GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A+ Grade by NAAC

12-B Status from UGC

NAAC GRADE A+

Accredited University

Computer Organization Notes

Module 1 & 2

By Devarshi Shukla

(Assistant Professor)

(Electronics and Communication Department)

GLA University, Mathura

BCSC1005: COMPUTER ORGANIZATION

Objective: *This course aims to introducing the concept of computer organization. In particular, it focuses on basic hardware architectural issues that affect the nature and performance of software.*

Credits: 03 **L–T–P-J: 3–0–0-0**

Module No.	Content	Teaching Hours
I	**Basic Organization:** Basic organization of the computer and Block level description of the functional units, Number representation; 1's and 2's Complement, Integer Representation, Arithmetic Addition & Subtraction with overflow. fixed and floating-point number representation, IEEE standard floating point representation. Introduction to Combinational Circuit- half adder, full adder, binary adder/subtractor, carry look ahead adders. Multiplexer and Demultiplexer, Register, bus and memory transfer. **Central Processing Unit:** Addition and subtraction of signed numbers, Multiplication: Signed operand multiplication, Booths algorithm. Processor organization, general registers organization, stack organization, Three, Two, One & Zero address instruction. Addressing modes, Micro-operations (Arithmetic, Logical & Shift) and its applications.	20
II	**Multiprogramming and Multiprocessing:** Flynn's classification, Introduction to pipelined operation. Instruction types, formats, Instruction cycles. **Control Unit:** Execution of a complete instruction. Hardwired and micro programmed control unit. Unconditional and Conditional branching. Microinstruction with next address field, pre-fetching microinstructions, Concept of horizontal and vertical microprogramming. **Memory:** Basic concept of Memory and its hierarchy, RAM memories, 2D, 2 & 1/2D memory organization. ROM memories. Cache memories: concept and design issues, performance, address mapping and replacement. Virtual memory: concept and implementation. **Input/Output:** Peripheral devices, I/O interface, I/O ports, Interrupts: interrupt hardware, types of interrupts and exceptions. Buses, bus architecture, types of buses and bus arbitration. Modes of Data Transfer: Programmed I/O, interrupt initiated I/O and Direct Memory Access., I/O channels and processors. Standard communication interfaces.	20

Text Books:

- M. Mano, "Computer System Architecture", 3rd Edition, PHI,1996

By Dr. Devarshi Shukla
Assistant Professor Electronics and Communication

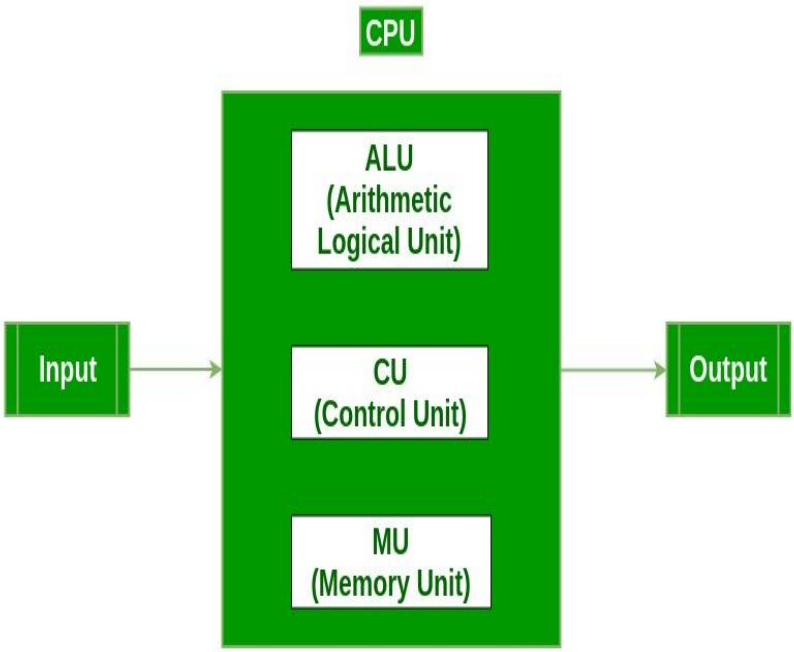
1. Basic Computer Organization

A computer is a collection of hardware and software resources that are designed to work together seamlessly in order to provide the user with a range of different functionality. The term "hardware" refers to the physical components of a computer, such as the processor, memory devices, display, keyboard, and so on. In contrast, "software" refers to the collection of programmes or instructions that are essential for the hardware resources to carry out their intended functions. Input, processing, and output are the three stages of a computer's working cycle; they are the functional components of a computer.

There are a few fundamental components that help with the computer's working cycle (the Input-Process-Output Cycle). It requires a certain input, processes that input, and then creates the output that is required. The output unit is responsible for producing the output, while the input unit is responsible for taking the input data and passing it on to the central processing unit to be processed. The data and instructions that are currently being processed are stored in the memory unit.

The term "digital computer" refers to a programmable device that reads binary data that is given as instructions, processes that binary data, and then displays the resultant digital output. As a result, computers that deal with digital data are known as digital computers.

1.1.Details of Functional Components of a Digital Computer



A. Input Unit: The input unit is made up of a variety of input devices that are connected to the computer. These devices accept information and turn it into a language that the computer can comprehend, which is called binary. The keyboard, the mouse, the joystick, the scanner, and other similar items are all examples of typical input devices.

B. Central Processing Unit (CPU)

- After the information has been fed into the computer via the input device, the information is processed by the central processing unit, also known as the CPU. The Central

Processing Unit (CPU) of a computer is sometimes referred to as the "brain" of the computer due to its role as the control centre of the computer.

- It first reads the instructions that have been retrieved from memory, and then it interprets those instructions so that it may determine what actions are required. The data is retrieved from memory or the input device, depending on whether or not it is necessary. After that, the CPU will either execute the needed calculation or do the required computation, and then it will either store the result or show it on the output device.

- The central processing unit (CPU) is comprised of three primary components, each of which is responsible for a distinct set of operations: Registers, Arithmetic Logic Unit (ALU), and the Control Unit (CU).

a. **Arithmetic and Logic Unit,** as its name indicates, is responsible for performing mathematical computations and making logical determinations. Addition, subtraction, multiplication, and division are the four operations that comprise arithmetic computations.

b. **Control Unit:** The Control unit is responsible for coordinating and controlling the flow of data into and out of the CPU. Additionally, it is responsible for controlling all of the operations of the ALU, registers, as well as the input/output units. In addition to that, it is accountable for carrying out each and every instruction that is saved within the programme. It decodes the instruction that was fetched, interprets the instruction, and then provides control signals to the input/output devices in the system until the needed operation is completed correctly by the ALU and the memory.

c. **Registers:** A register is a unit of memory that is only used temporarily by the central processing unit. These are used to store the data that is directly utilised by the processor, and their purpose is to prevent that data from being lost. Registers may be of varying sizes (16 bits, 32 bits, 64 bits, and so on), and each register inside the CPU has a particular purpose, such as storing data, storing an instruction, storing the address of a place in memory, and so on.

Keeping in Mind: The memory that is coupled to the central processing unit (CPU) is referred to as the "internal memory," and it is utilised to store both data and instructions. The internal memory is broken up into a lot of different storage sites, and each one of those storage places may hold either data or instructions. When a programme is run, the data it contains are transferred to the computer's internal memory and remain there until the programme has completed its execution. This process is known as "execution."

C. Output Unit: The output unit is made up of several output devices that are connected to the computer in some way. It takes the binary data that is produced by the CPU and transforms it into a form that can be understood by humans. Monitors, printers, plotters, and other electronic equipment fall under the category of common output devices.

2. Number representation; 1's and 2's Complement

In general, the binary number can be represented in two ways.

1. Unsigned Binary Numbers
2. Signed Binary Numbers

In digital computers, complements are used to make subtraction easier and to do logical operations. There are two kinds of complements in the Binary (base-2) number system: $1^{'s}$ complement and $2^{'s}$ complement. 1^{s} complement and 2^{s} complement are way of representing the signed binary numbers.

When using an unsigned binary number format, it is only possible to express positive binary integers. When dealing with unsigned n-bit binary integers, the magnitude of the number is represented by all n bits of the number itself.

1. *Unsigned Binary Numbers*

For instance, if we were to express the decimal value 12 as an unsigned five-bit integer, then $(12)_{10}$ would equal $(01100)_2$. In this case, each of the number's 5 bits is being used to indicate its magnitude.

Note: *Using n bits to represent unsigned binary numbers, we can show numbers from 0 to* 2^{n-1}. *In unsigned binary number format, for example, we can show numbers from 0 to 15 with 4 bits*.

2. *Signed Binary Numbers*

Signed binary numbers can be used to describe both positive and negative numbers. In a signed binary number format, the most significant bit (MSB) of the number is a sign bit. When a number is positive, the sign bit is 0, and when a number is negative, it is 1.

The signed binary numbers can be written in three different ways.

- *Signed Magnitude Form*
- *1's Complement Form*
- *2's Complement Form*

Sign Magnitude Representation

The Most Significant bit of the number is a sign bit in sign-magnitude format, and the remaining bit reflects the magnitude of the number in true binary form. For example, if a signed integer is written in an 8-bit sign-magnitude format, the MSB bit is a sign bit, and the remaining 7 bits reflect the magnitude of the number in true binary form.

Here is the representation of + 34 and -34 in a 8-bit sign-magnitude form.

+ 34 = 0 0 1 0 0 0 1 0

- 34 = 1 0 1 0 0 0 1 0

Similarly, in sign magnitude representations, there are two different representations for 0.

+ 0 = 0 0 0 0 0 0 0 0

- 0 = 1 0 0 0 0 0 0 0

Using n-bits, the range of numbers that can be represented in Sign Magnitude Representation is from **– (2^{n-1} – 1) to (2^{n-1} – 1)**.

1's Complement Representation

The 1's complement can be obtained by Simply invert the provided number to obtain a binary number's complement of 1.

For example, if we want to represent -34 in 8-bit 1's complement form, then first write the positive number (+34). And invert all 1s in that number by 0s and 0s by 1s in that number. The corresponding inverted number represents the -34 in 1's complement form. It is also called 1s complement of the number +34.

To represent -34 in 1's complement form

+ 34 = 0 0 1 0 0 0 1 0
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
- 34 = 1 1 0 1 1 1 0 1 (1's complement of + 34)

Using n-bits, the range of numbers that can be represented in 1's complement form is from – (2n-1 – 1) to (2n -1 – 1). For example, using 4-bits, it is possible to represent integers numbers from -7 to +7 in a 1's complement form representation.

Similar to sign-magnitude form, there are two different representations of 0 in 1's complement form representation.

+ 0 = 0 0 0 0 0 0 0 0

- 0 = 1 1 1 1 1 1 1 1

2's Complement Representation

There is a simple algorithm to convert a binary number into 2's complement. To get 2's complement of a binary number, simply invert the given number and add 1 to the least significant bit (LSB) of given result.

For example, Binary representation of + 7 is 0 111. Here, the most significant bit 0 represents the plus sign, and the rest of the three bits represent 7. Now, 1's complement of + 7 is 0 000, where 000 is the 1's complement of 111 and Most Significant Bit 0 represents the plus sign. Finally, we will add 1 to the LSB. There 2's complement of + 7 is 1 000 + 1 = 1 001.

3. Binary arithmetic's

- **Singed binary addition using 2's Complement**

a. ***CASE I**: If both the numbers are positive*

Example: Let's take 5-bit numbers (+7) and (+4)

Step1: Converter the given decimal number into its signed binary format

(+7):→ 0 0111

(+4):→ 0 0100

Step2: Add the both binary numbers obtained from step1.

$$\begin{array}{r} 0\ 0111 \\ +\ 0\ 0100 \\ \hline 0\ 1011 \end{array}$$

Step3: If carry bit generated at last, discard that carry bit.

b. ***CASE II:*** *If larger number is positive and smaller number is negative*

Example: Let's take 5-bit numbers (+7) and (-4)

Step1: Takes the 2's complement of negative number

(+4)→ 0 0100 then the 2's complement will be

(-4)→ 1 1100

Step 2: Convert the second positive number into its signed binary formate

(+7):→ 0 0111

Step3: Add the both binary numbers obtained from step1 and step 2.

$$\begin{array}{rr} & 0\ 0111 \\ & +\ 1\ 1100 \\ \hline \textit{Discard that carry} \leftarrow 1 & 0\ 0011 \end{array}$$

Step3: If carry bit generated at last, discard that carry bit.

c. ***CASE III:*** *If larger number is negative and smaller number is positive*

Step1: Takes the 2's complement of both the negative numbers

(+7)→ 0 0111 then the 2's complement will be

(-7)→ 1 1001

Step 2: Convert the second positive number into its signed binary format

(+4)→ 0 0100

Step3: Add the both binary numbers obtained from step1 and step 2.

$$\begin{array}{r} 1\ 1001 \\ +\ 0\ 0100 \\ \hline 1\ 1101 \end{array}$$

Step3: Takes the 2's complement of the magnitude of the result except the sign bit. Thus the final result will be $(1\ 0011)_2$

d. ***CASE IV:*** *If both the numbers are negative*

Example: Let's take 5-bit numbers (-7) and (+4)

Step1: Takes the 2's complement of both the negative numbers

(+7)→ 0 0111 then the 2's complement will be

(-7)→ 1 1001

Similarly

(+4)→ 0 0100 then the 2's complement will be

(-4)→ 1 1100

Step2: Add the both binary numbers obtained from step1. If carry generated then we discard that carry

$$\begin{array}{r} 1\ 1001 \\ +\ 1\ 1100 \\ \hline \textit{discard that carry bit} \leftarrow 1\ \ 1\ 0101 \end{array}$$

Step4: Takes the 2's complement of the magnitude of the result except the sign bit. Thus, the final result will be $(1\ 1011)_2$

Note: we take 2's complement of the result only in the case 3 and case 4.

- **Signed binary Subtraction using 2's complement**

The step carried out for subtraction are as follows

Step1: Find the 2's complement of subtrahend. This will change the subtrahend to its equivalent value of opposite sign. If the minuend is negative then also find its 2's complement

Step2: Add the value of minuend and complemented subtrahend.

Step 3: The above sum produced carry then discard it:

(i) *If the sign bit is 0 then the result obtained is positive, and the remaining bits shows the magnitude of the result.*

(ii) *If the sign bit is 1, the then above result is negative, and the result obtained is in its 2's complement value. Find the 2's complement of the magnitude of the result.*

Examples: Let's take 5-bit binary number.

(a). Subtract (+7) – (+4)

$$\begin{array}{r} 0\ 0111 \\ +\ 1\ 1100 \\ \hline \textit{discard that carry bit} \leftarrow 1\ 0\ 0011 \end{array}$$

(b). Subtract (+7) – (-4) this will become addition of (+7) and (+4)

$$\begin{array}{r} 0\ 0111 \\ +\ 0\ 0100 \\ \hline 0\ 1011 \end{array}$$

(c). Subtract (-7) - (+4) this will become (-7) and (-4)

$$\begin{array}{rr} & 1\ 1001 \\ + & 1\ 1100 \\ \hline \textit{discard that carry bit} \leftarrow 1 & 1\ 0101 \end{array}$$

Takes the 2's complement of the magnitude of the result except the sign bit. Thus, the final result will be $(1\ 1011)_2$

4. Addition and subtraction with Overflow

In signed binary 2's complement number format, the lower and upper limit is **-2^{n-1} to $2^{n-1}-1$.** If the addition or subtraction of the result fall beyond the above range then the condition of overflow will be occurring. Overflow is a phenomenon that arises in computer organization when the value that is produced as a result of performing an operation on valid representations of numbers goes outside of the given range limit.

let's take an example,

If you are representing numbers using 4 bits as unsigned binary, the minimum value is 0 (i.e., (0000), while the maximum value is 15 (1111). Any operation that results in a number larger than 15 or smaller than 0 has overflowed.

Overflow Detection –

- Overflow occurs when:

  1. Two negative numbers are added and an answer comes positive or
  2. Two positive numbers are added and an answer comes as negative.

  So, overflow can be detected by checking Most Significant Bit (MSB) of two operands and answer.

- Overflow can also be detected using 2 Bit Comparator just by checking Carry-in(C-in) and Carry-Out(C-out) from MSB's. Consider the N-Bit Addition of 2's Complement number.

Figure of overflow condition

So, overflow Occurs when C-in ≠ C-out.

Examples:

(i). Lets represent (-17) and (-19) in six bits and add them then

Take the 2's complement of (-17) = (101111) and (-19) = (101101)

```
    1111
  1 01111
+ 1 01101
-----------
  1 0 11100
```

If we look carefully then, C-in on MSB is 0 but C-out is 1 that means C-in ≠ C-out. Therefore, overflow will occur.

5.Floating point representation

5.1. Floating point number represented as

Sign(S)	Exponent (E)	Mantissa (M)

Where,

Sign: Its value shows the number is positive or negative, if the number is positive or negative then S=0 or S=1 respectively.

Mantissa: mantissa is the signed normalization. The signed normalization of is of two types: Explicit and Implicit normalization.

Exponent: It is stored in Biased form, where ***biased form means unsigned number***. The formula for biasing is **E=e+bias**, that means if we take exponent of K bits then value of bias will be 2^{K-1} and the value of small "e" will be getting from normalization.

What do you mean by Normalization of mantissa (M)?

Let's take an example 19.25 if we normalize this number then it looks like (0.1925×10^2). Similarly, if we convert above decimal number into binary (10011.01) and then we normalize this number then it looks like ($0.1001101 * 2^5$). Therefore, the value of mantissa (M) will be (M= 1001101).

As we know the normalized mantissa are two types

Explicit form	Implicit from
✓ Explicit normalization said that move the decimal point to the left and there should always be 1 after the decimal point. ✓ Let's take (10011.01) then explicit normalization will be ($0.1001101 * 2^5$) ✓ Then the value of Mantissa in explicit normalized form will be (M=1001101)	✓ Implicit normalization said that move the decimal point to the left and there should always be 1 before the decimal point. ✓ Let's take (10011.01) then implicit normalization will be ($1.001101 * 2^4$) ✓ Then the value of Mantissa in implicit normalized form will be (M=001101)

What do you mean by Biased exponent (E)?

The formula to find the biased Exponent is **(E = e + Bias).**

Where, **E** is stored or biased exponent, small "**e**" is original exponent and its value is obtained from the normalization. **Bias** value is obtained from the number of bits assigned to represent the exponent. Therefore, the **Bias** = 2^{K-1} where, K is the number of bits used to represent exponent.

Let's take the same example 19.25 = 10011.01, after the normalization it will be $0.1001101 * 2^5$. Then the value of **e=5 and** let's assume 8 bit is used to represent exponent then

bias = $2^{8-1}=2^7=64$**.** Therefore, **biased exponent (E)** = 5+64= 69**.** Thus, convert this biased exponent value in binary then E = 01000101

Steps to represent floating point number:

Step 1: If the number is given in decimal form, then convert into its binary form.

Step 2: Normalize the binary number obtained from step 1. This will give the value of Mantissa. If normalization time is not mentioned then we always normalized into its explicit form.

Step3: After the normalization, find the biased exponent (E= e+Bias). And convert the biased exponent value into binary form.

Step 4: Find the sign value. And placed all the value in the floating representation format.

Example 1: Consider a 16-bit register used to store floating point number. The mantissa is normalized signed number. And the 6 bit is used to represent biased exponent. What is the16-bit value for +19.25 in the register.

Solution:
Total register bits is 16

S=1bit	E= 6 bits	M= 9 bits

Step 1: Convert +19.25 into binary

$(+19.25)_{10} = (10011.01)_2$

Step 2: Normalized the converted binary number obtained from step 1.

Explicit normalized value will be $(0.1001101 * 2^5)$.

Therefore, Mantissa (M =1001101) but mantissa is off 9 bits thus we have to add 2 00 to the right-hand side of the binary therefore, (M = 100110100)

Step 3: From the normalization the value of original exponent **(e) is 5** because 2 to the power is **5** and further, the **Bias** value will be $2^{6-1}=31$, because **6** bits are used to represent biased exponent. So, according to the biased exponent **(E) = e + Bias**. Put all the values we get E = 5+31= 36. Convert 36 into binary 100100 but biased exponent represents with 6 bits. Therefore, (E = 100100)

Step4: Sign bit value is 0 because number is positive.

Sign	Exponent	Mantissa
0	100100	100110100

5.2 IEEE Standard 754 Floating Point Numbers.

IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.

1. Single Precision (IEEE 32-bit Floating-Point Representation)

In single precision, sign is represented by 1 bit, exponent represented by 8 bits and mantissa represented by 23 bits

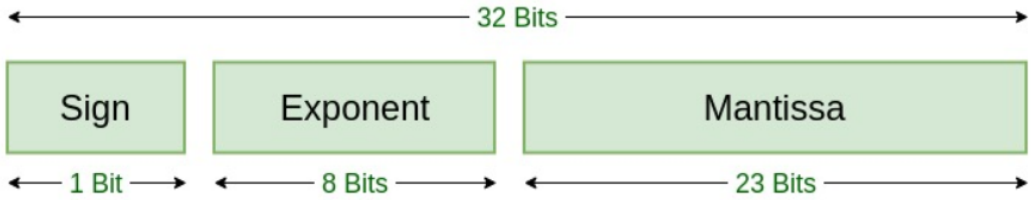


Figure 2: *Shows the single precision or IEEE 32-bit floating point representation.*

Example 1. Write +85.125 in single precision format

Step 1: Convert the magnitude of the given decimal number into its binary form +85.125 is

85 = 1010101
0.125 = 001
85.125 = 1010101.001

Step 2: Do the Implicit Normalization of the binary number obtained from step 1.
=1.010101001 x 2^6 Thus the value of Mantissa will be M = 010101001. But in single precision the value of M is 23 bits thus we add zeros (0) to the right-hand side of the obtained mantissa therefore the final value of mantissa will be (M=01010100100000000000000).

Step 3: Bias the exponent

To get the biased exponent we simply do (E= e+127)
In above step after the normalized the value of power of 2 is the value of small "e" i.e., e = 6. Thus, the biased exponent (E) = 6+127 =133. Convert this decimal value into its binary form that is (E=10000101).

Step 4: If the sign in positive or negative then put sign bit 0 or 1 respectively.

sign = 0

Step 5: write all the values of sign, exponent and mantissa in single precision format

Sign	Exponent	Mantissa
0	10000101	01010100100000000000000

2. Double Precision (IEEE 64-bit Floating-Point Representation)

In double precision, sign is represented by 1 bit, exponent represented by 11 bits and mantissa represented by 52 bits

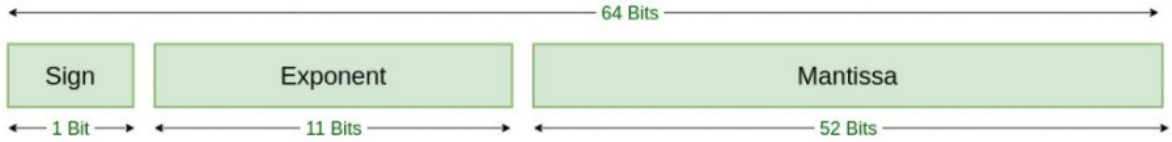


Figure 3: *Shows the double precision or IEEE 64-bit floating point representation.*

Example 1. Write +85.125 in double precision format

Step 1: Convert the magnitude of the given decimal number into its binary form
+85.125
85 = 1010101
0.125 = 001
85.125 = 1010101.001

Step 2: Do the Implicit Normalization of the binary number obtained from step 1.
=1.010101001 x 2^6 Thus the value of Mantissa will be M = 010101001. But in single precision the value of M is 52 bits thus we add zeros (0) to the right-hand side of the obtained mantissa therefore the final value of mantissa will be:
(M= 0101010010000000000000000000000000000000000000000000).

