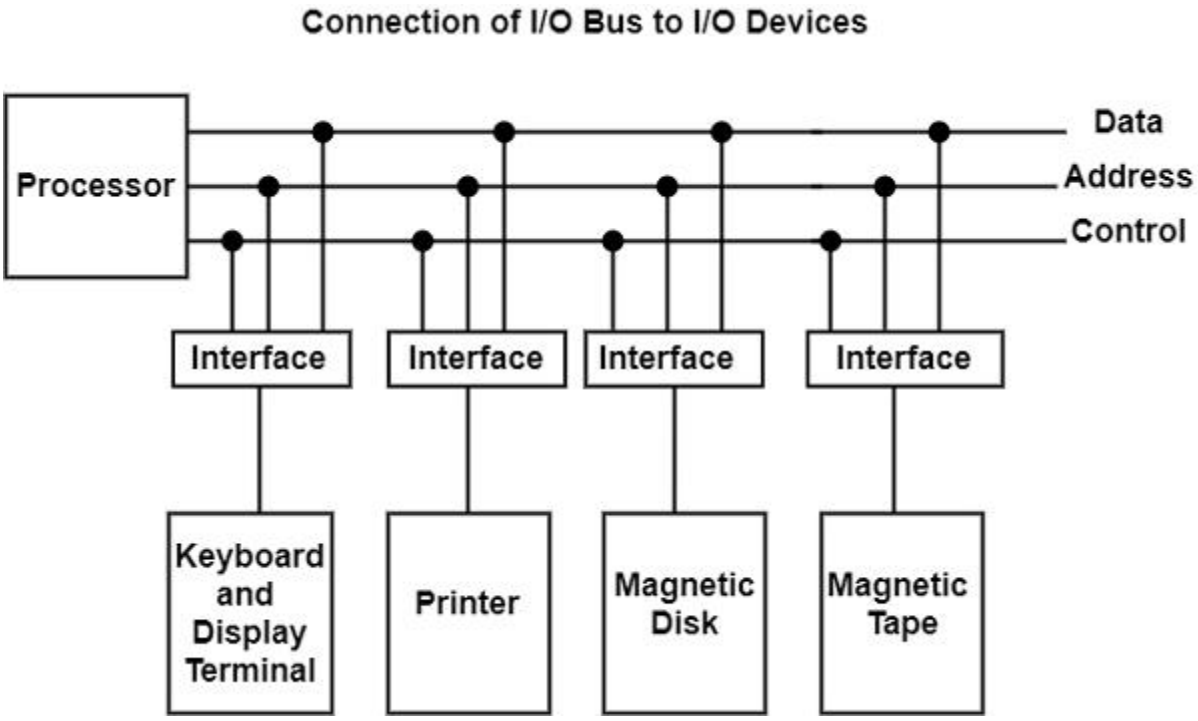
Input-Output Interface

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called **input-output interface units** because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

The I/O interface supports a method by which data is transferred between internal storage and external I/O devices.

I/O Bus and Interface Modules

The I/O bus is the route used for peripheral devices to interact with the computer processor. A typical connection of the I/O bus to I/O devices is shown in the figure.



The I/O bus includes data lines, address lines, and control lines. In any general-purpose computer, the magnetic disk, printer, and keyboard, and display terminal are commonly employed.

Each peripheral unit has an interface unit associated with it. Each interface decodes the control and address received from the I/O bus.

It can describe the address and control received from the peripheral and supports signals for the peripheral controller. It also conducts the transfer of information between peripheral and processor and also integrates the data flow.

The I/O bus is linked to all peripheral interfaces from the processor. The processor locates a device address on the address line to interact with a specific device. Each interface contains an address decoder attached to the I/O bus that monitors the address lines.

When the address is recognized by the interface, it activates the direction between the bus lines and the device that it controls. The interface disables the peripherals whose address does not equivalent to the address in the bus.

An interface receives any of the following four commands −

1. Control − A command control is given to activate the peripheral and to inform its next task. This control command depends on the peripheral, and each peripheral receives its sequence of control commands, depending on its mode of operation.
2. Status − A status command can test multiple test conditions in the interface and the peripheral.
3. Data Output − A data output command creates the interface counter to the command by sending data from the bus to one of its registers.
4. Data Input − The data input command is opposite to the data output command. In data input, the interface gets an element of data from the peripheral and places it in its buffer register.

The internal operations in an individual unit of a digital system are synchronized using clock pulse. It means clock pulse is given to all registers within a unit. And all data transfer among internal registers occurs simultaneously during the occurrence of the clock pulse. Now, suppose any two units of a digital system are designed independently, such as CPU and I/O interface.

If the registers in the I/O interface share a common clock with CPU registers, then transfer between the two units is said to be synchronous. But in most cases, the internal timing in each unit is independent of each other, so each uses its private clock for its internal registers. In this case, the two units are said to be asynchronous to each other, and if data transfer occurs between them, this data transfer is called Asynchronous Data Transfer.

But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data. These two methods can achieve this asynchronous way of data transfer:

Strobe control: A strobe pulse is supplied by one unit to indicate to the other unit when the transfer has to occur.

Handshaking: This method is commonly used to accompany each data item being transferred with a control signal that indicates data in the bus. The unit receiving the data item responds with another signal to acknowledge receipt of the data.

The strobe pulse and handshaking method of asynchronous data transfer is not restricted to I/O transfer. They are used extensively on numerous occasions requiring the transfer of data between two independent units. So, here we consider the transmitting unit as a source and receiving unit as a destination.

For example, the CPU is the source during output or write transfer and the destination unit during input or read transfer.

Therefore, the control sequence during an asynchronous transfer depends on whether the transfer is initiated by the source or by the destination.

So, while discussing each data transfer method asynchronously, you can see the control sequence in both terms when it is initiated by source or by destination. In this way, each data transfer method can be further divided into parts, source initiated and destination initiated.

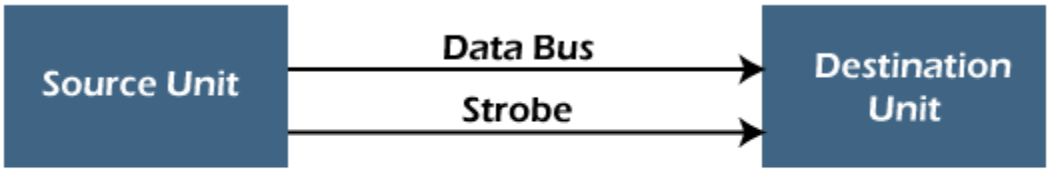
Asynchronous Data Transfer Methods

The asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate when they send the data. Thus, the two methods can achieve the asynchronous way of data transfer.

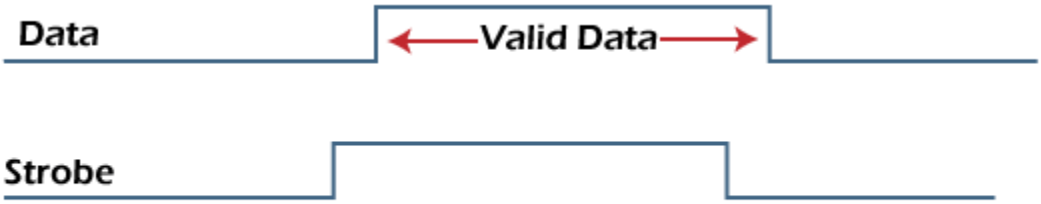
1. Strobe Control Method

The Strobe Control method of asynchronous data transfer employs a single control line to time each transfer. This control line is also known as a strobe, and it may be achieved either by source or destination, depending on which initiate the transfer.

