# Computer Organisation and Architecture

Computer Architecture: conceptual design + fundamental operational structure

Computer Organisation: implementation of the architecture is called as organisation.
↓
it deals with interconnection of physical devices to improve the performance

| CO | CA |
|---|---|
| → I/O organisation | → CPU design |
| → Memory organisation | → instructions |
| → Performance | → addressing mode |
| | → data format |

- Computer Organisation basically refers to the structural design and functionality of a computer system.
- Basically, it explains how the computer hardware components interact with each other to perform execution of program efficiently.

e.g.- what happens when we press the power button on the system?

Step 1: Power Supply Activation

i) Pressing the power button → electrical signal generation
↓
it will convert the electrical signal into the usuable power (for the H/w devices)

Step 2: Power On Self Test (POST)

BIOS (located on motherboard) → runs the POST process
↓
of CPU, RAM, etc.

Step 3: Initialization of the firmware
- (i) Operating System → booting
- (ii) Identification of the hardware components
- (iii) Booting Order Determination

Step 4: Loading the bootloader
→ small program responsible for booting of the operating system
→ stored on the storage devices

Step 5: Loading the Operating System
OS Kernel — loads into → memory

Step 6: System is ready to use with login screen

## Key Components of Computer Organisation

(I) CPU
- (i) Control Unit (CU)
- (ii) Arithmetic and Logic Unit (ALU)
- (iii) Registers

(II) Memory
- (i) Primary
- (ii) Secondary
- (iii) Cache

(III) Input/Output Devices

(IV) System Wires Bus
↓
collection of wires
→ provides pathway for data transmission
e.g. - data bus → carries data
control bus → carries control signals
address bus → carries memory addresses

(V) External Storage Devices

## Important Concepts of Computer Organisation

(i) Instruction Cycle
- Fetch
- Decode
- Execute
- Store

(ii) Instruction Set Architecture (ISA)
→ it explains how the software communicates with the hardware

(iii) Pipelining
→ it is a technique where multiple instructions are processed/run to increase the CPU performance simultaneously.

(iv) Memory Hierarchy
- → Registers
- → Cache
- → RAM
- → Hard-disk
- → Other External Storage Devices

(↓)
- increasing size
- decreasing access speed
- decreasing cost

(v) Parallelism
modern systems use parallel processing to perform the multiple tasks simultaneously

Q.) Differentiate b/w Pipelining and Parallelism?
In Pipelining, multiple stages of a process are executed in an overlapping manner.

whereas,

In Parallelism, multiple processes/tasks are executed simultaneously

Importance of Computer Organization

i) Increasing the resource utilization
ii) Performance Optimization
iii) System Design
iv) Problem Solving

Microarchitecture

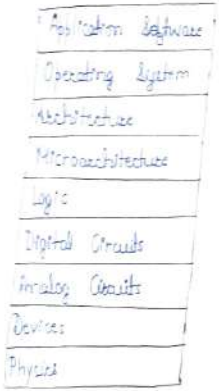



Fig. System Designing + Interaction b/w hardware & software

Physics: It utilises principles of electrical circuits
Devices: ALUs, Multiplexers, etc
Analog Circuits: continuous signals
Digital Circuits: discrete signals (0's and 1's)
Logic gates (like AND, OR, NOT, etc)
↓
main purpose — control the data flow and perform boolean operations

Microarchitecture: It deals with how to implement the architecture in H/w.
Architecture: It refers to the high-level design and functionality of the system while,
microarchitecture is the detailed implementation of the architecture focusing on how the components (ALUs, registers, caches) are organized and interact b/w the H/w

- Architecture is the programmer's view of machine while
Microarchitecture is the hardware designer's view.

Microarchitecture:
The computer's processors are designed and organized at the H/w level providing details about its components and their interface implements by mechanism of interaction for execution of the instructions

Different Microarchitecture for Single Architecture

(a) Single Cycle Microarchitecture → executes all necessary operations in a single cycle
(b) Multi Cycle Microarchitecture ↓ fetch, decode, execute, store/memory access
(c) Pipelined Microarchitecture

Single Cycle Microarchitecture

- Each instruction executes within a single cycle with all necessary operations like fetch, decode, execute, store

Multi Cycle Microarchitecture

- Each instruction is executed over multiple clock cycles with different stages of the instruction spread across these cycles.
- Therefore, it can be observed that each instruction uses same hardware but at different times.
- The same functional units (like ALU) are reused for different stages of the instruction reducing hardware duplication
- Only one instruction is processed at a time but it takes multiple cycles to complete it.
- Throughput — low (data is processed sequentially)

## Pipelined Microarchitecture

- Multiple instructions are executed concurrently and each instruction is involved with pipelined stages (occupying a different stage of the pipeline at some time)
- Throughput – High
  - e.g. – while one instruction is being fetched, another is decoded followed by execution and write-back to the memory
- It can be seen that different functional units are active simultaneously allowing higher h/w utilisation and increased throughput.

## Computer System Architecture

Deals: end-to-end structure of the system
↓
- completely processed

- It ensures execution of the different tasks
  ↓
  data process/information flow

Main Purpose: Simplification of the complex task

Outcome: Maximized performance and reduced cost
+
reduction in power consumption

Based on:

processor/general-purpose processor → computer system implemented

- General-purpose processors classified into 3 categories:

(i) Single-processor system: one main CPU → capable for executing general-purpose instructions.
+
instructions from the user (different microprocessor)

(ii) Multiprocessor system

(iii) Clustered system

## Single-Processor System

- In single processor system, there one main CPU which is capable of executing general-purpose instructions followed by set of instructions from user process
- Microprocessors are the specific processors that are present and capable of performing device-specific tasks
  - e.g. – human to machine language conversion
    (pressing of a key on keyboard)

## Multi-Processor System

- In this, two or more general-purpose processors are involved.
- Also called as "parallel systems" or "tightly coupled systems".
- In this, when two or more processors are involved, they are having closed communication,
  (capable of sharing buses, memory, clockcycles, etc.)
  - Closed Communication
  - Synchronization

Advantages:

(I) Increased throughput → analyzing performance of the computer system

(II) More economic

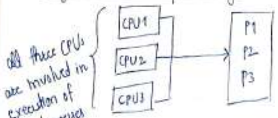
(III) Increased reliability → if any processor fails, multi-processor systems have backup, but in case of single-processor systems, it completely goes down

Types of multi-processor systems:

(I) Symmetric Multiprocessing

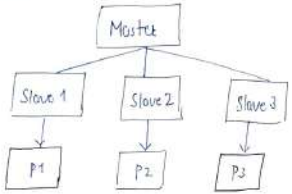
(II) Asymmetric Multiprocessing

### Symmetric Multiprocessing



## Asymmetric Multiprocessing

"Master-Slave Approach"




## Clustered Systems

- Like multiprocessor systems, clustered systems are involved together having multiple CPUs to accomplish different tasks.
- It provides high availability.
- Also structured as:
  (i) Symmetric
  (ii) Asymmetric

## Unsigned and Signed Integer Representation

Binary format: signed integers

3 ways:

(i) Sign and Magnitude
(ii) 1's complement
(iii) 2's complement

① Sign and Magnitude

Further divided into two types:
(i) Unsigned
(ii) Signed

- In signed integer, the integer with a positive or negative associated is represented.
- Since, the computer only understands binary format, therefore it is necessary to represent these signed integers in a binary format.