Step 3: Bias the exponent
To get the biased exponent we simply do (E= e+127)
In above step after the normalized the value of power of 2 is the value of small "e" i.e., e = 6. Thus, the biased exponent (E) = 6+1023 =1029. Convert this decimal value into its binary form that is (E= 10000000101).

Step 4: If the sign in positive or negative then put sign bit 0 or 1 respectively.
sign = 0

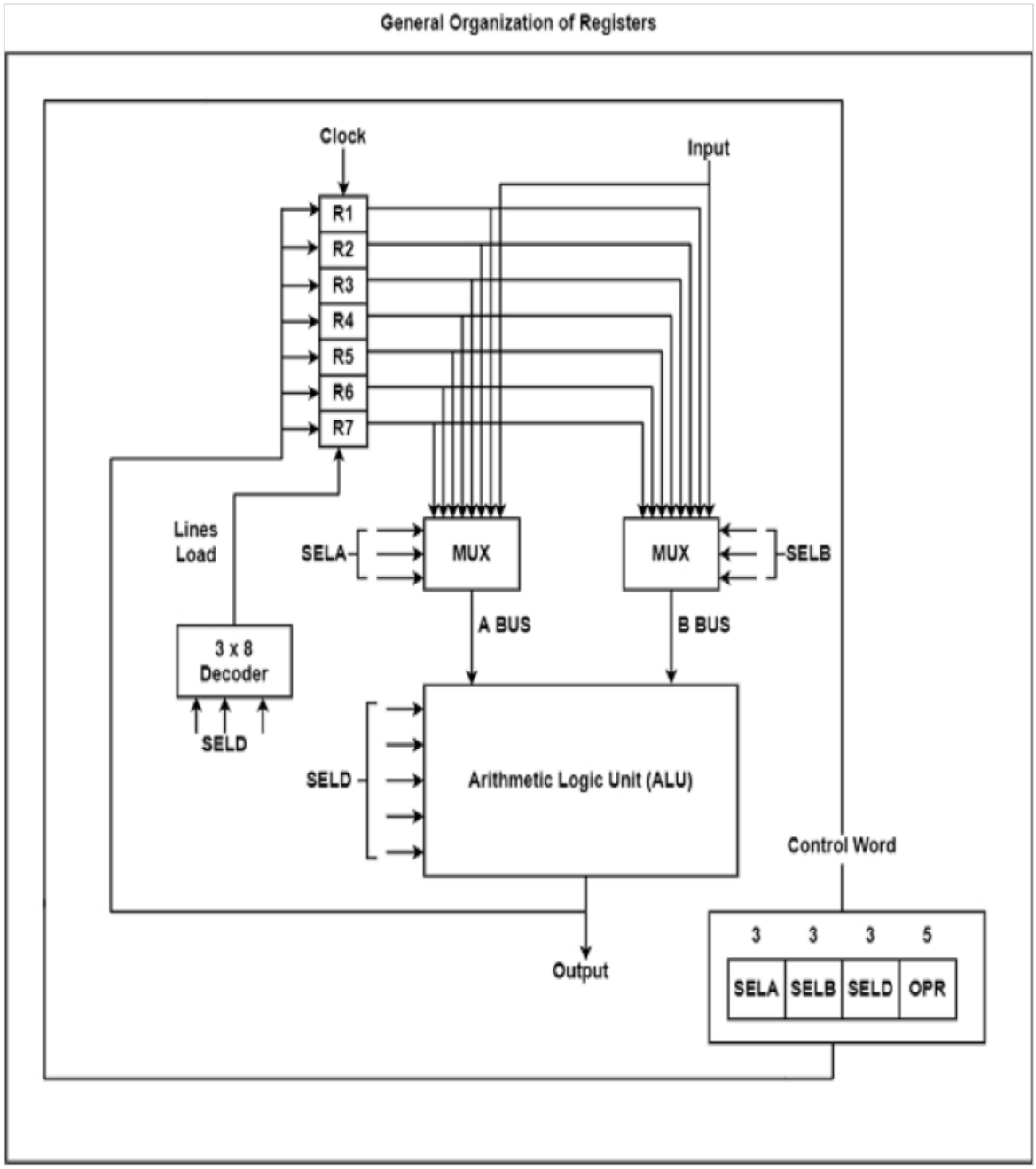
Step 5: write all the values of sign, exponent and mantissa in double precision format

Sign	Exponent	Mantissa
0	10000000101	0101010010000000000000000000000000000000000000000000

Note: *In IEEE 754 floating point representation we always do Implicit normalization. And bias value is always 127 and 1023 for single precision and double precision respectively.*

6. What is General Register Organization?

- A register is made up of flip-flops. A register is a special type of high-speed storage region within the CPU.
- Combinational circuits are used to implement data processing. Before processing, the information is always defined in a register.
- The registers help to accelerate program execution.
- Registers perform two essential tasks in CPU operation, which are as follows:
  - **(i)** It can serve as a temporary data storage location. This enables quick data access for directly implementing programs, if required.
  - **(ii)** It can save the CPU's status and information about the program's implementation.



A general organization of seven CPU registers is displayed in the figure.

Example: Consider R1 ← R2 + R3, the following are the functions implemented within the CPU-

MUX A Selector (SELA) − It can place R2 into bus A.

MUX B Selector (SELB) − It can place R3 into bus B.

ALU Operation Selector (OPR) − It can select the arithmetic addition (ADD).

Decoder Destination Selector (SELD) − It can transfer the result into R1.

Control word: The state of 14 binary selection inputs determines the control word. The 14-bit control word defines a micro-operation.

The encoding of register selection fields is specified in the table.

Encoding of Register Selection Field

Binary Code	SELA	SELB	SELD
000	Input	Input	None
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

Similarly, there are several micro-operations are implemented by the ALU. Few of the operations implemented by the ALU are displayed in the table.

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	ADD A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

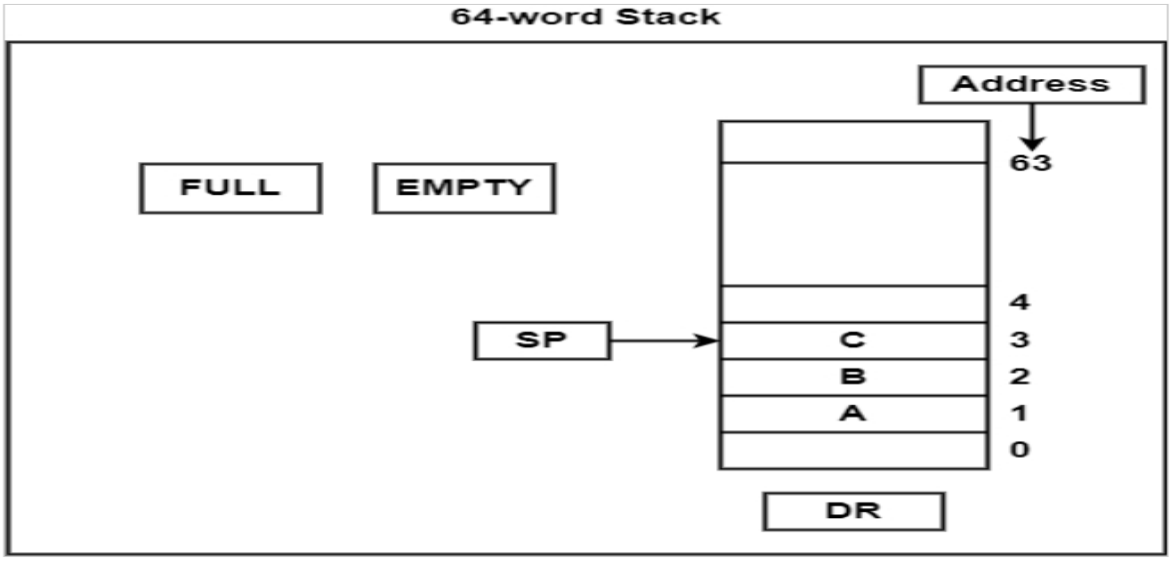
Therefore, Control world register will be look like:

Micro-operation	SELA	SELB	SELD	OPR	Control Word			
$R_1 \leftarrow R_2 + R_3$	R_2	R_3	R_1	ADD	010	011	001	00010

7.What is General Register stack Organization

The stack is also known as the LIFO (Last In First Out) list. It is the most essential component of the CPU. It stores data in such a way that the last element saved is retrieved first. Stacks are memory units that contain an address register and it is referred to as the Stack Pointer (SP). The stack pointer pointing the top address of the stack memory continuously.

Stack has two operations; insertion of data and deletion of data from the stack. The insertion operation is known as **push operation** and the deletion operation is known as **pop operation**. In a computer stack, when some data is inserted or push then stack pointer incremented. whereas when pop operations are performed then Stack pointer is decremented.



- Consider a 64-word register stack arranged as displayed in the figure. Then the stack pointer includes 6 bits, because $2^6 = 64$, and the SP cannot exceed 63 (111111 in binary). If it exceeded from 63 then it again reset to 0 that is 0(111111 + 1 = 1000000). SP holds only the six least significant bits.
- When the stack is full, the one-bit register 'FULL' is set to 1. If the stack is null, then the one-bit register 'EMTY' is set to 1.
- The data register (DR) holds the binary information which is composed into or readout of the stack.

The push operation is executed as follows –

SP←SP + 1	It can increment stack pointer
K[SP] ← DR	It can write element on top of the stack
If (SP = 0) then (FULL ← 1)	Check if stack is full
EMTY ← 0	Mark the stack not empty

- The stack pointer is incremented by 1 and the address of the next higher word is saved in the SP. The word from DR is inserted into the stack using the memory write operation.
- The first element is saved at address 1 and the final element is saved at address 0. If the stack pointer is at 0, then the stack is full and 'FULL' is set to 1.
- This is the condition when the SP was in location 63 and after incrementing SP, the final element is saved at address 0. During an element is saved at address 0, there are no more empty registers in the stack. The stack is full and set to 1 and the 'EMTY' is set to 0.

The pop operation includes the following sequence of micro-operations –

DR←K[SP]	It can read an element from the top of the stack
SP ← SP – 1	It can decrement the stack pointer
If (SP = 0) then (EMTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full

- The top element from the stack is read and transfer to DR and thus the stack pointer is decremented.
- If the stack pointer reaches SP address 0, then the stack is empty and 'EMTY' is set to 1. This is the condition when the element in location 1 is read out and the SP is decremented by 1.

6.1 Notations

1. **Infix notation :** A+B; In this notation operator (+) will be in between operand (A,B).
2. **Prefix notation:** +AB; In this notation operator (+) will written before the operand (A,B).
3. **Postfix notation:** AB+; In this notation operator (+) will written after the operand (A,B).

Note: *Prefix notation is also called as Polish Notation and Postfix Notation is also called as Reverse Polish Notation (RPN).*

6.1.1 Reverse Polish Notation (RPN).

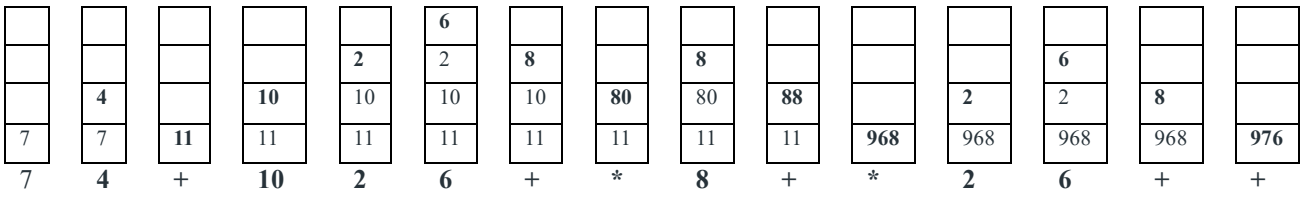
A stack effectively implements the arithmetic expression in digital computers.

Question Convert the following arithmetic expression from infix to Reverse Polish notation (RPN) and perform the stack operation to evaluate the expression. The given expression is **(7+4)*(10*(2+6)+8)+(2+6).**

Solution:

Infix→ **(7+4)*(10*(2+6)+8)+(2+6)**

RPN:→ **7 4 + 10 2 6 + * 8 +* 2 6 + +**



7. Register Transfer Language, micro-operations and its application

Registers:

- Register is the storage element in digital computer. It stores the data which going to be executed during execution cycle and also it stores the result after the execution.

- Computer registers are designated by upper case letters to denote the function of the register. For example, **R_1**, **R_2**.
- The register that holds an address for the memory unit is usually called a memory address register and is designated by the name **MAR**.

Registers Transfer:

- Information transfer from one register to another is designated in symbolic form is called as register transfer.

- **For example:** The statement $R_2 \leftarrow R_1$ denotes a transfer of the content of register R_1 into register R_2.

- Register transfer is the ability of hardware logic circuits to perform a given micro-operation and move the result of that operation to the same register or to another register.

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

Note: *A comma is used to separate two or more operations that are executed at the same time.*

Registers Transfer Language:

- Register transfer language is a symbolic notation for describing micro-operation transfers between registers.

- Programmers introduced the term "language" to designate programming languages. This programming language serves as a means of representing a computational procedure using symbolic representations.

Micro-operations:

- Micro-operations are operations performed on data stored in registers. A micro-operation is a simple operation that is carried out on data contained in one or more registers. This operation may be arithmetic, logical & shift operation.

Types of Micro-operations

The following are the different types of micro-operations:

1. Micro-operations that move binary data from one register to another are known as register transfers.
2. In registers, arithmetic micro-operations operate on numeric data stored.

3. Bit manipulation operations on non-numeric data are performed by logic micro-operations.
4. Shift micro-operations are data-based shift micro-operations.

A. Arithmetic

1. ***Addition micro-operation***–

$R_3 \leftarrow R_1 + R_2$	Contents of R_1 plus R_2 transferred to R_3
$R_3 \leftarrow R_1 - R_2$ $R_3 \leftarrow R_1 + (\overline{R}_2 + 1)$	Contents of R_1 minus R_2 transferred to R_3. Or we take the 2's complement of the value of the register R_2 and add with the register value of R_1.
$R_3 \leftarrow \overline{R}_2$	1's complements the contents of R_2 and transfer to R_3
$R_3 \leftarrow \overline{R}_2 + 1$	2's complements the contents of R_2 and transfer to R_3
$R_3 \leftarrow R_1 + 1$	Increment the content of R_1 by 1 and transfer to R_3
$R_3 \leftarrow R_1 - 1$	Decrement the content of R_1 by 1 and transfer to R_3

Application of Arithmetic operation

- Addition and Subtraction: Used in programming for counting, indexing, or calculating addresses.

2. ***Logic micro-operation***–

$R_1 \leftarrow 0$	Clear
$R_1 \leftarrow R_2 \wedge R_3$	AND
$R_1 \leftarrow R_2 \wedge \overline{R_3}$	R_2 AND with 1's complemented R_3
$R_1 \leftarrow \overline{R_2} \vee R_3$	1's complemented R_2 AND R_3
$R_1 \leftarrow \overline{R_2 \wedge R_3}$	NAND
$R_1 \leftarrow R_2 \vee R_3$	OR
$R_1 \leftarrow R_2 \vee \overline{R_3}$	R_2 OR with 1's complemented R_3
$R_1 \leftarrow \overline{R_2} \vee R_3$	1's complemented R_2 OR R_3
$R_1 \leftarrow \overline{R_2 \vee R_3}$	NOR
$R_1 \leftarrow R_2 \oplus R_3$	Ex-OR
$R_1 \leftarrow \overline{R_2 \oplus R_3}$	Ex-NOR
$R_1 \leftarrow all\ 1s$	Set all to 1

Application of Logic operation

- Logic operations are useful to manipulate individual bits or apportion of a world stored in a register.
- They can be used to change a bit value, delete a group of bits or insert a new bit value in a register.

3. Shift Operations

R←shl R	Shift-left register R
R←shr R	Shift-right register R
R←cil R	Circular shift left registers R
R←cir R	Circular shift right registers R
R←ashl R	Arithmetic Shift-left register R
R←ashr R	Arithmetic Shift-right register R

Difference between Shift left/right and arithmetic shift left/right

Logical Shift left/right	**Arithmetic shift left/right**
• During a logical shift right, the leftmost bit (the most significant bit or MSB) is always filled with a zero. Similarly, • During a logical shift left, the right most bit (the Least significant bit or LSB) is always filled with a zero.	• During an arithmetic shift right, the MSB (leftmost bit) is duplicated (i.e., filled with the same value as the original MSB). Similarly, During an arithmetic shift left, the LSB is duplicated (i.e., filled with the same value as the original LSB).

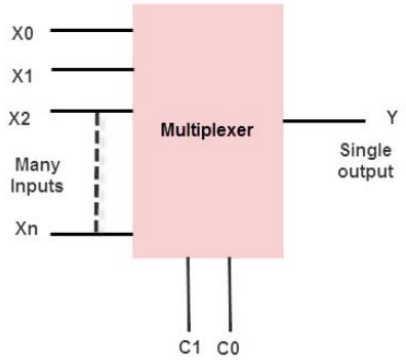
Application of shift operation

- Arithmetic shifts are used for signed integer values to maintain the integrity of the sign bit while shifting.
- Logical shifts are typically used for unsigned integer values to divide them by powers of 2.

8. Multiplexer, Demultiplexer, and overview of decoders

Multiplexer

- The multiplexer is a combinational circuit that has multiple inputs and single line output.
- The select lines determine which input is connected to the output. it is also called a data selector.
- In a multiplexer, there are 2^n input lines and 1 output line, where n is the number of select lines.
- Therefore, a multiplexer is a combinational circuit which is designed to switch one of the many input lines to a single output line by the use of a control signal.

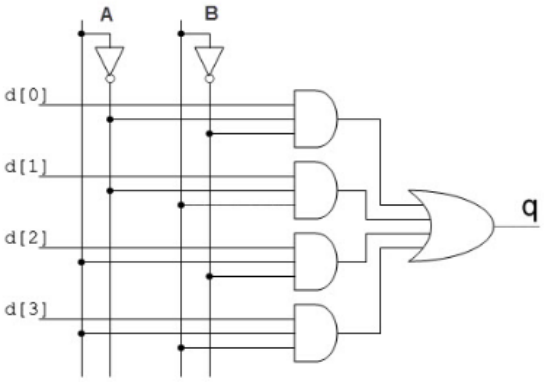


4-to-1 Multiplexer

The 4X1 multiplexer comprises 4-input bits, 1- output bit, and 2- control bits. The four input bits are namely D0, D1, D2, and D3, respectively; only one of the input bits is transmitted to the output. The output 'q' depends on the value of control input AB. The control bit AB decides which of the input data bit should transmit the output.

The following figure shows the 4X1 multiplexer circuit diagram using AND gates. For example,

- when the control bits AB =00, then the higher AND gates are allowed while remaining AND gates are restricted. Thus, data input D0 is transmitted to the output 'q"



4X1 Mux

- If the control input is changed to 01, then all gates are restricted except the second AND gate from the top. In this case, D1 is transmitted to the output, and q=D1.
- Similarly, If the control input is changed to AB =10, all gates are disabled except the third AND gate. In this case, D2 is transmitted to the output, and q = D2.
- Last, If the control input is changed to 11, then all gates are restricted except the bottom AND gate. In this case, D3 is transmitted to the output, and q=D3.

Applications of Multiplexers

Multiplexers are used in various applications wherein multiple-data need to be transmitted by using a single line.

1. *Communication System*

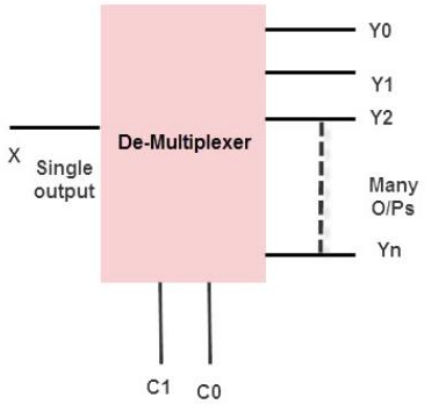
A communication system has both a communication network and a transmission system. By using a multiplexer, the efficiency of the communication system can be increased by allowing the transmission of data, such as audio and video data from different channels through single lines or cables.

2. Computer Memory

Multiplexers are used in computer memory to maintain a huge amount of memory in the computers, and also to reduce the number of Buses which are required to connect the memory to other parts of the computer.

Demultiplexer

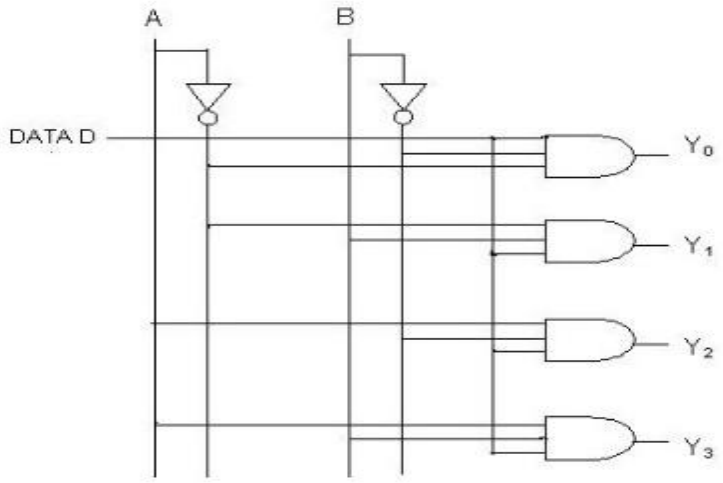
De-multiplexer is also a combinational circuit with one input and multiple output lines. It is used to send a signal to one of the many devices. The main difference between a multiplexer and a de-multiplexer is that a multiplexer takes two or more signals and encodes them on a wire, whereas a de-multiplexer does reverse to what the multiplexer does.



Demultiplexer

1to4 Demultiplexer

The 1-to-4 demultiplexer comprises 1- input bit, 4-output bits, and control bits. The 1X4 demultiplexer circuit diagram is shown below.



1X4 Demux

The input bit is considered as Data D. This data bit is transmitted to the output lines, which depends on the AB value and the control input.

- When the control input AB = 00, the upper first AND gate is permitted while the remaining AND gates are restricted. Thus, only data bit D is transmitted to the output, and Y0 = Data. If the data bit D is low, the output Y0 is low. IF data bit D is high, the output Y0 is high. The value of the output Y0 depends upon the value of data bit D, the remaining outputs are in a low state.

- If the control input changes to AB = 01, then all the gates are restricted except the second AND gate from the top. Then, data bit D is transmitted only to the output Y2; and, Y2 = Data.

- If the control input changes to AB = 10, then all the gates are restricted except the third AND gate from the top. Then, data bit D is transmitted only to the output Y2; and, Y2 = Data.

