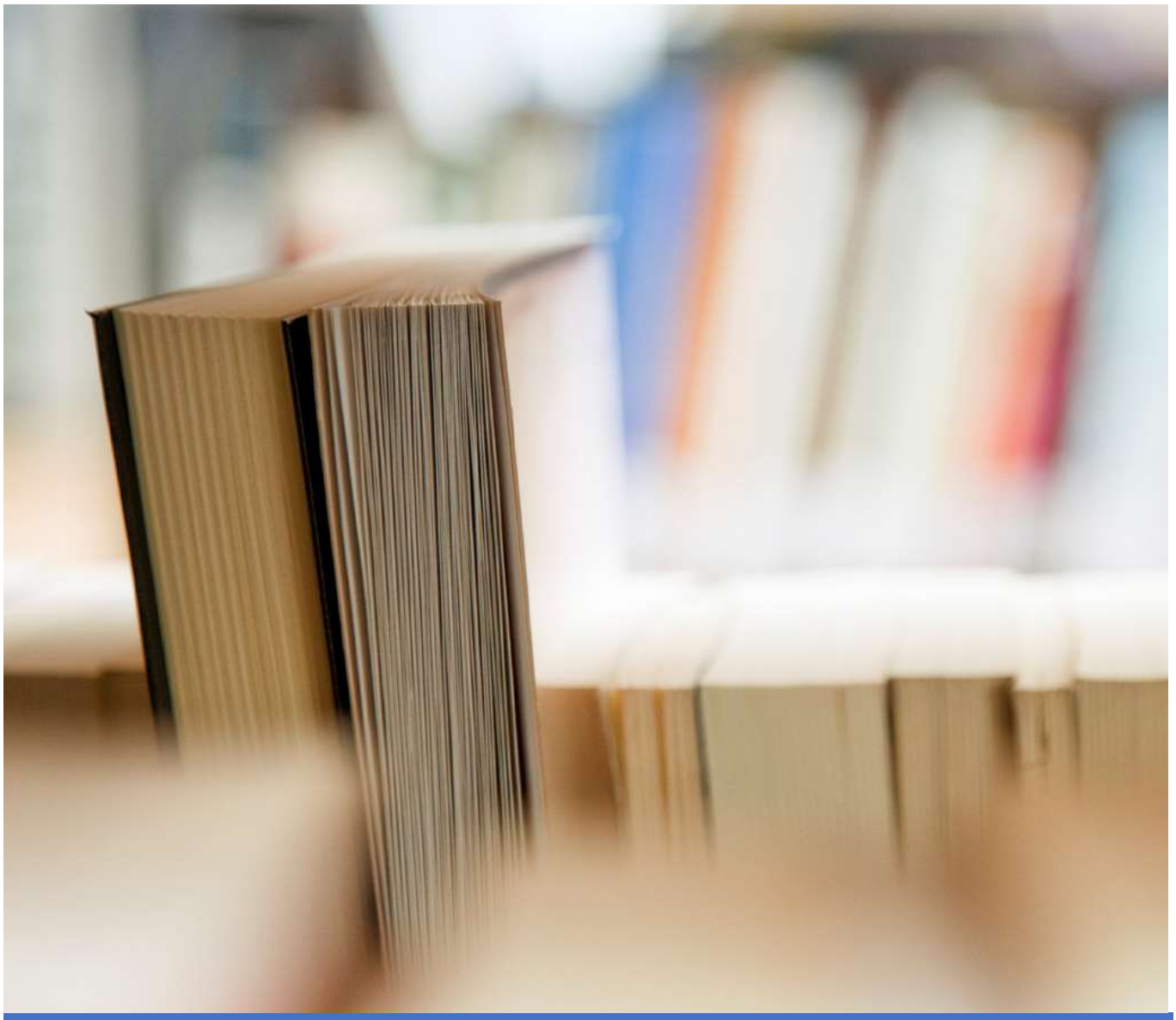


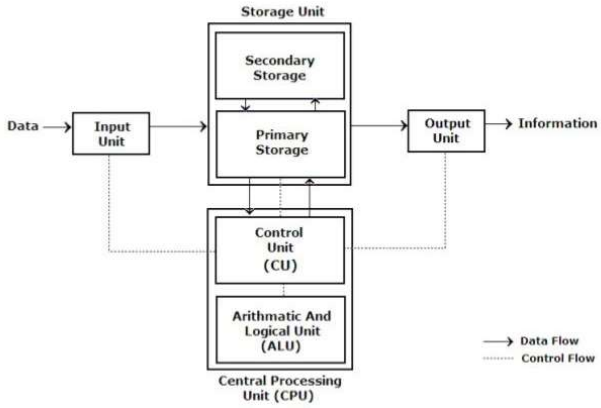
Computer organization and Architecture

UNIT -0

Computer

- computer is a programmable machine processing data through hardware and software, using binary code and interacting via input, output, and networks.

Block diagram of computer



Input Devices:

- It accepts instructions and data from outside of the world
- Keyboard,Mouse,touchpad, touchscreen

CPU (Central Processing Unit):

- Executes instructions and manages operations.
- It has two parts : 1.control unit 2. Arithmetic logic Unit
- **Control Unit**: Manages the execution of instructions, controls data flow, and coordinates internal operations within a computer's central processing unit (CPU).
- **Arithmetic Logic Unit (ALU)**: Performs arithmetic and logic operations on binary data within a computer's central processing unit (CPU).

Storage UNIT

- Stores data and instructions temporarily (RAM) or permanently (storage devices).
- It is also divided into two parts
- **Primary Memory**:
  - Temporary and volatile storage for actively used data and instructions.
  - RAM (Random Access Memory),ROM (Read-Only Memory),Cache memory
- **Secondary Memory:**
  - Permanent and non-volatile storage for long-term data retention.
  - Example : Hard Disk Drive (HDD),Solid State Drive (SSD) ,Optical drives (e.g., CD/DVD drives)

Output Devices:

- Present or convey processed information from a computer system to the user, such as through displays, printers, or speakers.

Computer organization and architecture

It provides in-depth knowledge of internal working, structuring, and implementation of a computer system.

Computer Architecture:

- Concerned with how hardware components are connected to form a computer system.
- Acts as the interface between hardware and software.
- Helps understand the functionalities of a system.
- Viewed by programmers in terms of instructions, addressing modes, and registers.
- Considered first in the design of a computer system.
- Deals with high-level design issues.
- Involves logic, including instruction sets, addressing modes, data types, and cache optimization.

Computer Organization:

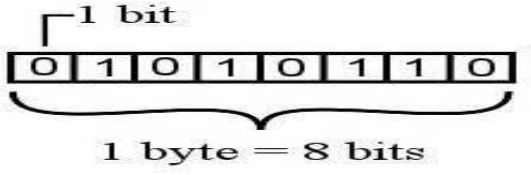
- Concerned with the structure and behavior of a computer system as seen by the user.
- Deals with the components and their connections in a system.
- Tells how all units in the system are arranged and interconnected.
- Expresses the realization of architecture.
- Done on the basis of architecture.
- Deals with low-level design issues.
- Involves physical components, including circuit design, adders, signals, and peripherals.

Binary numbers

- It is a base-2 numerical system using only 0s and 1s.

Bytes VS Bits

- One byte has 8 bits (2^3), providing 256 possible combinations (2^8).

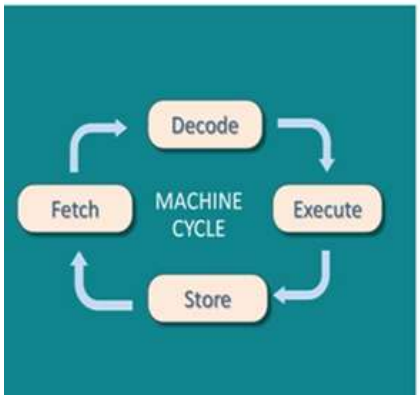


Register

A register is a small, fast storage location within the CPU used for quick data access and manipulation during program execution.

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

How Computer Works:



Fetch:

- The Program Counter (PC) holds the address of the next instruction.
- The Control Unit fetches the instruction from the memory location pointed to by the PC.
- The fetched instruction is stored in the Instruction Register (IR).

Decode:

- The Control Unit interprets the instruction in the IR, determining the operation to be performed.

Execute:

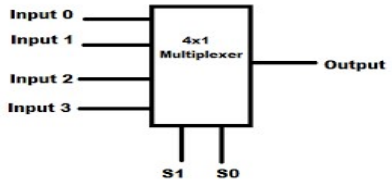
- The Control Unit coordinates with the ALU and other components to execute the instruction.
- Data may be fetched from memory into registers, processed by the ALU,

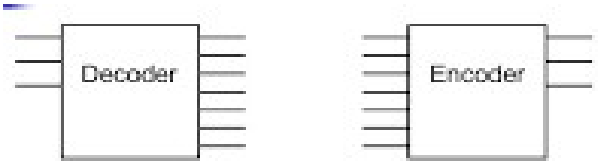
Store

- result stored back in memory or registers.

Encoder VS Decoder VS Multiplexer

- Encoder Converts digital data to an analog signal.
- Decoder Converts an analog signal to digital data.
- Multiplexer selects one from many.





OPR selector in Instruction

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	AND *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

Logic gates

Logic gates are fundamental building blocks of digital circuits, and they perform logical operations on one or more binary inputs to produce a binary output. Here are some common types of logic gates:

AND Gate:

Output is true (1) only if all inputs are true (1).

Symbol: ∧

Truth Table:

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

OR Gate:

Output is true (1) if at least one input is true (1).

Symbol: ∨

Truth Table:

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

NOT Gate:

Output is the opposite (complement) of the input.

Symbol: ¬ or $\overline{A}$

Truth Table:

A	Output
0	1
1	0

NAND Gate:

Output is false (0) only if all inputs are true (1).

Symbol: A $\overline{\wedge}$ B or ¬(A ∧ B)

Truth Table:

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

NOR Gate:

Output is false (0) if at least one input is true (1).

Symbol: A $\overline{\vee}$ B or ¬(A ∨ B)

Truth Table:

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

XOR Gate (Exclusive OR):

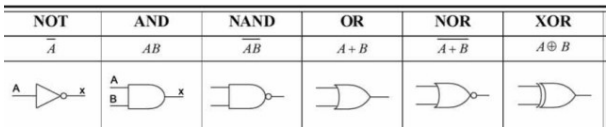
Output is true (1) if inputs are different.

Symbol: A ⊕ B or (A ∨ B) ∧ ¬(A ∧ B)

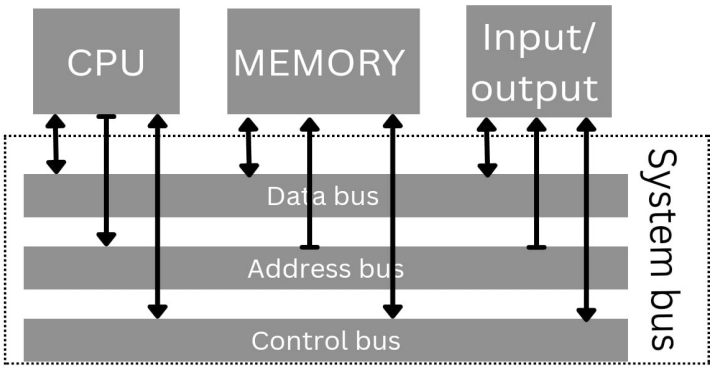
Truth Table:

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

Diagram of the all logic gates



System Bus



- System Bus Facilitates data and control signal transfer among computer components, serving as a communication highway for information exchange.
- The system bus consists of three main types of buses:

1. Address Bus

- **Purpose**: Carries memory addresses, with each signal combination representing a unique memory location.
- **Size**: Defines the maximum addressable memory; e.g., a 32-bit address bus can access 2^32 (4 gigabytes) of memory.
- **Direction**: Unidirectional, from CPU to memory or I/O devices.

2. Data Bus

- **Purpose**: Transmits data between the CPU, memory, and peripherals, carrying both instructions and processed data.
- **Size**: Specifies the parallel data transmission capacity; common sizes include 8-bit, 16-bit, 32-bit, or 64-bit data buses.
- **Direction**: Bidirectional, allowing data transfer in both directions.

3. Control Bus

- **Purpose**: Transmits control signals for coordinating and managing activities among connected components, encompassing read/write signals, interrupt requests, and clock signals.
- **Signals**: Includes common signals like Read (RD), Write (WR), Clock (CLK), Interrupt Request (IRQ), and others.
- **Direction**: Bidirectional, though some signals may be unidirectional.