3 bits

$2^3$ possible combinations = 8

0 → 000
1 → 001
2 → 010
3 → 011
4 → 100
5 → 101
6 → 110
7 → 111

⇒ represent natural numbers
⇒ no bit reserved for representation of sign

Range: $0 - (2^n - 1)$

### Signed Integer

- MSB reserved for representation of sign


Range: $(-2^{n-1}+1)$ to $(2^{n-1}-1)$

Drawback: '0' is represented twice that can result in data overflow.

Rules: In signed integer representation, the following rules are followed

(i) MSB represents the sign of the integer

(ii) Magnitude is represented by the other (n-1) bits

(iii) If the no. is +ve, MSB = 0
-ve, MSB = 1

(i) Write signed bit representation for positive and negative numbers (3 bits)

| Unsigned | Signed |
|---|---|
| 0 → 0 0 0 | -3 → 1 1 1 |
| 1 → 0 0 1 | -2 → 1 1 0 |
| 2 → 0 1 0 | -1 → 1 0 1 |
| 3 → 0 1 1 | 0 → 1 0 0 (or) 0 0 0 |
| 4 → 1 0 0 | 1 → 0 0 1 |
| 5 → 1 0 1 | 2 → 0 1 0 |
| 6 → 1 1 0 | 3 → 0 1 1 |
| 7 → 1 1 1 | |

Drawbacks:

(i) For "0", there are two representations, i.e. −0 and +0 which should not be the case as 0 is neither +ve nor -ve.

(ii) Out of the "n" bits, we are able to utilise (n-1) bits (for magnitude)

(iii) For -ve numbers, sign extension does not work

(iv) Numbers are not in cyclic order, i.e. after the largest no., the next no. is not the least no.

- In other words, when numbers are arranged in a cyclic order after reaching the largest no., the next no. in the sequence should be the smallest no. to create loop-like structure

e.g. - sign representation of +5 and -5 in 4 bits and 5 bits

| 4 bits | 5 bits |
|---|---|
| +5 → 0 1 0 1 | 0 0 1 0 1 |
| -5 → 1 0 0 1 | 1 0 1 0 1 |

## 1's complement

Syntax: [Sign bit | 1's complement of the rest]

-7 → 1 0 0 0
Sign | 7's complement

- In 1's complement, -ve values are obtained by complementing each bit of the +ve no.

Rules:

(i) For +ve no., the representation rules are the same as the signed integer representation

(ii) For -ve no., write the +ve no. in binary and take its 1's complement

Drawback:

- For "0" there are two representations − "-0" and "+0"
- Out of $2^n$ bits for representation, only (n-1) bits are utilised.

## 2's complement

Syntax: [Sign bit | 2's complement]

7 → 1 1 1

2's complement → 1 0 0 1

∴ -7 = 1 0 0 1

Rules:

(i) For +ve no., the representation rules are the same as signed integer representation

(ii) For -ve no., there are two different ways we can represent the no.:

(a) write an unsigned representation of $(2^n - X)$ for $-X$ representation

(b) write representation of $+X$ and take 2s complement

(iii) The range of representation is $-2^{n-1}$ to $(2^{n-1}-1)$.

Range: $-2^{n-1}$ to $2^{n-1}-1$

Advantages:

- No ambiguity in the representation of "0"
- No.'s are in cyclic order.
- Range: $-2^{n-1}$ to $2^{n-1}-1$, using n-bits (range is high)

(i) Describe utilisation of logic gates used to build fundamental components like multiplexers, decoders and memory.

(ii) Also discuss how analog components integrated with digital components in microarchitecture.

(i) Logic gates are the basic building blocks that are used to build the various fundamental components like multiplexer, decoders and memory.

Multiplexer

- A multiplexer is a combinational circuit that consists of multiple inputs and a single output, that selects one of the various input signals and routes it to the single output path
- AND gate: used to enable the input lines based on the selection lines.
- OR gate: combines the outputs of the AND gates to give the final output
- NOT gate: used to invert the input signals

Decoder

- A decoder is a combinational circuit that is used to convert binary number system into other number systems.
- AND gate: used for, each (grouping) binary combination output.
- OR gate: used to combine AND gate outputs for final output
- NOT gate: used to invert input signals

Memory

- Memory units are used to store and retrieve binary data for future access
- Latches and Flip-Flops are the basic memory element used to store one bit of data, created using cross-coupled NAND/NOR gates

(i) Logic gates serve as the fundamental components to build components like multiplexers, decoders and memory in digital circuits.

Multiplexers

- Multiplexer is a combinational circuit that is used to select one input from multiple inputs based on the control signals (select lines) and routes it to the output.
- Logic gates like AND, OR, NOT are used to create the selection mechanism, ensuring only the appropriate input is selected/activated while the other inputs are disabled.

Decoders

- Decoder is a combinational circuit that is used to convert binary number system to other number systems.
- A single output is activated for a given input combination.
- Logic gates like AND, OR, NOT are used to create the mechanism to activate the corresponding output for a unique input combination.

Memory

- Memory elements are ~~sh~~ used to store data for future access.
- Latches and Flip-Flops are the basic memory elements used to store a single bit of data.
- Logic gates like NAND/NOR are used in cross-coupled fashion to design memory elements that can maintain a stable state, i.e. store data and also allows controlled state changes based on clock pulses.

(ii) Analog components are integrated with digital components in micro architectures to ensure communication with the real world. The integration points include.

(a) Data Conversion

- Analog-to-Digital Convertors (ADCs) are used to convert the continuous analog signals (from sensors) into discrete digital data for the computers to process.
- Digital-to-Analog Convertors (DACs) are used to convert discrete digital data back into continuous analog signals for analog devices.

(b) Clock Generation

- Analog components like oscillators are used to generate the clock signals required for the synchronized working of digital systems

(c) Power Management

- Analog components like voltage regulators are used to ensure the system gets a stable power supply, required for digital components to function efficiently.

(d) Signal Processing

- Analog components like filters and amplifiers are used to filter and amplify the signals, to ensure it is suitable for digital processing.

(e) Security

- Advanced microarchitectures incorporate hardware-based cryptographic modules (like AES), that use hardware (analog) elements for specific functions.
- True Random Number Generators (TRNGs) often use analog components to generate randomness from physical processes like thermal noise or voltage fluctuations, which can be then used in cryptographic techniques.