Source initiated strobe: In the below block diagram, you can see that strobe is initiated by source, and as shown in the timing diagram, the source unit first places the data on the data bus.



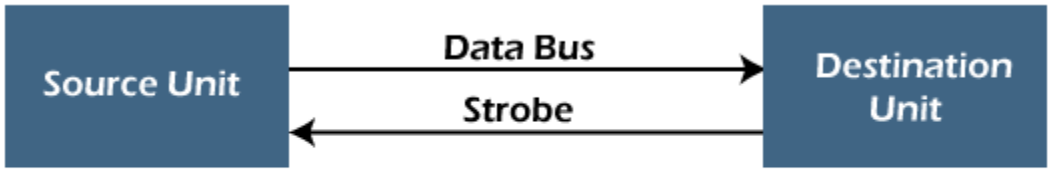
(a) Block Diagram



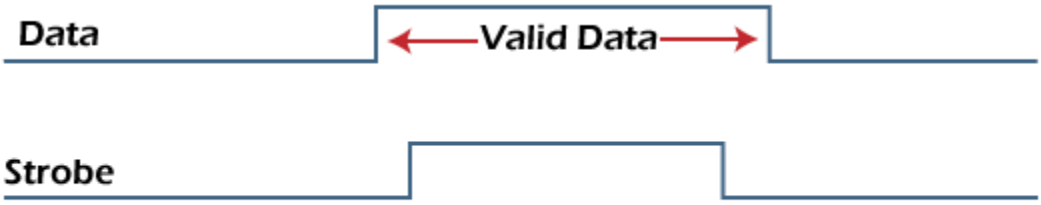
(b) Timing Diagram

After a brief delay to ensure that the data resolve to a stable value, the source activates a strobe pulse. The information on the data bus and strobe control signal remains in the active state for a sufficient time to allow the destination unit to receive the data. The destination unit uses a falling edge of strobe control to transfer the contents of a data bus to one of its internal registers. The source removes the data from the data bus after it disables its strobe pulse. Thus, new valid data will be available only after the strobe is enabled again. In this case, the strobe may be a memory-write control signal from the CPU to a memory unit. The CPU places the word on the data bus and informs the memory unit, which is the destination.

2. Destination initiated strobe: In the below block diagram, you see that the strobe initiated by destination, and in the timing diagram, the destination unit first activates the strobe pulse,informing the source to provide the data.



(a) Block Diagram



(b) Timing Diagram

The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain on the bus long enough for the destination unit to accept it. The falling edge of the strobe pulse can use again to trigger a destination register. The destination unit then disables the strobe. Finally, and source removes the data from the data bus after a determined time interval.

In this case, the strobe may be a memory read control from the CPU to a memory unit. The CPU initiates the read operation to inform the memory, which is a source unit, to place the selected word into the data bus.

Handshaking Method

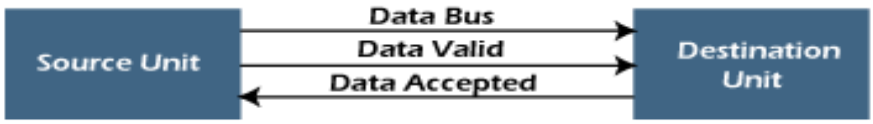
The strobe method has the disadvantage that the source unit that initiates the transfer has no way of knowing whether the destination has received the data that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has placed data on the bus.

So this problem is solved by the handshaking method. The handshaking method introduces a second control signal line that replays the unit that initiates the transfer.

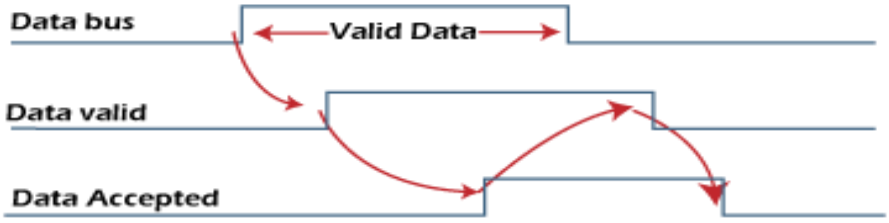
In this method, one control line is in the same direction as the data flow in the bus from the source to the destination. The source unit uses it to inform the destination unit whether there are valid data in the bus.

The other control line is in the other direction from the destination to the source. This is because the destination unit uses it to inform the source whether it can accept data. And in it also, the sequence of control depends on the unit that initiates the transfer. So it means the sequence of control depends on whether the transfer is initiated by source and destination.

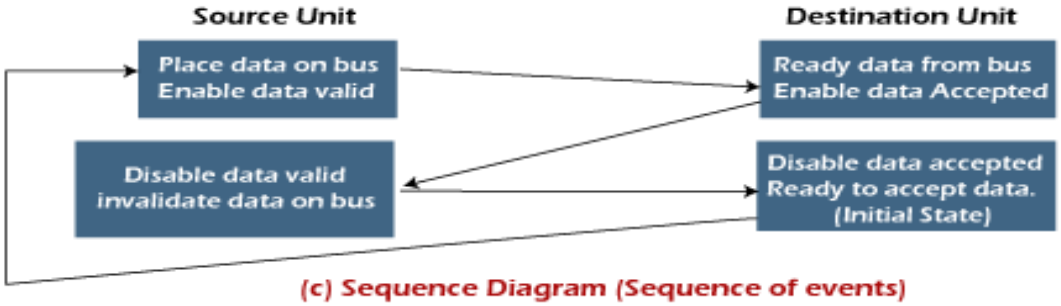
Source initiated handshaking: In the below block diagram, you can see that two handshaking lines are "**data valid**", which is generated by the source unit, and "**data accepted**", generated by the destination unit.



(a) Block Diagram



(b) Timing Diagram

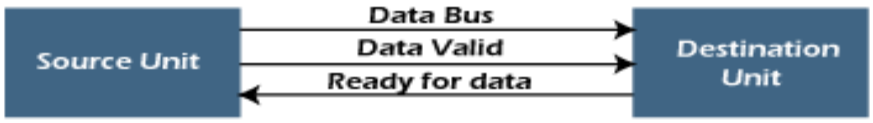


(c) Sequence Diagram (Sequence of events)

The timing diagram shows the timing relationship of the exchange of signals between the two units. The source initiates a transfer by placing data on the bus and enabling its data valid signal. The destination unit then activates the data accepted signal after it accepts the data from the bus. The source unit then disables its valid data signal, which invalidates the data on the bus. After this, the destination unit disables its data accepted signal, and the system goes into its initial state. The source unit does not send the next data item until after the destination unit shows readiness to accept new data by disabling the data accepted signal. This sequence of events described in its sequence diagram, which shows the above sequence in which the system is present at any given time.

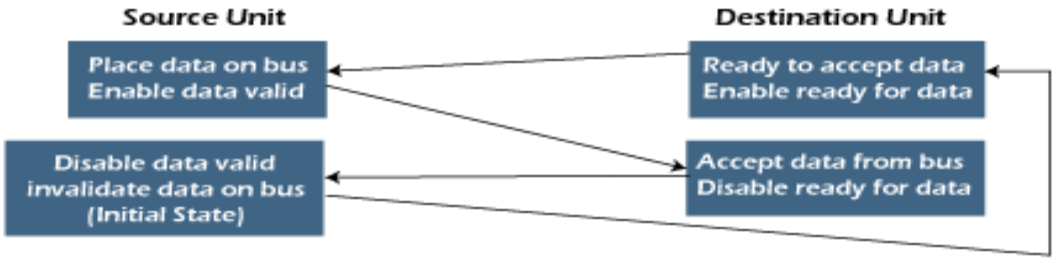
Destination initiated handshaking: In the below block diagram, you see that the two handshaking lines are "**data valid**", generated by the source unit, and "**ready for data**" generated by the destination unit.