Single Shared Bus

- All components (CPU, memory, peripherals) share a single communication pathway (bus).
- **Advantage**: Simplicity in design and lower cost.
- **Disadvantage**: Potential for congestion and slower performance as all data must use the same pathway.

Multiple Shared Buses

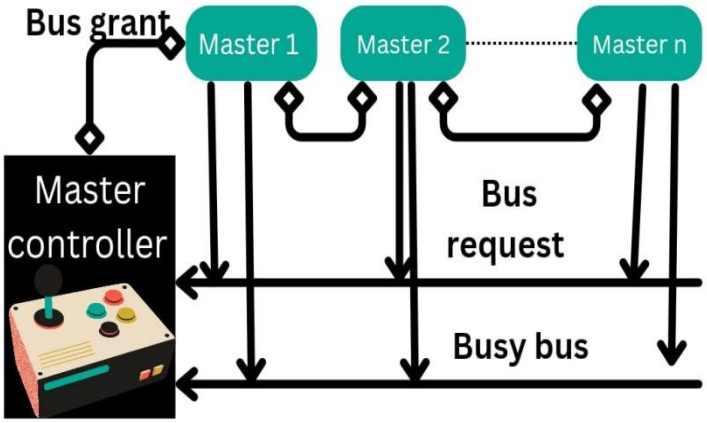
- Different components use separate communication pathways (buses).
- **Advantage**: Reduced congestion, faster communication between specific components.
- **Disadvantage**: Increased complexity in design and higher cost due to multiple buses.

Bus arbitration

- Bus arbitration is a mechanism which decides the selection of current master to access bus.
- Multiple master or slave units connected to a shared bus may concurrently request access to the bus.
- In such situation, bus access is given to the master having highest priority.
- Three different mechanisms are commonly used for this:

1. Daisy chaining method

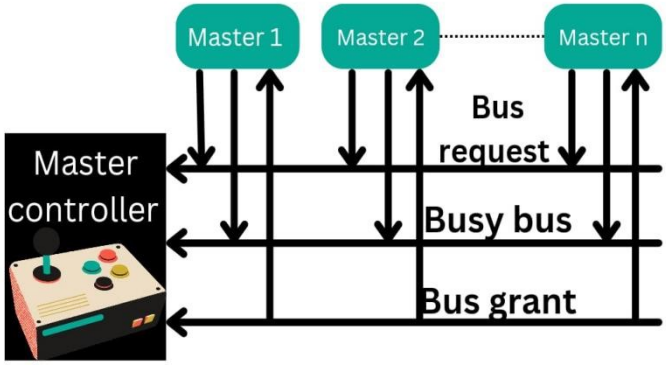
- all masters make use of the same line for bus request.
- The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus.
- This master blocks the propagation of the bus grant signal, activates the busy line and gains control of the bus.
- Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access.
- During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.



- **Avantages**
  - It is a simple and cheaper method.
  - It requires the least number of lines and this number is independent of the number of masters in the system.
- **Disadvantages**
  - Propagation delay arises in this method.
  - If one device fails then the entire system will stop working.

2. Polling method

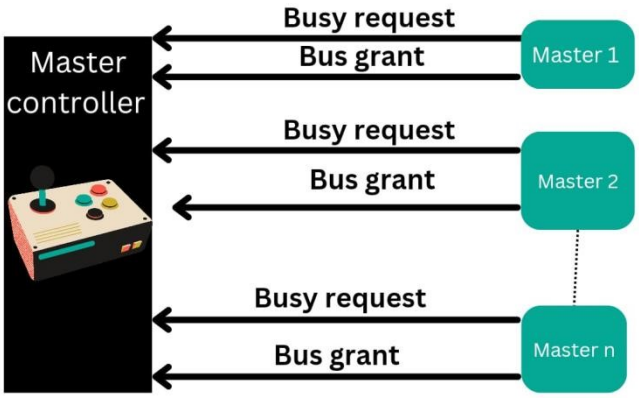
- The polling method involves sequentially checking each master to determine which one is ready to access the bus.



- **Advantages :**
  - If the one module fails entire system does not fail.
  - The priority can be changed by altering the polling sequence stored in the controller.
- **Disadvantages :**
  - It requires more bus request and grant signals (2 x n signals for n modules).
  - Polling overhead can consume a lot of CPU time.

3. Independent Request or Fixed Priority Method -

- A unique pair of bus requests and bus grant lines are provided to each master, and each pair is given a priority.
- The controller's built-in priority decoder chooses the utmost priority request and then asserts the matching bus grant signal.
- **Advantage**:
  - This technique produces a quick response.
- **Disadvantage**:
  - A significant number of control lines are needed, which raises the cost of the hardware.



Memory transfer

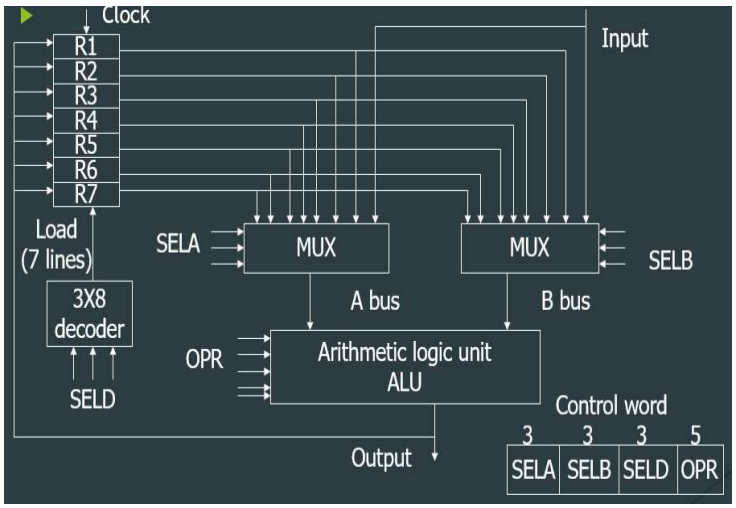
- Memory transfer involves two main operations: reading (fetching) and writing (storing) data.
- Reading transfers a copy of data from a memory location to the CPU.
- Writing transfers information from the CPU to a specific memory location, replacing the previous content.
- Two important registers are used for memory transfer:
- the Address Register (AR) and the Data Register (DR).
- T**he Address Register (AR)** holds the memory location address from which data needs to be read or where data needs to be written.
- **The Data Register (DR)** is used to temporarily store the data being transferred between memory and the CPU.
- **Read Operation:**To perform a read operation:

  DR ← M[AR]
- **Write Operation**: To perform a write operation:

  M[AR] ← DR

General Register-Based CPU Organization:

- In a general register-based CPU organization:
- Multiple general-purpose registers are used instead of a single accumulator register.
- Two or three address fields are present in the instruction format.
- Each address field specifies a general register or a memory word.
- The availability of many CPU registers enables efficient storage of heavily used variables and intermediate results.
- This organization minimizes memory references, leading to increased program execution speed and reduced program size



- **Example Instruction:**
  - Example assembly language instruction: MULT R1, R2, R3
  - MULT: Mnemonic for multiplication.
  - R1, R2, R3: Register operands.
  - R1 <-- R2 * R3: Indicates that the result of the multiplication is stored in register R1.
- **Advantages**:
  - Supports efficient data manipulation.
  - Reduces the need for frequent memory accesses.
  - Common in modern processor architectures, especially Reduced Instruction Set Computing (RISC) architectures.

Register Stack

- An ordered set of elements with a variable length.
- Accessible one element at a time from the top.
- Also known as a pushdown list or Last-In-First-Out (LIFO) list.

Registers Used in Stack Organization:

- **Stack Pointer Register (SP):**

Contains a 6-bit binary value representing the address of the top of the stack.

Limited to values from 000000 to 111111 (0 to 63).

- **FULL Register:** 1-bit register.

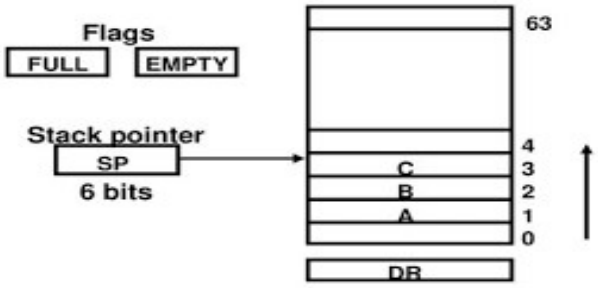
Set to 1 when the stack is full.

- **EMPTY Register:** 1-bit register.

Set to 1 when the stack is empty.

- **Data Register (DR):**

Holds data to be written into or read from the stack.



Working of POP and PUSH:

- **PUSH ():**

```
if FULL == 0:
    SP = SP + 1        // Increment stack pointer
    M[SP] = DR   // Write item Data Register to top of  stack
    if SP == 0:
        FULL = 1       // Mark the stack as full
    EMPTY = 0          // Mark the stack as not empty
```

- **POP ():**

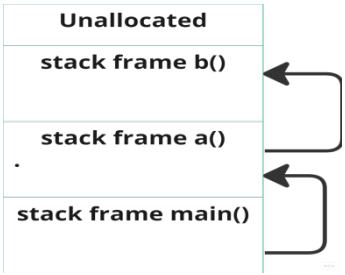
```
if EMPTY == 0:
    DR = M[SP]  //Read item from top of stack to Data Register
    SP = SP - 1        // Decrement stack pointer
    if SP == 0:
        EMPTY = 1      // Mark the stack as empty
    FULL = 0           // Mark the stack as not full
```

Memory stack

- Memory stack is a dynamic data structure used in processor processes.
- Maintains automatic process-state data, aiding program execution.
- Manages the flow of control in subroutines.
- Stores return addresses during subroutine calls and pops them upon returning.
- Reflects the execution state of the process.
- Stores subroutine arguments and local variables.
- Information pushed onto the stack as a result of a function call is called a "frame."
- The address of the current frame is stored in the frame pointer register.
- Dynamic structure supports any level of nesting within memory constraints.

- **Example of memory stack :**

```
1   b(){  .....   }
2   a(){
3       b();
4   }
5   main(){
6       a();
7   }
```



Register stack VS MEMORY stack

Register Stack	Memory Stack:
: Generally within the CPU	Typically in RAM.
Momentary spaces for internal processes in the CPU.	Series of memory spaces used in processor processes.
Provides fast access to data.	Data is temporarily stored in registers.
Limited in size.	Larger in size.

Addressing mode

- A technique used in computer architecture to specify how operands are chosen for instructions
- Determining how the processor references memory to fetch or store data.

1. Immediate Addressing Mode:

- Operand is specified explicitly in the instruction.
- Example: MOV AX, #5 (moves the immediate value 5 into register AX).

2. Register Addressing Mode:

- Operand is the content of a register.
- Example: ADD BX, CX (adds the contents of registers BX and CX).

3. Direct Addressing Mode:

- Operand is the content of the memory location directly specified by the instruction.
- Example: MOV AX, [1000] (moves the content of memory location 1000 into register AX).

4. Indirect Addressing Mode:

- Operand is the content of the memory location whose address is specified by a register or another memory location.
- Example: MOV AX, [BX] (moves the content of the memory location whose address is in register BX into register AX).

5. Indexed Addressing Mode:

- Operand is found at the sum of a base register and an index register, possibly scaled by a factor.
- Example: MOV AX, [SI + 100] (moves the content of the memory location whose address is the sum of the base register SI and the immediate value 100 into register AX).

6. Relative Addressing Mode:

- Operand is at a location whose address is given by the sum of the address in a register and a constant offset.
- **Example**:
  - Jump+3 if accumulator == 2
  - If accumulator != 2 then Jump +5
  - This is called a conditional jump and it is making use of relative addressing .

7. Stack Addressing Mode:

- Operand is implicitly the top of the stack.
- Example: PUSH AX (pushes the content of register AX onto the stack).

8. Displacement Addressing Mode:

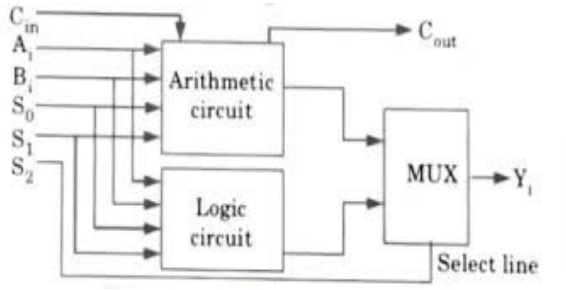
- The effective address of the operand is calculated as the sum of the content of register BX and the displacement value (10), resulting in the memory address from which data is to be loaded into the AX register.
- Operand Address = Base Register + Displacement
- MOV AX, [BX + 10]
- In this instruction: BX is the base register, 10 is the displacement value.