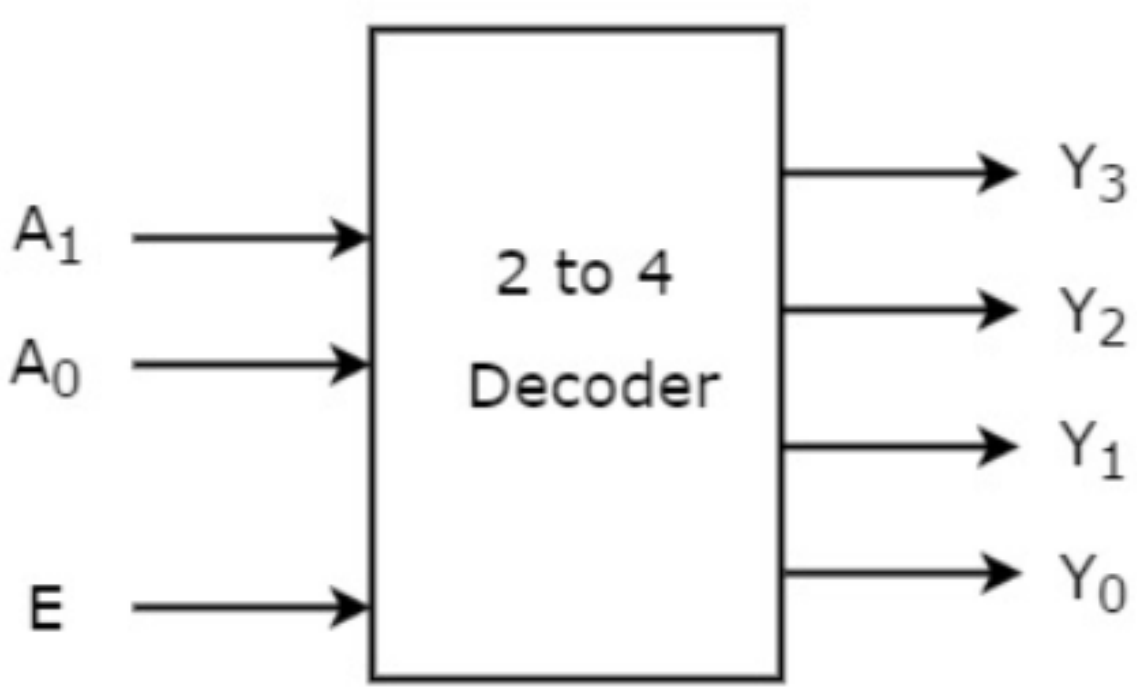
- If the control input changes to AB = 11, then all the gates are restricted except the bottom AND gate from the top. Then, data bit D is transmitted only to the output Y3; and, Y3 = Data.

Overview of Decoders

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The block diagram of 2 to 4 decoder is shown in the following figure.



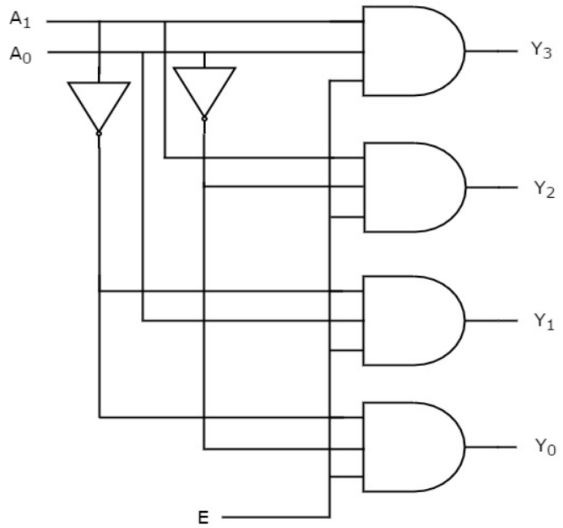
The Truth table of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

$Y3=E.A_1.A_0$
$Y2=E.A_1.A_0'$
$Y1=E.A_1'.A_0$
$Y0=E.A_1'.A_0'$

Therefore, form the above expression the decoder logic diagram will be



Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

Example

Let's as design 2 to 8 decoder using 2 to 4 decoder. Thus, the formula for designing higher order decoders using lower order decoders are as follows

$$Required\ number\ of\ \ lower\ order\ decoders = \frac{m_2}{m_1}$$

Where,

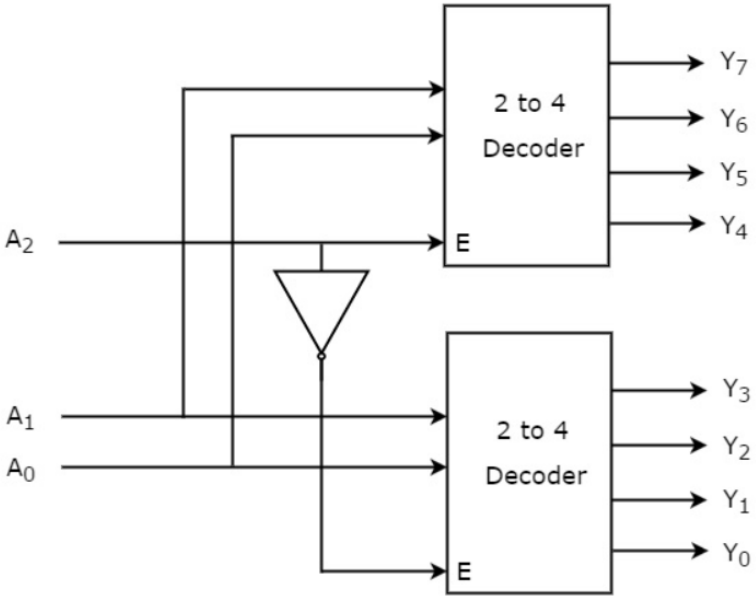
m_1 is the number of outputs of lower order decoder.

m_2 is the number of outputs of higher order decoder.

Then, m_1 is 4 and m_2 is 8

$$Required\ number\ of\ \ lower\ order\ decoders = \frac{8}{4} = 2$$

Hence, two 2 to 4 decoder is required to design 3 to 8 decoder and the diagram is:



E	Input			Output							
	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
Bottom decoder enables	0	0	0	1	0	0	0	0	0	0	0
	0	0	1	0	1	0	0	0	0	0	0
	0	1	0	0	0	1	0	0	0	0	0
	0	1	1	0	0	0	1	0	0	0	0
Top decoder enables	1	0	0	0	0	0	0	1	0	0	0
	1	0	1	0	0	0	0	0	1	0	0
	1	1	0	0	0	0	0	0	0	1	0
	1	1	1	0	0	0	0	0	0	0	1

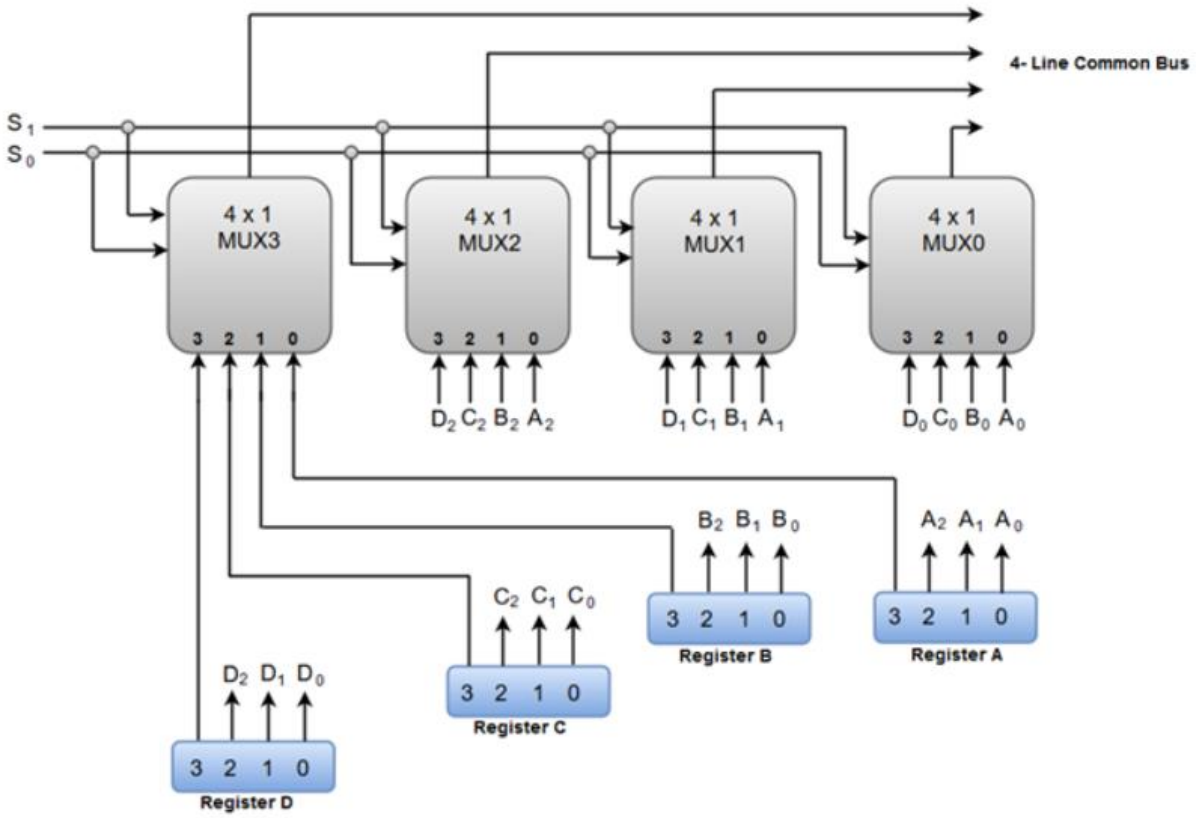
9. Bus and Memory Transfer

BUS:

- A digital system composed of many registers, and paths must be provided to transfer information from one register to another.
- A bus structure, is more efficient for transferring information between registers in a multi-register configuration system.

Bus design using multiplexer

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S_1 and S_2).



- The two selection lines S_1 and S_2 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 both select lines are low (S_1S_0 = 00), the 0 data inputs of all four multiplexers are chosen and sent to the bus. This causes the contents of register A to be sent to the bus lines, because the all the outputs of register A are linked to the 0 data inputs of the all the multiplexers.

- Similarly, when selected S_1S_0 = 01,10 and 11, register B, register C and register D are selected respectively, and the bus lines received the data provided by selected register.

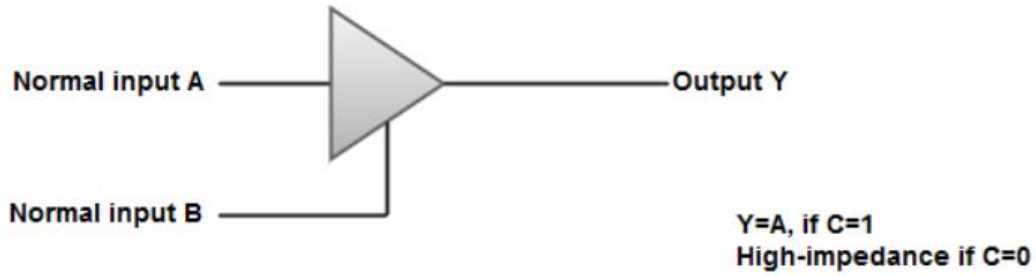
Note: *The number of multiplexers needed to construct the bus is equal to the number of bits in each register. The size of each multiplexer must be 'k * 1' where k is number of registers. For instance, a common bus for 32 registers of 8 bits each requires 8 multiplexers, one for each line in the bus. Each multiplexer must have 32 data input lines and 5 selection.*

Bus design using three state gates with decoder circuit

A bus system can also be constructed using three-state gates instead of multiplexers.

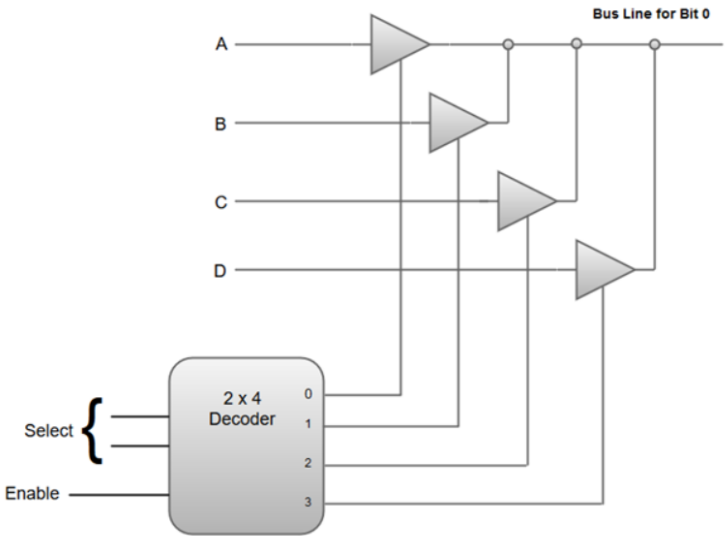
The three state gates can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state.

The graphical symbol of a three-state buffer gate can be represented as:



The following diagram demonstrates the construction of a bus system with three-state buffers.

Bus line with three state buffer:



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

Memory Transfer

Most of the standard notations used for specifying operations on memory transfer are stated below.

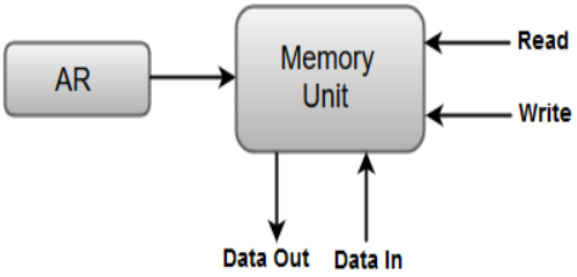
- The transfer of information from a memory unit to the user end is called a **Read operation**.
- The transfer of new information to be stored in the memory is called a **Write operation**.
- A memory word is designated by the letter M.
- We must specify the address of memory word while writing the memory transfer operations.
- The address register is designated by AR and the data register by DR.
- Thus, a read operation can be stated as:

Read: DR ← M [AR]

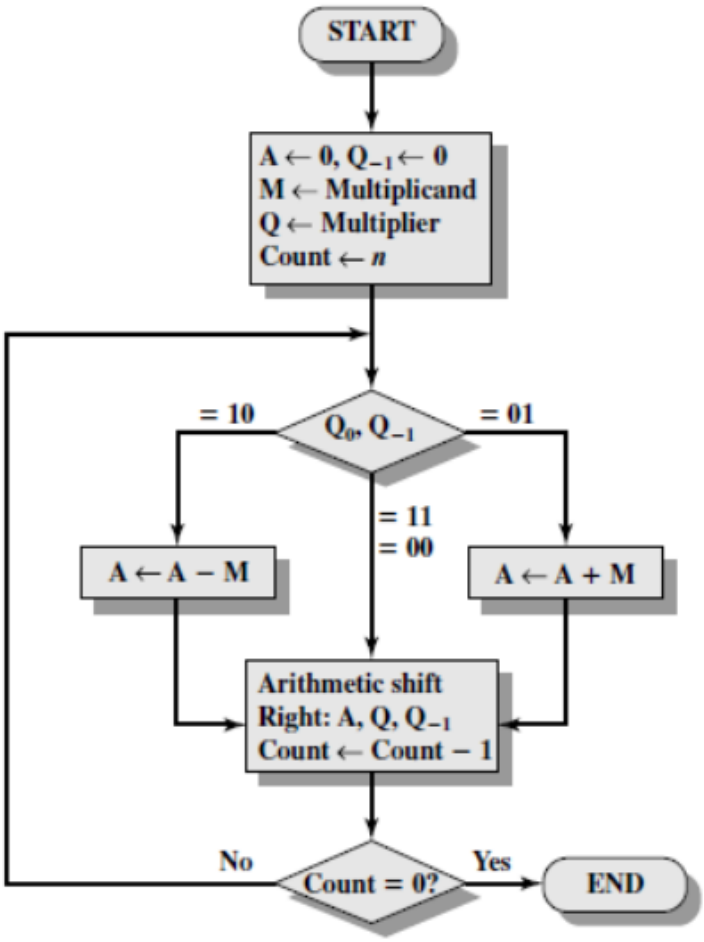
The Read statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR)

Example let say address of the memory location is stored in [AR] which is [2500]. And the content on this memory location is [2500] = [05]. Then according to read operation DR will contains [DR] = [05].

Similarly, the write statement means, a transfer of information from register R1 into the memory word (M) selected by address register (AR). Thus, a write operation can be stated as:
Write: M [AR] ← R1



10. Booth Algorithm



Following are the steps to solve multiplication using booth algorithm

Step1: Convert the given decimal number into binary.

Step 2: Assign the binary value of multiplicand (M) and multiplier (Q) and also find the 2's complement of multiplicand (-M).

Step 3: Initialize the A←0 and Q_{-1}←0. Where, **A** is represented by **n bits**. The value on **n** will be decided by number of bits is used to represent the multiplicand and multiplier. $\mathbf{Q_{-1}}$ is represented with **1 bit**.

Step 4: Write the value of A, Q and Q_{-1} in the below format and initiate the Count value C←n

After that, according to algorithm, we have to check the value of $\mathbf{Q_0.Q_{-1}}$. Where, $\mathbf{Q_0}$ is the Least significate bit of **Q.**

(i). If $Q_0.Q_{-1}$ = 10 then we have to find the new value of A.

A← A-M
Then, we place the new value of A in the above format mentioned in step 4. Further, we do the Arithmetic right shift (A, Q, Q_{-1}). Decrement the count←Count-1 gain check for count to become zero if not then place the last value of (A, Q, Q_{-1}) after the ARS.

Similarly,

(ii). If $Q_0.Q_{-1}$ = 01 then we have to find the new value of A.

A← A+M
Then, we place the new value of A in the above format mentioned in step 4. Further, we do the Arithmetic right shift (A, Q, Q_{-1}). Decrement the count←Count-1 gain check for count to become zero if not then place the last value of (A, Q, Q_{-1}) after the ARS

Else,

If $Q_0.Q_{-1}$ = 00/11 then we directly do the Arithmetic right shift (A, Q, Q_{-1}). Decrement the count←Count-1. When the count become zero (C←0) then stop the process.

Example: - Multiply $(7)_{10}$ and $(3)_{10}$ using booth Algorithm.

Step 1: Convert the given decimal number into binary.

$(7)_{10}$ ← $(0111)_2$
$(3)_{10}$ ← $(0011)_2$

Step 2: Assign the binary value of multiplicand (M) and multiplier (Q) and also find the 2's complement of multiplicand (-M).

(M) ← $(0111)_2$
(Q) ← $(0011)_2$
(-M)← $(1001)_2$

Step 3: Initialize the A←0 and Q_{-1}←0. Where, **A** is represented by **n bits**. The value on **n** will be decided by number of bits is used to represent the multiplicand and multiplier. **Q_{-1}** is represented with **1 bit**.

(A)←0000 and **(Q_{-1})←0**

Step 4: Write the value of A, Q and Q_{-1} in the below format and initiate the Count value

C←4

A	Q	Q_{-1}
0000	0011	0

(i). $Q_0.Q_{-1}$ = 10 then we have to find the new value of A.

A← A-M

A← 1001

A	Q	Q_{-1}
1001	0011	0

(ii) ARS:

A	Q	Q_{-1}
1100	1001	1

Decrement the count←Count-1 and place the last value of (A, Q, Q_{-1}) after the ARS.

C←3

A	Q	Q_{-1}
1100	1001	1

(i), $Q_0.Q_{-1}$ = 11

(ii) ARS:

A	Q	Q_{-1}
1110	0100	1

Decrement the count←Count-1:

C←2

A	Q	Q_{-1}
1110	0100	1

(i), $Q_0.Q_{-1}$ = 01
A← A+M

A← 0101

A	Q	Q_{-1}
0101	0100	0

(ii) ARS:

A	Q	Q_{-1}
0010	1010	0

Decrement the count←Count-1:

C←1

A	Q	Q_{-1}
0010	1010	0

(i), $Q_0.Q_{-1} = 00$

(ii) ARS:

A	Q	Q_{-1}
0001	0101	0

Decrement the count←Count-1:

C←0

Stop the process and note down the value of A,Q only. Therefore, Answer will be $(00010101)_2$ this is nothing but $(21)_{10}$

Module 2

By Dr. Devarshi Shukla
Assistant Professor Electronics and Communication

11. Introduction to Memory

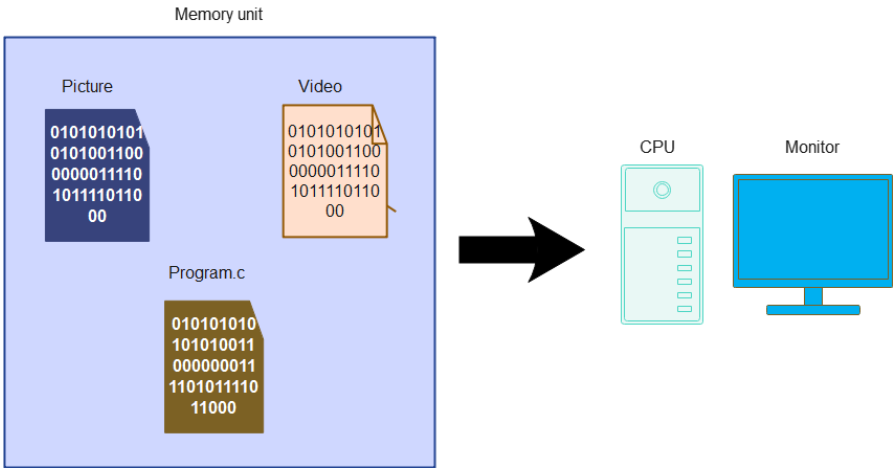


Figure 1: File store in the form of 0s and 1s in memory

All the files (audio, video, text, program, applications) stored in computer memory in the form of 0s and 1s bits and this information is process by the microprocessor.

All these files are of different size. So, we might think that large memory storage will be the solution to this situation. But as the size of the memories increases, then the access time of the data stored in those memories are also increases exponentially (Shown in figure 2) and make the system slow.

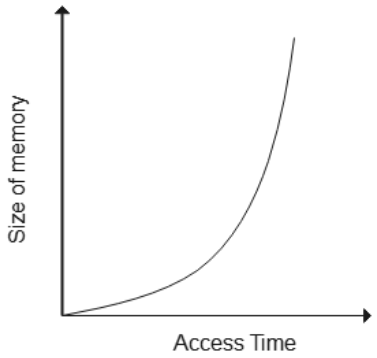


Figure 2: size Vs. Access time

To understand above figure let assume that the clock frequency of microprocessor is 2GHz. Then the time taken to perform the single task will be 0.5ns. This show that the processor speed is very fast. Now if we associate a large memory then the access time of data would be more and processor will be in ideal mode most of the time during the data access time.

Constraint: -

Constraint is that the speed of the processor is high thus, we have required memory unit which is faster and synchronize with the processor speed. On the other hand, if the speed increases, then the size of the memory decreases and cost will increase.

So, to provide the optimized solution in terms of cost, speed, space and needs, different memories has been developed according to their uses and place it in modern computer.

11.1. Types of Memory

The Memory Design is divided into 2 main types:

External Memory or Secondary Memory:

- Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e., peripheral storage devices which are accessible by the processor via an I/O Module.
- They are Non-volatile in nature i.e., they store data permanently.
- Large amount of information can be stored in these memories and their size varies from Mega Byte (MB) to Giga Byte (GB).

Internal Memory or Primary Memory:

- Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.
- They are volatile in nature i.e., if power goes off all the data stored in these memories are also removed.
- Their size varies from Byte to GB.