Note that the name of signal data accepted generated by the destination unit has been changed to ready for data to reflect its new meaning.



(a) Block Diagram

(b) Timing Diagram



(c) Sequence Diagram (Sequence of events)

The destination transfer is initiated, so the source unit does not place data on the data bus until it receives a ready data signal from the destination unit. After that, the handshaking process is the same as that of the source initiated.

The sequence of events is shown in its sequence diagram, and the timing relationship between signals is shown in its timing diagram. Therefore, the sequence of events in both cases would be identical.

Advantages of Asynchronous Data Transfer

Asynchronous Data Transfer in computer organization has the following advantages, such as:

It is more flexible, and devices can exchange information at their own pace.

In addition, individual data characters can complete themselves so that even if one packet is corrupted, its predecessors and successors will not be affected.

It does not require complex processes by the receiving device. Furthermore, it means that inconsistency in data transfer does not result in a big crisis since the device can keep up with the data stream.

It also makes asynchronous transfers suitable for applications where character data is generated irregularly.

Asynchronous Serial Transmission:

The transfer of data between two units is serial or parallel.

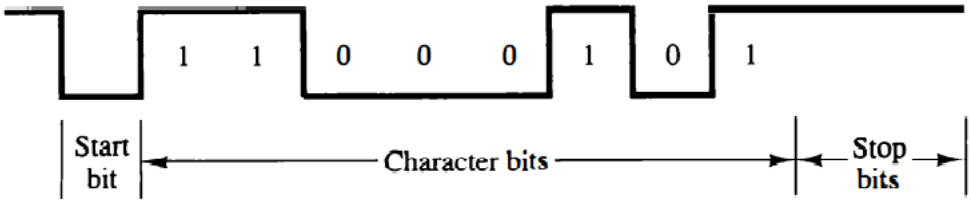
In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time. Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.

In this technique each character consists of three points :

i. Start bit
ii. Character bit
iii. Stop bit

i. Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.
ii. Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.
iii. Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Serial Transmission of Asynchronous is done by two ways:

a) Asynchronous Communication Interface

b) First In First out Buffer

Asynchronous Communication Interface:

It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register. The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission. The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register. CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO):

A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates. When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate. If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

2. Programmed I/O.
3. Interrupt- initiated I/O.
4. Direct memory access(DMA).

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

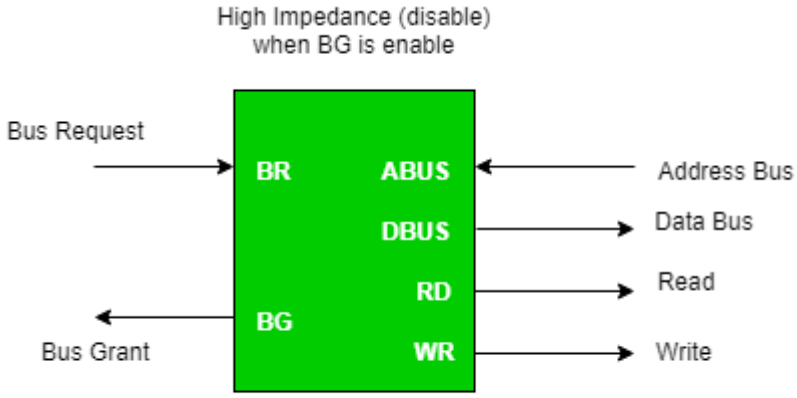


Figure - CPU Bus Signals for DMA Transfer

Bus Request: It is used by the DMA controller to request the CPU to relinquish the control of the buses.

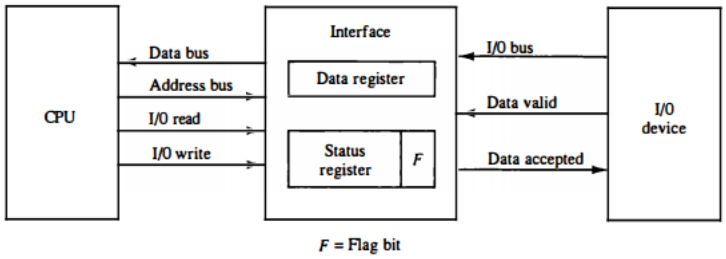
Bus Grant: It is activated by the CPU to inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

Burst Transfer: DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

Steps involved are:
1. Bus grant request time.
2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.
3. Release the control of the bus back to CPU So, total time taken to transfer the N bytes = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Example of Data Transfer:



- When a byte of data is available, the device places it in the I/O bus and enables its data valid line.
- The interface accepts the byte into its data register and enables the data accepted line. The interface sets a bit in the status register to as F or flag bit.
- If the flag is equal to 1, the CPU reads the data from the data register. The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed.
- The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface.

Flow chart:

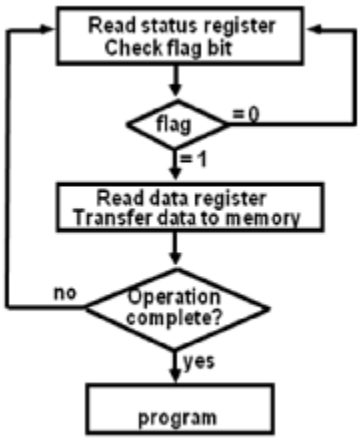


Fig: Flowchart for CPU program to input data

- The transfer of each byte requires three instructions:
    1. Read the status register.
    2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
    3. Read the data register.

Input-output processor (IOP)

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors.

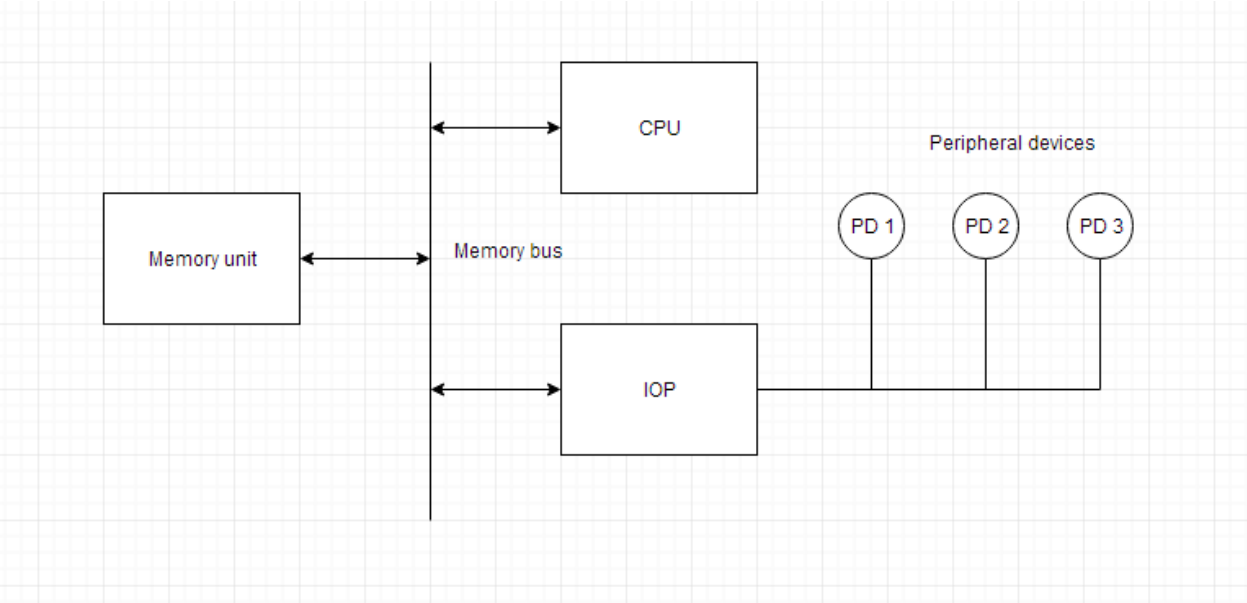
Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram of I/O Processor

Below is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor.