## Integer addition and subtraction

→ by ALU
→ using signed no.'s
→ using 2s complement

### Addition

- To add treat the numbers as if they were unsigned integers.
- If the result is positive, the 2s complement no. looks the same as the unsigned integer.
- If the result is negative, it appears as a (-ve) no in 2s complement form.
- Sometimes there is an extra bit beyond the word length, this extra bit is discarded.

* Overflow rule: if two no's are added and they both are (+ve) or both are (-ve), then overflow occurs iff the result has the opposite sign.

e.g.- (+6) + (+7) (case I: both no. are (+ve))

```
  00110
  00111
  ------
  01101
   (+ve)
```

- (+9) + (+7) (case II: addition of two (+ve) no. with overflow condition)

```
  01001
  00111
  ------
  10000  (Invalid)
   (-ve) → overflow
```

i) (+ve) > (-ve) (case III: addition of (+ve) and (-ve) no.)

(+13) + (-6)

```
  01101
  11010
  ------
 [1]0011
discarded
```

ii) (-ve) > (+ve)

(-15) + (+9)

```
  10001
  01001
  ------
  11010
```

Case III: addition of two (-ve) no's

• (-8) + (-4)

8 → 01000
-8 → 10111 + 1 = 11000

4 → 00100
-4 → 11011 + 1 = 11100

```
  11000
  11100
 1 10100
 ↑
discarded
```

Case IV: addition of two (-ve) no with overflow

• (-8) + (-9)

8 → 01001
-9 → 10110 + 1 = 10111

```
  11000
  10111
 1 21111 → Invalid
 ↑
discarded
```

(-ve) → overflow

e.g. - (i) (-7) + (+5) → 4 bits

7 → 0111
-7 → 1000 + 1 = 1001

```
  1001
  0101
  1110
```

(ii) (-3) - (+4)

```
  0011
  0100
  0111
```

(iii) (-5) + (+4)

```
  0101
  0100
 1 1001 → invalid (overflow)
```

(iv) (-7) + (-6)

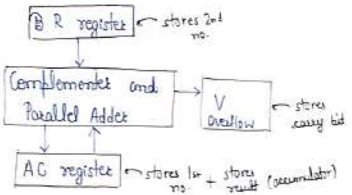
6 → 0110
-6 → 1001 + 1 = 1010

```
  1001
  1010
 1 0011 → invalid (overflow)
 ↑
discarded
```

Subtraction

A − B → A + 2s complement of B
= A + $\overline{B}$ + 1

Hardware Implementation of Addition and Subtraction




Mode value: 0 → addition
1 → subtraction

Addition: (i) if M=0, then h/w → adder

$$O + AC + BR \rightarrow AC$$

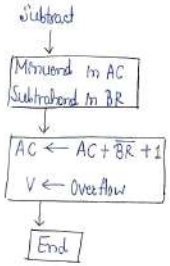
- mode control i/p is used to perform addition and subtraction operation
- if M=0, then the H/W implementation can work as adder
  M=1, then the H/W implementation can work as subtractor

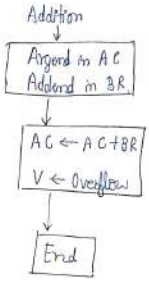
Subtraction: (ii) if M=1, then h/w → subtraction

- content of AC register goes to parallel adder
- content of BR register is sent to complementer first to perform the complement of BR

$$1 + AC + \overline{BR} \rightarrow AC$$

Flowchart




Q) Calculate M−S

(i) M = 2 , S = -7
(ii) M = 5 , S = -2
(iii) M = -5 , S = -2
(iv) M = 5 , S = 2
(v) M = -7 , S = 7
(vi) M = -6 , S = -4

(i) M − S = 2 − (−7)
= 2 + 7
= 0010
0111
1001 ← invalid

(ii) M − S = 5 − (−2)
= 5 + 2
= 0101
0010
0111

(iii) M − S = (−5) − (−2)
= (−5) + 2
= 1011
0010
1101

5 → 0101
−5 → 1010
+ 1
1011

(iv) M − S = 5 − 2
= 5 + (−2)
= 0101
1110
[1] 0011
discard

2 → 0010
−2 → 1101
+ 1
1110

(v) M − S = 7 − 7
= 7 + (−7)
= 0111
1001
[1] 0000
discard

7 → 0111
−7 → 1000
+ 1
1001

(vi) M − S = (−6) − (−4)
= −6 + 4
= 1010
+0100
1110

6 → 0110
−6 → 1001
+ 1
1010

## Multiplication using Add and Shift Method

Calculate 6 × 3.

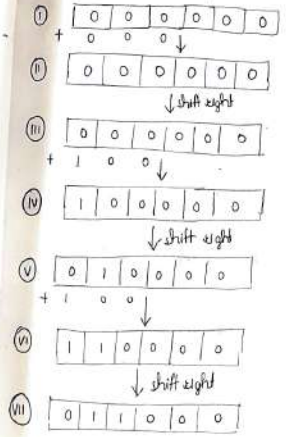
0 1 1 0 ← Multiplicand
× 0 0 1 1 ← Multiplier
0 1 1 0 ← partial product
0 1 1 0 x
0 0 0 0 x x
0 0 0 0 x x x
0 0 1 0 0 1 0 = 18

Calculate 4 × 6

1 0 0
1 1 0
000
100x
100xx
1100 0

∵ n = 3
∴ 2n = 6

(initialized with 0)
6 bit register




- no of registers required:
  (i) store individual results obtained in each step (partial products) in a separate register and then add them to get the final result, but this methodology isn't used as the cost'll become very high
  (ii) use single register and apply → add, shift right operations
- length of register = 2n bits (max) where, n → no. of bits in operands
- if we have n-bit multiplicand and n-bit multiplier, then total space required to store the result is 2n.
- no. of additions required completely depends on no. of 1s appearing in the multiplier.
- no. of shifts required depends on the no. of digits present in the multiplier

multiplicand ↑ 11 × 13 = 143 ← product (AQ)
└ multiplier

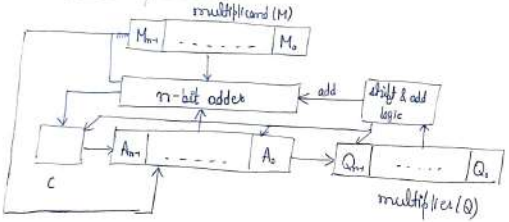
11 → 1011
13 → 1101

Tracing Table

+ accumulator initialized = 0
+ if $Q_0$ (LSB of Q) = 1 ⇒ perform add + shift right

| M | C | A | Q | Operation |
|---|---|---|---|---|
| 1011 | 0 | 0000 | 1101 | First Cycle: Add M with A (A = A+M) |
| | 0 | 1011 | 1101 | Shift Right: CAQ |
| | 0 | 0101 | 1110 | |
| 1011 | 0 | 0010 | 1111 | Second Cycle: Shift right CAQ |
| 1011 | 0 | 1101 | 1111 | Third Cycle: A = A+M |
| | 0 | 0110 | 1111 | Shift Right: CAQ |
| 1011 | 01 | 0001 | 1111 | Fourth Cycle: A = A+M |
| | 0 | 1000 | 1111 | Shift Right: CAQ |

Hardware Implementation




(i) The control logic reaches the bits of the multiplier (Q register) one at a time, starting with LSB.