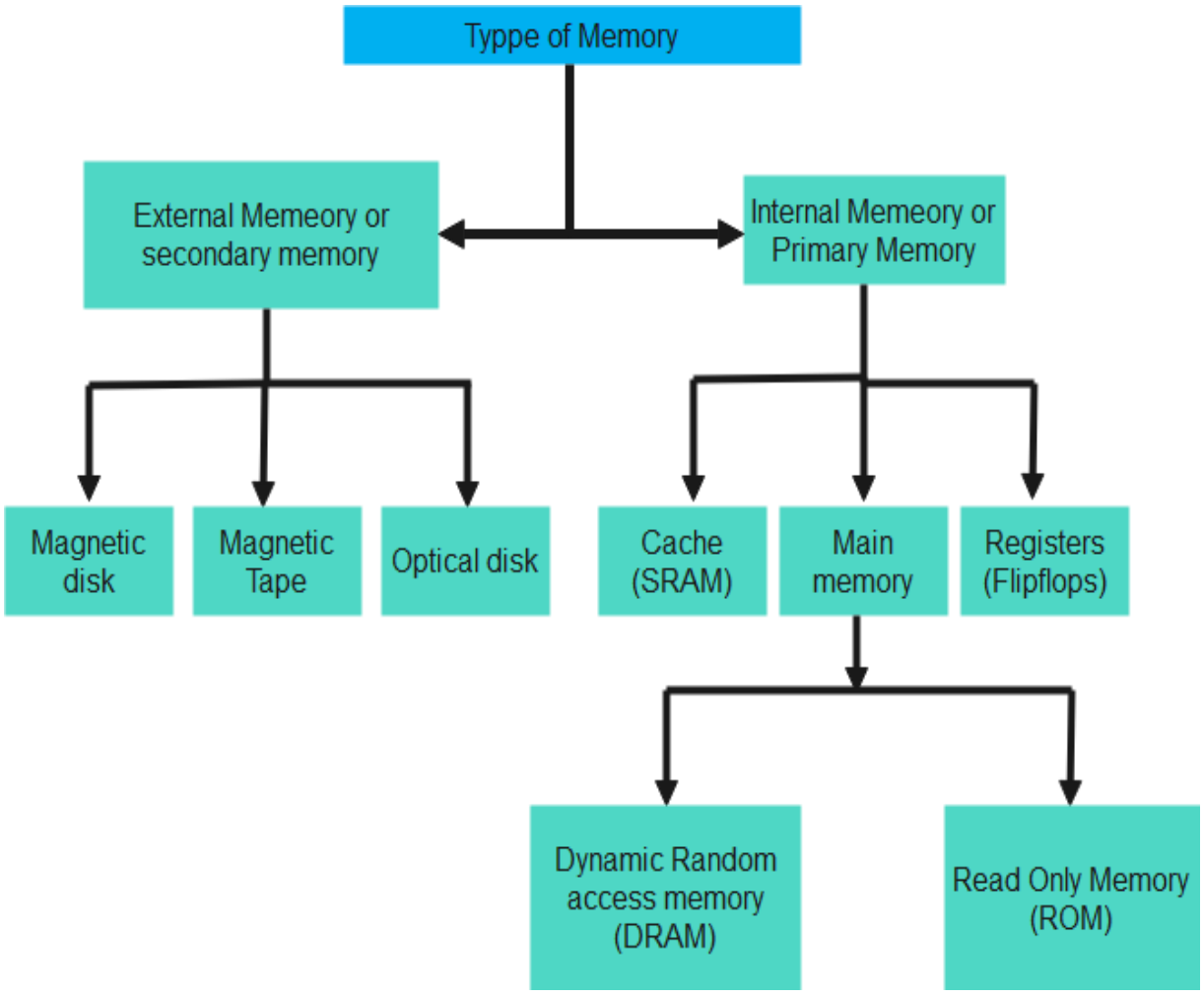
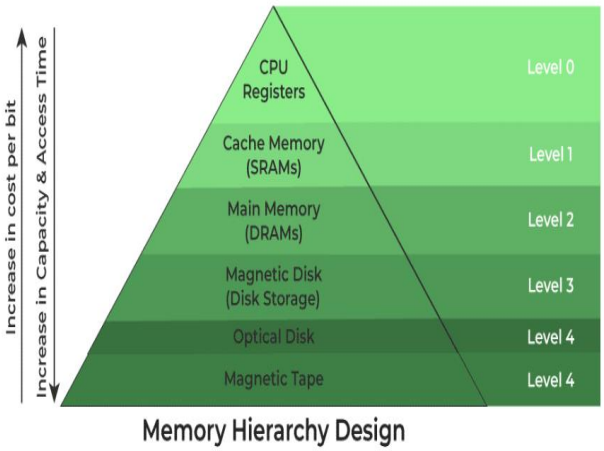


Figure 3 Types of Memory

11.2. Memory Hierarchy Design



Memory Hierarchy Design

11.1.1. Registers

Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 8 to 64 bits.

11.1.2. Primary Memory

Primary memory is volatile in nature and managed by Operating system. It is used to store data and instructions that are currently in use by the CPU.

Types of Primary Memory

Primary memory consists of Main memory and Cache. The main memory comprises of Dynamic Random Access Memory (DRAM) and Read only Memory (ROM).

Dynamic RAM: It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds this is why it is called as dynamic RAM. It is also called as Main memory.

Read Only Memory

11.1.3. Cache Memory

Static RAM*:* Static RAM stores the binary information and information remains valid until power is supplied. It has a faster access time as compared to dynamic RAM and is used in implementing cache memory. Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

11.1.4. Secondary Storage

Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

11.2. Characteristics of Memory Hierarchy

Capacity: It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

Access Time: It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

Performance: Earlier when the computer system was designed without a Memory Hierarchy design, the speed gap increased between the CPU registers and Main Memory due to a large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

Cost Per Bit: As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

11.3. Advantages of Memory Hierarchy

- It helps to manage the memory in a better way.
- It helps in spreading the data all over the computer system.
- It saves the consumer's price and time.

11.4. Memory Interfacing

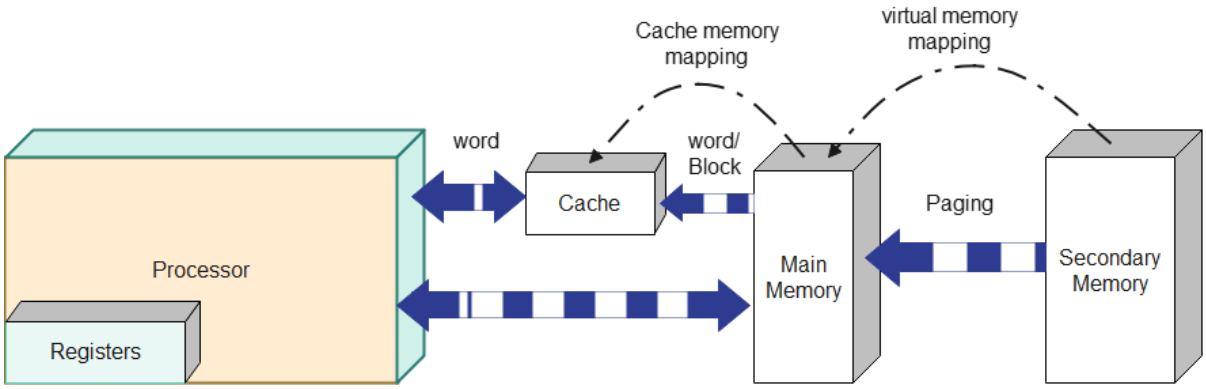


Figure 3: Memory Hierarchy

- Processor have registers which are way faster than Main memory (Dynamic RAM) and secondary memories (HDD, SSD). However, registers are small in size and they even don't not able to store a single instruction. They simply store temporarily data or result.

- Main memory comprises of Dynamic RAM, ROM. It is larger in size due to this it can able to store large instructions as compared to registers. However, the speed of main memory is still slower for processor. Therefore, for faster accessing the data, designer placed another memory between processor and main memory called cache.

- Static RAM is used as cache. It is faster than Main memory and secondary memory. The data which are frequently needed during processing are placed in cache from Main Memory.

Communication Technique between Cache and main memory

The main memory communicate with cache is the form of data word or blocks and the way these communications take place called as memory mapping.

- Registers, cache, main memory does not store data permanently then another type of memory required called as secondary memory. These memories are non-volatile in nature than mean they store the data for long period of time. They are lager in terms of space as compared to all the memories but slower as compared to all the memories.

Communication Technique between Main memory and secondary memory

Virtual memory mapping technique has been used by operating system to communicate between the main memory and secondary memory in the form of paging.

Question on hit ratio??

11.5. Main Memory

The main memory acts as the central storage unit in a computer system. It is a relatively large and fast memory which is used to store programs and data during the run time operations. The primary is classified into two major units.

- Dynamic RAM (Random Access Memory) integrated circuit chips
- ROM (Read Only Memory) integrated circuit chips.

A. Dynamic RAM

The dynamic RAM exhibits the binary information in the form of electric charges that are applied to capacitors. The capacitors are integrated inside the chip by MOS transistors. The dynamic RAM consumes less power and provides large storage capacity in a single memory chip.

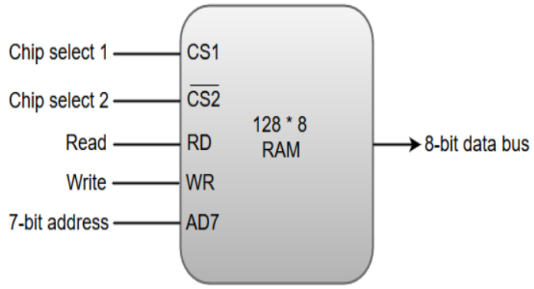


Figure 4. Dynamic RAM of (128 x 8) size.

- A 128 * 8 RAM chip has a memory capacity of 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus.
- The 8-bit bidirectional data bus allows the transfer of data either from memory to CPU during a **read** operation or from CPU to memory during a **write** operation.

- The **read** and **write** inputs specify the memory operation, and the two chip select (CS) control inputs are for enabling the chip only when the microprocessor selects it.
- The bidirectional data bus is constructed using **three-state buffers**.
- The output generated by three-state buffers can be placed in one of the three possible states which include a signal equivalent to logic 1, a signal equal to logic 0, or a high-impedance state.

The following table specifies the RAM operations

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

From the functional table, we can conclude that the unit is in operation only when CS1 = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0.

B. ROM Chip

A ROM memory is used for keeping programs and data that are permanently resident in the computer. Apart from the permanent storage of data, the ROM portion of main memory is needed for storing an initial program called a **bootstrap loader**. The primary function of the bootstrap loader program is to start the computer software operating when power is turned on. ROM chips are also available in a variety of sizes and are also used as per the system requirement.

The following block diagram demonstrates the chip interconnection in a 512 * 8 ROM chip.

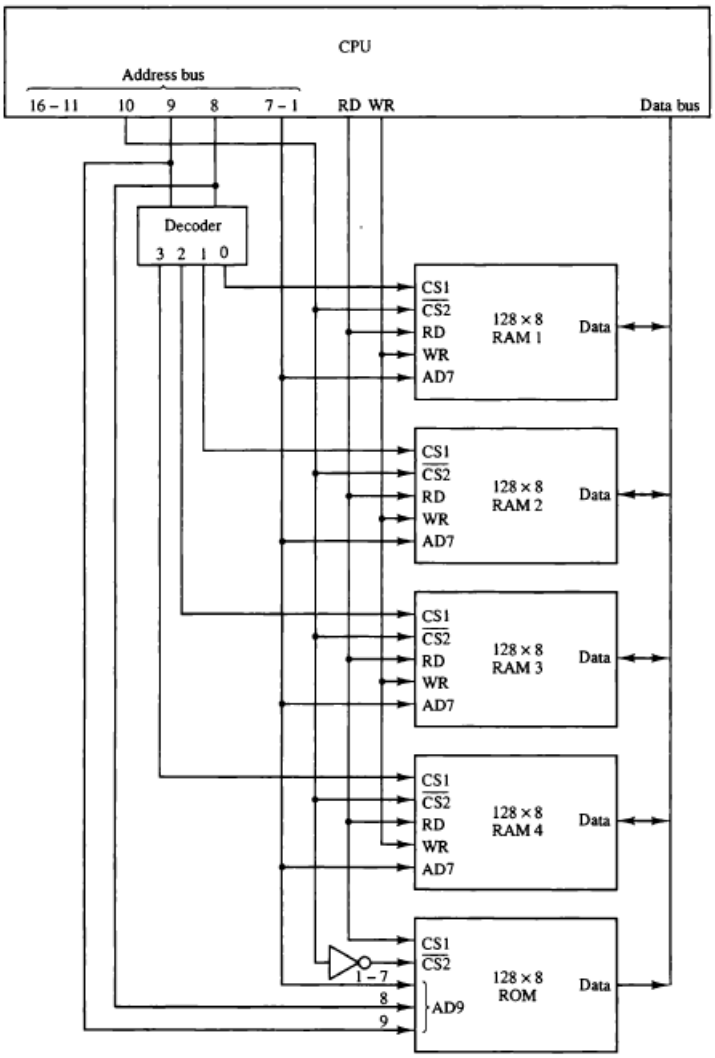
Figure 5. ROM chip of (512 x 8) size.

- A ROM chip has a similar organization as a RAM chip. However, a ROM can only perform read operation; the data bus can only operate in an output mode.
- The 9-bit address lines in the ROM chip specify any one of the 512 bytes stored in it.

- The value for chip select 1 and chip select 2 must be 1 and 0 for the unit to operate. Otherwise, the data bus is said to be in a high-impedance state.

Question on Memory Interfacing

Q.1.



Cache Memory

The data or contents of the main memory that are used frequently by processor are stored in the cache memory so that the processor can easily access that data in a shorter time.

The basic operation of a cache memory is as follows:

- When the processor needs to access memory, the cache is examined. If the word is found in the cache it is called as HIT.
- If the word addressed by the processor is not found in the cache it is called as MISS, and then the main memory is accessed to read the word.
- The performance of the cache memory is frequently measured in terms of a quantity called hit ratio. The ratio of the number of hits divided by the total processor references to memory (hits plus misses) is the hit ratio.

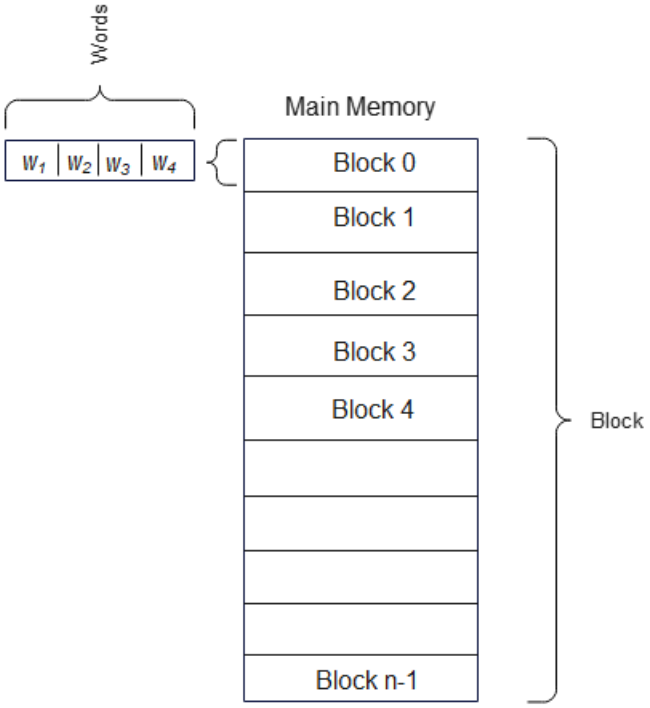
Concepts:

1. **Conversion**

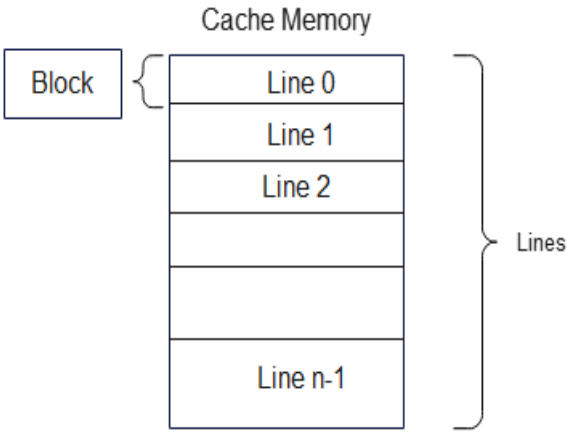
Size		In binary
1 Byte	**8 Bits**	2^3
1 Word	**1 Byte**	2^3
1Kilo	**1024 Byte**	2^{10}
1Mega	**1024 KB**	2^{20}
1Gega	**1024 MB**	2^{30}
1Tera	**1024GB**	2^{40}

2. Concept of "LINE" and "BLOCKS"

Main memory divided into blocks each block may contains one or more than one words.



and cache memory divided into lines and each line can contain one block at a time.

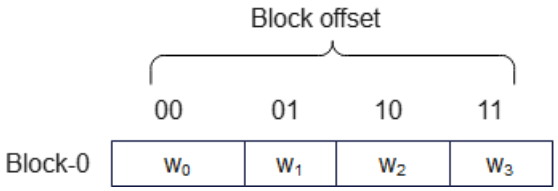


3. **Physical address.**

Physical address will be generated by processor to access the data from the cache memory. The number of bits require to represent the physical address is equal to the number of bits require to represent the size of the main memory.

Terminology used in Physical address format:

1. **Block offset:**
   It provides the address or location of the word (required by the processor) of the particular block. It is show in below figure.



   The number of bits to represent the block offset is determined from the number of words is present in each block. From above figure block-0 has four words the number of bits require to represent the block offset will be 2 (i.e.,$4 = 2^2$).
2. **Index/ Line number:**
   The index provides the line number of the cache from where the desired word will be access by the processor. The number of bits to represent the index is determined from the number of lines in the cache memory. Let say if the number of lines in cache memory is 8 then number of bits require to represent the index will be 3 (i.e., $8 = 2^3$).

Cache Mapping:

As we know that the size of cache memory is smaller than the Main memory so all the blocks cannot be assigned to the cache memory at once. To do so we require some technique to assign all the blocks to the cache is called as mapping. All the blocks are mapped in cache from main memory in round robin fashion.

There are three different types of mapping used for the purpose of cache memory which are as follows:

- Direct mapping,
- Associative mapping
- Set-Associative mapping

A. Direct mapping

(i). Mapping: - In direct mapping, the formula used to map each block from main memory onto the cache memory lines is (*K* **mod** *n*). where K is blocks number of main memory and n is number of lines in cache memory.

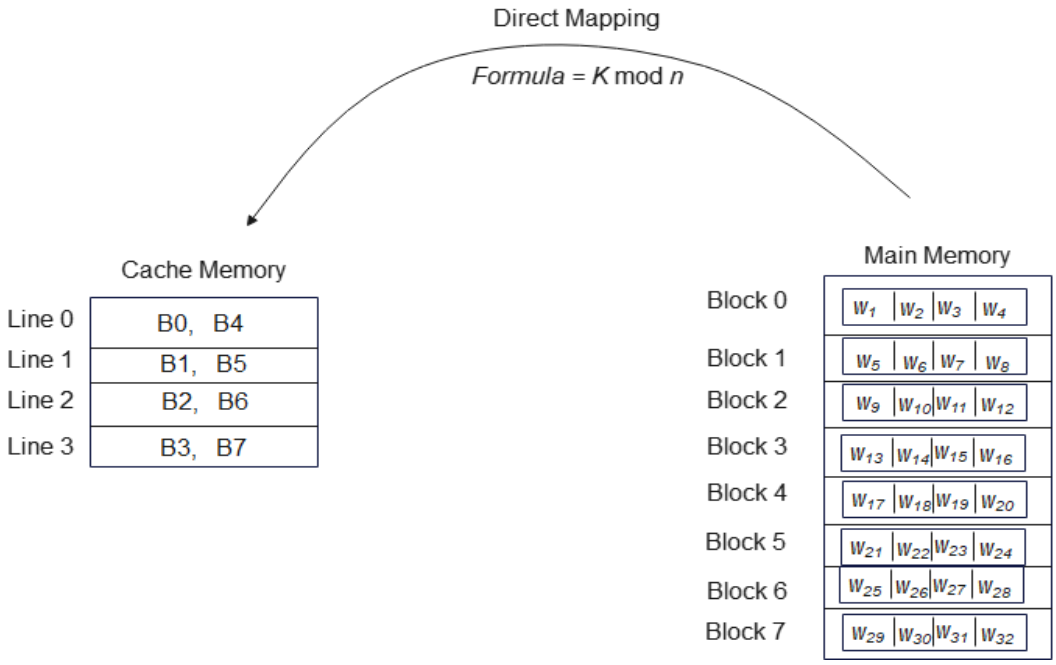
For Example: Let say that if the size on memory is of 32 Byte and each block size is 4 Byte. Also, the cache memory is of 16 Byte. First, we have to find the number of blocks in main memory (K)

$$Total\ number\ of\ blocks\ in\ MM\ (K) = \frac{Total\ size\ of\ \ Main\ Memeory}{Size\ of\ each\ block} = \frac{32B}{4B} = 8$$

Also,

$$\text{The number of lines in cache memory (n)} = \frac{Total\ size\ of\ \ cache\ Memeory}{Size\ of\ each\ line} = \frac{16B}{4B} = 4$$

Here, size of line in cache memory is equal to block size of main memory.

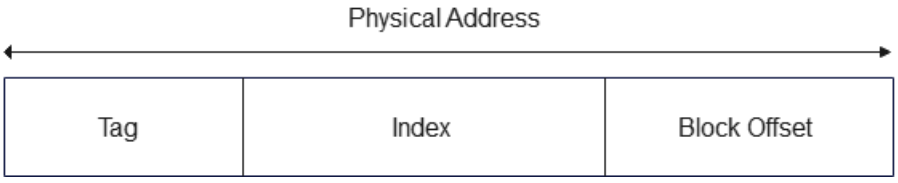


So, B0 will be place in cache memory line number is given by 0 mod 4= 0 therefore Block-0 will be placed in line number 0 of the cache memory. B1 will be place in cache memory line number is given by 1 mod 4 =1, this means block Block-1 placed in line number 1of the cache memory. Similarly, B2, B3, B4, B5, B6 and B7 will be placed in cache memory line number 2,3,0,1,2,3 respectively. Thus, from the above figure it is found that (B0, B4) will be mapped in line 0, similarly, (B1, B5) (B2, B6) and (B3, B7) will be mapped only in Line1, Line2, Line3 respectively.

It should also keep in mind that at any instant of time only one block will be placed in each line and all the words of that particular block will be in their respective cache line. That is if Block-0 is placed in line 0 then all the word of Block-0 will be in cache memory line 0.

(ii). How processor will access the word in cache in direct mapping?