The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program.

The IOP operates independent from CPU and transfer data between peripherals and memory.



The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation.

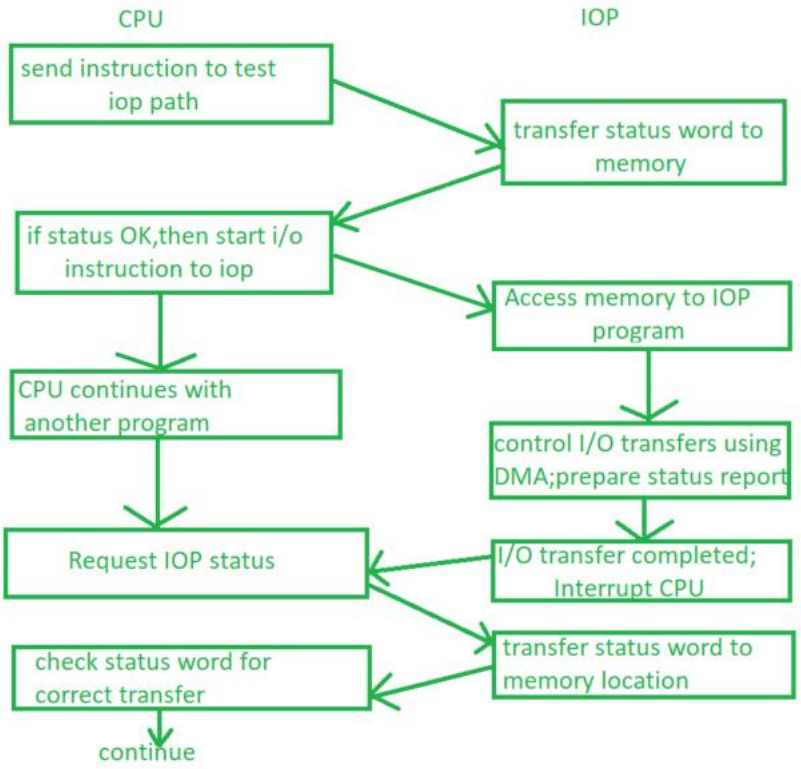
The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt.

Instructions that are read from memory by an IOP are also called commands to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

CPU – IOP Communication:

There is a communication channel between IOP and CPU to perform task which come under computer architecture. This channel explains the commands executed by IOP and CPU while performing some programs. The CPU do not executes the instructions but it assigns the task of initiating operations, the instructions are executed by IOP. I/O transfer is instructed by CPU. The IOP asks for CPU through interrupt. This channel starts by CPU, by giving "test IOP path"

instruction to IOP and then the communication begins as shown in diagram:



between IOP and CPU Whenever CPU gets interrupt from IOP to access memory, it sends test path instruction to IOP. IOP executes and check for status, if the status given to CPU is OK, then CPU gives start instruction to IOP and gives it some control and get back to some another (or same) program, after that IOP is able to access memory for its program. Now IOP start controlling I/O transfer using DMA and create another status report as well. AS soon as this I/O transfer completes IOP once again send interrupt to CPU, CPU again request for status of IOP, IOP check status word from memory location and gives it to CPU. Now CPU check the correctness of status and continues with the same process.

UNIT 5

MEMORY ORGANIZATION

❖ MEMORY CLASSIFICATIONS:

The computer memory stores different kinds of data like input data, output data, intermediate results, etc., and the instructions. Binary digit or bit is the basic unit of memory. A bit is a single binary digit, i.e., 0 or 1. A bit is the smallest unit of representation of data in a computer. However, the data is handled by the computer as a combination of bits. A group of 8 bits form a byte. One byte is the smallest unit of data that is handled by the computer. One byte (8 bit) can store 256 different combinations of bits, and thus can be used to represent 256 different symbols. In a byte, the different combinations of bits fall in the range 00000000 to 11111111. A group of bytes can be further combined to form a word. A word can be a group of 2, 4 or 8 bytes.

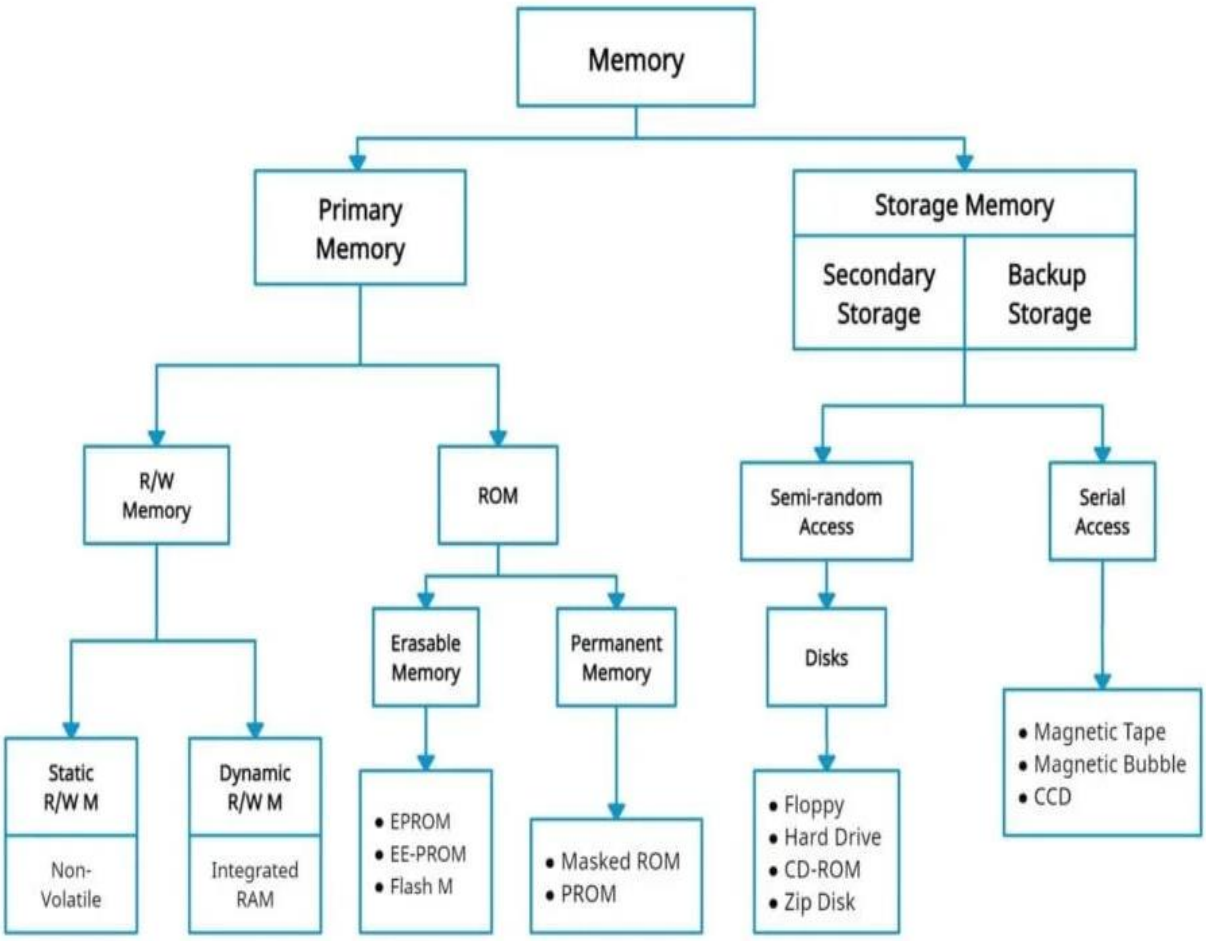
The memory is made up of registers and each register has a group of flipflop that store bits of information; these flip flops are called memory cells.