(ii) If $Q_0 = 1$, add the value in the M register (multiplicand) to the A register
(iii) store the result back in A register
(iii) the C register handles the carry bit generated from the addition.

(iv) If $Q_0 = 0$, no addition is performed

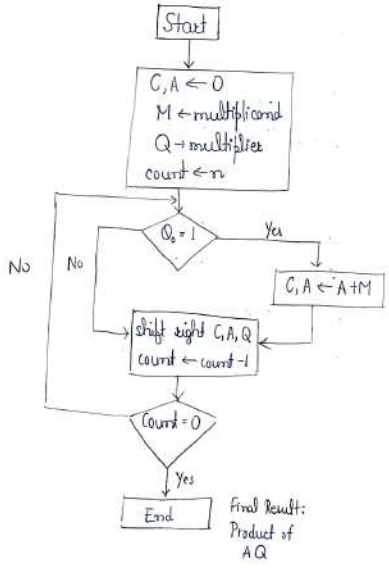
(ii) regardless of whether an addition occurs, the contents of C, A, Q register are shifted to the right by 1 bit. The steps are:
(a) The C-bit moves into the leftmost position of A-register
(b) The rightmost bit of A-register moves into the leftmost position of the Q register
(c) The rightmost bit of Q-register ($Q_0$) is discarded.

(iv) Repeat the process for each bit of the multiplier (Q-register). The result of the multiplication is stored across the A-register and Q register at the end.

Remark:

- This method avoids unnecessary additions when the multiplier bit is zero making it more efficient as compared to the traditional approach of adding the multiplicand repeatedly.

Flowchart



- At the start of the process, the count is set to the no. of bits in the multiplier.
- Each time the algorithm processes 1 bit of multiplier ($Q_0$), the count is reduced by 1.
- Basically, it tracks how many bits are left.
- When the count reaches 0, it means all bits of the multiplier have been processed and the algorithm ends.

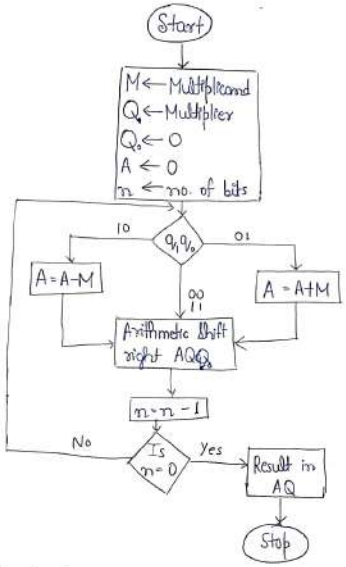
## Addition and Subtraction with signed magnitude data

(i) A → (+ve), B → (+ve)
then actual operation, A−B/A+B

(ii) A → (+ve), B → (+ve)
or,
A → (−ve), B → (+ve)
then, opposite operation will be performed

| Operations | Add Magnitude | Subtract Magnitude | | |
|---|---|---|---|---|
| | | A>B | A<B | A=B |
| (i) (+A)+(+B) | (A+B) | | | |
| (ii) (+A)+(−B) | | sign of A +(A−B) | −(B−A) | +(A−B) |
| (iii) (−A)+(+B) | | −(A−B) | +(B−A) | +(A−B) |
| (iv) (−A)+(−B) | −(A+B) | | | |
| (v) (+A)−(+B) | | +(A−B) | −(B−A) | +(A−B) |
| (vi) (+A)−(−B) | +(A+B) | +(A+B) | +(B+A) | +(A−B) |
| (vii) (−A)−(+B) | −(A+B) | −(A+B) | −(B+A) | +(A−B) |
| (viii) (−A)−(−B) | | −(A−B) | +(B−A) | +(A−B) |

## Booth's Algorithm




Q.) (−7) X (+3)
multiplicand, multiplier

M = 1001 Q = 0011 n = 4
−M = 0111

| n | A | Q | $Q_0$ | Action/Comment |
|---|---|---|---|---|
| 4 | 0000 | 0011 | 0 | initialize |
| | 0111 | 0011 | 0 | A = A−M |
| | 0011 | 1001 | 1 | arithmetic right shift AQ$Q_0$ |
| 3 | 0001 | 1100 | 1 | arithmetic right shift AQ$Q_0$ |
| 2 | 1010 | 1100 | 1 | A = A+M |
| | 1101 | 0110 | 0 | arithmetic right shift AQ$Q_0$ |
| 1 | 1110 | 1011 | 0 | arithmetic right shift AQ$Q_0$ |

21 → 01 01 01
−21 → 10 10 10 + 1
10 10 11
11 01 01 1

(-4) X (-2) -M= 0100 Q=1110 n=4

| n | A | Q | $Q_0$ | Action/Comment |
|---|---|---|---|---|
| 4 | 0000 | 1110 | 0 | initialize |
| | 0000 | 0111 | 0 | right shift |
| 3 | 0100 | 0111 | 0 | A=A-M |
| | 0010 | 0011 | 1 | right shift |
| 2 | 0001 | 0001 | 1 | right shift |
| 1 | 0000 | 1000 | 1 | right shift |

Q.) 3 X (-4)

M=0011 -Q= 0100

-M= 1101 Q= 1011 n=4

1 1

1100

| n | A | Q | $Q_0$ | Action/Comment |
|---|---|---|---|---|
| 4 | 0000 | 1100 | 0 | initialize |
| | 0000 | 0110 | 0 | right shift |
| 3 | 0000 | 0011 | 0 | right shift |
| 2 | 1101 | 0011 | 0 | A=A-M |
| | 1110 | 1001 | 1 | right shift |
| 1 | 1111 | 0100 | 1 | right shift |