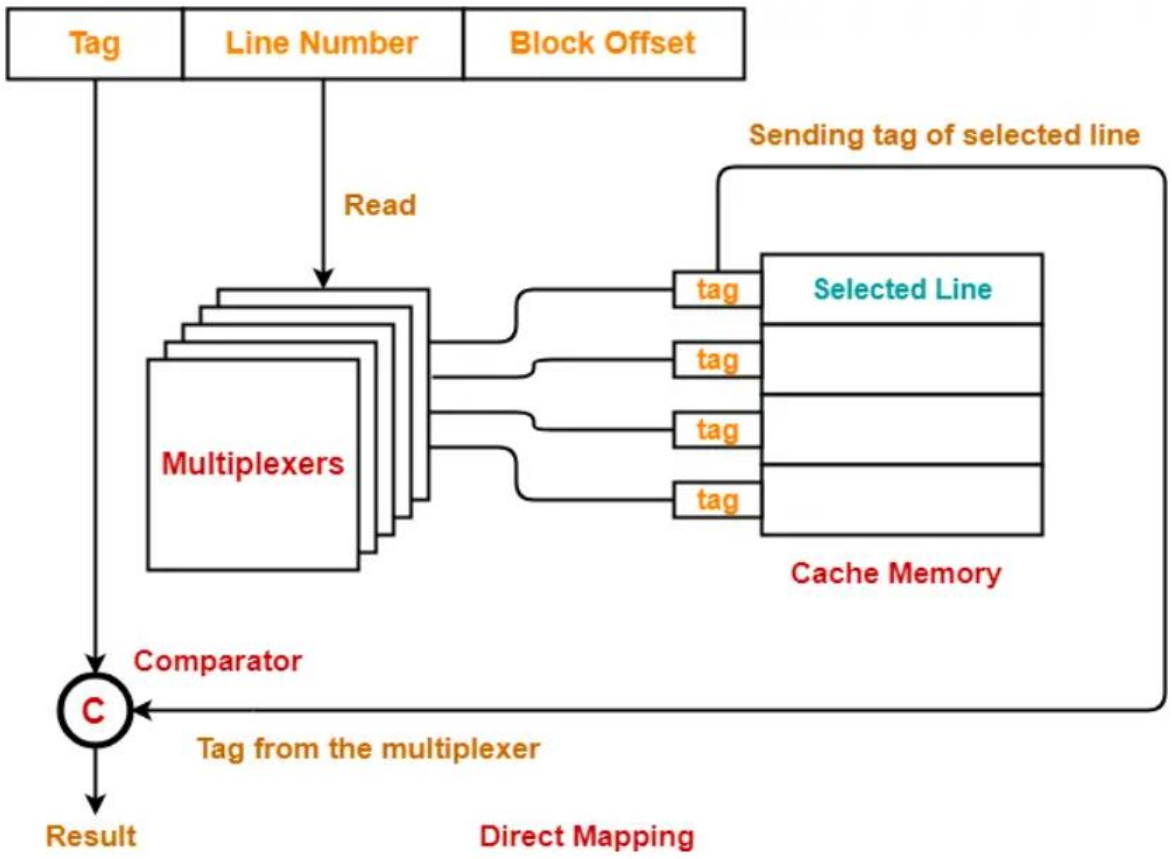
Processor will generate the physical address to read the particular word. The physical address has three parts called as Tag, index/Line number and Block offset as shown in below figure.



The following steps explain the working of direct mapped cache-
After CPU generates a memory request,

- The line number/ index number field of the address is used to access the particular line of the cache.
- The tag field of the CPU address is then compared with the tag of the line.
- If the two tags match, a cache hit occurs and the desired word is found in the cache.

- If the two tags do not match, a cache miss occurs.
- In case of a cache miss, the required word has to be brought from the main memory.
- It is then stored in the cache together with the new tag replacing the previous one.



B. Associative mapping

In associative mapping any block can be placed in any line of the cache memory. Whenever a data is requested, the Tag bits are simultaneously compared with all stored tags in each line of the cache memory using the internal logic. If a match is found, the corresponding is read out. Otherwise, the main memory is accessed if address is not found in cache.

C. Set Associative mapping

(i). Mapping: -

k-way set associative mapping,

- Cache lines are grouped into sets where each set contains k number of lines.
- A particular block of main memory can map to only one particular set of the cache.
- However, within that set, the memory block can map to any freely available cache line.
- The set of the cache to which a particular block of the main memory can map is given by-

Cache set number

= (Main Memory Block Address) Modulo (Number of sets in Cache)

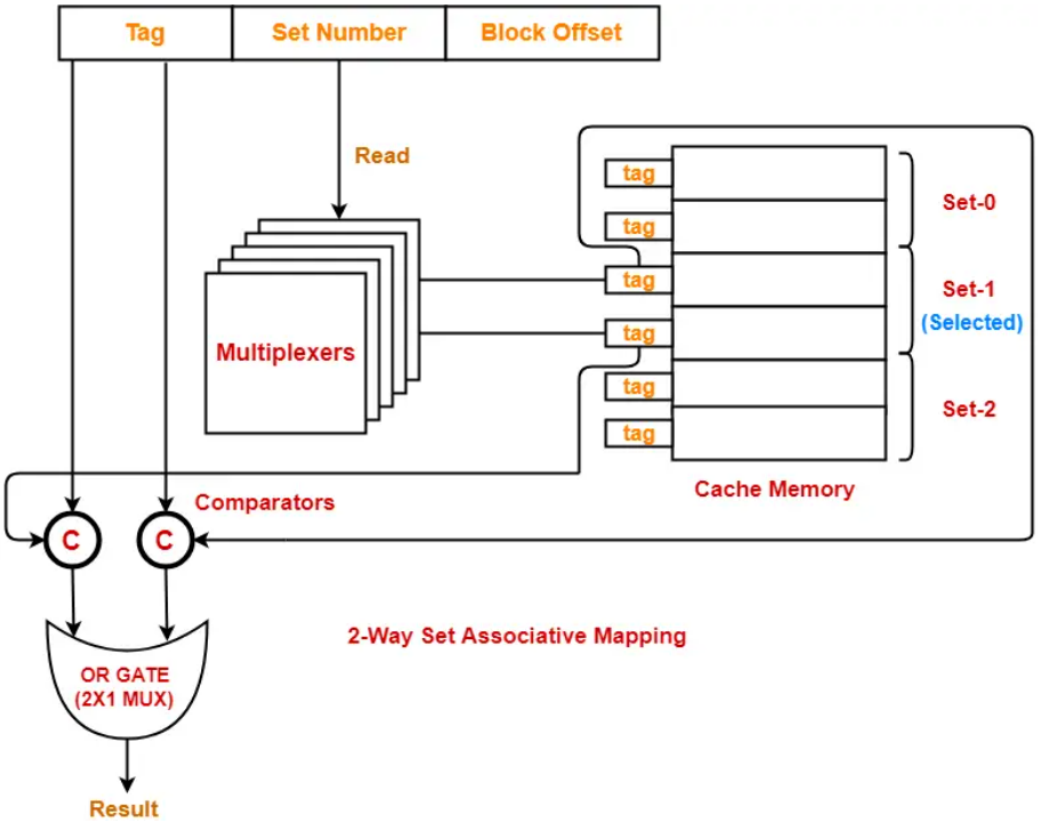
Example: Lets we have a 2-way set associative mapping. The size of the main memory is 32 Byte/ 32 words, block size is 4 byte/ 4 words, and cache memory size is 16 bytes. Then the number of blocks in main memory will be 8 and number of lines in cache memory is 4. Further, according to 2-way set associative mapping the number of lines in each set will be 2. Therefore, the number of set will be 2 is given by

No. of set = (Number of lines in cache / k)

where k is number of lines in each set

(ii). How processor will access the word in cache in direct mapping?

The following diagram shows the implementation of 2-way set associative cache-



Step-01:

- Each multiplexer reads the set number from the generated physical address using its select lines in parallel.
- To read the set number of S bits, number of select lines each multiplexer must have = S.

Step-02:

- After reading the set number, each multiplexer goes to the corresponding set in the cache memory.

Step-03:

- Tag bits of Physical address compared in comparator circuit with the tag bits of selected lines of the selected set number of the cache memory.

Step-04:

- The output result of each comparator is fed as an input to an OR Gate.
- OR Gate is usually implemented using 2 x 1 multiplexer.
- If the output of OR Gate is 1, a cache hit occurs otherwise a cache miss occurs.

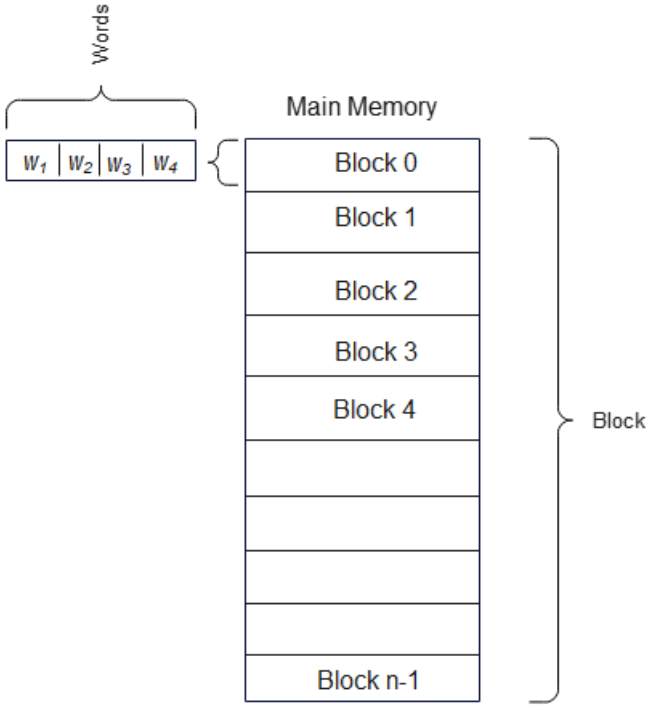
Numerical on Memory interfacing

1. **Conversion**

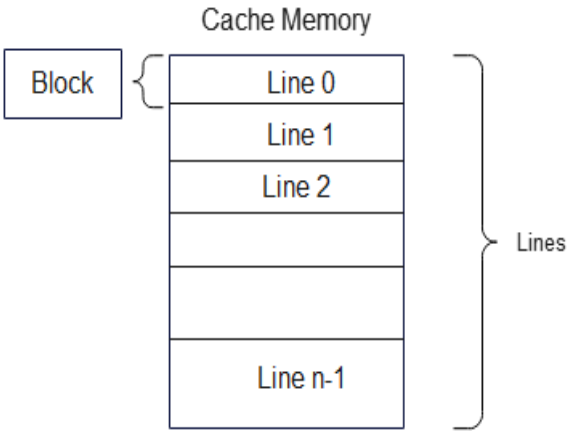
Size		In binary
1 Byte	8 Bits	2^3
1 Word	1 Byte	2^3
1Kilo	1024 Byte	2^{10}
1Mega	1024 KB	2^{20}
1Gega	1024 MB	2^{30}
1Tera	1024GB	2^{40}

2. Concept of "LINE" and "BLOCKS"

Main memory divided into blocks each block may contains one or more than one words.



and cache memory divided into lines and each line can contain one block at a time.



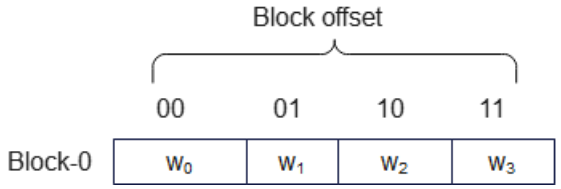
Numerical on Memory interfacing

1. **Physical address.**

Physical address will be generated by processor to access the data from the cache memory. The number of bits require to represent the physical address is equal to the number of bits require to represent the size of the main memory.

Terminology used in Physical address format:

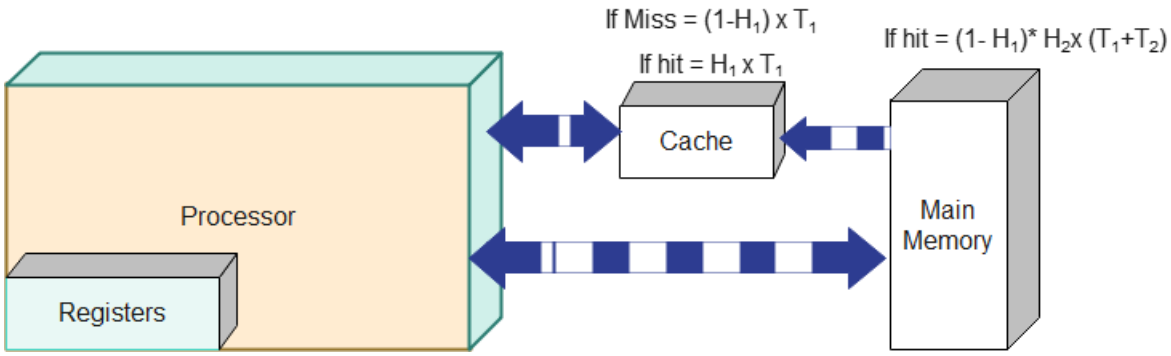
1. **Block offset:**
   It provides the address or location of the word (required by the processor) of the particular block. It is show in below figure.



   The number of bits to represent the block offset is determined from the number of words is present in each block. From above figure block-0 has four words the number of bits require to represent the block offset will be 2 (i.e.,$4 = 2^2$).
2. **Index/ Line number:**
   The index provides the line number of the cache from where the desired word will be access by the processor. The number of bits to represent the index is determined from the number of lines in the cache memory. Let say if the number of lines in cache memory is 8 then number of bits require to represent the index will be 3 (i.e., $8 = 2^3$).

3. **Tag Number:** A unique identifier for a group of data. Because different regions of memory may be mapped into a block, the tag is used to differentiate between them.

Q1. What is the average memory access time for a machine with a cache hit rate of 80% and cache access time of 5 ns and main memory access time of 100 ns when



Solution: -

Average Access Time (T_{avg}) = $H_1xT_1 + (1- H_1) * H_2 x (T_1+T_2)$.

Given,

$H_1 = 0.8$ and $H_2 = 1$.

$T_1 = 5ns$ and $T_2 = 100ns$

Put all the given value in above formula

$T_{avg} = 0.8 \times 5ns + (1- 0.8) * 1 \times (105ns).$

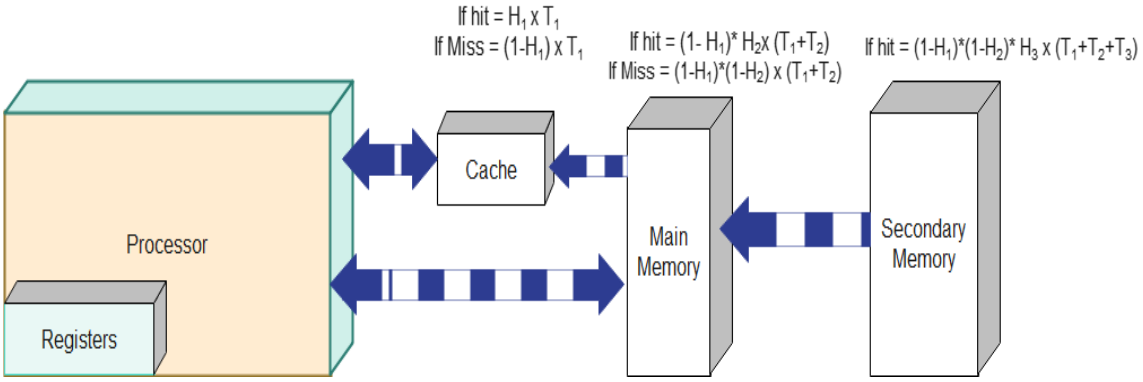
$= 4ns + 0.2 \times 105 ns$

$= 4ns + 21ns$

T_{avg} = 25ns Ans

Q.2. A computer has a cache, main memory and a disk used for virtual memory. An access to the cache takes 10 ns. An access to main memory takes 100 ns. An access to the disk takes 10,000 ns. Suppose the cache hit ratio is 0.9 and the main memory hit ratio is 0.8. The effective access time required to access a referenced word on the system is

Solution:



Effective memory access time

$= H1 \times T1 + (1 – H1) \times H2 \times (T1 + T2) + (1 – H1) \times (1 – H2) \times H3 \times (T1 + T2 + T3)$

$= 0.9 \times 10 ns + (1 – 0.9) \times 0.8 \times (10 ns + 100 ns) + (1 – 0.9) \times (1 – 0.8) \times 1 \times (10 ns + 100 ns + 10000 ns)$

$= 9 ns + 8.8 ns + 202.2 ns$

$= 220 ns$

Q3. Solve the following questions:

a. How many 128 x 8 RAM chips are needed to provide a memory capacity of 2048 bytes?

b. How many lines of the address bus must be used to access 2048 bytes of memory?

c. How many of these lines will be common to all chips?

d. How many lines must be decode for chip select?

e. Specify the size of the decoders.

Solution: -

Numerical on Memory interfacing

A. $Number\ of\ RAM\ Chip = \frac{Total\ Size\ of\ Memory}{Size\ of\ singe\ RAM\ chip} = \frac{2048\ Byte}{128\ X\ 8} = 16\ chips$

B. The number of address line must be needed to access (n) $= 2048 = 2^n = 2^{11} = 11$

C. The number of line common to all chip $=$ number of address lines of single chip $= 2^{AD}$

Where, AD is the number of address line of single RAM chip 128 = $2^{AD7} = 7$.

D. The of lines of decoder to select the chip = Total address line of memory – the number of address line of single RAM chip. Therefore,

$$\text{Total line of the decoder to select the chip } (x) = 11 - 7 = 4$$

E. The size of the decoder $= x * 2^x$. Where, *x* is the number of Decoder line/ input line

Q.4. A computer has 16-bit processor and employ RAM chip of 128 X 8 and ROM chip of 512 X 8. The computer system needs 512 Byte of RAM and 512 Byte of ROM.

a. How may RAM and ROM chips are needed.

b. How many addresses line will be in each RAM and ROM chip.

c. How many Inputs line in decoder for chip selection and what is the size of decoder.

d. Draw the interface diagram of RAM and ROM chips with processor along with address mapping.

Solution: -

A. $Number\ of\ RAM\ Chip = \frac{Total\ Size\ of\ Memory}{Size\ of\ singe\ RAM\ chip} = \frac{512\ Byte}{128\ X\ 8} = 4\ chips$

$Number\ of\ ROM\ Chip = \frac{Total\ Size\ of\ Memory}{Size\ of\ singe\ ROM\ chip} = \frac{512\ Byte}{512\ X\ 8} = 1\ chips$

B. The number of address line in RAM (n) $= 128 = 2^n = 2^7 = 7$

The number of address line in ROM (n) $= 512 = 2^n = 2^9 = 9$

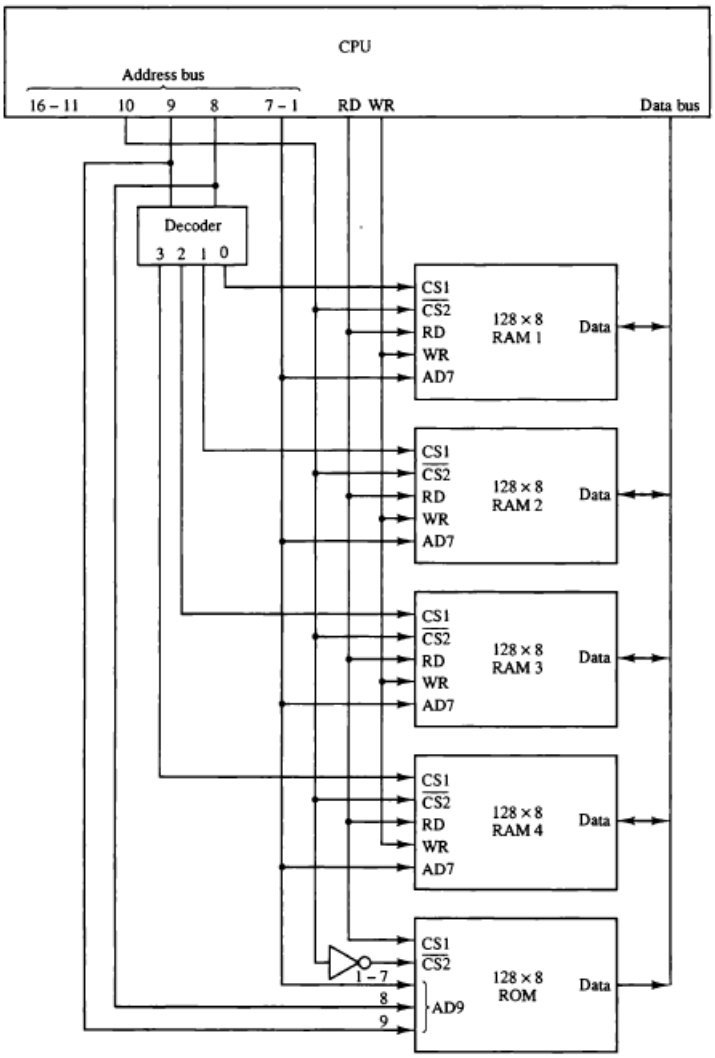
C. The of lines of decoder to select the chip = (Total address line of memory) – (the number of address line of single RAM chip). Therefore,

$$\text{Total line of the decoder to select the chip } (x) = 9 - 7 = 2$$

Then size of decoder = 2 X 4.

D. The interfacing of RAM and ROM with the processor will be

Numerical on Memory interfacing



Component	Hexadecimal Address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM1	0000-007F	0	0	0	×	×	×	×	×	×	×
RAM2	0080-00FF	0	0	1	×	×	×	×	×	×	×
RAM3	0100-017F	0	1	0	×	×	×	×	×	×	×
RAM4	0180-01FF	0	1	1	×	×	×	×	×	×	×
ROM	0200-03FF	1	×	×	×	×	×	×	×	×	×

Q.4. A computer employs RAM chips of 256 x 8 and ROM chips of 1024 x 8. The computer system needs 2K bytes of RAM, 4K bytes of ROM, and four interface units, each with four registers. A memory-mapped I/O configuration is used. The two highest-order bits of the address bus are assigned 00 for RAM and 10 for interface registers.

a. How many RAM and ROM chips are needed?

b. Draw a memory-address map for the system.

c. Give the address range in hexadecimal for RAM, ROM and interface.