The number of bits stored in a register is called memory word. Memory chips are available in various word sizes. The user can use this memory to hold program and store data.

❖ CHARACTERISTICS OF MEMORIES:

Volatility
o Volatile {RAM}
o Non-volatile {ROM, Flash memory}
Mutability
o Read/Write {RAM, HDD, SSD, RAM, Cache, Registers…}
o Read Only {Optical ROM (CD/DVD…), Semiconductor ROM}
Accessibility
o Random Access {RAM, Cache}
o Direct Access {HDD, Optical Disks}
o Sequential Access {Magnetic Tapes}

- **MEMORY CLASSIFICATION:**



➢ PRIMARY MEMORY (Main Memory)

Primary memory is the main memory of computer. It is a chip mounted on the motherboard of computer. Primary memory is categorized into two main types:
Random access memory (RAM) and read only memory (ROM). RAM is used for the temporary storage of input data, output data and intermediate results. The input data entered into the computer using the input device, is stored in RAM for processing. After processing, the output data is stored in RAM before being sent to the output device. Any intermediate results generated during the processing of program are also stored in RAM. Unlike RAM, the data once stored in ROM either cannot be changed or can only be changed using some special operations. Therefore, ROM is used to store the data that does not require a change.

- **Types of Primary Memory**

1. **RAM (Random Access Memory)**

The Word "RAM" stands for "random access memory" or may also refer to short term memory. It's called "random" because you can read store data randomly at any time and from any physical location. It is a temporal storage memory. RAM is volatile that only retains all the data as long as the computer powered. It is the fastest type of memory. RAM stores the currently processed data from the CPU and sends them to the graphics unit.

There are generally two broad subcategories of RAM:

• **Static RAM (SRAM):** Static RAM is the form of RAM and made with flip-flops and used for primary storage are volatile. It retains data in latch as long as the computer powered. SRAM is more expensive and consumes more power than DRAM. It used as Cache Memory in a computer system. As technically, SRAM
uses more transistors as compared to DRAM. It is faster compared to DRAM due to the latching arrangement, and they use 6 transistors per data bit as compared to DRAM, which uses one transistor per bit.

• **Dynamic Random Access Memory (DRAM):** It is another form of RAM used as Main Memory, its retains information in Capacitors for a short period (a few milliseconds) even though the computer powered. The Data is Refreshed Periodically to maintain in it. The DRAM is cheaper, but it can store much more information. Moreover, it is also slower and consumes less power than SRAM.

2. **ROM (Read Only Memory)**

ROM is the long-term internal memory. ROM is "Non-Volatile Memory" that retains data without the flow of electricity. ROM is an essential chip with permanently written data or programs. It is similar to the RAM that is accessed by the CPU. ROM comes with pre-written by the computer manufacturer to hold the instructions for booting-up the computer.

There is generally three broad type of ROM:

• **PROM (Programmable Read Only Memory):** PROM stands for programmable ROM. It can be programmed only be done once and read many. Unlike RAM, PROMs retain their contents without the flow of electricity. PROM is also Non volatile memory. The significant difference between a ROM and a PROM is that a ROM comes with pre-written by the computer manufacturer whereas PROM manufactured as blank memory. PROM can be programmed by PRsOM burner and by blowing internal fuses permanently.

• **EPROM (Erasable Programmable Read Only Memory):** EPROM is pronounced ee-prom. This memory type retains its contents until it exposed to intense ultraviolet light that clears its contents, making it possible to reprogram the memory.

• **EEPROM (Electrically Erasable Programmable Read Only Memory):** EEPROM can be burned (programmed) and erased by first electrical waves in a millisecond. A single byte of a data or the entire contents of device can be erased. To write or erase this memory type, you need a device called a PROM burner.

➢ **Flash Memory**

Flash memory is a non-volatile memory chip used for storage and for transferring data between a personal computer (PC) and digital devices. It has the ability to be electronically reprogrammed and erased. It is often found in USB flash drives, MP3 players, digital cameras and solid-state drives. Flash memory is a type of electronically erasable programmable read only memory (EEPROM), but may also be a standalone memory storage device such as a USB drives. EEPROM is a type of data memory device using an electronic device to erase or write digital data. Flash memory is a distinct type of EEPROM, which is programmed and erased in large blocks

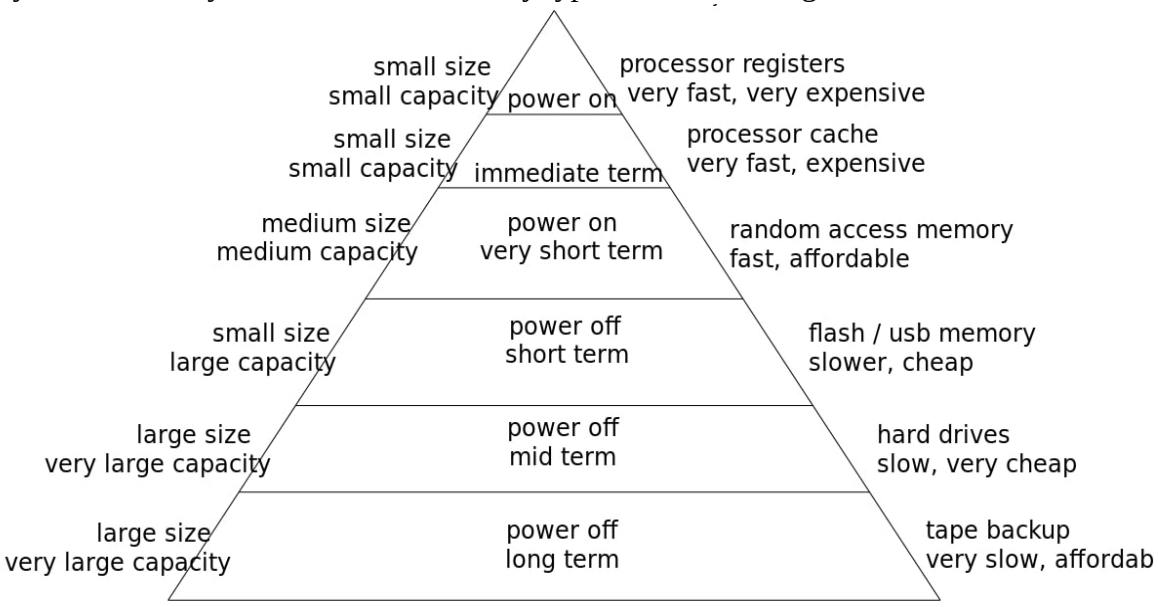
❖ MEMORY HIERARCHY

The memory is characterized on the basis of two key factors: capacity and access time.

 Capacity is the amount of information (in bits) that a memory can store.

 Access time is the time interval between the read/ write request and the availability of data.

The lesser the access time, the faster is the speed of memory. Ideally, we want the memory with fastest speed and largest capacity. However, the cost of fast memory is very high. The computer uses a hierarchy of memory that is organized in a manner to enable the fastest speed and largest capacity of memory. The hierarchy of the different memory types is shown in Figure.