9. Implied Addressing Mode:

- Operands are specified implicitly in the definition of the instruction .
- Zero address instruction is a stack organized computer are implied mode instruction since the operands are implied to be on the top of the stack .

Sequential Artihmetic and Logic Unit

- ALU is a component within a computer's CPU.
- It performs arithmetic and logical operations on binary data.
- **Arithmetic Operations**: Includes addition, subtraction, multiplication, and division.
- **Logical Operations:** Handles logical functions like AND, OR,NOT.
- **Sequential Execution**: Operations are carried out one after another, not simultaneously.



Arithmetic Mode (s2 = 0):

- ALU functions as an arithmetic circuit.
- The output of the arithmetic circuit is transferred as the final output.
- This implies that arithmetic operations, such as addition, subtraction, multiplication, or division, are performed.

Logic Mode (s2 = 1):

- In this mode, the ALU acts as a logic circuit.
- The output of the logic circuit is transferred as the final output.
- Unlike in arithmetic operations, carry input or carry output is not required in logic operations.

Half adder

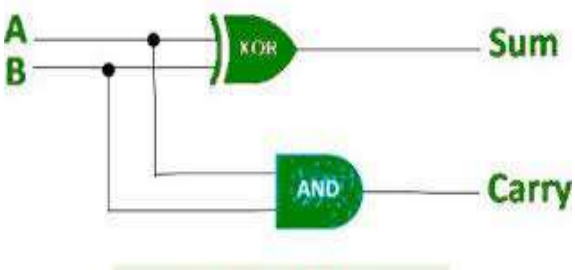
Sum (S): The XOR of A and B.

S = A XOR B

Carry (C): The AND of A and B.

$Cout_t$ = A AND B

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



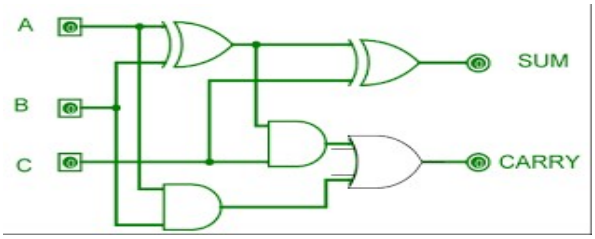
Full adder

sum (S): The XOR of the three inputs (A, B, and Cin).

S=A XOR B XOR Cin

Carry (Cout):

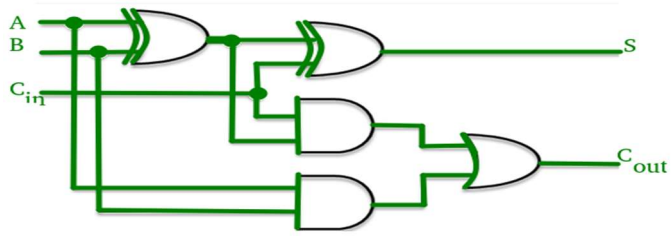
Cout= (A AND B) + (A XOR B) AND Cin



Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Carry Lookahead Adder (CLA)

- A Carry Lookahead Adder (CLA) is a type of adder in digital logic.
- It improves speed by reducing the time needed to determine carry bits.
- CLA calculates carry bits before the sum, minimizing waiting time for the result of higher-value bits.
- Two variables, propagator and generator, are used in CLA for efficient carry computation.



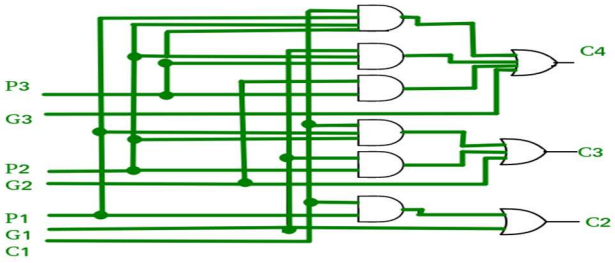
- Addition of two binary numbers in parallel allows all bits to be available for computation simultaneously.
- Carry propagation time is a critical attribute of adders.
- Reducing carry delay time is a key advantage of CLA, contributing to overall speed improvement.

$$P_i = A_i \oplus B_i \qquad S_i = P_i \oplus C_i$$
$$G_i = A_i B_i \qquad C_{i+1} = G_i + P_i C_i$$

where G_i produces the carry when both A_i, B_i are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to C_{i+1}.

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

- C4 is computed simultaneously with lower-order carries .
- No waiting for carry propagation; they are determined in parallel.
- Boolean expressions for carries use a sum of products approach.
- Implemented with one level of AND gates for product terms and an OR gate.
- Boolean functions are designed for minimal delay.
- Carry-out is ready for the next bit without waiting for sequential propagation.

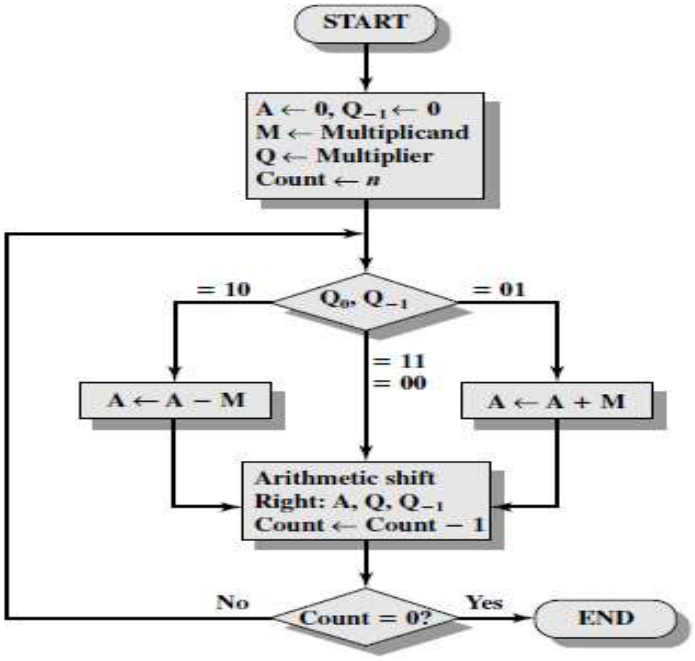


Booths algo

- Booth's Algorithm is a multiplication algorithm that efficiently performs binary multiplication using a series of steps.

  - Initialize variables: Multiplier (Q), Multiplicand (M), Accumulator (A), and a counter (N).
  - Set the counter (count) to the bit length of the multiplier (Q).
  - Start a loop that iterates N times.
  - Check the rightmost two bits of the multiplier (Q0 ,Q-1).
    - If Q0Q-1 = 10, perform the operation A=A−M.
    - If Q0 Q-1 = 01, perform the operation A=A+M.
  - Right shift Q and A by 1 bit.
  - Decrement (count) by 1.
  - Check if the counter (count) is greater than 0.
  - If true, go back to the "Loop Start" step; otherwise, exit the loop.

- The final product is in the Accumulator (A) and the Multiplier (Q).



Example : Multiply (7 x 3)

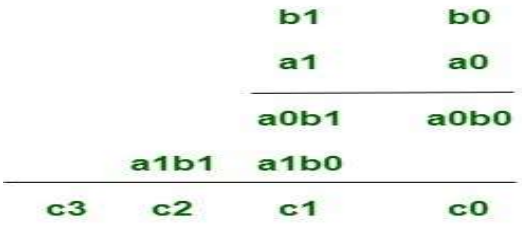
Initialized value :

A=0000
Q=0011
Q-1=0
M=0111
-M = 1001

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	First cycle
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	Third cycle
0001	0101	0	0111	Shift	Fourth cycle

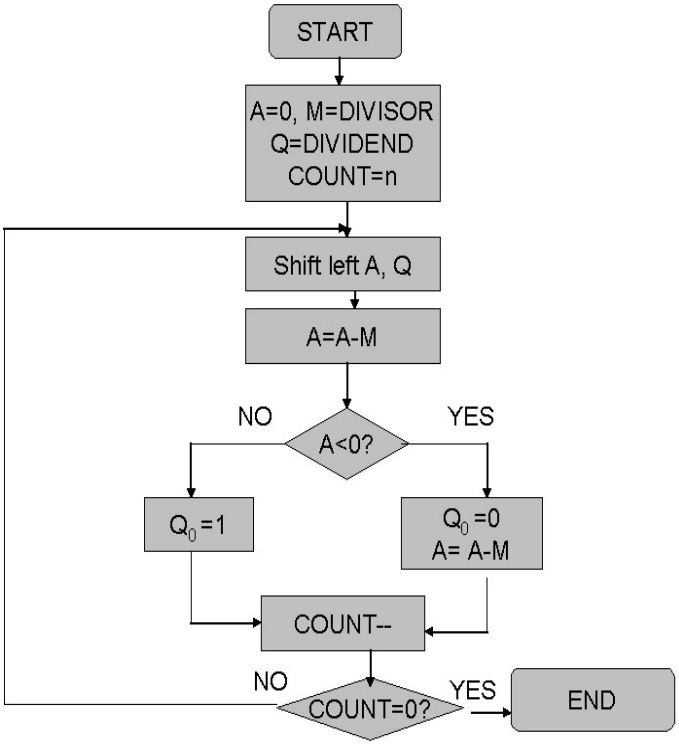
Array multiplier

- An array multiplier is a digital circuit that performs binary multiplication using an array of logic gates. The most common array multiplier architecture is the Wallace Tree Multiplier.
- **Partial Products:**



Restroing division algorithm

- Restoring division is a binary division algorithm that involves restoring partial remainders during each step of the division process.



Example : divide (11 / 3) using restoring

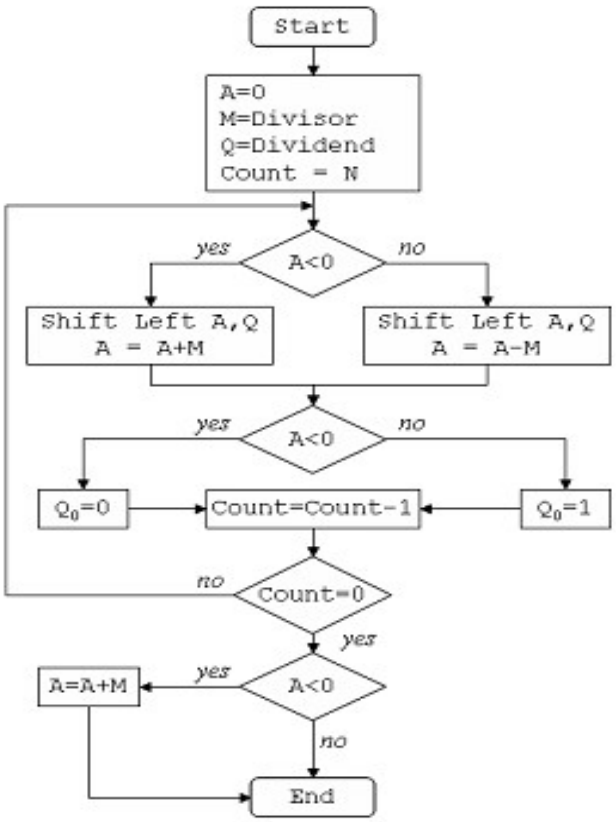
Initialized value :

A=0000
Q=1011
M=00011 , -M = 11101

N	A	Q	ACTION
4	00000	1011	initialize
	00001	011_	ShL AQ
	11110	011_	A=A-M
3	00001	0110	Restore Q[0]=0
	00010	110_	ShL AQ
	11111	110_	A = A - M
2	00010	1100	Restore, Q[0] = 0
	00101	100_	ShL AQ
	00010	100_	A = A - M
1	00010	1001	Q[0] =1
	00101	001_	ShL AQ
	00010	001_	A = A - M
0	00010	0011	Q[0] = 1

Non restoring method

- Non-restoring division is more complex than restoring division algorithmically.
- However, its hardware implementation simplifies the process.
- Non-restoring division involves only one decision and addition/subtraction per quotient bit.
- After subtraction, there are no additional restoring steps, leading to a simpler hardware design.