Solution: -

A. $$Number\ of\ RAM\ Chip = \frac{Total\ Size\ of\ Memory}{Size\ of\ singe\ RAM\ chip} = \frac{2K\ Byte}{256\ X\ 8} = 8\ chips$$

$$Number\ of\ ROM\ Chip = \frac{Total\ Size\ of\ Memory}{Size\ of\ singe\ ROM\ chip} = \frac{4KByte}{1024\ X\ 8} = 4\ chips$$

B. Number of address line in RAM = 8 and in ROM =10.
C. Decoder line from RAM will be 3 X 8 and for ROM will be 2 X 4.
D. 4 interface each with four registers therefore, 4 * 4 = 16

Component	Address in Hexadecimal	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
RAM	0000-07FF	0	0	0	0	0	3 x 8 decoder			x	x	x	x	x	x	x	x
ROM	1000-1FFF	0	1	0	0	2 x 4 Decoder		x	x	x	x	x	x	x	x	x	x
Register interface	2000-200F	1	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x

Q5. Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find-

A. Number of bits in tag
B. Tag directory size

Solution: -

Given-

Cache memory size = 16 KB

Block size = Line size = 256 bytes

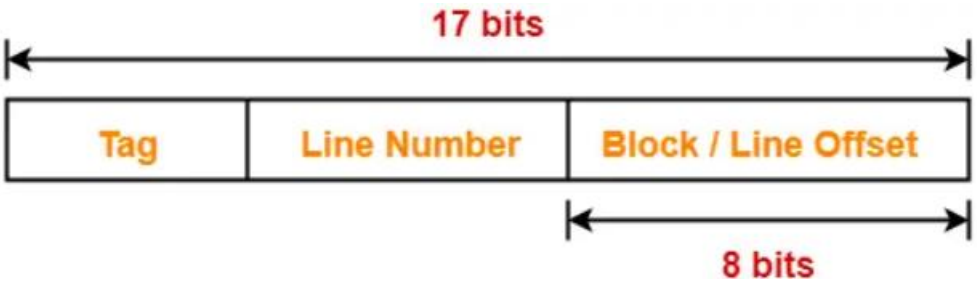
Main memory size = 128 KB

We consider that the memory is byte addressable

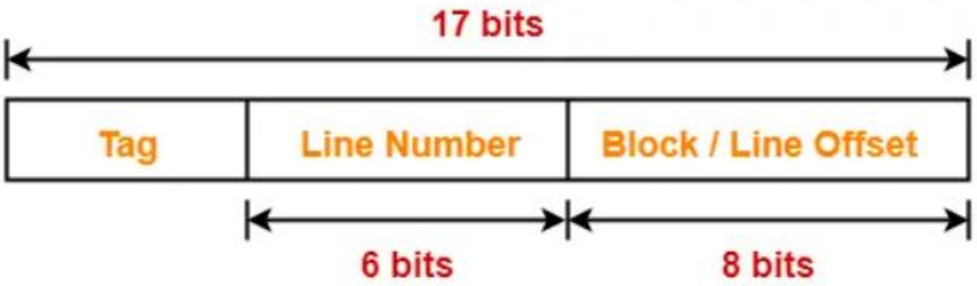
We have, Size of main memory = 128 KB = 2^{17} bytes. Thus, Number of bits in physical address = 17 bits

Numerical on Memory interfacing

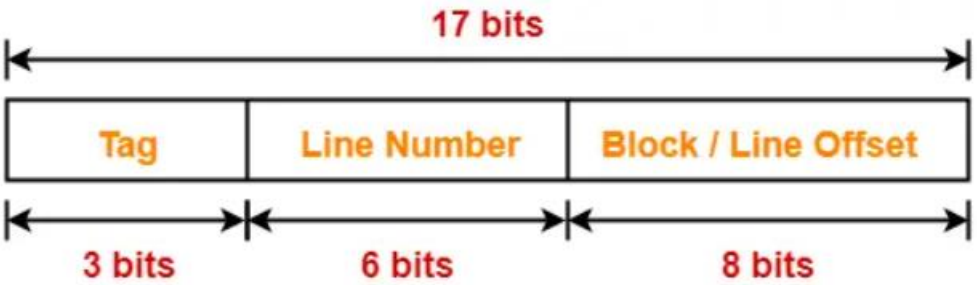
Number of Bits in Block Offset- We have, Block size = 256 bytes = 2^8 bytes. Thus, Number of bits in block offset = 8 bits.



Number of Bits in Line Number- Total number of lines in cache = Cache size / Line size = 16 KB / 256 bytes = 2^{14} bytes / 2^8 bytes = 2^6 lines. Thus, Number of bits in line number = 6 bit



Number of Bits in Tag- Number of bits in tag = Number of bits in physical address – (Number of bits in line number + Number of bits in block offset) = 17 bits – (6 bits + 8 bits) = 17 bits – 14 bits = 3 bits. Thus, Number of bits in tag = 3 bits.



Tag Directory Size- Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag = 26 x 3 bits = 192 bits = 24 bytes. Thus, size of tag directory = 24 bytes.

Q.7. Consider a direct mapped cache of size 512 KB with block size 1 KB. There are 7 bits in the tag. Find-

1. **Size of main memory**
2. **Tag directory size**

Solution-

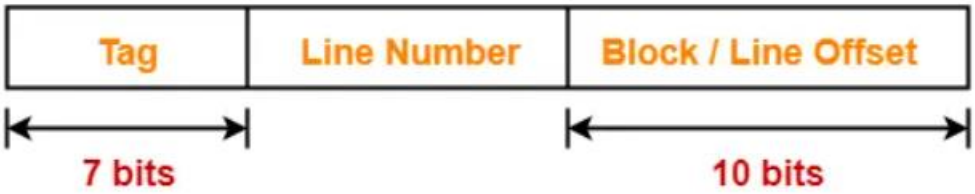
Given-

- Cache memory size = 512 KB

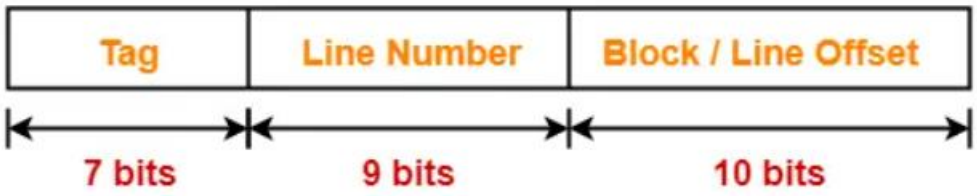
- Block size = Line size = 1 KB
- Number of bits in tag = 7 bits

We consider that the memory is byte addressable.

Number of Bits in Block Offset- We have, Block size = 1 KB = 2^{10} bytes. Thus, Number of bits in block offset = 10 bits.



Number of Bits in Line Number- Total number of lines in cache = Cache size / Line size = 512 KB / 1 KB. = 2^9 lines Thus, Number of bits in line number = 9 bits.



Number of bits in physical address = Number of bits in tag + Number of bits in line number + Number of bits in block offset = 7 bits + 9 bits + 10 bits = 26 bits. Thus, Number of bits in physical address = 26 bits. We have, Number of bits in physical address = 26 bits. Thus, Size of main memory = 2^{26} bytes = 64 MB.

Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag = 2^9 x 7 bits = 3584 bits = 448 bytes**.** Thus, size of tag directory = 448 bytes

Q.8. Consider a direct mapped cache with block size 4 KB. The size of main memory is 16 GB and there are 10 bits in the tag. Find-

1. **Size of cache memory**
2. **Tag directory size**

Solution:-

Given-

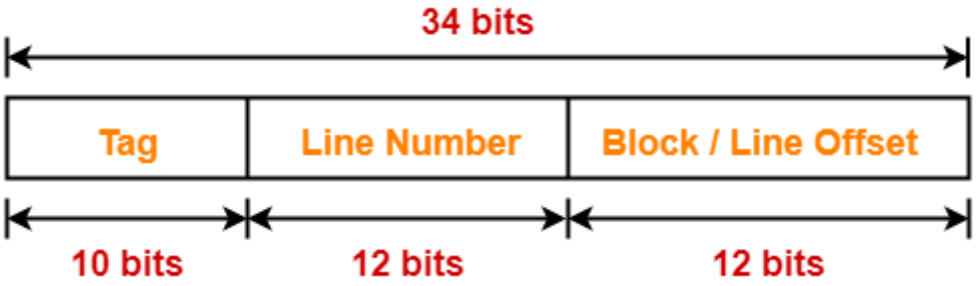
- Block size = Frame size = Line size = 4 KB
- Size of main memory = 16 GB
- Number of bits in tag = 10 bits

We consider that the memory is byte addressable.

Number of Bits in Physical Address- We have, Size of main memory = 16 GB = 2^{34} bytes**.** Thus, Number of bits in physical address = 34 bits**.**
Number of Bits in Block Offset- We have, Block size = 4 KB = 2^{12} bytes. Thus, Number of bits in block offset = 12 bits.

Numerical on Memory interfacing

Number of Bits in Line Number- Number of bits in line number = Number of bits in physical address – (Number of bits in tag + Number of bits in block offset) = 34 bits – (10 bits + 12 bits) = 34 bits – 22 bits = 12 bits**.** Thus, Number of bits in line number = 12 bits



Number of Lines in Cache- We have Number of bits in line number = 12 bits**.** Thus, Total number of lines in cache = 2^{12} lines.
Size of Cache Memory- Size of cache memory = Total number of lines in cache x Line size = 2^{12} x 4 KB = 2^{14} KB = 16 MB Thus, Size of cache memory = 16 MB.
Tag Directory Size- Tag directory size = Number of tags x Tag size = Number of lines in cache x Number of bits in tag = 2^{12} x 10 bits = 40960 bits = 5120 bytes. Thus, size of tag directory = 5120 bytes.

Q.9. Consider a 4-way set associative mapped cache with block size 4 KB. The size of main memory is 16 GB and there are 10 bits in the tag. Find-

1. **Size of cache memory**
2. **Tag directory size**

Solution-

Given-

- Set size = 4
- Block size = Frame size = Line size = 4 KB
- Main memory size = 16 GB
- Number of bits in tag = 10 bits

We consider that the memory is byte addressable.

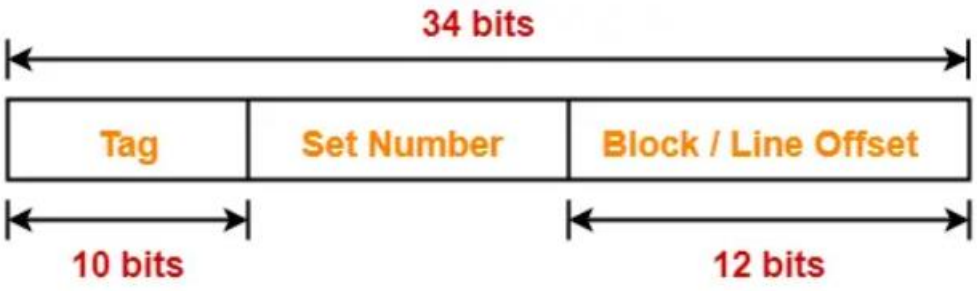
Number of Bits in Physical Address-

We have, Size of main memory = 16 GB = 2^{34} bytes. Thus, Number of bits in physical address = 34 bit.

Number of Bits in Block Offset-

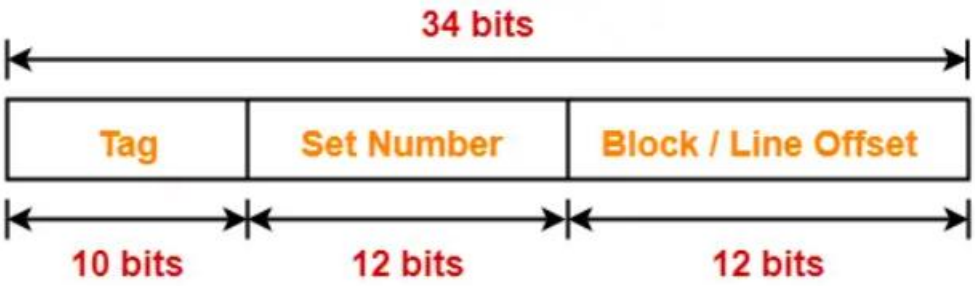
We have, Block size = 4 KB = 2^{12} bytes. Thus, Number of bits in block offset = 12 bits

Numerical on Memory interfacing



Number of Bits in Set Number-

Number of bits in set number = Number of bits in physical address – (Number of bits in tag + Number of bits in block offset) = 34 bits – (10 bits + 12 bits) = 34 bits – 22 bits = 12 bits. Thus, Number of bits in set number = 12 bits.



Number of Sets in Cache-

We have, Number of bits in set number = 12 bits. Thus, Total number of sets in cache = 2^{12} sets

Number of Lines in Cache-

We have, Total number of sets in cache = 2^{12} sets. Each set contains 4 lines

Thus, Total number of lines in cache = Total number of sets in cache x Number of lines in each set = 2^{12} x 4 lines = 2^{14} lines

Size of Cache Memory-

Size of cache memory = Total number of lines in cache x Line size = 2^{14} x 4 KB = 2^{16} KB = 64 MB

Thus, Size of cache memory = 64 MB

Q.10. A two-way set associative cache memory uses blocks of four words. The cache can accommodate a total of 2048 words from main memory. The main memory size is 128Kx32. Formulate all pertinent information required to construct the cache memory. What is the size of cache memory.

Solution:

Given: -

- Set size = 2
- Block size = Line size = 4 Words
- Main memory size = 128K x 32
- Number of words accommodate in cache is 2048.

Number of Bits in Physical Address-

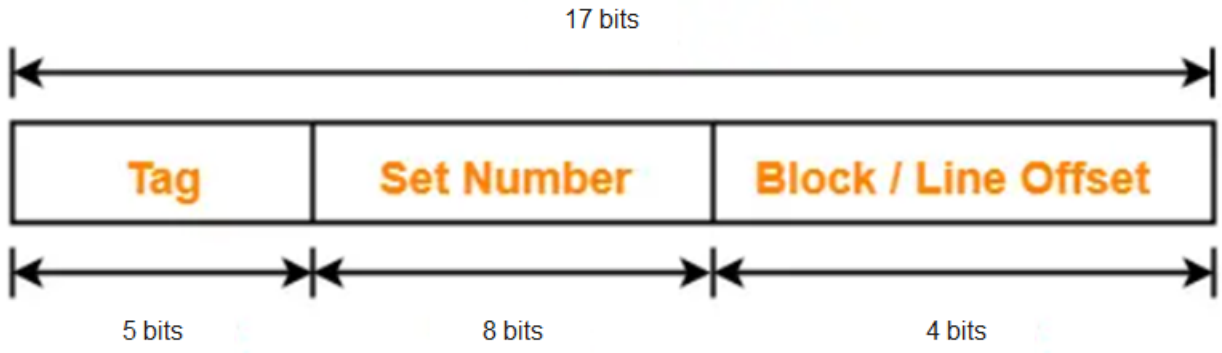
The Size of the main memory is 128K X32 so number of bits is used to represent memory space in main memory will be 2^{17}. Therefore, Number of bits in physical address = 17 bit

Number of Bits in Block Offset-

We have, Block size = 4 words and size of word is 32 bits this means that each word is of 4 bytes therefore, Block size will be 4 X4Byte =16 Byte = 2^4 byte. Thus, Number of bits in block offset = 4 bits

Number of Bits in Set Number-

Cache memory accommodate 2048 words and size of each line is 4 words thus, total number of lines in cache memory will be 512. Moreover, the 2-way set associative cache memory has been used thus, total number of sets will be 512/2 =256 = 2^8. Therefore, Number of bits in set number = 8 bits. And the remain bit are used to represent Tag = 5 bits.



Number of Sets in Cache-

We have, Number of bits in set number = 8 bits

Thus, Total number of sets in cache = 2^8 sets = 256

Number of Lines in Cache-

Total number of sets in cache = 256 and each set contains 2 lines. Thus, total number of Line is cache memory will be 256 X2= 512= 2^9

Size of Cache Memory-

Size of cache memory = Total number of lines in cache x Line size = 2^9 X 2^4 Byte = 2^{13} B = 8 KB

Thus, Size of cache memory = 8KB

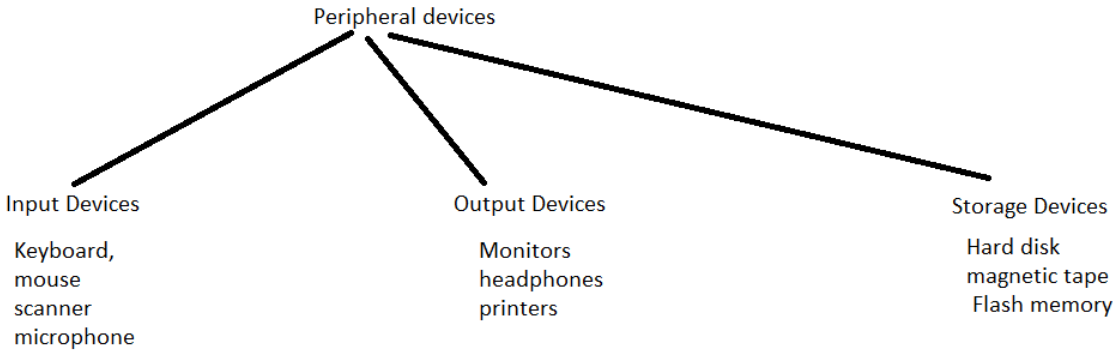
UNIT 5 (NOTES)

Peripheral Devices

A Peripheral Device is defined as the device which provides input/output functions for a computer. It communicate with the outside world microcomputers use peripherals (I/O devices). Commonly used peripherals are: A/D converter, D/A converter, CRT, printers, Hard disks, floppy disks, magnetic tapes etc. A peripheral device is a device that is connected to a computer system but is not part of the core computer system architecture.

Classification of Peripheral devices:

It is generally classified into 3 basic categories which are given below:



1. **Input Devices:**
   The input devices is defined as it converts incoming data and instructions into a pattern of electrical signals in binary code that are comprehensible to a digital computer.
   Example:
   Keyboard, mouse, scanner, microphone etc.

2. **Output Devices:**
   An output device is generally reverse of the input process and generally translating the digitized signals into a form intelligible to the user. The output device is also performed for sending data from one computer system to another. For some time punched-card and paper-tape readers were extensively used for input, but these have now been supplanted by more efficient devices.
   Example:
   Monitors, headphones, printers etc.

3. **Storage Devices:**
   Storage devices are used to store data in the system which is required for performing any operation in the system. The storage device is one of the most requirement devices and also provide better compatibility.
   Example:
   Hard disk, magnetic tape, Flash memory etc

Advantage of Peripherals Devices:

- It is helpful for taking input very easily.
- It is also provided a specific output.
- It has a storage device for storing information or data
- It also improves the efficiency of the system.

Input-Output Interface (I/O interface.)

- Input-output interface provides a method for transferring information between internal storage and external I/0 devices.
- Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.
- The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral

The major differences are: ***(Why I/O interface are needed)***

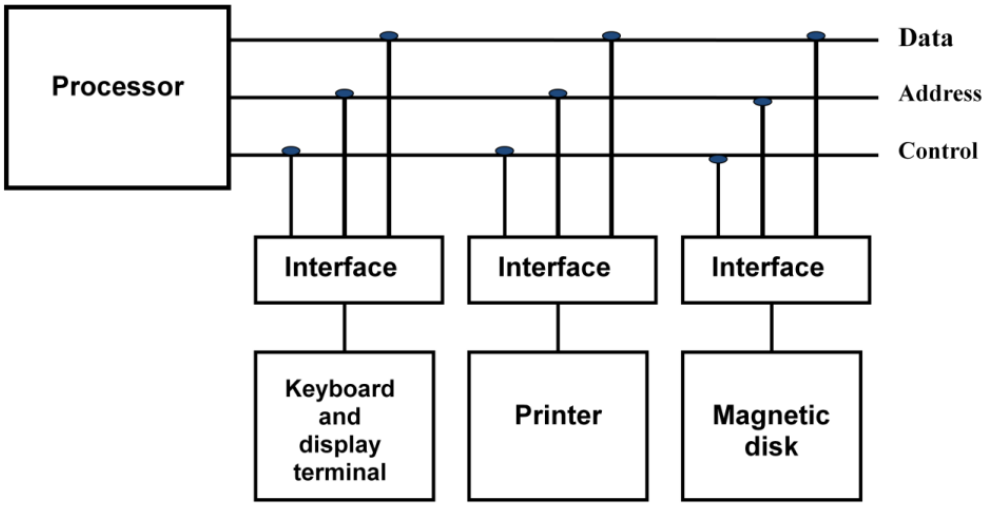
1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units.

I/O BUS and Interface Module

- The I/O bus consists of data lines, address lines, and control lines.
- The I/O bus from the processor is attached to all peripherals interface.
- To communicate with a particular device, the processor places a device address on address lines.
- Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.
- It is also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller.
  For example, the printer controller controls the paper motion, the print timing

- 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.



Connection of I/O bus to input-output devices

An interface receives any of the following four commands (I/O command.) –

1. **Control Command**: 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 Command**: A status command is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated.
3. **Output data :** A data output command creates the interface counter to the command by sending data from the bus to one of its registers.
4. **Input Data** − 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

I/O versus Memory Bus

- In addition to communicating with VO, the processor must communicate with the memory unit.
  There are three ways that computer buses can be used to communicate with memory and I/O:
  1. Separate set of address, control and data bus to I/O and memory.
     In This case it is simple because both have different set of address space and instruction but require more buses. This is done by using separate I/O Processor (IOP) in addition to CPU.
  2. Have common bus (data and address) for I/O and memory but separate control lines. (Isolated I/O)
  3. Have common bus (data, address, and control) for I/O and memory. (Memory Mapped I/O)

Isolated I/O

- It has common bus (data and address) for I/O and memory but separate read and write control lines for I/O
- when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O.
- The address for I/O here is called ports.

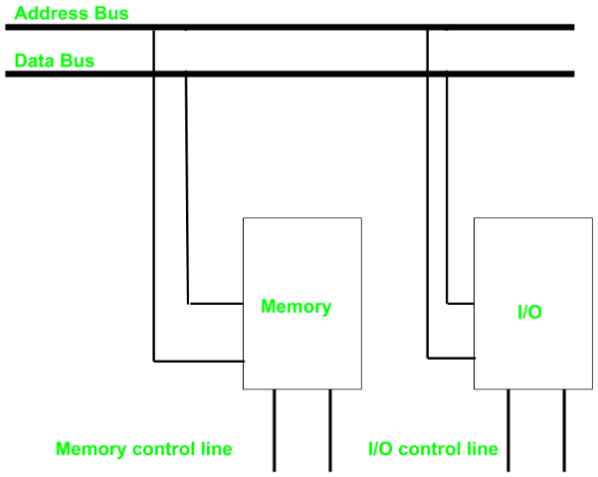


Figure: Isolated I/O

Memory Mapped I/O

- The isolated 110 method isolates memory and 110 addresses so that memory address values are not affected by interface address assignment since each has its own address space.
- In Memory mapped I/O , the CPU uses same address space for both memory and I/O.
- The computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses.
- The computer treats an interface register as being part of the memory system.
- The assigned addresses for interface registers cannot be used for memory words, which reduces the memory address range available.

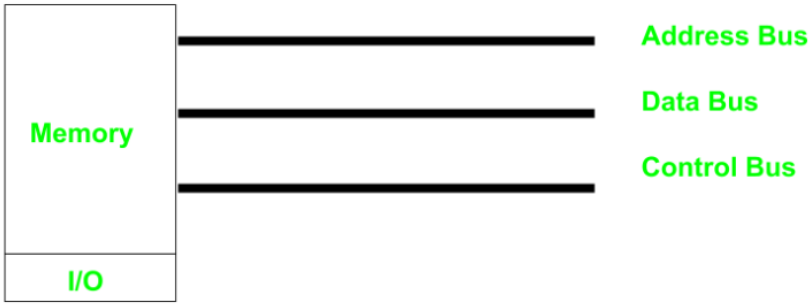
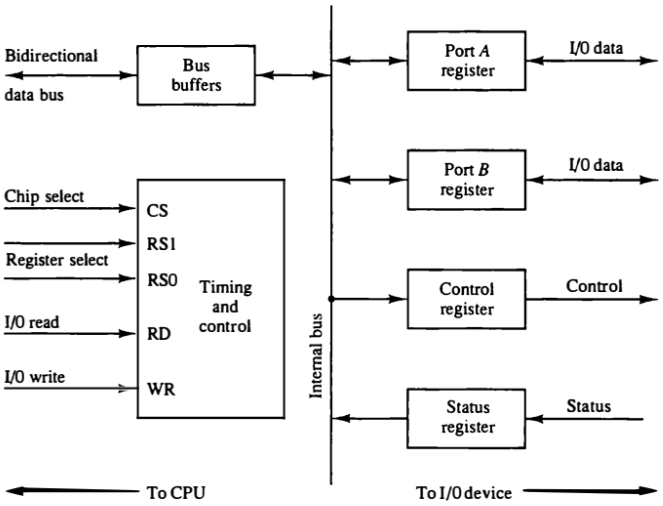


Figure: Memory Mapped I/O

Isolated I/O	Memory-mapped I/O
Different address spaces are used for computer memory and I/O devices. I/O devices have dedicated address space.	Same address space is used for memory and I/O devices.
Separate control unit and control instructions are used in case of I/O devices.	Control units and instructions are same for memory and I/O devices.
More complex and costlier than memory -mapped I/O as more bus are used.	Easier to build and cheap as it's less complex.
Entire address space can be used by memory as I/O devices have separate address space.	Some part of the address space of computer memory is consumed by I/O devices as address space is shared.
Computer memory and I/O devices use different control instructions for read write.	Computer memory and I/O devices can both use same set of read and write instructions.
Separate control bus is used for computer memory and I/O devices. Though same address and data bus are used.	Address, data and control bus are same for memory and I/O devices.