The Internal Memory and External Memory are the two broad categories of memory used in the computer. The Internal Memory consists of the CPU registers, cache memory and primary memory. The internal memory is used by the CPU to perform the computing tasks. The External Memory is also called the secondary memory. The secondary memory is used to store the large amount of data and the software.

In general, referring to the computer memory usually means the internal memory.

➢ **Internal Memory**

The key features of internal memory are:

1. Limited storage capacity.
2. Temporary storage.
3. Fast access.

4. High cost.

Registers, cache memory, and primary memory constitute the internal memory. The primary memory is further of two kinds: RAM and ROM. Registers are the fastest and the most expensive among all the memory types. The registers are located inside the CPU, and are directly accessible by the CPU. The speed of registers is between 1-2 ns (nanosecond). The sum of the size of registers is about 200B. Cache memory is next in the hierarchy and is placed between the CPU and the main memory. The speed of cache is between 2-10 ns. The cache size varies between 32 KB to 4MB. Any program or data that has to be executed must be brought into RAM from the secondary memory. Primary memory is relatively slower than the cache memory. The speed of RAM is around 60ns. The RAM size varies from 512KB to 64GB.

- **Secondary Memory**

The key features of secondary memory storage devices are:

1. Very high storage capacity.
2. Permanent storage (non-volatile), unless erased by user.
3. Relatively slower access.
4. Stores data and instructions that are not currently being used by CPU but may be required later for processing.
5. Cheapest among all memory.

To get the fastest speed of memory with largest capacity and least cost, the fast memory is located close to the processor. The secondary memory, which is not as fast, is used to store information permanently, and is placed farthest from the processor. With respect to CPU, the memory is organized as follows:

- Registers are placed inside the CPU (small capacity, high cost, very high speed)
- Cache memory is placed next in the hierarchy (inside and outside the CPU)
- Primary memory is placed next in the hierarchy
- Secondary memory is the farthest from CPU (large capacity, low cost, low speed)

The speed of memories is dependent on the kind of technology used for the memory. The registers, cache memory and primary memory are semiconductor memories. They do not have any moving parts and are fast memories. The secondary memory is magnetic or optical memory has moving parts and has slow speed.

- **CPU REGISTERS**

Registers are very high-speed storage areas located inside the CPU. After CPU gets the data and instructions from the cache or RAM, the data and instructions are moved to the registers for processing. Registers are manipulated directly by the control unit of CPU during instruction execution. That is why registers are often referred to as the CPU's working memory. Since CPU uses registers for the processing of data, the number of registers in a CPU and the size of each register affect the power and speed of a CPU. The more the number of registers (ten to hundreds) and bigger the size of each register (8 bits to 64 bits), the better it is.

- **CACHE MEMORY**

Cache memory is placed in between the CPU and the RAM. Cache memory is a fast memory, faster than the RAM. When the CPU needs an instruction or data during processing, it first looks in the cache. If the information is present in the cache, it is called a cache hit, and the data or instruction is retrieved from the cache. If the information is not present in cache, then it is called a cache miss and the information is then retrieved from RAM.

Type of Cache memory

Cache memory improves the speed of the CPU, but it is expensive. Type of Cache Memory is divided into different levels that are L1, L2, L3:

Level 1 (L1) cache or Primary Cache

L1 is the primary type cache memory. The Size of the L1 cache very small comparison to others that is between 2KB to 64KB, it depends on computer processor. It is an embedded register in the computer microprocessor (CPU). The Instructions that are required by the CPU that are firstly searched in L1 Cache. Example of registers are accumulator, address register, Program counter etc.

Level 2 (L2) cache or Secondary Cache

L2 is secondary type cache memory. The Size of the L2 cache is more capacious than L1 that is between 256KB to 512KB. L2 cache is located on computer microprocessor. After searching the Instructions in L1 Cache, if not found then it searched into L2 cache by computer microprocessor. The high-speed system bus interconnecting the cache to the microprocessor.

Level 3 (L3) cache or Main Memory

The L3 cache is larger in size but also slower in speed than L1 and L2, its size is between 1MB to 8MB. In Multicore processors, each core may have separate L1 and L2, but all core share a common L3 cache. L3 cache double speed than the RAM. The advantages and disadvantages of cache memory are as follows:

Advantages

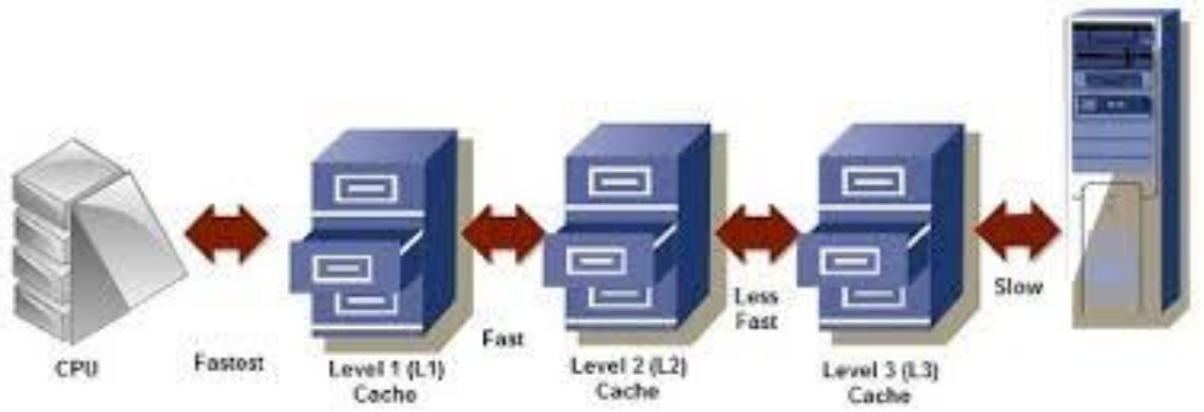
The advantages of cache memory are as follows:

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use

Disadvantages

The disadvantages of cache memory are as follows:

- Cache memory has limited capacity.
- It is very expensive



PERFORMANCE PARAMETERS OF CACHE MEMORY:

Cache Hit:

A cache hit is when a computer processor finds the data it needs inside cache memory. When a program requests data from memory, the processor will first look in the cache. If the memory location matches one of the tags in a cache entry the result is a cache hit and the data is retrieved from the cache. Cache hits improve performance by retrieving data from a smaller and faster memory source.

Cache Miss

A cache miss is when a computer processor does not find the data it needs in cache memory and must request it from the main memory. The main memory places the memory location and data in as an entry in the cache. The data is then retrieved by the processor from the cache.

Cache hit rate

A cache hit ratio is calculated by dividing the number of cache hits by the total number of cache hits and misses, and it measures how effective a cache is at fulfilling requests for content.

$$\frac{\text{Total number of cache hits (requests)}}{\text{(Total number of cache hits (requests) + Number of cache misses)}} = \text{Cache hit ratio}$$

Cache Miss rate

A miss ratio is the flip side of this where the cache misses are calculated and compared with the total number of content requests that were received.

Miss rate = (1 – hit rate)

Hit Time

The time required to access the requested information in a given level of memory.

Miss Penalty

The time required to process a miss, which includes replacing a block in upper level of memory, plus the additional time to deliver the requested data to the processor.

- **Secondary Memory**

We have read so far, that primary memory is volatile and has limited capacity. So, it is important to have another form of memory that has a larger storage capacity and from which data and programs are not lost when the computer is turned off. Such a type of memory is called secondary memory. In secondary memory, programs and data are stored. It is also called auxiliary memory. It is different from primary memory as it is not directly accessible through the CPU and is non-volatile. Secondary or external storage devices have a much larger storage capacity and the cost of secondary memory is less as compared to primary memory.