Example : divide (11 / 3) using non-restoring

Initialized value :

A=0000
Q=1011
M=00011 , -M = 11101

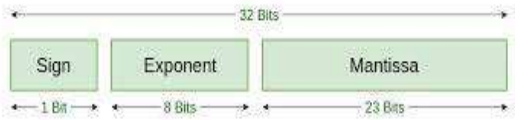
N	A	Q	ACTION
4	00000	1011	initialize
	00001	011_	ShL AQ
	11110	011_	A=A-M
3	11110	0110	Q[0]=0
	11100	110_	ShL AQ
	11111	110_	A = A + M
2	11111	1100	Q[0] = 0
	11111	100_	ShL AQ
	00010	100_	A = A +M
1	00010	1001	Q[0] = 1
	00101	001_	ShL AQ
	00010	001_	A = A - M
0	00010	0011	Q[0] = 1

IEEE standard for floating point arithmetic

- IEEE standard for floating point arithmetic is a technical standard for floating point computation .

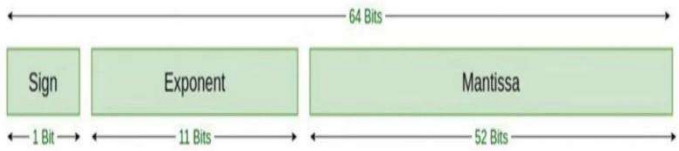
Single Precision (32 bit):

- The format consists of three components: the sign bit, the exponent, and the fraction (also known as the significand or mantissa).
- 1. **1 sign bit** 2. **8 bit exponent** 3. **23 bit mantissa**
- The formula for the value of a single-precision floating-point number is: $(-1)^S x 1.f \times 2^{(e-127)}$



Double Precision (64 bit):

- 1. **1 sign bit** 2. **11 bit exponent** 3. **52 bit mantissa**
- The formula for the value of a single-precision floating-point number is: $(-1)^S x 1.f \times 2^{(e-1023)}$



Example: ***Suppose we want to represent the decimal number 6.75 in IEEE 754 single-precision format.***

number is positive, then sign bit **s = 0**

Convert the absolute value to binary:

$(6.75)_{10}$ = $(110.11)_2$ in binary.

Normalize the binary representation:

$110.11 = 1.1011 \times 2^2$.

exponent (e): e=2+127=129.

exponent in binary: $129_{10} = 10000001_2$

Single precision format :

0	10000001	1011……..

Example: ***Suppose we want to represent the decimal number -6.75 in IEEE 754 double-precision format.***

number is positive, then sign bit **s = 1**

Convert the absolute value to binary:

$(6.75)_{10}$ = $(110.11)_2$ in binary.

Normalize the binary representation:

$110.11 = 1.1011 \times 2^2$.

exponent (e): e=2+1023=1025.

exponent in binary: $1025_{10} = (10000000001)_2$

Double precision format :

1	10000000001	1011……..

OVERFLOW :

- Overflow occurs when the result of an arithmetic operation is too large (in absolute value) to be represented within the available number of bits.
- There are two primary types of overflow: signed overflow and unsigned overflow.
- **Signed Overflow**: Occurs when the result of an operation exceeds the maximum representable positive value or falls below the minimum representable negative value for the given number of bits.
- **Unsigned Overflow**: Occurs when the result of an operation exceeds the maximum representable value for the given number of bits, considering all values as non-negative.

Instruction

- Instruction is a command to the processor to perform a given task on specified data.
- A computer performs tasks on the basis of the instruction provided.
- Instruction format :
  1. Opcode specifies the operation to be performed.
  2. Operand specifies that data to be operated upon .
  3. The Mode specifies how the operand will be located.

Mode	Opcode	Operand/ address of Operand

Types of operation performed by instruction

- Data transfers between memory and CPU registers .
- Arithmetic and logic operation on data .
- Program sequencing and control :manage the execution of instructions in a program
- I/O transfers

Example

ADD R1, R2, R3
ADD is the opcode
R1, R2 , R3 is the register or operand where the data for the operation is stored

Types of instruction

1. Three addressing instruction :

- Instructions specify three operands.
- The operation is performed between the two source operands, and the result is stored in the destination operand.

Example : X=(A+B* (C+D)

ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

2. Two addressing instruction :

- Instructions specify two operands.
- The operation is performed between the two explicitly specified operands, and the result is stored in one of the operands.

Example : X=(A+B* (C+D)

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = M[C]
ADD	R2, D	R2 = R2 + M[D]
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

3. One addressing instruction :

- Instructions specify one operand.
- The operation is performed between the operand and an implicit accumulator or register.

Example : X=(A+B* (C+D)

```
LOAD    A       AC = M[A]
ADD     B       AC = AC + M[B]
STORE   T       M[T] = AC
LOAD    C       AC = M[C]
ADD     D       AC = AC + M[D]
MUL     T       AC = AC * M[T]
STORE   X       M[X] = AC
```

4. Zero addressing instruction :

- stack is the zero addressing instruction
- Stack organized computer does not use an address field for the instructions ADD and MULL .

Example : X=(A+B* (C+D)

```
PUSH    A       TOP = A
PUSH    B       TOP = B
ADD             TOP = A+B
PUSH    C       TOP = C
PUSH    D       TOP = D
ADD             TOP = C+D
MUL             TOP = (C+D)*(A+B)
POP     X       M[X] = TOP
```

Numarical : Evaluate arithmetic statement X=(A+B* (C+D) using a general register computer with three , two , one address instruction format a program to evaluate the expression

Intruction cycle :

- Instruction cycle is a complete process of instruction execution .
- It is a basic operational process of a computer .
- Instruction cycle is divided into three sub cycles :

Fetch cycle :

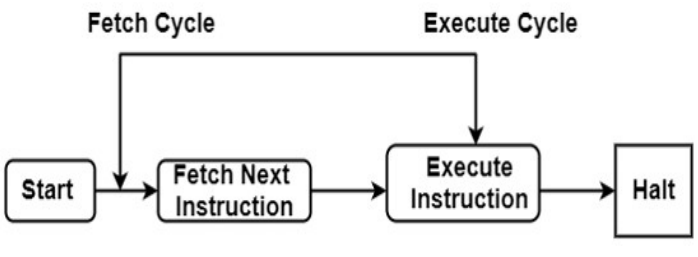
- The CPU retrieves the next instruction from memory.
- Address is determined by the Program Counter (PC) register.

Decode cycle :

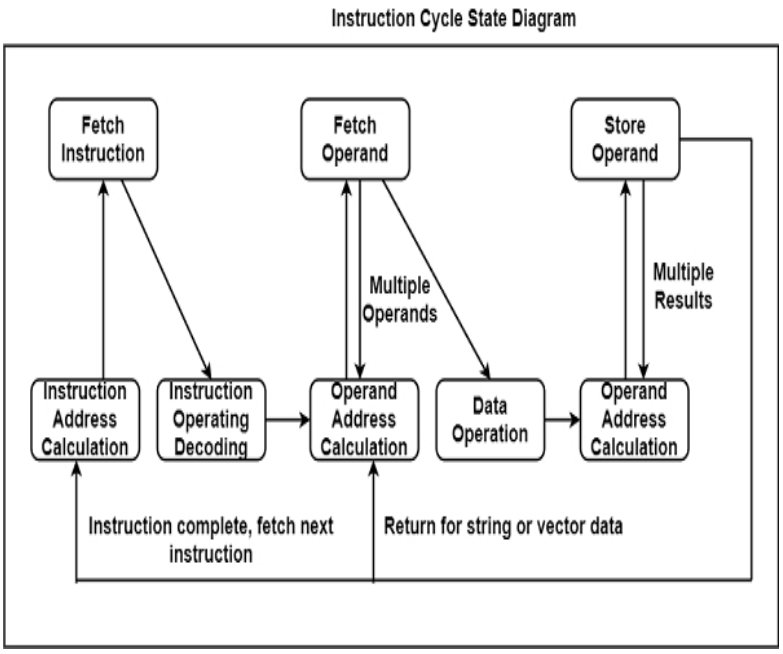
- The fetched instruction is analyzed to identify the operation and operands.
- Opcode (operation code) and addressing modes are determined.

Execute cycle :

- The CPU carries out the operation specified by the instruction.
- Involves arithmetic, logical operations, data movement, or control transfers

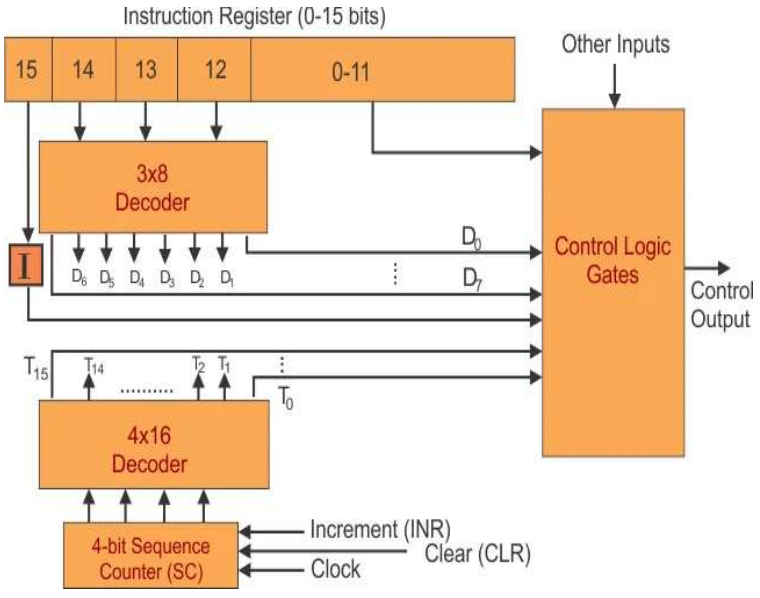


State diagram of the instruction cycle



How instruction will be decoded or executed by a control unit :

- Control unit consists of :
  1. Instruction register
  2. Number of control logic gates
  3. Two decoders
  4. 4-bit sequence counter
- The instruction is fetched from memory and stored in the Instruction Register (IR).
- IR is divided into I bit, opcode, and operand (address part).
- The first 12 bits (0-11) are sent to control logic gates.
- Process the first 12 bits to perform specific operations.
- Opcode bits (12-14) are decoded using a 3x8 decoder.
- Produces eight outputs (D0 through D7) for control logic.
- The last bit (15) of IR is transferred as an addressing mode.
- Sequence Counter Counts from 0 to 15 in binary.
- Output is decoded into 16 timing pulses (T0 through T15).
- Control of Sequence Counter Can be incremented by the INR input.
- Can be cleared by the CLR input synchronously.



Microoperations

- Microoperations are elementary operations that manipulate data stored in registers or perform control operations within the CPU.

Types of Microoperations:

1. Register Transfer Microoperations:

- Involve the transfer of data between registers.
- Examples: Load, Store, Transfer, Exchange.

2. Arithmetic Microoperations:

- Involve arithmetic operations on data in registers.
- Examples: Add, Subtract, Increment, Decrement.

3. Logic Microoperations:

- Involve logical operations on data in registers.
- Examples: AND, OR, NOT, XOR.

4. Shift Microoperations:

- Involve shifting the bits of data within registers.
- Examples: Shift Left, Shift Right.
-

Shift microoperations :

- Shift microoperations involve moving the bits of a binary number to the left or right within a register. There are two main types of shift operations: logical shifts , arithmetic shifts , circular shifts

1. Logical Shifts:

Left Logical Shift (LSL):

- In a left logical shift, zeros are shifted into the vacated bit positions on the right, and the leftmost bits are shifted out.
- **Example:** If 110110 is left logically shifted by 2 positions, the result is 011000.

Right Logical Shift (LSR):

- In a right logical shift, zeros are shifted into the vacated bit positions on the left, and the rightmost bits are shifted out.

Example: If 110110 is right logically shifted by 2 positions, the result is 001101.

2. Arithmetic Shifts:

Left Arithmetic Shift (ASL):

- In a left arithmetic shift, the bits are shifted to the left, and the vacated bit positions on the right are filled with the sign bit (the leftmost bit).

Example: If 110110 is left arithmetic shifted by 2 positions, the result is 011000.

Right Arithmetic Shift (ASR):

- In a right arithmetic shift, the bits are shifted to the right, and the vacated bit positions on the left are filled with the sign bit (the leftmost bit).

Example: If 110110 is right arithmetic shifted by 2 positions, the result is 111101.

3. Circular Shift

Circular Left Shift :

- each bit is shifted to the left by the specified number of positions. The bits that are shifted out on the left re-enter from the right.

Example: If 110110 is Left Circular shifted by 2 positions, the result is 011011.