Example of I/O Interface



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

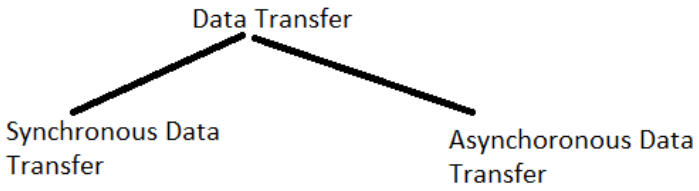
Example of I/O interface unit

- It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus.
- The chip select and register select inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output, respectively.
- The four registers communicate directly with the I/O device attached to the interface.
- The I/O data to and from the device can be transferred into either port A or port B
- The control register receives control information from the CPU
- The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer.

Data Transfer:

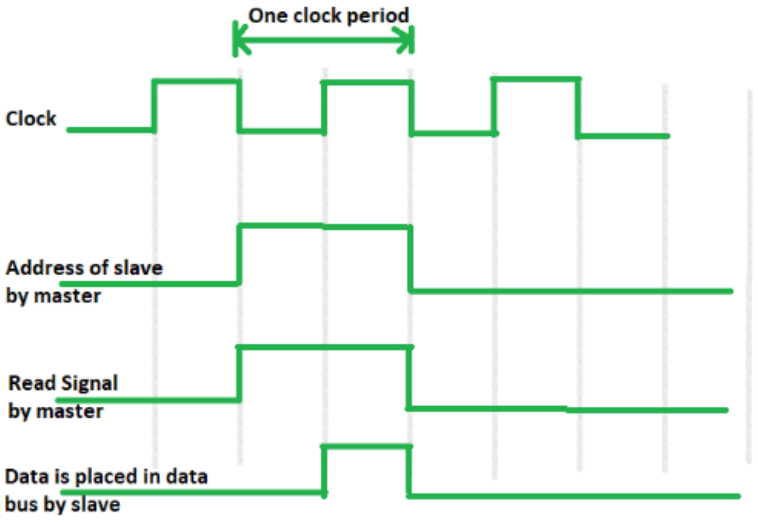
- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator. Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse.
- Two units, such as a CPU and an VO interface, are designed independently of each other.

Two type of Data Transfer takes place between peripherals:



Synchronous Data Transfer

- If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous.
  Example: In a Master-Slave Flip Flop
- The master is designed to supply the data at a time when the slave is definitely ready for it. Usually, the master will introduce sufficient delay to take into account the slow response of the slave, without any request from the slave.
- The master does not expect any acknowledgment signal from the slave when data is sent by the master to the slave



Timing diagram for Synchronous Read Operation

Advantages –

1. The design procedure is easy. The master does not wait for any acknowledges signal from the slave, though the master waits for a time equal to slave's response time.

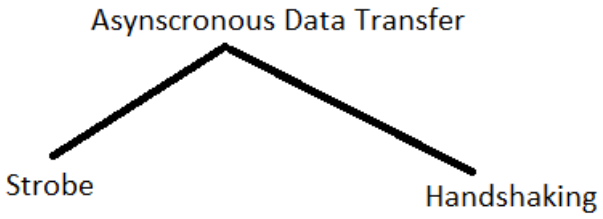
2. The slave does not generate an acknowledge signal, though it obeys the timing rules as per the protocol set by the master or system designer.

Disadvantages –

1. If a slow speed unit connected to a common bus, it can degrade the overall rate of transfer in the system.
2. If the slave operates at a slow speed, the master will be idle for some time during data transfer and vice versa

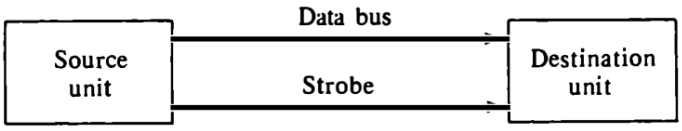
Asynchronous Data Transfer

- The internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be asynchronous to each other. This approach is widely used in most computer systems
  The Asynchronous data transfer is done by using two Method:
  1. Strobe
  2. Handshaking

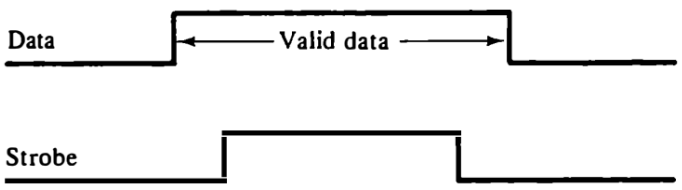


A. Strobe control

- The strobe control method of asynchronous data transfer employs a single control line to time each transfer.
- The strobe may be activated by either the source (source-initiated transfer) or the destination unit (Destination initiated transfer).
- In A source initiated transfer, the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse. The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- The source removes the data from the bus a brief period after it disables its strobe pulse.



(a) Block diagram



(b) Timing diagram

Figure Source-initiated strobe for data transfer.

- In Destination Initiated Strobe, data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it.
- The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval

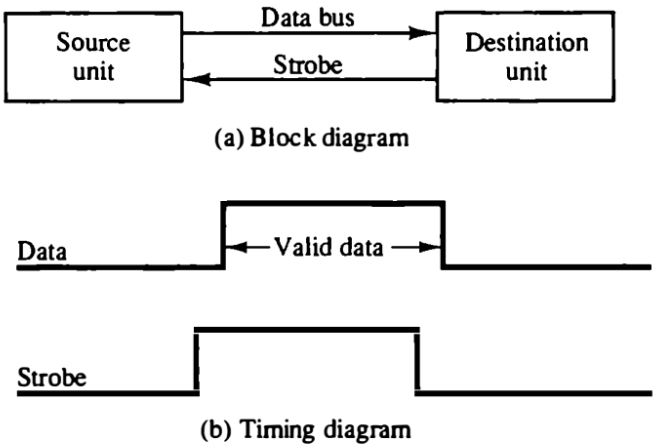


Figure Destination-initiated strobe for data transfer.

B. Handshaking Control

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus.
- Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.
- The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer.
- The basic principle of the two-wire handshaking method of data transfer is as follows.
  1. One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus.
  2. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data.
- The two handshaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit.

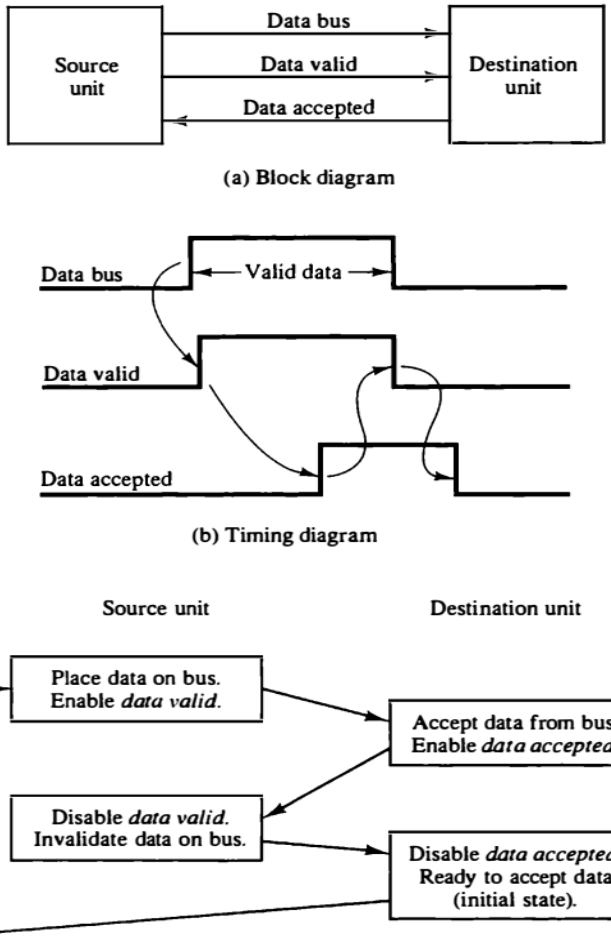


Figure Source-initiated transfer using handshaking.

- **In a Source initiated Transfer**, the source unit initiates the transfer by placing the data on the bus and enabling its data valid signal.
- The data accepted signal is activated by the destination unit after it accepts the data from the bus.
- The source unit then disables its data valid signal, which invalidates the data on the bus.
- The destination unit then disables its data accepted signal and the system goes into its initial state.
- **In a Destination initiated Transfer**, the source unit in this case does not place data on the bus until after it receives the **ready for data** signal from the destination unit.
- Here **Data accept line** of Source initiated Transfer is replaced by **ready for data signal**.

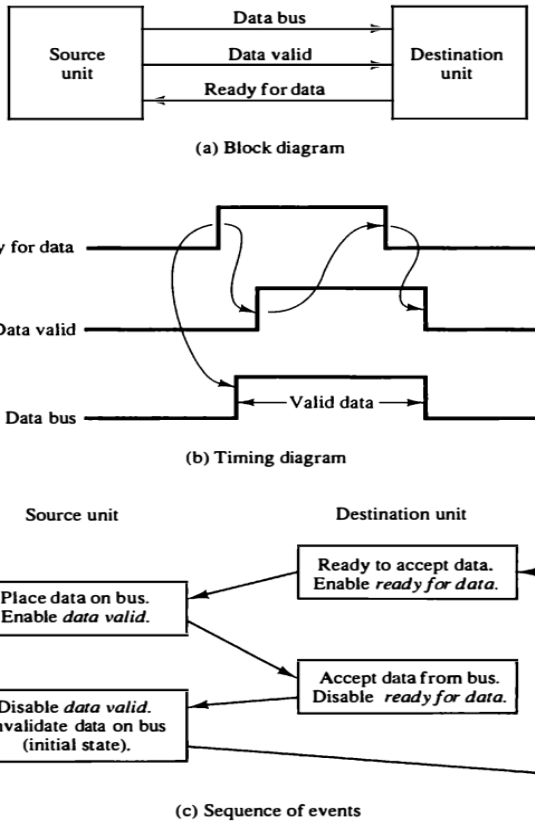
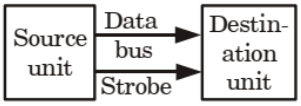
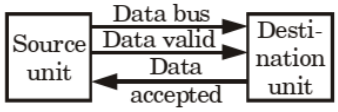
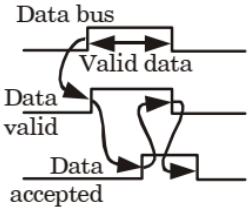


Figure Destination-initiated transfer using handshaking.

Advantage of the Handshaking method:

1. The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
2. If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a Timeout mechanism which provides an alarm if the data is not completed within time

S. No.	Parameter	Strobe control	Handshaking
1.	Control line	It employs a single control line to time each transfer.	It employs more than single control line to time each transfer.
2.	Acknowledgement	Reply message is not present.	Reply message is present.
3.	Block diagram		
4.	Timing diagram		

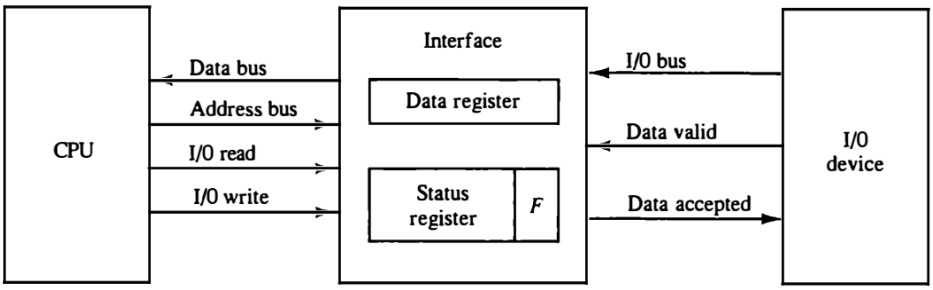
S. No.	Synchronous transmission	Asynchronous transmission
1.	Transmitter and receivers are synchronized by clock.	Transmitter and receivers are not synchronized by clock.
2.	Data bits are transmitted with synchronization of clock.	Bits of data are transmitted at constant rate.
3.	Data transfer takes place in blocks.	Data transfer is character oriented.

Modes of data Transfer (I/O Techniques)

- Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit.
- The CPU merely executes the instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit.
- Data transfer between the central computer and 110 devices may be handled in a variety of modes, these are:
    1. Programmed I/O
    2. Interrupt-initiated I/O
    3. Direct memory access (DMA)

Programmed I/O (Example of Programmed I/O)

- In the programmed I/O method, the I/O device does not have direct access to Memory.
- A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.
- The device transfers bytes of data one at a time which is placed on 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 that we will refer to as an F or "flag" bit
- 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. This is according to the handshaking Procedure.
- A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device.
- 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.
- Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte.



F = Flag bit

Figure Data transfer from I/O device to CPU.

- A flowchart of the program that must be written for the CPU is shown. It is assumed that the device is sending a sequence of bytes that must be stored in memory. 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 if set.
    3. Read the data register

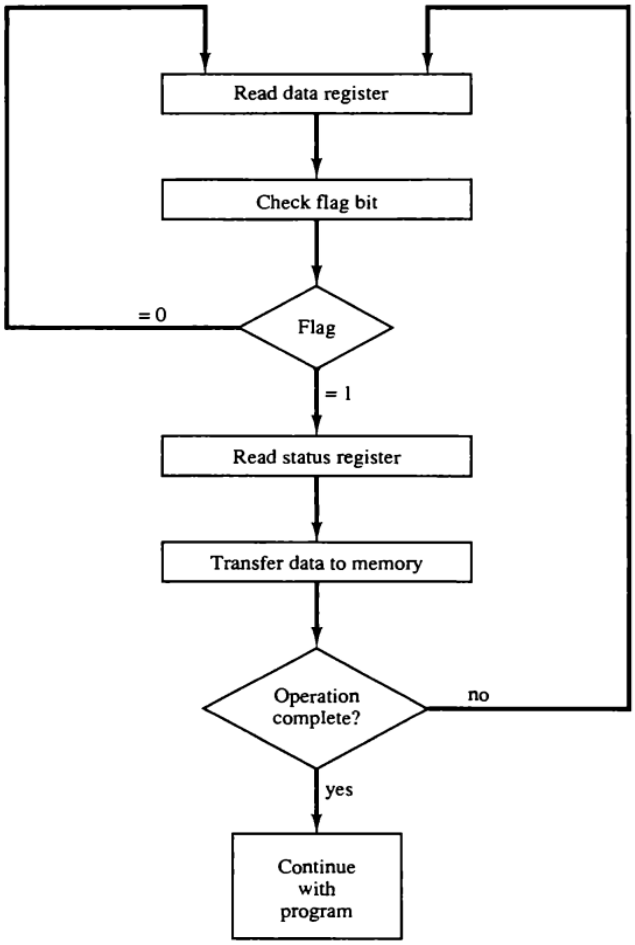


Figure Flowchart for CPU program to input data.

- The programmed VO method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously.
- The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

Interrupt-Initiated I/O

- An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility.
- While the CPU is running a program, it does not check the flag. However, when the flag is set, The CPU momentarily interrupted from proceeding with the current program.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.
- There are two ways of choosing the branch address:
    1. Vectored Interrupt
    2. Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

S. No.	Vectored interrupt	Non-vectored interrupt
1.	Vectored interrupt are those interrupt that generates the interrupt request, identifies itself directly to the processor.	Non-vectored interrupt are those in which vector address is not pre-defined.
2.	Vector interrupt have fixed memory location for transfer of control for normal execution.	Non-vectored interrupt do not have fixed memory location for transfer of control for normal execution.
3.	Vectored interrupt has memory address.	A non-vectored interrupt do not have memory address.
4.	The vectored interrupt allows the CPU to be able to know what ISR to carry out in software.	When a non-vectored interrupt received, it jump into the program counter to fixed address in hardware.
5.	Response time is low.	Response time is high.
6.	TRAP is a vectored interrupt.	INTR is non-vectored interrupt.

Programmed I/O	Interrupt Initiated I/O
Data transfer is initiated by the means of instructions stored in the computer program. Whenever there is a request for I/O transfer the instructions are executed from the program.	The I/O transfer is initiated by the interrupt command issued to the CPU.

Programmed I/O	Interrupt Initiated I/O
The CPU stays in the loop to know if the device is ready for transfer and has to continuously monitor the peripheral device.	There is no need for the CPU to stay in the loop as the interrupt command interrupts the CPU when the device is ready for data transfer.
This leads to the wastage of CPU cycles as CPU remains busy needlessly and thus the efficiency of system gets reduced.	The CPU cycles are not wasted as CPU continues with other work during this time and hence this method is more efficient.
CPU cannot do any work until the transfer is complete as it has to stay in the loop to continuously monitor the peripheral device.	CPU can do any other work until it is interrupted by the command indicating the readiness of device for data transfer
Its module is treated as a slow module.	Its module is faster than programmed I/O module.
It is quite easy to program and understand.	It can be tricky and complicated to understand if one uses low level language.
The performance of the system is severely degraded.	The performance of the system is enhanced to some extent.

Priority Interrupt

- There are number of I/O devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
  1. Using Software : by Polling Procedure
  2. Using Hardware : by Daisy-Chaining Priority and Parallel Priority Interrupt
- A pooling procedure is used to identify highest priority in software means.

Polling

- A polling procedure is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts.

- The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt.
- The highest-priority source is tested first, and if its interrupt signal is ON, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on.
- The **disadvantage** of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation

Hardware Procedure

- Hardware priority system function as an overall manager.
- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.
- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial (Daisy-Chaining Priority) or parallel (Parallel Priority Interrupt) connection of interrupt lines.

Daisy-Chaining Priority

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU.
- CPU then sends the INTACK signal which is applied to PI (priority in) of the first device.
- If it had requested the attention, it place its VAD (vector address) on the bus. And it block the signal by placing 0 in PO (priority out)
- If not it pass the signal to next device through PO (priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

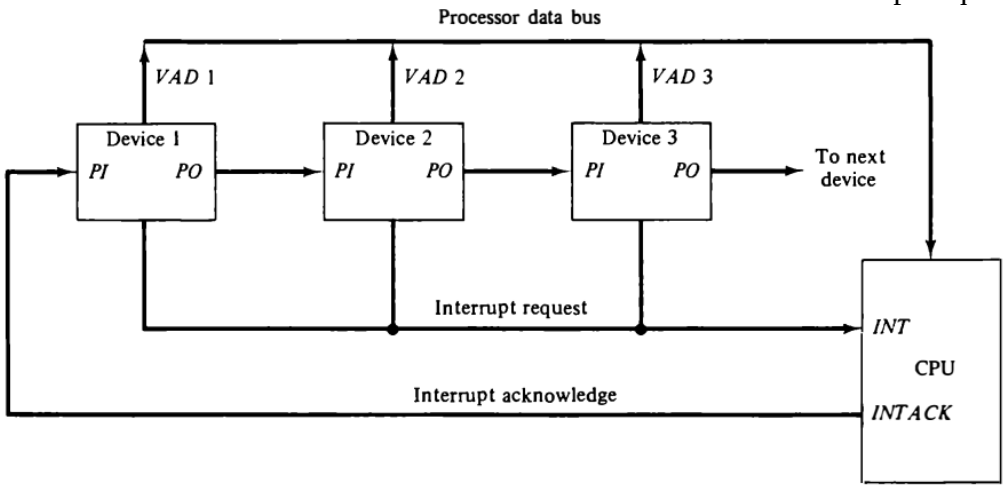
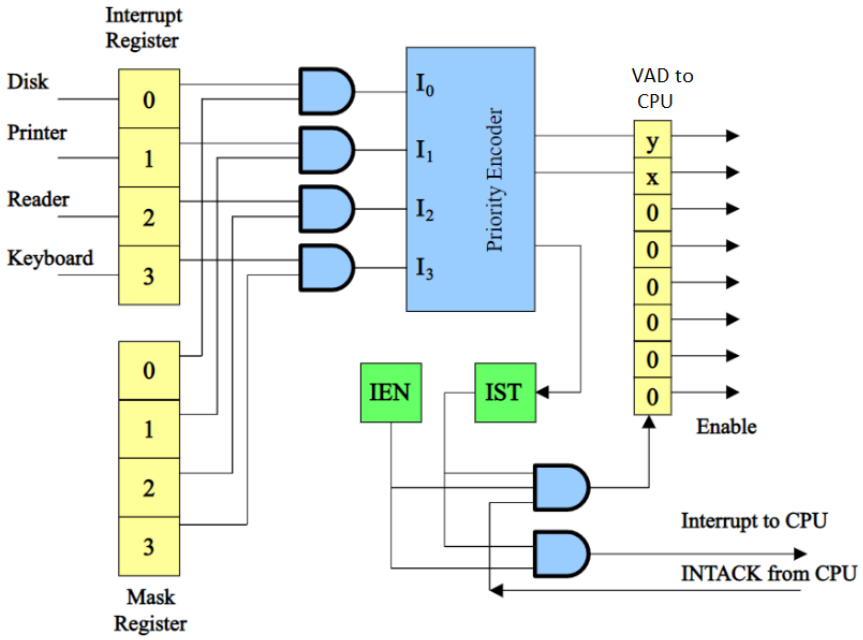


Figure Daisy-chain priority interrupt.

Parallel Priority Interrupt

- It consist of interrupt register whose bits are set separately by the interrupting devices.

- Priority is established according to the position of the bits in the register
- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.
- Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.
- Priority encoder generates two bits of vector address.
- Another output from it sets IST(interrupt status flip flop)



Parallel Priority Interrupt hardware

Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I_0' I_1'$
0	0	1	×	1	0	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

Direct Memory Access (DMA)

- The Transfer of Data between a fast storage devices is limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA).
- During DMA transfer, the CPU is idle and has no control of the memory buses.
- A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

- The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:
  A. **Bus Request:** The Bus Request (BR) input is used by the DMA controller to request the CPU to stop having control of bus. When this input is active, the CPU terminates the execution of the current instruction and place the address bus, the DATA bus and the read and write line into high impedance state.
  B. **Bus Grant**: The CPU activates the bus Grant (BG) output to inform the external DMA that the buses are in high impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention.
- These two control signals in the CPU that facilitates the DMA transfer.
- When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a high Impedance state. High Impedance state means that the output is disconnected.
- When DMA terminates the transfer, it disables the bus request line. The CPU disables the Bus Grant (BG), takes control of the bus, and returns to its normal operation.