- **Use of Secondary memory**

Secondary memory is used for different purposes but the main purposes of using secondary memory are:

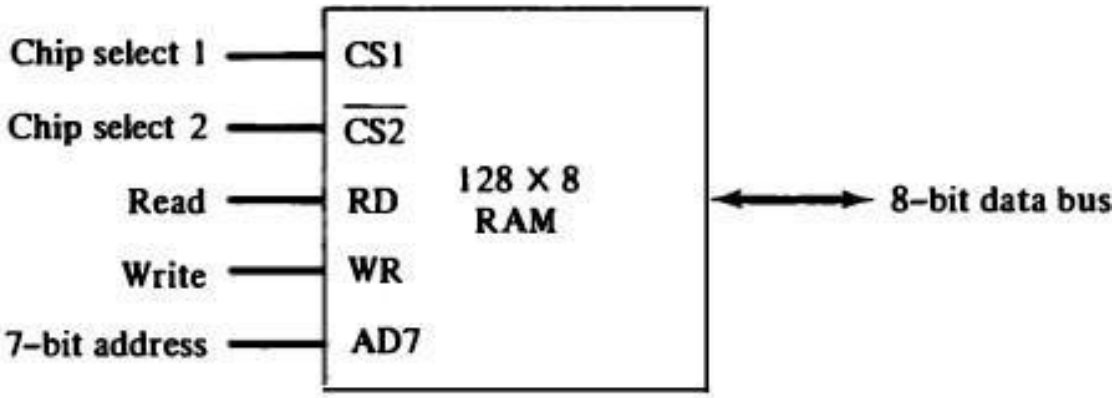
Permanent storage: As we know that primary memory stores data only when the power supply is on, it loses data when the power is off. So we need a secondary memory to stores data permanently even if the power supply is off.

Large Storage: Secondary memory provides large storage space so that we can store large data e900-like videos, images, audios, files, etc permanently.

Portable: Some secondary devices are removable. So, we can easily store or transfer data from one computer or device to another

- **RAM Chip:**

- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed.
- Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a *read* operation or from CPU to memory during a *write* operation.
- The capacity of the memory is 128 words ofeight bits (one byte) per word.
- It requires a 7-bit address bus and an 8-bit bidirectional data bus.
- The read and write inputs specifies the memory operation and the two chips select (CS) control inputsare for enabling the chip only when it is selected by the microprocessor.
- The availability of more than one control input to select the chip facilitates the decoding the addresslines when multiple chips are used in the microcomputer.
- The function table listed in figure 2(b) specifies the operation of the RAM chip.

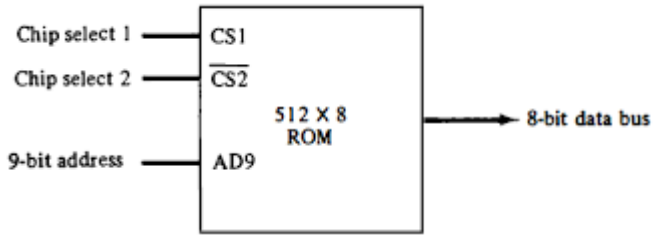


(a) Block diagram

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

ROM Chip:

- A ROM chip is organized externally in a similar manner of RAM chip.
- The block diagram of a ROM chip is shown in Figure3.
- The nine address lines in the ROM chip specify any one of the 512 bytes stored in it.
- The two chip select inputs must be CS1 = 1 and 2 = 0 for the ·unit to operate. Otherwise, the databus is in a high-impedance state.
- There is no need for a read or write control because the unit is read only.



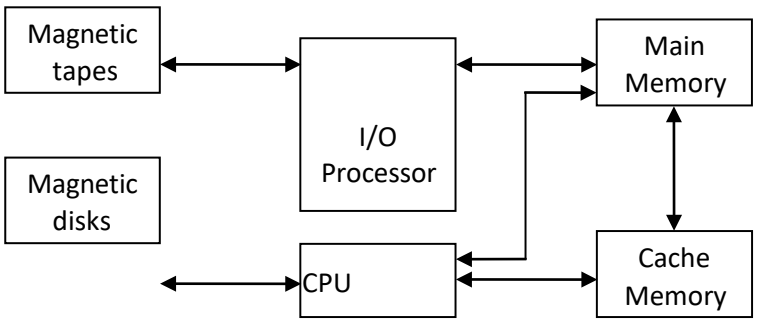
➢ Memory Address Map

The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a tutorial representation of assigned address space for each chip in the system. To demonstrate with a particular example,
Assume that a computer system needs 1024 bytes of memory (512 bytes of RAM by using 128 bytes RAM chips and 512 bytes of ROM by using a 512 ROM chips). So to solve this problem we make the sequence of solution below:

1. Number of RAM chips = total capacity of RAM / capacity for single chip
   = 512 bytes / 128 bytes
   = 4 chips.

   Number of ROM chips = total capacity of ROM / capacity for single chip
   = 512 bytes / 512 bytes
   = 1 chip

The memory hierarchy system is shown in the following figure.



(memory hierarchy system)

The memory unit that communicates directly with the CPU iscalled the *main memory*. Devices that provide backup storage are called *auxiliary memory*. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backupinformation.
Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

The part of the computer system that supervises the flow of information between auxiliary memoryand main memory is called the *memory management system.*

Auxiliary Memory

Auxiliary memory is known as the lowest-cost, highest-capacity and slowest-access storage in a computer system. Auxiliary memory provides storage for programs and data that are kept for long-term storage or when not in immediate use. The most common examples of auxiliary memories are magnetic tapes and magnetic disks.

A magnetic disk is a digital computer memory that uses a magnetization process to write, rewrite and access data. For example, hard drives, zip disks, and floppy disks.

Magnetic tape is a storage medium that allows for data archiving, collection, and backup for different kinds of data.

Associative memory

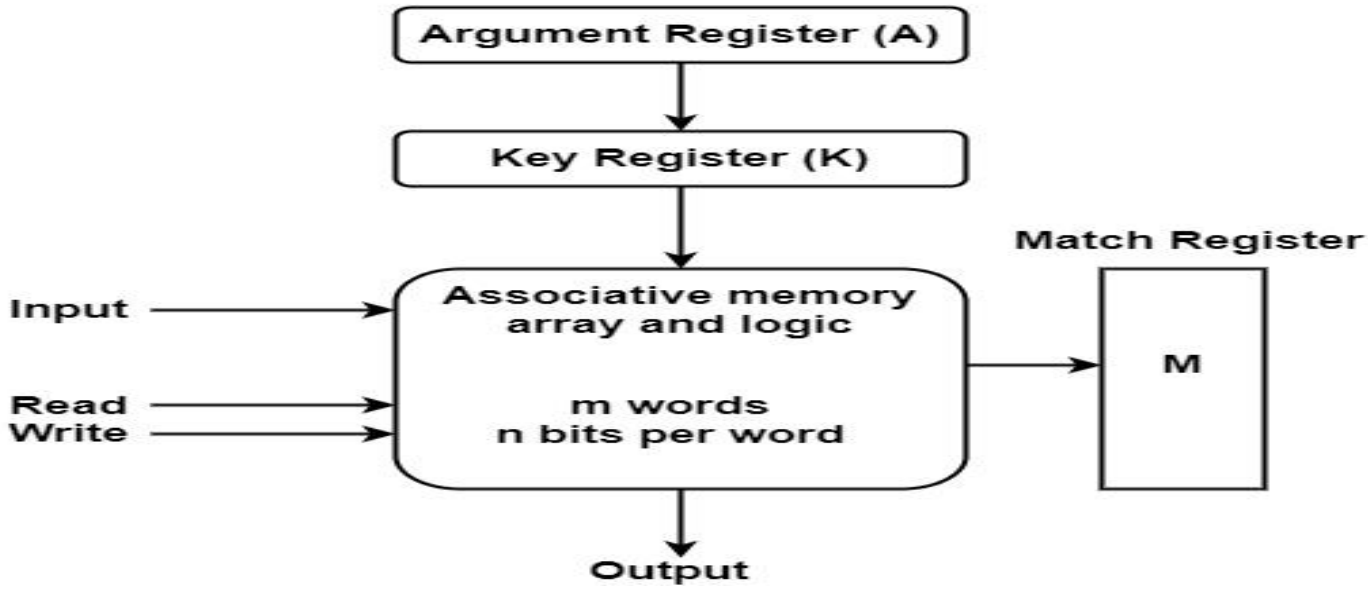
An associative memory can be treated as a memory unit whose saved information can be recognized for approach by the content of the information itself instead of by an address or memory location. Associative memory is also known as **Content Addressable Memory (CAM)**.