Circular Right Shift :

- each bit is shifted to the right by the specified number of positions. The bits that are shifted out on the right re-enter from the left.

Example: If 110110 is right Circular shifted by 2 positions, the result is 101101.

CISC

- CISC stands for Complex Instruction Set Computer.
- It features a complex instruction set, encompassing a wide range of operations.
- CISC instructions have a variable-length format.
- The execution of instructions in CISC architectures may take a varying number of clock cycles.
- CISC processors allow direct manipulation of operands residing in memory.
- This supports versatile data handling capabilities.
- CISC architectures aim to simplify the compiler's task.
- CISC architectures support a diverse set of instructions, including arithmetic, memory access, and complex data manipulation operations.
- Microprogrammed control is used in cisc

Example : MULADD R4, R1, R2, R3

Characteristics of CISC

- 100-250 instructions in the set.
- Inclusion of infrequently used specialized tasks.
- 5-20 diverse addressing modes.
- Variable-length instruction formats.
- Emphasis on manipulating operands in memory.

RISC

- RISC stands for Reduced Instruction Set Computer.
- Fixed-length instruction formats for simplicity.
- Data is stored in processor registers for computations.
- Results are transferred to memory using store instructions.
- All instructions in RISC have simple register addressing.
- A limited number of addressing modes are used.
- CISC architectures aim to simplify the processor's task.
- RISC architectures typically have a large number of registers in the processor unit.
- Hardwired control is used in risc

Example :

```
LDA R1, 1000(R2)
ADD R3, R1, R2
```

Characteristics of risc

- Limited instruction set, often around 50-100 instructions.
- Predominance of simple instructions with uniform execution time.
- Minimal addressing modes, typically 3-5 modes.
- Fixed-length instruction formats for simplicity.
- Emphasis on register-to-register operations, reducing memory access.

RISC VS CISC

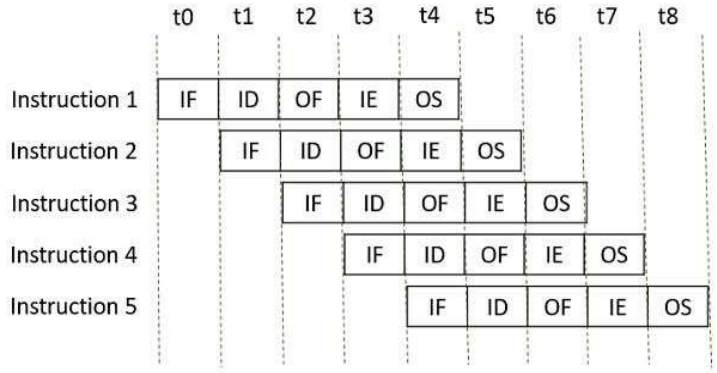
CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

Pipelining

- Pipelining involves breaking down a sequential process into smaller sub-operations.
- Each sub-operation is executed in its own segment.
- All segments run in parallel with each other, facilitating concurrent processing.
- Operations are organized in a way that facilitates concurrent execution, leading to overall time savings.
- Pipelining is like optimizing a production line for efficiency in assembling a product.
- Creates and organizes a pipeline of instructions for the processor to execute in parallel.
- Enables parallel processing of multiple instructions, enhancing overall performance.

An instruction pipeline has five stages:

- IF: Instruction Fetch
- ID: Instruction Decode
- OF: Operand Fetch
- IE: Instruction Execute
- OS: Operand Store



Pipelining of 5 Instructions

Types of pipelining

1. Arithmetic Pipelining:

- Designed for high-speed floating-point arithmetic operations, including addition, multiplication, and division.
- Multiple arithmetic logic units (ALUs) are integrated into the system to perform parallel computation in various data formats.

- This type of pipelining is particularly effective in scenarios where a significant number of arithmetic calculations need to be performed rapidly.

2. Instruction Pipelining:

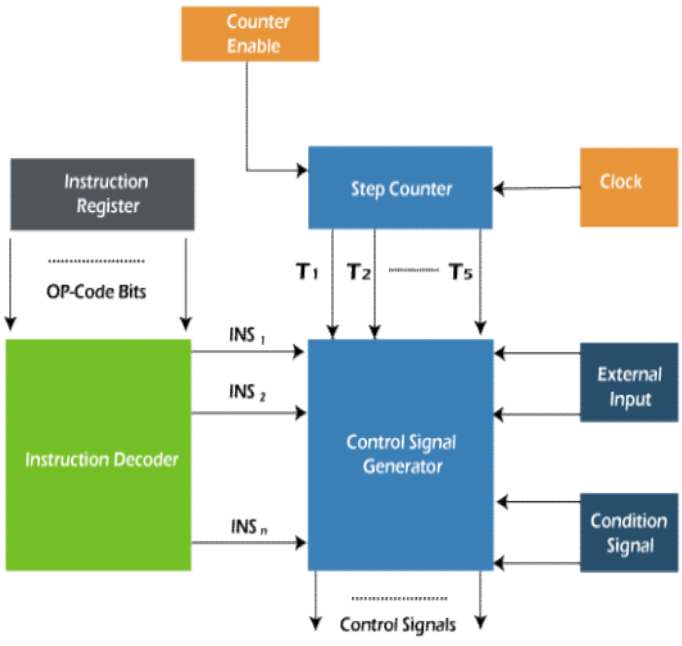
- Overlaps the execution of multiple instructions to improve throughput and efficiency.
- The execution of one instruction is overlapped with the execution of the subsequent instruction. It is also known as "instruction lookahead."
- Instruction pipelining is a common technique in modern processors to enhance overall instruction execution speed by breaking down the instruction processing into stages.

3. Processor Pipelining:

- Processors are pipelined to handle the same data stream.
- The data stream is initially processed by the first processor, and the result is stored in memory. The subsequent processors access and refine the result obtained by the previous processor.
- This type of pipelining is often used in scenarios where different processors specialize in different aspects of data processing, and the output of one processor becomes the input for the next.

Hardwired control unit

- It is a controller as a sequential logic circuit or a finite state machine that generates a sequence of control signals in response to the externally supplied instruction
- Control logic is implemented with gates, flip-flop , decoders , and other digital circuits .
- A hardwired control requires changes in the wiring among the various components if the design has to be modified or changed
- Each step in this sequence is completed in one clock cycle.
- A counter may be used to keep the track of control steps .



Hardwired Control Unit

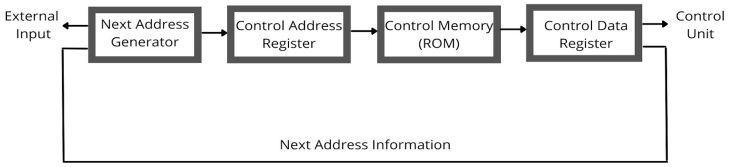
- **Instruction Register**: The instruction fetched from the main memory is placed in the instruction register and the instruction remains there till its execution is completed.
- **Instruction Decoder**: The instruction decoder interprets the opcode and the addressing mode from the instruction register and determines what actions have to be taken.
- **Step Counter:** Tracks progress in instruction execution.
  It Specifies the current step among instruction fetch, decode, operand fetch, execute, and operand store.
- **Signal Generator**: It is a combinational circuit that generates the control signals depending upon their input.
- **Clock**: The clock implement in the control circuitry is such that it completes one clock cycle for each step of instruction execution.
- **External Inputs**: The external input component acknowledges the control circuitry about the external signal such as interrupts.
- **Conditional signal :** These components help the control unit in generating the control signals for branching instructions.

There are four simplified and systematic methods for the design method :

- State table method or one-hot method .
- Delay element method
- Sequence counter method
- PLA method

Microprogrammed Control Unit

- It is a component within a computer's central processing unit (CPU) that uses microcode to control and execute instructions.



- Sequence of control signals to be generated by the controller can be stored in a special ROM also called Control MEMORY .
- Memory control word is written for each micro-operation and these control words are stored in a serial ascending memory location .
- Control word is accessed serially from the control memory .
- Control words are stored in the ROM permanently .
- The output of the control memory provides the required control signals .
- If the control memory is sequentially accessed by incrementing control memory location . then the sequence of control signals stored in successive word of ROM can be generated.

Hardwired vs Microprogrammed Control Unit

Microprogrammed Control Unit	Hardwired Control Unit
It is implemented by programming	It is a circuitry approach
CISC style instructions	RISC style instructions
Modifications are easy as it will require the change in the code section only.	Modification is difficult as the control unit is hardwired
It works well for complex instructions also	It works well for simple instructions
Implementing microprograms is not costly	Implementing hardwired structure is costly
Slower execution	Faster execution

Memory

- memory refers to electronic storage space used to store data and instructions that processor can access quickly.
- There are several types of memory in a computer, each serving different purposes.

Memory Hierarchy

- It is organized in layers, with the fastest and most expensive memory at the top and slower, larger at the bottom and more affordable memory types below:

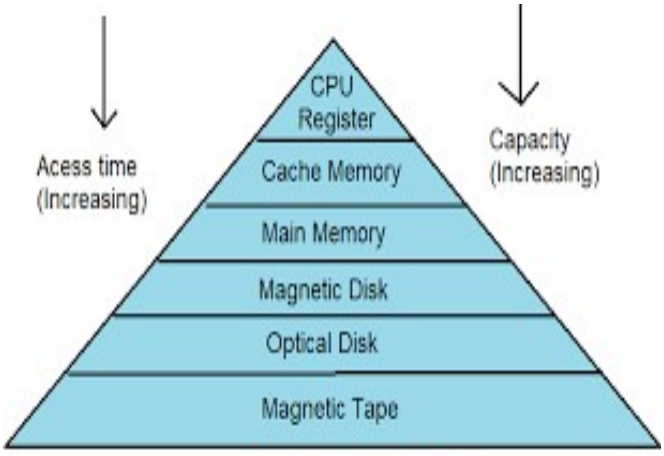


Fig:- Memory Hierarchy

- **Registers**: Extremely fast, small storage locations within the CPU for actively processed data.
- **Cache Memory**: Small-sized, high-speed memory near the CPU, storing frequently used programs and data.
- **Random Access Memory (RAM)**: Volatile, temporary memory for quick data access.
- **Solid-State Drives (SSD)**: Non-volatile storage using flash memory for faster access than traditional Hard Disk Drives (HDDs).
- **Hard Disk Drives (HDD):** Magnetic storage on spinning disks for larger but slower access.
- **Optical Drives, Tape Drives, and Cloud Storage:** Lower-speed options for long-term archival storage.
- **"The Purpose and Benefits of Organizing Computer Memory in a Hierarchical System"**

Cost	Speed	Capacity
Volatility	Hierarchy flexibility	Optimizing performance

Semiconductor RAM

- Semiconductor RAM refers to a class of computer memory technologies that utilize semiconductor materials for storing and accessing data.
- The semiconductor RAM family includes several types, with Dynamic RAM (DRAM) and Static RAM (SRAM) being two of the most common.

1. Dynamic RAM (DRAM):

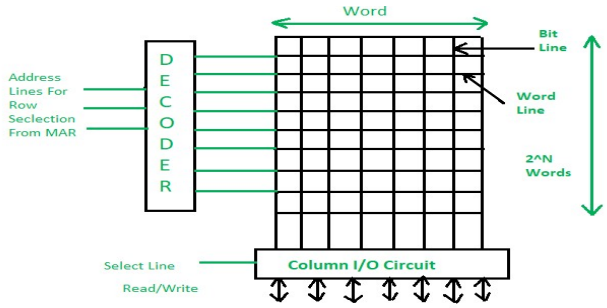
- DRAM is made up of capacitors and transistors.
- Each DRAM memory cell comprises a capacitor (for storing charge, representing a bit) and a transistor (controlling access to stored information).
- DRAM is volatile, requiring constant power; data is lost if power is interrupted.
- Periodic refresh cycles are necessary to counteract charge leakage and maintain data integrity.
- DRAM provides high memory density, suitable for main memory (RAM) in computers.
- DRAM is relatively cost-effective compared to other memory types.
- **Types of DRAM:**
- Each designed for specific applications and offering improvements in speed and efficiency.
  - Synchronous DRAM (SDRAM)
  - Double Data Rate (DDR) SDRAM
  - Graphics DRAM (GDDR),

2. Static RAM (SRAM):

- SRAM utilizes a flip-flop circuit made of transistors.
- SRAM is volatile, resulting in data loss when power is turned off.
- SRAM is faster than DRAM, providing quick access times, ideal for cache memory.
- SRAM is more complex and generally more expensive to manufacture than DRAM. It is often used in smaller quantities, particularly for cache memory.