## Unsigned Binary Division

Dividend

1011 ) 10010011 ( 00001101

Divisor − 1011↓ ↑ Quotient

1110

1011

X1111

Partial remainder 1011

0100 ← Remainder

1011 > 1 — 0

1011 > 10 — 0

1011 > 100 — 0

1011 > 1001 — 0

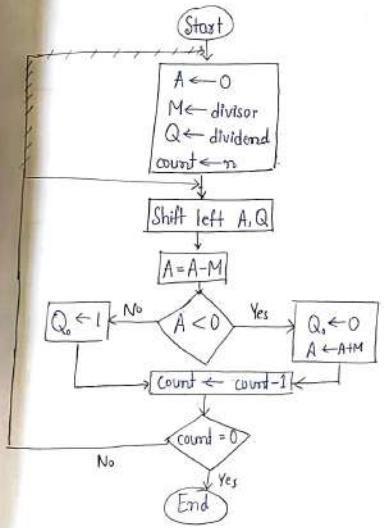
10010 > 1011 — 1

1110 > 1011 — 1

0111 < 1011 — 0

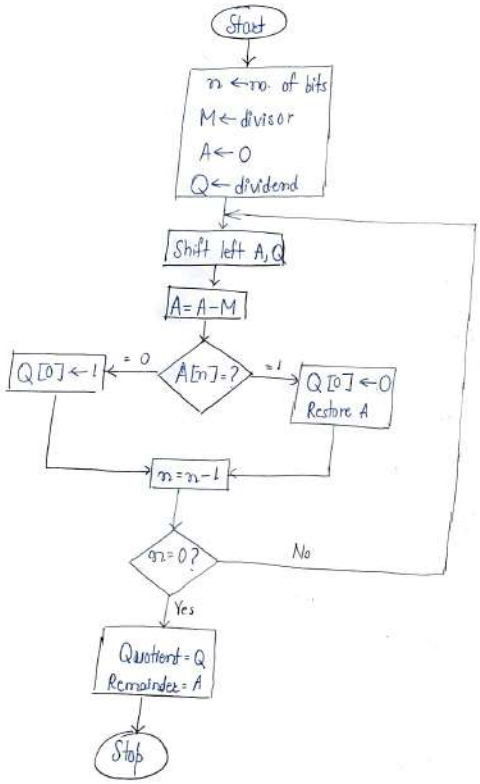
1111 > 1011 — 1

Flowchart:



## Restoring Division Algorithm

Flowchart:




Remarks:

(i) Size of A and M is n+1.
e.g. – if n = 4, A and M are 5 bits each.

(ii) Shift left: MSB of A → lost

(iii) A[n] → MSB of A

e.g. – 11 (dividend) / 3 (divisor) → 4 bits

result: 3 (quotient)
2 (remainder)

M = 3 = 00011
−M = 11101

Q = 1011
A = 00000

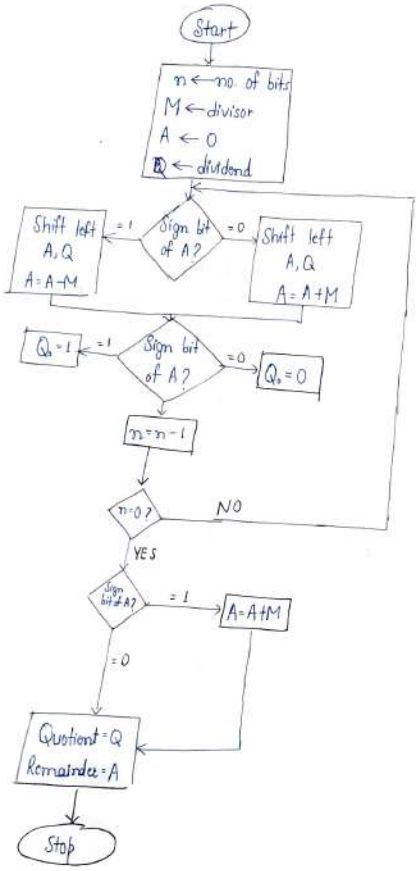
Tracing Table

| n | M | A | Q | Action |
|---|---|---|---|---|
| 4 | 00011 | 00000 | 1011 | Initialize |
| | | 00001 | 011_ | Shift left A,Q |
| | | 11110 | 011_ | A = A − M |
| | | 00001 | 0110 | Q[0] ← 0, Restore A |
| 3 | | 00010 | 110_ | Shift left A,Q |
| | | 11101 | 110_ | A = A − M |
| | | 00010 | 1100 | Q[0] ← 0, Restore A |
| 2 | | 00101 | 100_ | Shift left A,Q |
| | | 00010 | 100_ | A = A − M |
| | | 00010 | 1001 | Q[0] ← 1 |
| 1 | | 00101 | 001_ | Shift left A,Q |
| | | 00010 | 001_ | A = A − M |
| | | 00010 | 0011 | Q[0] ← 1 |

Quotient = 0011
Remainder = 00010

## Non-restoring Division Algorithm

### Flowchart




e.g - 11 (dividend) / 3 (divisor) → 4 bits
M = 00011 , -M = 11101

Tracing Table

| n | M | A | Q | Action |
|---|---|---|---|---|
| 4 | 00011 | 00000 | 1011 | Initialize |
| | | 00001 | 011_ | Shift left A,Q |
| | | 00100 | 011_ | A = A+M |
| | | 00100 | 0110 | $Q_0 = 0$ |
| 3 | | 01000 | 110_ | Shift left A,Q |
| | | 01011 | 110_ | A = A+M |
| | | 01011 | 1100 | $Q_0 = 0$ |
| 2 | | 10111 | 100_ | Shift left A,Q |
| | | 11010 | 100_ | A = A+M |
| | | 11010 | 1001 | $Q_0 = 1$ |
| 1 | | 10101 | 001_ | Shift left A,Q |
| | | 10010 | 001_ | A = A-M |
| | | 10010 | 0011 | $Q_0 = 1$ |
| 0 | | 10101 | 0011 | A = A+M |

# IEEE 754 Floating Point Representation

(i) Fixed and Floating Point Representation

(ii) Fixed Point Representation

$\pm$ 8 100.00000

↑ decimal/radix point

Eg. In a fixed point representation, both (+ve) and (-ve) no. can be represented by assuming a fixed position of decimal/binary followed by fractions

- This method has some limits, i.e it can't handle very large no. or very small fractions.

e.g. - when dividing two large no., part of the result could be lost.

- To resolve this problem, scientific representation or floating point no. can be adopted

## Floating Point Number

- It refers to the fact that a number's radix can float anywhere relative to its significant digits

Format for storing:

$\pm S \times B^{\pm E}$

- S → significand / mantissa
- B → base
- E → exponent

- Base will not be represented, only significand/mantissa.
- Therefore, to represent very small no. using fewer bits, floating point representation is suitable and useful.

Normalization: To standardize FP representation in computer memory.