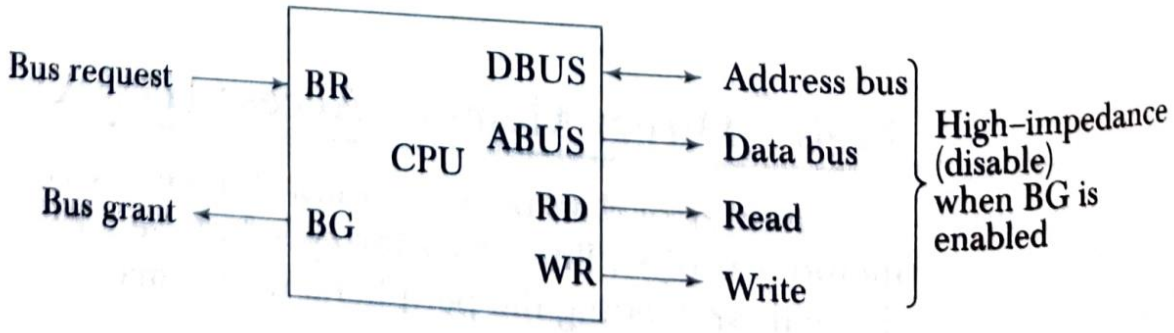


Figure: CPU bus signal for BUS Transfer

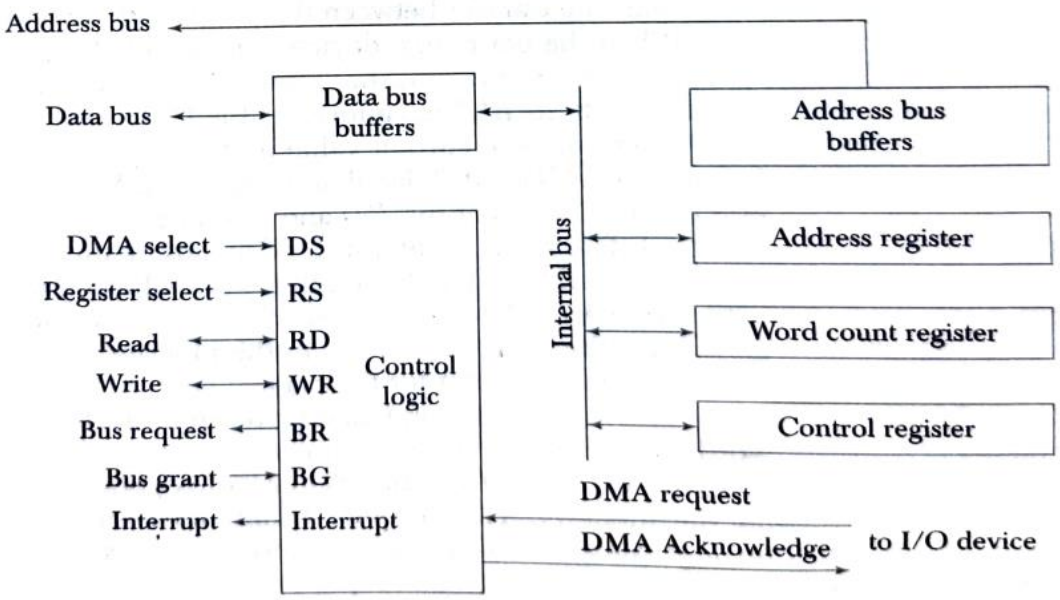
- The transfer of data from memory can be made in several ways:
  1. Burst Mode
  2. Cycle Stealing Mode

Burst Mode: In this mode, once the DMA controller gains the charge of the system bus, then it releases the system bus only after completion of data transfer. Till then the CPU has to wait for the system buses.

Cycle Stealing Mode: In this mode, the DMA controller forces the CPU to stop its operation and transfer the control over the bus for a short term to DMA controller. After the transfer of every byte, the DMA controller releases the bus and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte

DMA Controller

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it need address register, a word count register and a set of address lines.
- The Address register and address lines are used for direct communication with the memory.
- The data transfer may be done directly between the device and memory under control of the DMA.
- The DMA unit communicates with the CPU via the data bus and control lines.
- The DMA controller has three registers:
    1. Address Register
    2. Word Count Register
    3. Control Register
- Address Register: - Address Register contains an address to specify the desired location in memory.
- Word Count Register: - WC holds the number of words to be transferred. The register is increase/decrease by one after each word transfer and internally tested for zero
- Control Register :- Control Register specifies the mode of transfer
- The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.
- When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.
- When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



Block diagram of DMA controller.

DMA Transfer

- The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus.
- Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory
- When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to stop control on the buses.
- The CPU responds with its BG line, informing the DMA that its buses are disabled.
- The DMA then puts the current value of its address register into the address bus, Initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device.
- The direction of transfer depends on the status of the BG line.
- When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers.
- When BG= 1, the RD and WR are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.
- For each word that is transferred, the DMA increments its address register and decrements its word count register.
- If the word count register reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of interrupt.

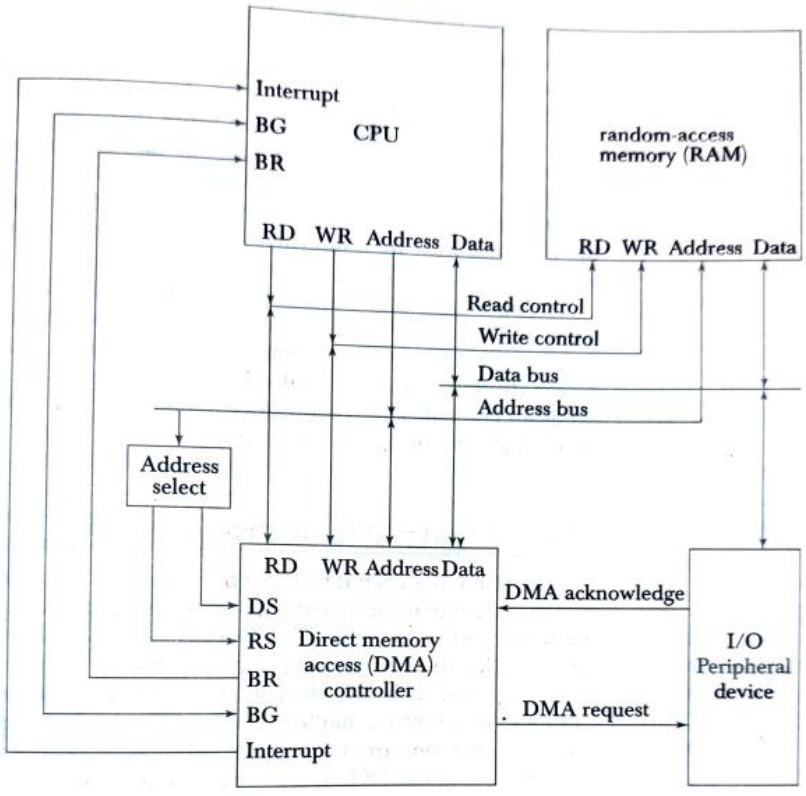


Figure DMA transfer in a computer system.

Input-Output Processor (IOP)

- Instead of having each interface communicate with the CPU, a computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices.
- An input-output processor (IOP) may be classified as a processor with direct memory access capability that communicates with I/O devices.
- The IOP is similar to a CPU except that it is designed to handle the details of I/O processing.
- Unlike the DMA controller that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions, which is specifically designed for I/O transfer.
- The block diagram indicated two processor (CPU and IOP), which can communicate with Memory unit directly.
- The CPU is responsible for processing data needed in the solution of computational tasks.
- The IOP provides the path for transfer of data between various peripherals devices and memory unit.
- The data formats of peripheral devices differ from memory and CPU data formats.
- After the input data are assembled into a memory word, they are transferred from lOP directly into memory by "stealing" one memory cycle from the CPU.
- Instructions that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU.

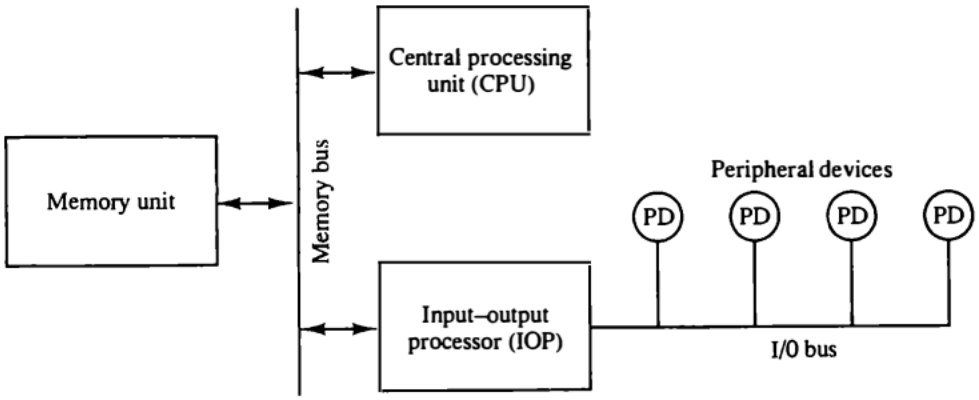


Figure Block diagram of a computer with I/O processor.

CPU-IOP Communication

- In most cases the memory unit acts as a message centre where each processor leaves information for the other.
- The sequence of operations may be carried out as shown in the flowchart for the communication between CPU and IOP.

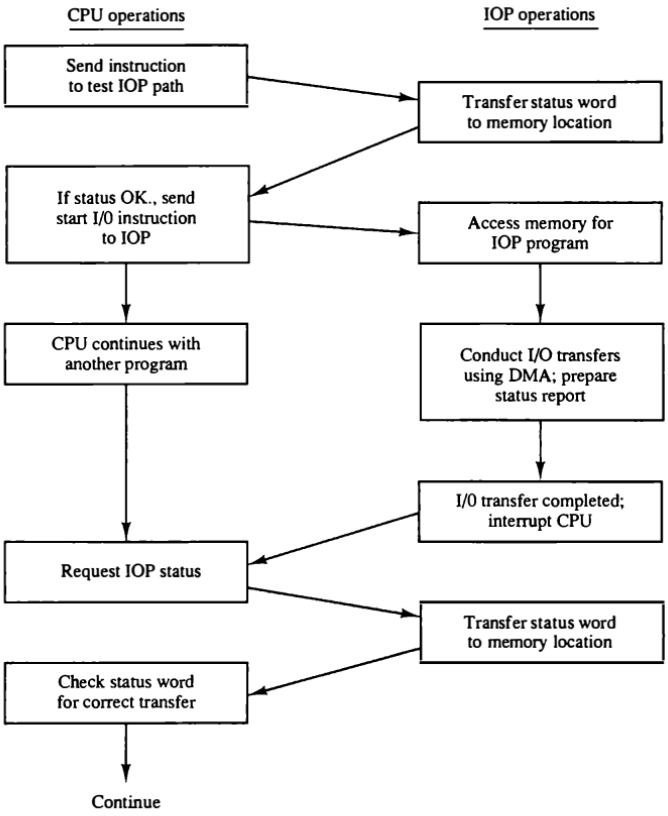


Figure CPU-IOP communication.

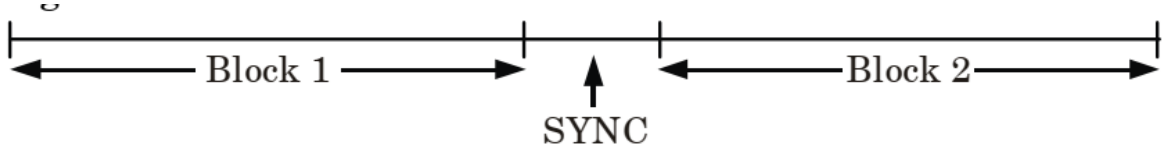
- The CPU sends an instruction to test the IOP path.
- The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer.
- The CPU refers to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer.
- The memory address received with this instruction tells the IOP where to find its program
- The CPU can now continue with another program while the IOP is busy with the I/O program.
- When the IOP terminates the execution of its program, it sends an interrupt request to the CPU.
- The CPU responds to the interrupt by issuing an instruction to read the status from the IOP.
- The IOP responds by placing the contents of its status report into a specified memory location.

Serial Communication

- Serial communication is the process of sequentially transferring the information/bits on the same channel. Due to this, the cost of wire will be reduced, but it slows the transmission speed.
- In serial communication, binary pulses are used to show the data
- The serial communication can either be asynchronous or synchronous.

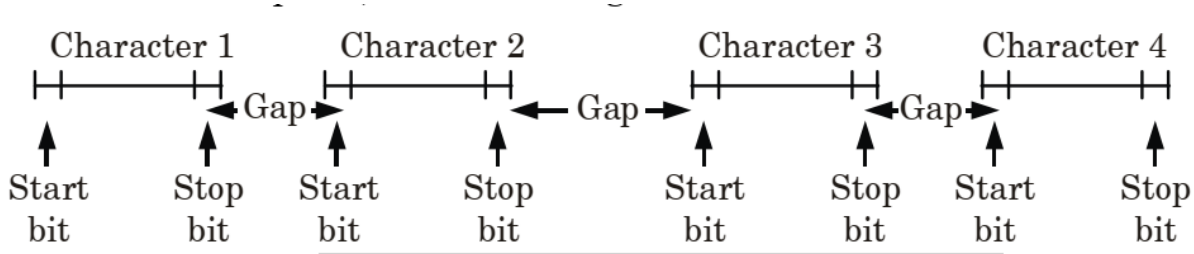
Synchronous Communication

In the synchronous communication scheme, after a fixed number of data bytes, a special bit pattern called SYNC is sent as shown Figure. There is no gap between adjacent characters in the synchronous communication. There is a continuous stream of data bits coming at a fixed speed in a synchronous communication scheme. Synchronous communication is used generally when two computers are communicating to each other or when a buffered terminal is communicating to the computer



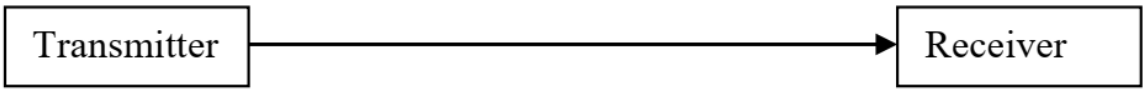
Asynchronous Serial data transmission:

In the asynchronous communication scheme, each character includes start and stop bits, as shown in Figure. There are some gaps between adjacent characters in the asynchronous communication. In the asynchronous communication scheme, the bits within a character frame (including start, parity and stop bits) are sent at the baud rate. Asynchronous communication is used when slow speed peripherals communicate with the computer.

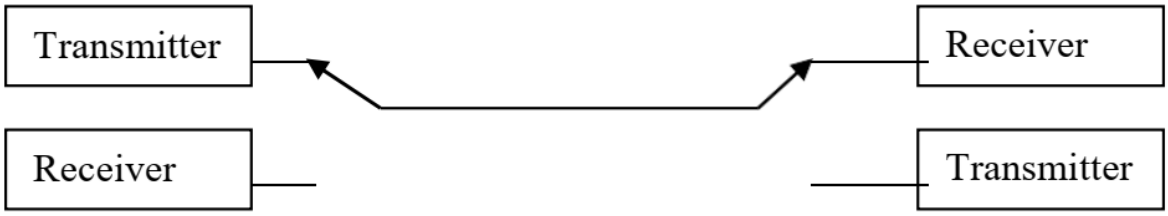


The Serial transmission modes are described as follows:

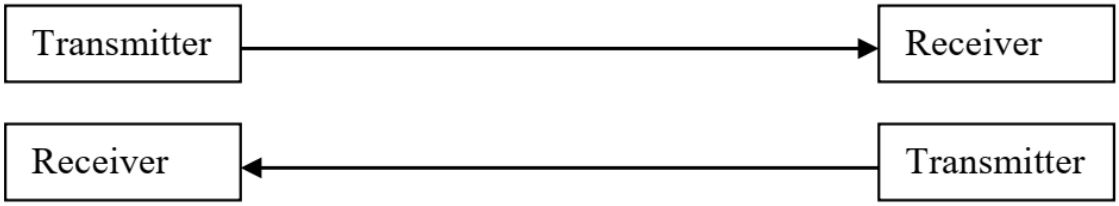
1. Simplex: In the simplex method, the data transmission can be performed only in one direction. A sender can only transmit the data, and the receiver can only accept that data. The receiver cannot reply back to the sender



2. **Half duplex communication link**: In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time.



3. **Full duplex communication link:** If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception



Interrupt

An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. It alerts the processor to a high priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the ***Interrupt Service Routine*** *(ISR).*

An interrupt in computer architecture is a signal that requests the processor to suspend its current execution and service the occurred interrupt. To service the interrupt the processor executes the corresponding interrupt service routine (ISR). After the execution of the interrupt service routine, the processor resumes the execution of the suspended program.

TYPES OF INTERRUPTS

There are many type of interrupts but basic type of interrupts are –

1. Hardware and software interrupts
2. Vectored and Non- vectored Interrupts
3. Mask able and non-maskable interrupts

Hardware and software interrupts

1. Hardware Interrupts: If the signal for the processor is generated from external device or hardware is called hardware interrupts. There are 5 Hardware Interrupts in processor. They are

1. INTR,
2. RST 7.5,
3. RST 6.5,
4. RST 5.5,
5. TRAP

1.TRAP(RST 4.5):-

- ❑ Non maskable

- Edge and level triggered .
- Edge and level triggered means that the TRAP must go high and remain high until it is acknowledged
- Highest priority
- vectored interrupt.
- In the case of sudden power failure, it executes a ISR (Interrupt service routine) and send the data from main memory to backup memory.

2.RST 7.5:-

- Second highest priority.
- Maskable
- edge triggered interrupt.
- Edge sensitive means input goes high and no need to maintain high state until it is recognized.

3.RST6.5 and RST5.5:-

- level triggered
- maskable interrupts.
- RST 6.5 has third highest priority and RST 5.5 has fourth highest priority.

4.INTR:-

- Level triggered
- Maskable interrupt.
- Lowest priority.

2. Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor.

- There are 8 software interrupts. They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

Software interrupt can also divided in to two types:

- **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.

- **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

Example: Division by zero, execution of an illegal opcode or memory related fault could cause exceptions.

Mask able and non-maskable interrupts

- **Maskable Interrupt:** The hardware interrupt that can be ignored or delayed for some time if the processor is executing a program with higher priority are termed

as maskable interrupts**.** These interrupts are either edge-triggered or level-triggered, so they can be disabled.

INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts.

- **Non Maskable Interrupt:** The hardware interrupts that can neither be ignored nor delayed and must immediately be serviced by the processor are termed as non-maskable interrupts**.** It consists of both level as well as edge triggering and is used in critical power failure conditions.

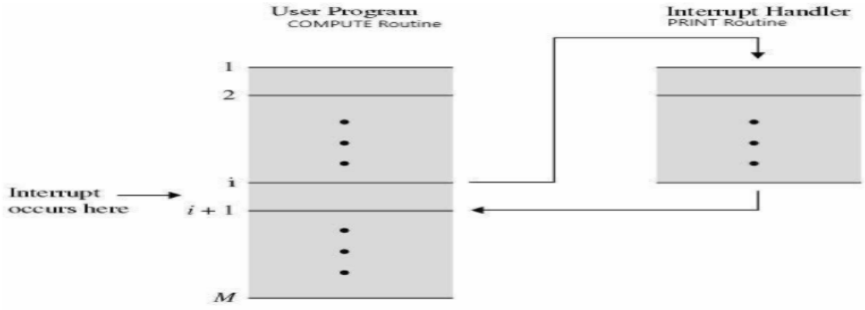
  TRAP is a non-maskable interrupt.

Vectored and Non- vectored Interrupts

- **Vectored Interrupts** are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address. When an interrupt is occurred and program control automatically branches the program execution to a specific address
- **Non-Vectored Interrupts** (Scalar Interrupt) are those in which vector address is not predefined. Interrupts that have a variable address. When an interrupt device have to provide an address from where the execution of program will begin. **INTR** is the only non-vectored interrupt

INTERRUPT HANDLING MECHANISM AND INTERRUPT SERVICE ROUTINE (ISR):

- The job of the interrupt handler in Interrupt handling is to service the device and stop it from interrupting. Once the handler returns, the CPU resumes what it was doing before the interrupt occurred.
- The routine that gets executed when an interrupt request is made is called as interrupt service routine.

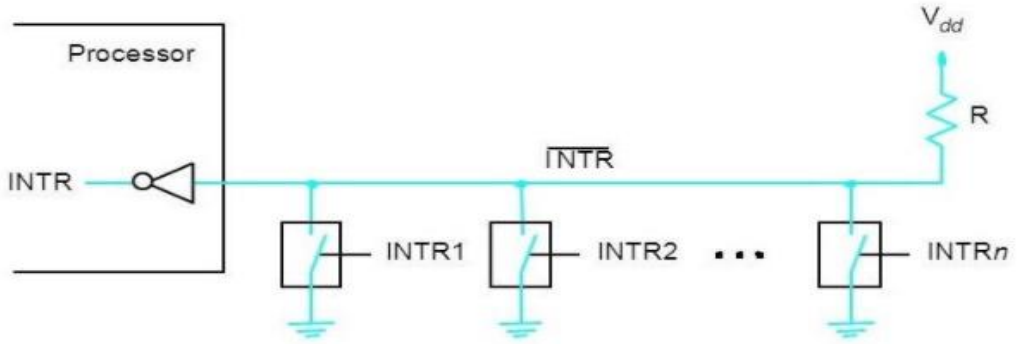


- Step 1: When the interrupt occurs the processor is currently executing i'th instruction and the program counter will be currently pointing to (i + 1)th instruction.
- Step 2: When the interrupt occurs the program counter value is stored on the processes stack.

- Step 3: The program counter is now loaded with the address of interrupt service routine.
- Step 4: Once the interrupt service routine is completed the address on the processes stack is pop and place back in the program counter.
- Step 5: Execution resumes from (i + 1)th line of compute routine

INTERRUPT HARDWARE

Many computers have facility to connect two or more input and output devices to it like laptop may have 3 USB slots. All these input and output devices are connected via switches as shown -



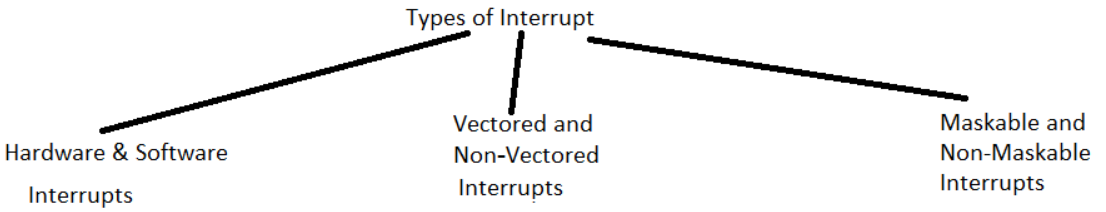
So there is a common interrupt line for all N input/output devices and the interrupt handling works in the following manner ->

1. When no interrupt is issued by the input/output devices then all the switches are open and the entire voltage from **Vdd** is flown through the single line **INTR** and reaches the processor. Which means the processor gets a voltage of **1V.**

2. When the interrupt is issued by the input/output devices then the switch associated with the input/output device is closed, so the entire current now passes via the switches which means the hardware line reaching the processes i.e **INTR** line gets **0 voltage.** This is an indication for the processor that an interrupt has occurred and the processor needs to identify which input/output device has triggered the interrupt

3. The value of **INTR** is a **logical NOT** of the requests from individual devices.

4. The **resistor R** is called as a pull up resistor because it pulls the line voltage to high voltage state when all switches are open(no interrupt state).



Hardware	Software	Vectored	Non-Vectored	Maskable	Non-Maskable
INTR, RST 7.5, RST 6.5, RST 5.5, TRAP	RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.	TRAP RST	Only INTR	Remaing all	only TRAP