2D memory organization:

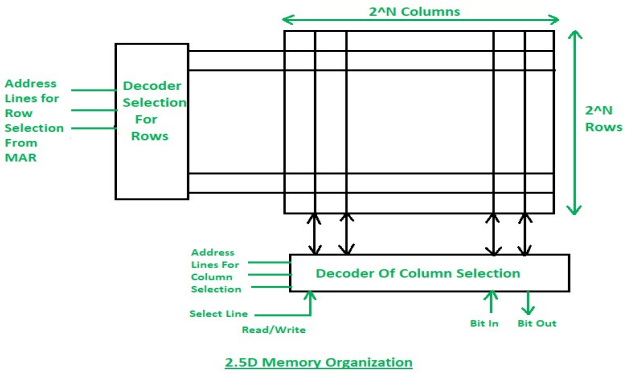
- The memory is structured as a 2D matrix, with rows(word lines) and columns(bit lines) .
- Each row in the matrix represents a word, which is a fixed-size unit of data.
- The decoder is a combinational circuit with n input lines and 2^n output lines.
- The n input lines take the address information from the Memory Address Register (MAR).
- The decoder processes the input address to generate 2^n output lines.
- One of the output lines is selected based on the specific address provided in the MAR.



- The selected output line from the decoder corresponds to a specific row in the matrix.
- This row is then accessed for reading or writing.
- Once the row is selected, the word of data in that row can be accessed through the data lines.
- For reading, the data is transferred from the selected row to the requesting entity (e.g., CPU or I/O controller).
- For writing, new data can be written into the selected word in the row.

2.5 D memory organization

- In 2.5D memory organization, there are two decoders: a column decoder and a row decoder.
- The column decoder selects the column, while the row decoder selects the row based on the address from the Memory Address Register (MAR).
- The selected cell is determined through the bit outline, enabling the read or write operation at that memory location using the data lines.



2.5D Memory Organization

- In reading mode, the word/bit represented by the Memory Address Register (MAR) becomes available on the data lines.
- This enables the retrieval of the desired data from the selected memory cell for reading.
- In write mode, data from the Memory Data Register (MDR) is sent to the memory cell addressed by the Memory Address Register (MAR).
- The select line helps in identifying the specific cell, facilitating the write operation with the data from the MDR.

Differences of 2d and 2.5d orgn

- In 2D orgn hardware is fixed but in 2.5D hardware changes.
- 2D orgn requires more gates while 2.5D requires less.
- 2D is more complex in comparison to the 2.5D orgn .
- Error correction is not possible in the 2D orgn but in 2.5D it could be done easily.
- 2D is more difficult to fabricate in comparison to the 2.5D orgn . this line write in different with bullets points

ROM

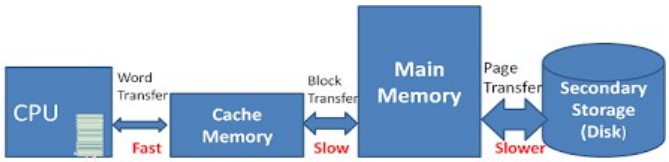
- ROM stands for Read-Only Memory.
- It is a type of non-volatile memory that retains its contents even when the power is turned off.

Types of ROM:

- **PROM (Programmable ROM):** Allows for a one-time programmability, where the data is written (or programmed) into the memory by the user. Once programmed, the data becomes permanent.
- **EPROM (Erasable Programmable ROM):** Users can erase and reprogram the memory multiple times. However, erasure typically involves exposure to ultraviolet light.
- **EEPROM (Electrically Erasable Programmable ROM):** Similar to EPROM but can be erased and reprogrammed electrically, without the need for ultraviolet light. EEPROM allows for more convenient updates.
- **Mask ROM:**In some cases, the ROM is created with fixed data during the manufacturing process and is referred to as mask ROM. The data is "masked" onto the ROM chip during fabrication.

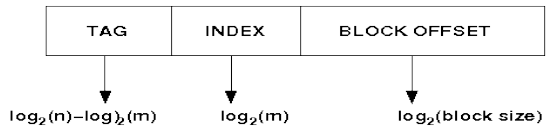
Cache memory

- Cache memory is high-speed volatile computer memory.It provides rapid data access to a processor.
- Stores frequently used computer programs, applications, and data.
- Crucial for improving overall computer system speed and performance.
- Store frequently accessed data and instructions close to the CPU.
- Minimizes the time to access data and instructions compared to main memory (RAM).



- There are typically three levels of cache in a modern computer system:
- **L1 Cache:** Smallest, fastest, on CPU chip, separate instruction and data caches for independent handling.
- **L2 Cache**: Slightly larger, slower than L1, often shared among different cores in a multi-core processor, provides additional storage for frequently accessed data.
- **L3 Cache:** Larger, slower than L2, shared among all cores in a multi-core processor, additional storage for frequently used data that may not fit in L1 and L2 caches.

- **Terminologies in cache Memory**



- **Cache Lines:** Fixed-size blocks storing small amounts of data.
- **Cache Size:** Total amount of data cache can hold, measured in KB or MB.
- **Hit and Miss:** Hit when data found, miss when not (requires retrieval from higher-level cache or main memory).
- **Tag, Index, and Offset:** Divides physical address for address mapping in the cache.

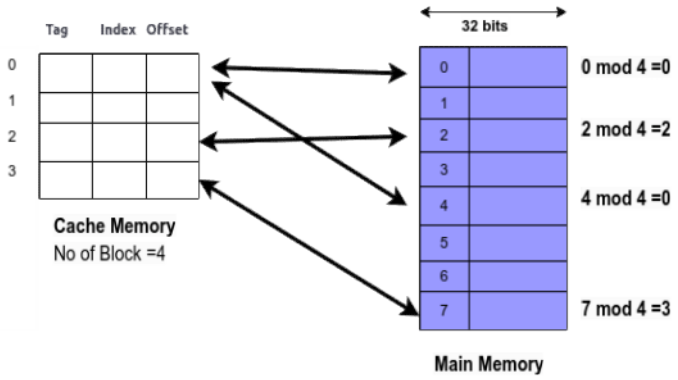
- **How performance of cache memory is measured :**
- Perfomance of cache memory is frequently measured in terms of a quantity called hit ratio.
- When the cpu refers to memory and finds the word in cache , it is said to produce a hit .
- If the word is not in cache , it is in main memory and it counts as a miss .
- Ratio of hit and miss is called hit ration
- This is the best way to measured the performance of the cache memory .

Cache mapping

- Cache mapping refers to the technique used to determine how data is mapped between the main memory (RAM) and the cache memory.
- The primary goal of cache mapping is to efficiently manage the transfer of data between these two levels of memory to optimize overall system performance.

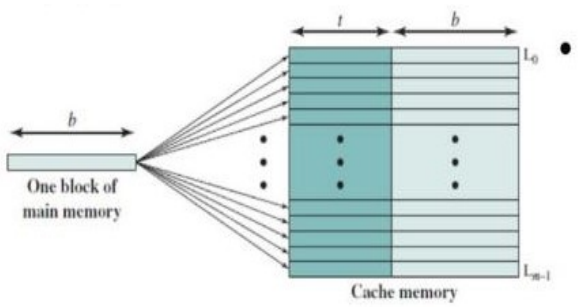
1. Direct Mapping

- Each block of main memory maps to a specific block in the cache.
- A simple and efficient technique, but it may lead to cache conflicts.
- One cache line corresponds to one specific block in main memory.
- **Advantages:**
- Simplicity in hardware implementation.
- Lower hardware cost.
- **Disadvantages:**
- Limited flexibility may lead to more conflicts and cache misses.



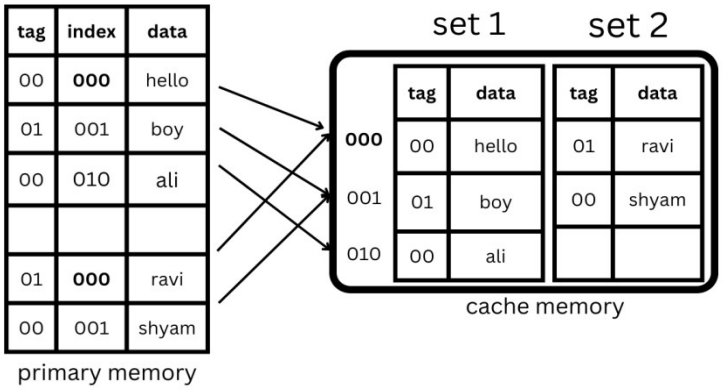
2. Fully Associative Mapping

- Any block of main memory can be placed in any line of the cache.
- Offers maximum flexibility but is more complex to implement.
- Reduces the likelihood of cache conflicts.
- **Advantages:**
- Maximum flexibility, reducing conflicts.
- Potential for high cache hit rates.
- **Disadvantages:**
- More complex and expensive hardware.



3. Set-Associative Mapping

- Combines aspects of direct and fully associative mapping.
- Divides the cache into sets, with each set containing multiple lines (blocks).
- Each block of main memory can map to any line within a specific set.
- Provides a balance between simplicity and flexibility.
- **Advantages:**
- Offers a balance between flexibility and simplicity.
- Reduces conflicts compared to direct-mapped caches.
- **Disadvantages:**
- Moderately complex hardware.

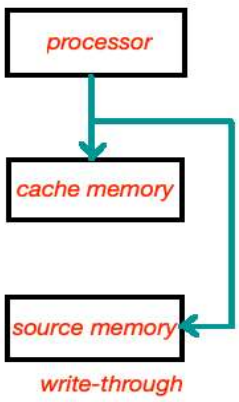


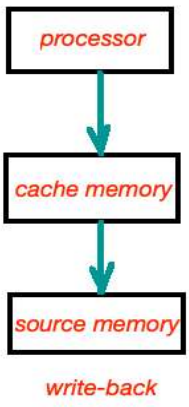
Write-Through:

- Definition: Simultaneous update of both cache and main memory with every write operation.
- Process:
- Cache modification immediately reflected in main memory.
- Ensures main memory holds the most up-to-date information.
- Advantages:
- Guarantees data consistency between cache and main memory.
- Simplifies management and recovery in system failures.
- Disadvantages:
- Slower write performance due to dual updates.
- Higher bus traffic from frequent main memory updates.

Write-Back:

- Cache-only update with "dirty" marking for outdated main memory.
- Read operation on "dirty" location triggers cache content write-back to main memory.
- Delayed writes reduce write frequency, minimizing bus traffic.
- Potentially faster write performance, updating only the cache.
- Reduces bus traffic, beneficial for bandwidth-limited systems.
- Potential for data inconsistency until write-back occurs.
- Complex management of dirty blocks, possible recovery issues after system failures.

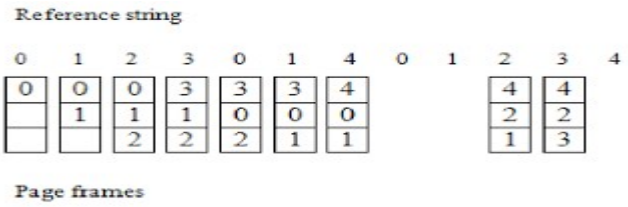




REPLACEMENT POLICY

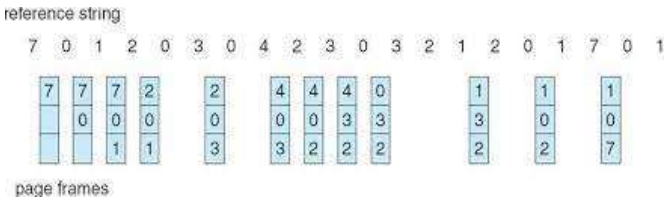
1. FIFO (First-In-First-Out)

- Oldest page in the cache is the first to be replaced.
- Track the order of page arrivals.
- When a page needs replacement, replace the oldest one.
- Simple and easy to implement.
- May not consider recent usage, leading to suboptimal performance.



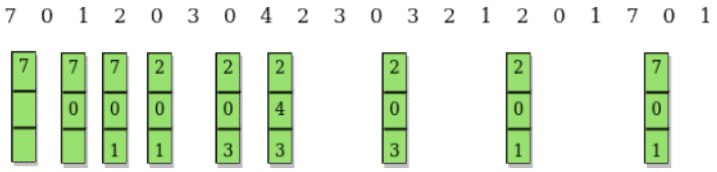
2. LRU (Least Recently Used)

- The least recently used page is replaced.
- Track the order of page accesses.
- Replace the least recently used page when needed.
- Adapts well to temporal locality.
- Implementation complexity.
- Overhead in tracking access history.