- (i) Explicit
- (ii) Implicit

e.g. - (i) $101.1 \longrightarrow 0.1011 \times 2^{3}$

Mantissa


Formula:

$(-1)^{S} \times 0.M \times 2^{E}$

sign bit — exponent — mantissa

e.g. - 5.875

$5 \rightarrow 101$

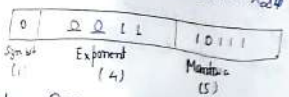
$0.875 \times 2 = 1.750$

$0.750 \times 2 = 1.500$

$0.500 \times 2 = 1.000$

$\therefore (5.875)_{10} = (101.111)_{2}$

Normalize (explicit) $\rightarrow 0.101111 \times 2^{3}$



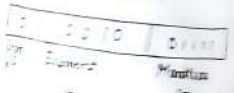

Solutions: (1) Increase size of mantissa → NOT reliable

(2) Implicit normalization

## Implicit Notation for Normalization

$1.01111$

Normalize $\therefore 1.01111 \times 2^{2}$

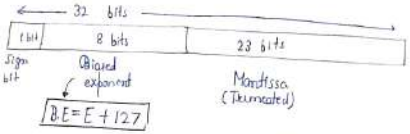


# IEEE 754 Floating Point Representation

(i) 32-bit format
(ii) 64-bit format
(iii) 128-bit format

754 → specification to represent the binary no.'s

## (i) 32-bit format



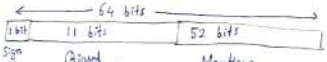

BE = E + 127

- Range for 2s complement: $-2^{n-1}$ to $2^{n-1}-1$
  - $-128$ to $127$
  - Unconditionally add 128,
  - $\Rightarrow -128+128$ to $127+128$
  - $\Rightarrow$ 0 to 255

# Remarks:

- The exponent is stored in a biased format, it means that a fixed value called the "bias" is subtracted to get the real exponent.
- Biasing is done to avoid storing negative exponents directly and make comparisons easier.
- Since all exponents are positive, comparing FP numbers becomes simpler in H/W.

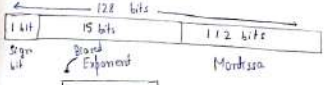
## (ii) 64-bit format




BE = E + 1023

- Range (2s complement): $-1024$ to $1023$
  - Unconditionally add 1024
  - $\Rightarrow -1024+1024$ to $1023+1024$
  - $\Rightarrow$ 0 to 2047

## (iii) 128-bit format




BE = E + 16383

- Range (for 2s complement): $-16384$ to $16383$
  - Adding 16384,
  - $\Rightarrow -16384+16384$ to $16383+16384$
  - $\Rightarrow$ 0 to 32767

Q.) Express the given decimal no. in IEEE 754 32-bit format

(i) $(76.125)_{10}$

| | | |
|---|---|---|
| 2 | 76 | 0 |
| 2 | 38 | 0 |
| 2 | 19 | 1 |
| 2 | 9 | 1 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

$0.125 \times 2 = 0.250$
$0.250 \times 2 = 0.500$
$0.500 \times 2 = 1.000$

$\therefore (76.125)_{10} = (1001100.001)_2$

$= 1.001100001 \times 2^6$

| | | |
|---|---|---|
| 2 | 133 | 1 |
| 2 | 66 | 0 |
| 2 | 33 | 1 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

| 0 | 10000101 | 001100001...zeros 0 |
|---|---|---|

(ii) $(3066.25)_{10}$

| | | |
|---|---|---|
| 2 | 3066 | 0 |
| 2 | 1533 | 1 |
| 2 | 766 | 0 |
| 2 | 383 | 1 |
| 2 | 191 | 1 |
| 2 | 95 | 1 |
| 2 | 47 | 1 |
| 2 | 23 | 1 |
| 2 | 11 | 1 |
| 2 | 5 | 1 |
| 2 | 2 | 0 |
| | 1 | |

$0.25 \times 2 = 0.50$
$0.50 \times 2 = 1.00$

$BE = E + 127$
$= 138$

$\therefore (3066.25)_{10} = (101111111010.01)_2$

$= (1.0111111101001 \times 2^{11})$

| | | |
|---|---|---|
| 2 | 138 | 0 |
| 2 | 69 | 1 |
| 2 | 34 | 0 |
| 2 | 17 | 1 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

| 0 | 10001010 | 01111101001...3 |
|---|---|---|

| 2 | 236 | 0 |
|---|---|---|
| 2 | 118 | 0 |
| 2 | 59 | 1 |
| 2 | 29 | 1 |
| 2 | 14 | 0 |
| 2 | 7 | 1 |
| 2 | 3 | 1 |
| | 1 | |

$0.75 \times 2 = 1.50$

$0.50 \times 2 = 1.00$

$\therefore (236.75)_{10} = (11101100.11)_2$

$= (1.110110011 \times 2^7)_2$

BE = E + 127

= 134

| 2 | 134 | 0 |
|---|---|---|
| 2 | 67 | 1 |
| 2 | 33 | 1 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

| 0 | 10000110 | 11011001...0 |
|---|---|---|

← 1 bit → ← 8 bits → ← 23 bits →

Q.) Express the given decimal no. in IEEE 754 64-bit format:

i) $(320.0625)_{10}$

| 2 | 320 | 0 |
|---|---|---|
| 2 | 160 | 0 |
| 2 | 80 | 0 |
| 2 | 40 | 0 |
| 2 | 20 | 0 |
| 2 | 10 | 0 |
| 2 | 5 | 1 |
| 2 | 2 | 0 |
| | 1 | |

$0.0625 \times 2 = 0.1250$

$0.1250 \times 2 = 0.2500$

$0.2500 \times 2 = 0.5000$

$0.5000 \times 2 = 1.0000$

| 2 | 1031 | 1 |
|---|---|---|
| 2 | 515 | 1 |
| 2 | 257 | 1 |
| 2 | 128 | 0 |
| 2 | 64 | 0 |
| 2 | 32 | 0 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

$\therefore (320.0625)_{10} = (101000000.0001)_2$

$= (1.01000000001 \times 2^8)$

BE = 8 + 1023 = $(1031)_{10}$

$= (10000000111)_2$

| 0 | 10000000111 | 0100000000001 ---- 0 |
|---|---|---|

ii) $[A0F3.0E3]_{16}$

$= (1010\ 0000\ 1111\ 0011.\ 0000\ 1110\ 0011)_2$

$= (1.010\ 0000\ 1111\ 0011\ 0000\ 1110\ 0011 \times 2^{15})_2$

B.E = 15 + 1023

= $(1038)_{10}$

= $(10000001110)_2$

| 2 | 1038 | 0 |
|---|---|---|
| 2 | 519 | 1 |
| 2 | 259 | 1 |
| 2 | 129 | 1 |
| 2 | 64 | 0 |
| 2 | 32 | 0 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

| 0 | 10000001110 | 0100000111100001110001... 0 |
|---|---|---|

iii) $[FAFA.01]_{16}$

$= (1111\ 1010\ 1111\ 1010.\ 0000\ 0001)_2$

$= (1.111\ 1010\ 1111\ 1010\ 0000\ 0001 \times 2^{15})_2$

B.E = 15 + 1023

= $(1038)_{10}$

= $(1000\ 0001\ 110)_2$

| 0 | 1000 0001 110 | 1111010111110100000001...0 |
|---|---|---|

← 52 bits →

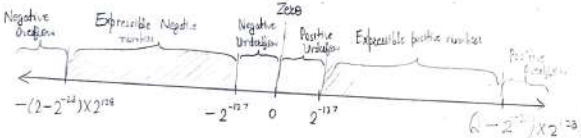
Q) Why is the mantissa truncated in floating point representation?