The block diagram of associative memory is shown in the figure. It includes a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word.

The match register M has m bits, one for each memory word. Each word in memory is related in parallel with the content of the argument register.

The words that connect the bits of the argument register set an equivalent bit in the match register. After the matching process, those bits in the match register that have been set denote the fact that their equivalent words have been connected.

Reading is proficient through sequential access to memory for those words whose equivalent bits in the match register have been set.



Cache Mapping

The process of transferring the data from main memory to cache is known as mapping process. There are various cache mapping techniques like associative mapping, direct mapping, set-associative mapping.

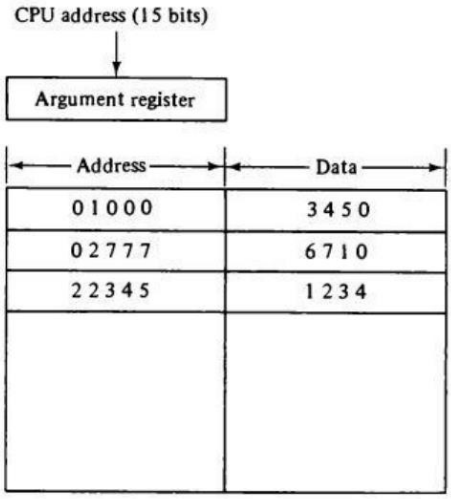
The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

Associative Mapping

The fastest and most flexible cache organization uses an associative memory. This organization is illustrated in figure. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.

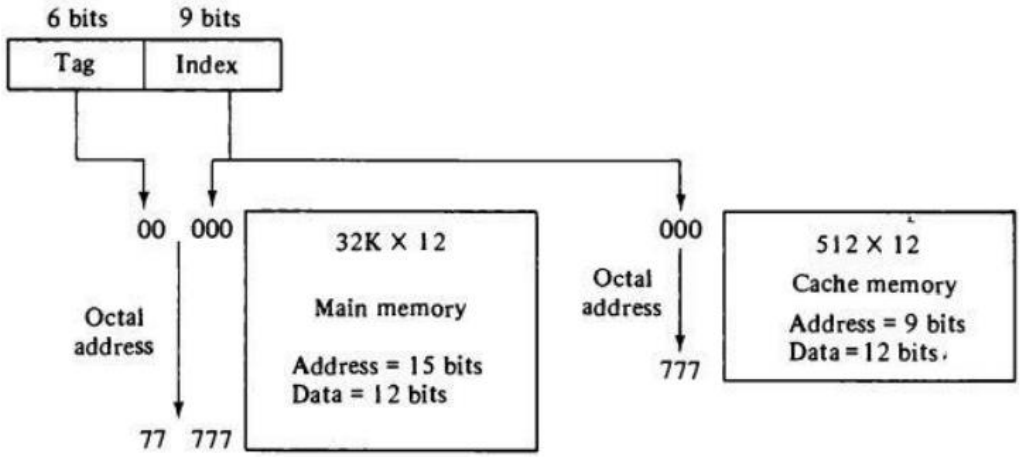
The diagram shows three words presently stored in the cache. The address value of 15-bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15-bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory.

If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache. The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache. A simple procedure is to replace cells of the cache in round robin order whenever a new word is requested from main memory. This constitutes a first in first out (FIFO) replacement policy.



Direct Mapping

The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and the remaining six bits form the tag field.



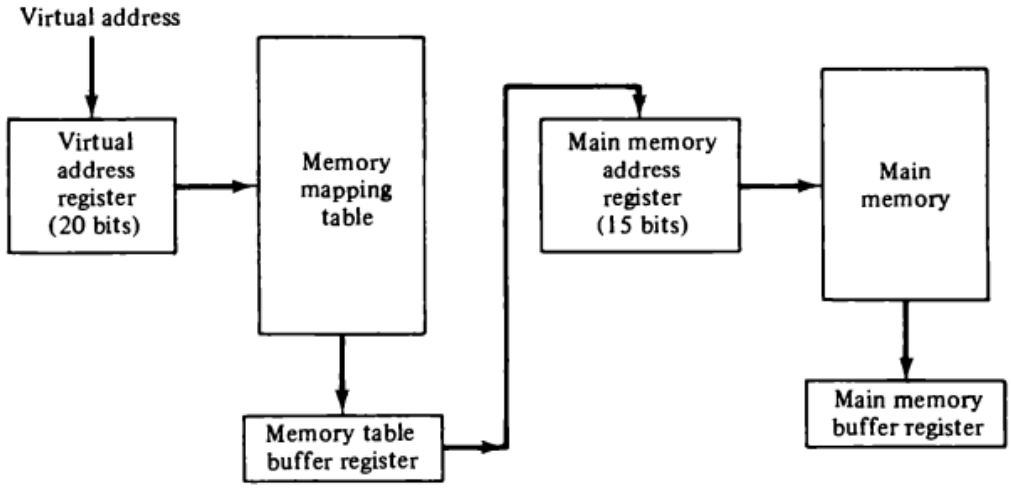
Virtual memory

A virtual memory system attempts to optimize the use of the main memory (the higher speed portion) with the hard disk (the lower speed portion). In effect, virtual memory is a technique for using the secondary storage to extend the apparent limited size of the physical memory beyond its actual physical size. It is usually the case that the available physical memory space will not be enough to host all the parts of a given active program. Virtual memory gives programmers the illusion that they have a very large memory and provides mechanism for dynamically translating program-generated addresses into correct main memory locations. The translation or mapping is handled automatically by the hardware by means of a mapping table.

Address Space and Memory Space

An address used by the programmer is a virtual address and the set of such addresses is the Address Space. An address in main memory is called a location or physical address. The set of such locations is called the Memory Space. Thus, the address space is the set of addresses generated by the programs as they reference instructions and data; the memory space consists of actual main memory locations directly addressable for processing. Generally, the address space is larger than the memory space.

Example: consider main memory: 32K words (K = 1024) = 215 and auxiliary memory 1024K words = 220. Thus, we need 15 bits to address physical memory and 20 bits for virtual memory (virtual memory can be as large as we have auxiliary storage).



(virtual memory system)

In virtual memory system, address field of an instruction code has a sufficient number of bits to specify all virtual addresses. Mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by CPU.
The figure shows that main memory needs an address that includes both the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory. The n-bit memory address is divided into two fields: k-bits for the index field and n-k bits for the tag field.

The direct mapping cache organization uses the n bit address to access the main memory and the k-bit index to access the cache. Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address