3. Optimal

- Theoretical algorithm with complete knowledge of future page accesses.
- Select the page not used for the longest period in the future.
- Serves as a benchmark for other algorithms.
- Theoretically optimal decisions.
- Impractical in real-world scenarios.
- Not implementable due to the need for future access knowledge.



Auxiliary memory

- Auxiliary memory, also known as secondary storage or secondary memory, refers to non-volatile storage devices used to store data permanently or semi-permanently.
- Unlike primary memory (RAM), which is volatile and loses its contents when the power is turned off, auxiliary memory retains data even when the power is off. Common types of auxiliary memory include:

- **Hard Disk Drives (HDDs):**
- HDDs are magnetic storage devices with spinning disks and read/write heads.
- High capacity, relatively slower access times compared to RAM

- **Solid State Drives (SSDs):**
  - SSDs use NAND-based flash memory for storage, providing faster access times than HDDs.
  - Faster read/write speeds, lower mechanical failure risk compared to HDDs.

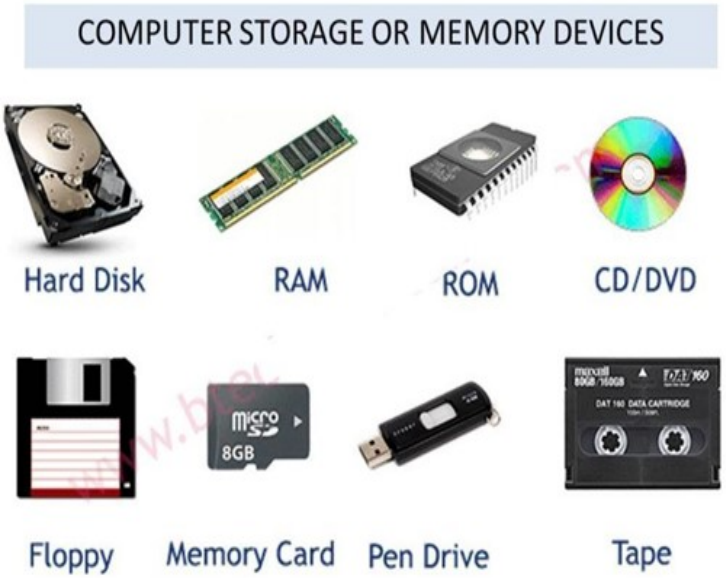
- **Optical Storage (CDs, DVDs, Blu-ray):**
  - Optical discs store data through pits and lands on the disc surface, read using lasers.
  - Portable, widely used for distribution, slower access compared to HDDs and SSDs.

- **USB Flash Drives:**
  - Flash drives use NAND-based flash memory for portable data storage.
  - Portable, small form factor, faster than optical storage.

- **Magnetic Tapes:**
  - Magnetic tapes use sequential access for data storage.
  - Used for backup and archival purposes, slower access times.

- **Memory Cards (SD cards, microSD cards):**
  - Compact storage devices commonly used in cameras, smartphones, and other devices.
  - Portable, small form factor, widely used in mobile devices.

- **External Hard Drives:**
  - Portable hard drives for additional storage capacity.
  - Characteristics: Combines high capacity with portability.



Virtual Memory

- Virtual Memory is a storage scheme that creates an illusion of a large main memory by treating a portion of secondary memory as if it were the main memory.
- Enables loading larger processes than available main memory.
- Allows the execution of multiple processes with an illusion of abundant memory.
- Instead of loading one large process entirely into main memory, the OS loads different parts of multiple processes.
- Supports efficient multitasking.
- Enhances the illusion of abundant memory for user processes.
- When insufficient main memory for page loading:
- OS identifies least-used or unreferenced RAM areas.
- Copies those areas to secondary memory to create space for new pages.
- All procedures happen automatically, providing the computer with a perception of virtually unlimited RAM.

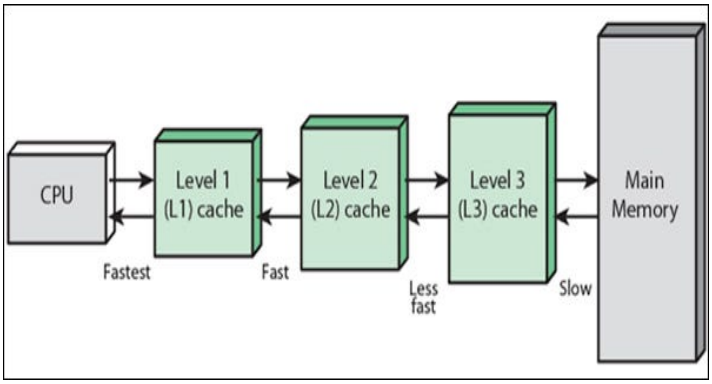
Interleaved Memory:

- Interleaved memory is a memory organization technique where consecutive memory addresses are distributed across multiple memory modules or banks.
- Data is stored in a round-robin fashion across the modules, allowing for parallel access to multiple modules simultaneously.
- Enhances memory bandwidth by enabling concurrent access to different memory banks.
- Distributes memory access load across modules.
- Increased complexity in memory controller design.

Associative Memory (CAM):

- Associative memory, or Content-Addressable Memory (CAM), is a type of computer memory that allows content-based addressing. It enables the retrieval of data based on its content rather than a specific memory address.
- CAM compares data input with stored content and returns the corresponding memory location.
- Enables fast data retrieval based on content.
- Useful in applications like network routing tables and cache memory.
- Higher cost and complexity compared to traditional memory.

Memory organization :



Question 1: A memory system has 2048 memory locations. Calculate the number of address lines needed.
Question 2: A RAM chip has 4096 cells, each storing 8 bits of data. Determine the number of output lines.

Question 3: An EPROM has 8192 memory locations, and each location can store 16 bits. Calculate the number of input lines required.
Question 5: A ROM chip has 256 address lines. Calculate the total number of unique memory locations it can address.
Answers:

Question 1:

Number of Address Lines=log 2(Number of Memory Locations)
Number of Address Lines=log2 (Number of Memory Locations)
Number of Address Lines=log 2(2048)=11Number of Address L ines=log2 (2048)=11 address lines.

Question 2:

Number of Output Lines=Data WidthNumber of Output Lines=Da ta Width
Number of Output Lines=8Number of Output Lines=8 output lines.

Question 3:

Number of Address Lines=log2 (8192)=13

Question 5:

Number of Memory Locations=2Number of Address Lines
Number of Memory Locations=2Number of Address Lines
Number of Memory Locations=2256Number of Memory Location s=2256 unique memory locations (a very large number).

PERIPHERAL DEVICES

- Peripheral devices are external components connected to computers through interfaces like **USB, HDMI** and **Ethernet**.
- They serve various functions such as input, output, storage, and communication.
- **Common examples include:**
- **Input Devices:**Keyboards,Mice,Scanners,Digital cameras
- **Output Devices:**Monitors,Printers,Projectors,Speakers
- **Storage Devices:**Hard drives,Solid-state drives (SSDs),USB flash drives,External hard drives
- **Communication Devices:**Network cards,Modems,Wireless adapters
- **Other Devices:**External CD/DVD drives,Webcams,Microphones

I/O interfaces

- I/O interfaces facilitate communication between a computer system and peripheral devices for input and output.
- They include physical connectors, communication protocols, and software interfaces:
- **Physical Connectors:**
  - ✓ USB (Universal Serial Bus)
  - ✓ HDMI (High-Definition Multimedia Interface)
  - ✓ VGA (Video Graphics Array)
  - ✓ Ethernet
  - ✓ Thunderbolt
- **Communication Protocols:**
  - ✓ Serial communication (RS-232)
  - ✓ Parallel communication
  - ✓ SATA (Serial advanced tech. attachment) for storage devices
  - ✓ PCIe (Peripheral Component Interconnect Express) for high-speed data transfer
- **Software Interfaces:**
  - ✓ Device drivers for OS communication with specific hardware
  - ✓ APIs (Application Programming Interfaces) for software and peripheral interaction

key requirements for an I/O module

- **Compatibility:**The I/O module must be compatible with the computer system's architecture, addressing, data formats.
- **Addressability:**The module should have a unique address that the CPU can use to identify and communicate with it.
- **Data Buffering:**Adequate buffering capabilities to temporarily store data during the transfer between the CPU and the peripheral device preventing data loss or overflows.
- **Power Management:**Support for power management features to optimize energy consumption during idle periods.

- **Data Rate Matching:**
- The ability to match the data transfer rates between the CPU and the peripheral device to avoid bottlenecks and ensure smooth communication.
- **Interrupt Handling:**
- Support for interrupt handling mechanisms to inform the CPU when the peripheral device requires attention or when a data transfer is complete.
- **Error Handling:**
- Error detection and correction mechanisms to ensure data integrity during the transfer process.
- **Control and Status Registers:**
- Dedicated registers for controlling the I/O operation and storing the status of the I/O module, allowing the CPU to monitor and manage the I/O process.
- **Scalability:**
- Scalability to accommodate a variety of different types and numbers of peripheral devices.

I/O Bus

- Communication pathway for CPU and peripheral devices.
- Facilitates CPU communication with external devices.
- Enables dedicated data transfer to/from peripherals.
- Specific connectors or slots on the motherboard.
- Varying data transfer speeds and protocols.
- **Examples:** PCI ,USB , SATA

I/O Commands

- CPU instructions for data flow with peripherals.
- Directs I/O bus for specific actions (read, write, control).
- Coordinates timing and sequencing of data transfer.
- **Types:**
- **Read**: Data transfer from peripheral to CPU.
- **Write**: CPU instructs I/O for data transfer to peripheral.
- **Control**: Manages overall peripheral operation.
- Execution:
- Generated by CPU during program execution.
- Executed through I/O instructions in the program.

Interrupts

- Interrupts are signals sent to the CPU, temporarily halting its current execution to handle a specific event or request from a peripheral or software.
- Allows the CPU to respond promptly to events without constant polling.
- Enhances multitasking and responsiveness in computer systems.

Types of Interrupts:

- **Vectored Interrupts:**
  - ✓ Unique vector for each source.
  - ✓ Directly jumps to specific ISR.
- **Non-Vectored Interrupts:**
  - ✓ Common ISR entry point.
  - ✓ Requires additional identification.
- **Maskable Interrupts:**
  - ✓ Can be disabled by the CPU.

- ✓ Allows selective blocking.
- ✧ **Non-Maskable Interrupts:**
  - ✓ Cannot be disabled.
  - ✓ Indicates critical events.

- ✧ **Hardware Interrupts:**
  - ✓ External devices signaling.
  - ✓ Asynchronous requests.
- ✧ **Software Interrupts:**
  - ✓ Generated by software.
  - ✓ Requests OS services.

Identifying the Source of Interrupts:

1.Polled Interrupts:

- ✧ Iterative checking of status or interrupt request lines of devices to identify pending interrupts.
- ✧ Systematically polling devices to detect interrupt requests.
- ✧ CPU periodically checks each device for interrupt status, determining if an interrupt is pending.

2.Vectored Interrupts:

- ✧ Each interrupt source is assigned a unique vector stored in an interrupt vector table.
- ✧ The interrupting device sends a signal with its specific vector to the CPU.
- ✧ CPU uses this vector to directly jump to the corresponding **Interrupt Service Routine (ISR).**
- ✧ Fast and direct identification of interrupt sources.
- ✧ Devices signal their interrupt through a unique vector, enabling direct access to specific ISRs.

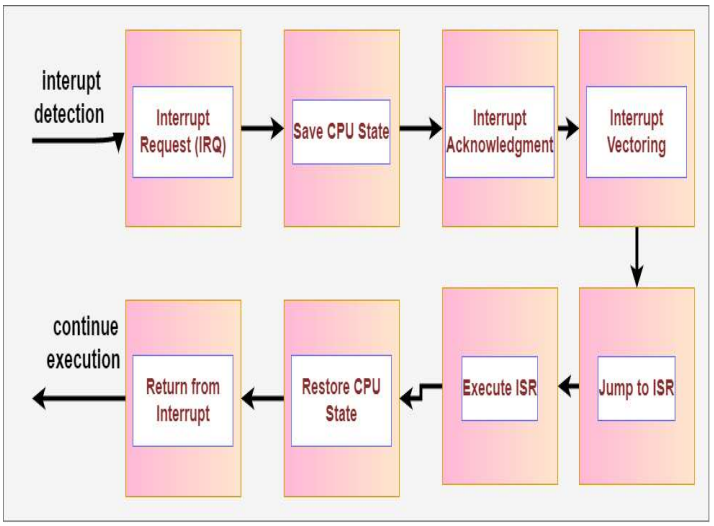
Resolving the Source of the Interrupt:

1.Polling Method:

- ✧ Iterative checking of devices to determine the source of the interrupt.
- ✧ CPU sequentially polls each device until the interrupting device is identified.
- ✧ Systematically resolving the interrupt source by checking each device in sequence.
- ✧ Devices are polled in a predetermined order until the interrupting device is found.