Ans:- The mantissa is truncated due to limited precision in storage.

- Since floating point no's must be fit in within a fixed no. of bits, only a certain no. of digits can be stored.
- So, if a no. has more precision than the available bits, truncation is performed to fit the number within the available bits.

Q) Explain positive and negative overflow and underflow using range specification graph.

- Overflow occurs when a number is too large (in magnitude) to be represented within the given range
  - Positive Overflow: Happens when the no exceeds the largest (+ve) value.
  - Negative Overflow: Happens when the negative no. is too large in magnitude.
- Underflow occurs when a number is too small (close to zero) to be represented within the given exponent range
  - Positive Underflow: A very small (+ve) no. becomes zero due to limited precision
  - Negative Underflow: A very small (-ve) no. becomes zero due to limited precision.




Range Specification for Single Precision Floats

*~~For 32-bit no,~~

* For normal non-zero floating point numbers,
  - In 32-bit format range is 1-254 (new) and actual range is -126 to 127
  - In 64-bit format range is 1-2046 (new) and actual range is -1022 to 1023
* For normal numbers, there is always 1 to the left of the decimal point in binary representation, i.e. 1 isn't actually stored it is implied. This allows the system to store 24 bits for single precision and 53 bits for double precision and 113 bits for 128-IEEE 754 format. Hence, it's a space saving technique in which leading 1 isn't represented and mantissa is allowed to store extra bits for higher precision.

Floating Point Arithmetic

$X = 0.3 \times 10^2$ $\quad Y = 0.2 \times 10^3$

(i) Addition

Formula: $X+Y = \left\{ X_s \times B^{X_E - Y_E} + Y_s \right\} \times B^{Y_E}, \forall\ X_E \leq Y_E$

$X_s \rightarrow$ significand of X
$Y_s \rightarrow$ " " Y
$X_E \rightarrow$ exponent of X
$X_Y \rightarrow$ " " Y
$B \rightarrow$ base

$\therefore X+Y = \left\{ 0.3 \times 10^{2-3} + 0.2 \right\} \times 10^3$
$= \left\{ 0.3 \times 10^{-1} + 0.2 \right\} \times 10^3$
$= 0.23 \times 10^3$
$= 2.30$

(ii) Subtraction

Formula: $X-Y = \left\{ X_s \times B^{X_E - Y_E} - Y_s \right\} \times B^{Y_E}, \forall\ X_E \leq Y_E$

$\therefore X-Y = \left\{ 0.3 \times 10^{2-3} - 0.2 \right\} \times 10^3$
$= \left\{ 0.03 - 0.2 \right\} \times 10^3$
$= -0.17 \times 10^3$
$= -170$

(iii) Multiplication

Formula: $X \times Y = \left\{ X_s \times Y_s \right\} \times B_E^{X_E + Y_E}$

$\therefore X \times Y = (0.3 \times 0.2) \times 10^5$
$= 0.06 \times 10^5$
$= 6000$

(iv) Division

Formula: $\frac{X}{Y} = \left\{ \frac{X_s}{Y_s} \right\} \times B_E^{X_E - Y_E}$

$\therefore \frac{X}{Y} = \frac{0.3}{0.2} \times 10^{-1}$
$= 0.15$

## Structural and Functional views of the system

structure → "what" ⟶ CPU, main memory, I/O devices
function → "how" ⟶ control unit
↳ responsible for executing micro instructions

* In the structural and functional views of the system, control unit runs micro instructions that defines its functionality.
* The structural components are CPU, main memory, I/O devices, system interconnection, etc.
* The functional components are data processing, storage, movement and control

Q) What is the decimal equivalent of floating point no.

i) 4=0400000H
↑ Hexadecimal

### Single Precision

Formula: $(-1)^S \times 2^{E-127} \times 1.M$

### Double Precision

Formula: $(-1)^S \times 2^{E-1023} \times 1.M$

$(4=0400000)_{16} = (0100=0000\ 0100\ 0000\ \ldots\ 0000)_2$
$= (1.00\ 0000\ 0100\ 0000 \ldots 0000 \times 2^{30})_2$
$\therefore [(-1)^0 \times 2^{30} \times 1.00000000100000\ldots0]_2$

BE = 30+127
= 157
$= (1001\ 1101)_2$

| 0 | 1001 1101 . | 00000010000...0 |
|---|---|---|

←23 bits→

ii) $(10)_{10}$

$(10)_{10} = (1010)_2$
$= (1.010 \times 2^3)_2$
$(-1)^0 \times 2^3 \times 1.010$

BE = 3+127
= 130

| 0 | 100 0000 10 | 010..... 0 |
|---|---|---|

←23 bits→

| 2 | 130 | 0 |
|---|---|---|
| 2 | 65 | 1 |
| 2 | 32 | 0 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

iii) 40A00000H

$= (0100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000)$
$= (1.00\ 0001\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{20})_2$

BE = 20+127
= 157

| 0 | 1001 1101 | 00 0001 0100 0000 0000 0000 |
|---|---|---|

←23 bits→

| 2 | 157 | 1 |
|---|---|---|
| 2 | 78 | 0 |
| 2 | 39 | 1 |
| 2 | 19 | 1 |
| 2 | 9 | 1 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| | 1 | |

# Unit-II

## Module 1: Control Unit and Instruction Set

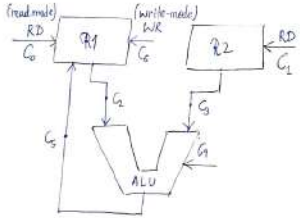
* Key functions that are performed by processors:
  - (i) Operations (OP code) } Instruction set
  - (ii) Addressing Modes
  - (iii) Registers
  - (iv) Input/Output Module Interface
  - (v) Memory Module Interface/Interactions } System Bus
  - (vi) Interrupts } System Bus + Operating System

- To understand how processor performs the above listed tasks, we need to look at the Control Unit (CU).
- The Control Unit manages and directs the processor's operations so that everything will work smoothly.

* Control Unit : It directs the main operations by sending control signals to the data path. It flows b/w I/O, CPU, memory subsystem, etc.