2.Daisy Chaining:

- ✧ Devices are connected in a chain, each with a unique priority level.
- ✧ When an interrupt occurs, the CPU identifies the source based on the priority level of the interrupting device in the chain.
- ✧ Efficiently resolving interrupt sources with a prioritized chaining mechanism.
- ✧ Devices are arranged in a daisy chain, and the CPU identifies the interrupting device by its position in the chain.



steps for handles the intrerrupts :

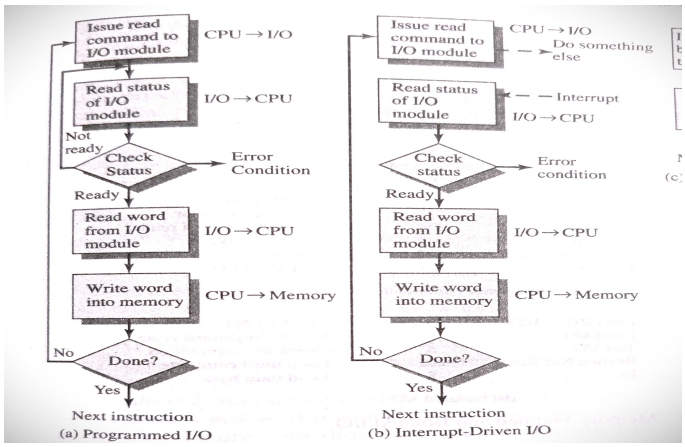
- ✧ **Interrupt Detection:** Continuous monitoring of interrupt lines or flags for device or system requests.
- ✧ **Interrupt Request (IRQ)**: Device signals CPU with an interrupt request, indicating an occurrence.
- ✧ **Save CPU State:** CPU preserves its current state, including program counter and register contents.
- ✧ **Interrupt Acknowledgment:** CPU confirms interrupt receipt, potentially sending a signal back to the device.
- ✧ **Interrupt Vectoring:** CPU uses interrupt vector to locate the ISR's address for specific interrupt source.
- ✧ **Jump to ISR:** CPU jumps to ISR's address specified by interrupt vector or fixed location for non-vectored interrupts.
- ✧ **Execute ISR:** ISR performs necessary operations, e.g., data read/write, flag updates.
- ✧ **Restore CPU State:** CPU reinstates saved state, including program counter and register contents.
- ✧ Return from Interrupt: CPU executes return-from-**interrupt instruction,** resuming control at the interrupted point.
- ✧ **Continue Execution:** With interrupt handled and CPU state restored, normal program or task execution continues.

Programmed I/O (PIO):

- ✧ PIO involves CPU actively controlling data transfer between peripheral and memory.
- ✧ CPU issues commands to the I/O device and waits for operation completion.
- ✧ Inefficiency arises as the CPU is occupied throughout I/O, potentially causing performance bottlenecks.
- ✧ **Advantages:**
  - ✓ Simplicity ,Controlled by the CPU
- ✧ **Disadvantages:**
  - ✓ High CPU Overhead ,Slower Performance

interrupt-Driven I/O (IDIO):

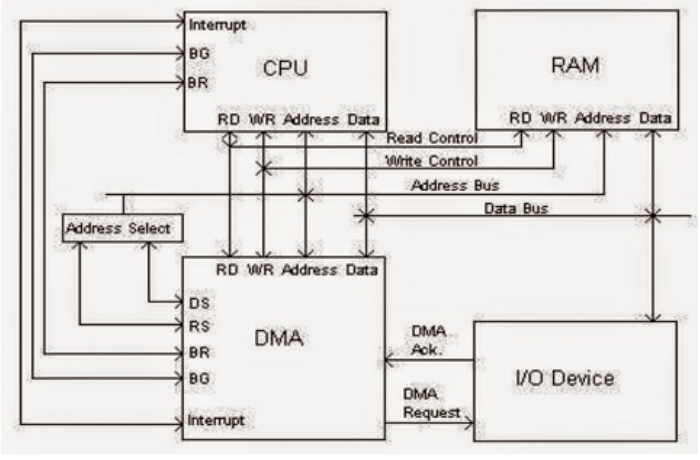
- ✧ IDIO, or interrupt-driven I/O, frees the CPU from actively waiting for I/O completion.
- ✧ I/O device generates an interrupt when ready to transfer data or on specific events.
- ✧ CPU, upon interrupt, transfers control to a dedicated ISR for handling the I/O event.
- ✧ IDIO enables the CPU to multitask during I/O operations, enhancing efficiency over programmed I/O.
- ✧ **Advantages:**
  - ✓ Efficiency ,Asynchronous Operation ,Flexible and Responsive
- ✧ **Disadvantages:**
  - ✓ Complexity,Potential for Latency ,Resource Intensive



Direct Memory Access (DMA):

- DMA is a technique that allows peripherals to transfer data directly to and from the system's memory without the intervention of the CPU.
- A DMA controller is used to manage the data transfer independently of the CPU.
- This method is highly efficient as it offloads the data transfer task from the CPU, allowing the CPU to focus on other processing tasks.
- DMA is particularly useful for large data transfers and is commonly employed in scenarios like disk I/O, network data transfer, and graphics processing.
- **Advantages:**
  - ✓ Increased Throughput: improved throughput and overall system performance.
  - ✓ Reduced CPU Overhead
  - ✓ Efficient for Block Transfers
- **Disadvantages:**
  - ✓ Complexity ,Limited Peripheral Support

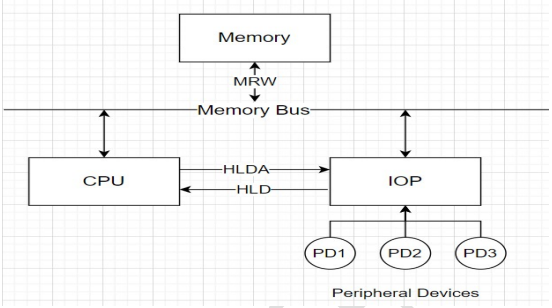
Working of dma



- DMA controller becomes bus master.
- CPU sends DMA select signal to controller.
- DMA requests bus by raising bus request signal.
- Controller acknowledges DMA request.
- CPU grants bus control to DMA by raising bus grant signal.
- CPU provides transfer details to DMA.
- CPU resumes execution; DMA is now bus master.
- DMA interacts with memory and devices for data transfer.
- DMA disables bus request after completion.
- CPU disables bus grant, regaining bus control.

I/O Processor

- I/O Processor is specialized in overseeing Input/Output operations in a computer system.
- Its primary function is to transfer I/O tasks away from the CPU, enhancing system efficiency and performance.
- Found in various computing environments, from personal computers to large-scale servers.



key characteristics and functions of I/O processors:

1.Offload CPU
2.Parallel Processing
3.Dedicated I/O
4. DMA Support
5.System Bus Connection:
6. Interrupt Handling:
7. Specialized Interfaces:
8. Control Buffering and Caching:
9. Communication Protocols

I/O channel

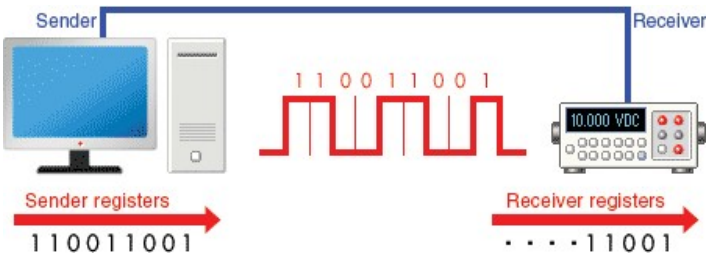
- An I/O channel, or controller, oversees communication between the CPU and peripherals, managing input and output operations.
- Responsible for coordinating data transfer between the CPU and external devices.
- Manages tasks like data transfer timing, buffer utilization, communication protocols, and interrupt handling.

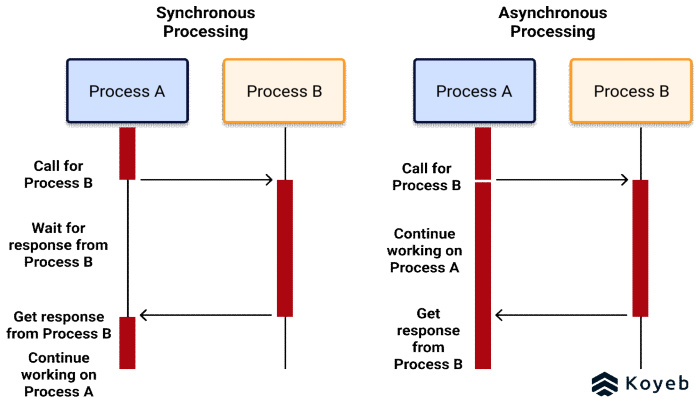
File Transfer Operation Workflow:

- CPU initiates file transfer by sending a request to the I/O Channel.
- I/O Channel, controlled by its controller, manages data transfer, handling timing and protocol for communication with the HDD.
- During read or write operations, I/O Channel may use a buffer for temporary data storage.
- I/O Processor monitors the process, handling interrupts from the I/O Channel and managing overall I/O flow.
- Leveraging DMA capabilities, I/O Processor efficiently transfers data between HDD and system memory with minimal CPU involvement.
- After file transfer completion, I/O Processor signals CPU through interrupts, indicating data readiness for further

Serial communication

- Serial communication uses one or two transmission lines for sending and receiving data, transmitting one bit at a time.
- Enables continuous, bit-by-bit data transmission.
- Merits include minimal signal wires, reducing wiring and equipment costs.





Modes of serial communication

1.Simplex:

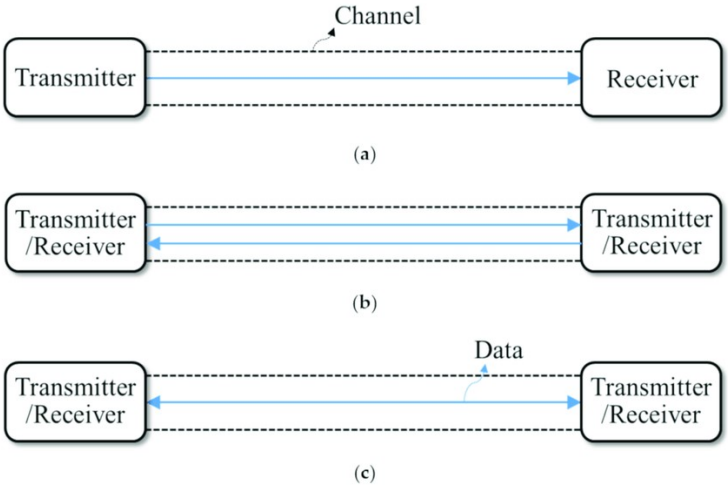
- ✓ Unidirectional communication.
- ✓ Data flows in one direction only, from sender to receiver.
- ✓ **Example** : tv , radio , keyboard

2.Half-Duplex:

- ✓ Communication in both directions, but not simultaneously.
- ✓ Devices take turns transmitting and receiving.
- ✓ **Example** : walky-tolky

3.Full-Duplex:

- ✓ Simultaneous communication in both directions.
- ✓ Allows for data transmission and reception concurrently.
- ✓ **Example** :phone conversation



Asynchronous Communication:

- ✧ Data is sent without a shared clock signal.
- ✧ Each character is framed by start and stop bits to synchronize.
- ✧ No common time reference between sender and receiver.
- ✧ Start and stop bits mark the beginning and end of each character.
- ✧ Widely used in serial communication.
- ✧ **Example**: email sending /receiving

Synchronous Communication:

- ✧ Data transmission is synchronized using a shared clock signal.
- ✧ Sender and receiver use the same clock signal for timing.
- ✧ Requires a clock signal for synchronization.
- ✧ Data is sent in chunks or frames, aligned with the clock.
- ✧ More complex but allows higher data rates.
- ✧ Often used in parallel communication.
- ✧ **Example**: phone conversation