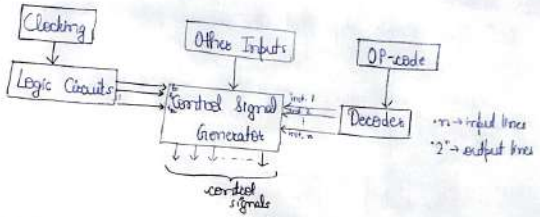
Working (CU):

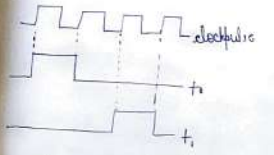
e.g.- ADD R1, R2
// R1 ← R1+R2




- RD: read-mode
- WR: write-mode
- R1, R2 : registers
- ALU: arithmetic operations
- $C_0 - C_6$ : control signals (can be in either order)

Block Diagram




- OP-code means operations and it is related to the instructions. It is decoded to know the purpose of the instruction. After decoding among multiple lines, only one line is enabled.
- Decoder is a combinational circuit having 'n' no. of inputs and ('2'ⁿ) no. of output lines.
- Depending on the instructions ($I_0 - I_n$) and time ($t - t_n$), the control signals are generated.
- Clock-pulses is basically a signal that synchronizes the operation of a processor. These pulses are utilised by the CPU to co-ordinate with different tasks such as fetching, decoding and executing different instructions.
- Time Period: duration of one complete cycle (measured in seconds or nanoseconds) for modern processors

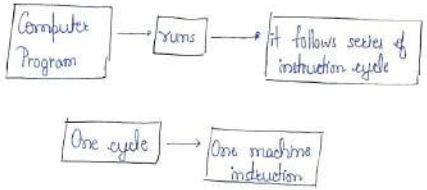
$$T = \frac{1}{f}$$

- $T$ → time period
- $f$ → clock frequency

- CU utilises clocking to breakdown sequence of instructions into microinstructions. (smallest step)
- The no. of clockpulses required depends on complexity of operation to be executed.

## Micro-operations

Def: Tiny simple steps that the processor performs to complete the instruct




Each instruction Cycle:

(I) Fetch
(II) Indirect
(III) Execute
(IV) Interrupt

Fetch

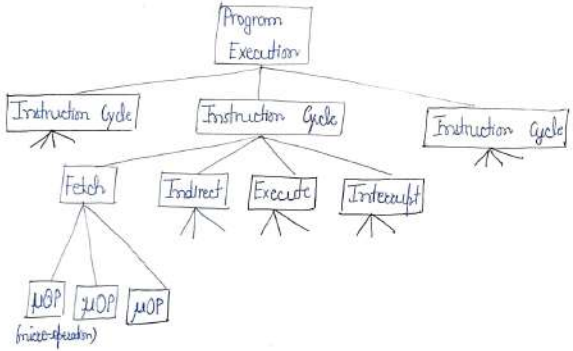
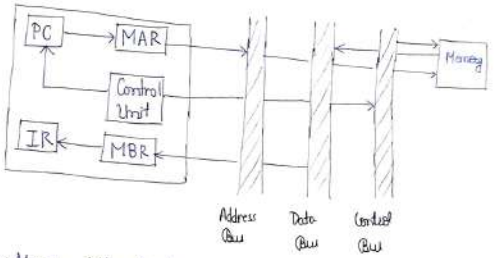



Fig: Constitutional elements of program execution

### (I) Fetch Cycle




- MAR: Memory Address Register
- MBR: Memory Buffer Register
- IR: Instruction Register → instruction decoding + execution
- PC: Program Counter

(I) The PC contains the address of the next instruction to be fetched.
- Then, this address is copied from PC to MAR.

(II) The address stored in MAR goes to memory via address bus

(III) The CU requests memory read operation and the instruction from memory is placed on the data bus and then stored in the MBR.

(IV) The instruction is then transferred to the IR from MBR

(V) PC will be incremented by 1, pointing to the instruction to be fetched for the next cycle.

Steps:

(I) PC → MAR
(II) MAR → address bus
(III) Memory → data bus → MBR
(IV) MBR → IR
(V) PC → PC + I

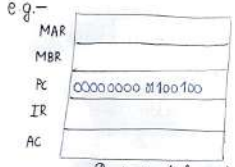
- Memory Address Register (MAR)
  - connected to the address bus
  - specifies the address in memory for read/write operations

- Memory Buffer Register (MBR)
  - connected to the data bus
  - contains value to be stored in memory or the last read value

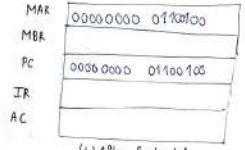
- Program Counter (PC)
  - it holds the address of the next instruction to be fetched

- Instruction Register (IR)
  - it holds the last instruction fetched

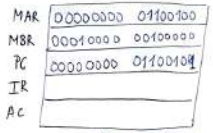
e.g.-



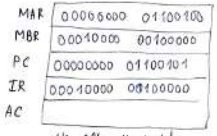
(a) Beginning (before $t_1$)



(b) After first step



(c) After second step



(d) After third step

$t_1$ : MAR ← PC

$t_2$ : MBR ← Memory

$t_3$ : PC ← PC + I

~~$t_4$~~ : IR ← MBR

## (ii) Indirect Cycle

Once instruction is fetched → execution

e.g.- assume,

one address instruction cycle + indirect addressing (NOT applicable for direct addressing)

- indirect addressing: address of address of memory (operand) is available
- direct addressing: address of operand is directly available

Steps:

(i) $t_1$ : MAR ← (IR(address)) — Address transfer to MAR
- System is ready to fetch operand address from memory — ~~Fetching~~

(ii) $t_2$ : MBR ← memory — Fetching the indirect address

(iii) $t_3$ : IR(address) ← MBR(address) — Updating IR with operand address
(MBR(address): operand address)

## (iv) Interrupt Cycle

Step (i): MBR ← PC (next instruction) — Save current PC value

Reason: CPU can resume with old PC value

Step (ii): MAR ← Save address — Update PC and save the old PC value
PC ← routine address

Step (iii): Memory ← MBR — Store old PC value in memory

- When CPU finishes executing an instruction, it'll check for any interrupts that exist before handling the next instruction
- If there's an interrupt the CPU needs to pause its current task, handle the interrupt and then return to the previous task.

Steps: (i) PC holds the address of the next instruction to be executed, therefore CPU copies this address to MBR (it ensures that after interruption handling the CPU can return wherever left off)
— Save current program counter value

(iii) Update PC and save the old PC value

- Old PC value which was stored in MBR in (i) needs to be written in the memory.
- To do that MAR is loaded with "Save Address" i.e it specifies specific memory location.
- PC is then updated with "Routine Address" which is the starting address of the interrupt handling routine (It is a piece of code that deals with the interrupt). Now, the CPU'll switch to the interrupt handling routine

(iv) Store Old PC value in memory

- Old PC value saved in MBR is now stored in memory at "Save Address" location. It ensures that after handling the interrupt, the CPU will retrieve it.

## (III) Execution Cycle

e.g.- ADD instruction (ADD R1, X)
instruction

ADD R1, 'X'

→ Take the value stored at memory location 'X'
→ Add it to the value in R1
→ R1 ← R1 + X
↳ (storing the value back in R1)

Micro-operations:

(i) $t_1$ : MAR ← IR (address)
(ii) $t_2$ : MBR ← memory (~~instructions~~)
(iii) $t_3$ : R1 ← MBR + R1

## Functional Requirements of Control Unit (CU)

- Generalised Meaning: Tasks performed by CU
- Designing & Implementation:

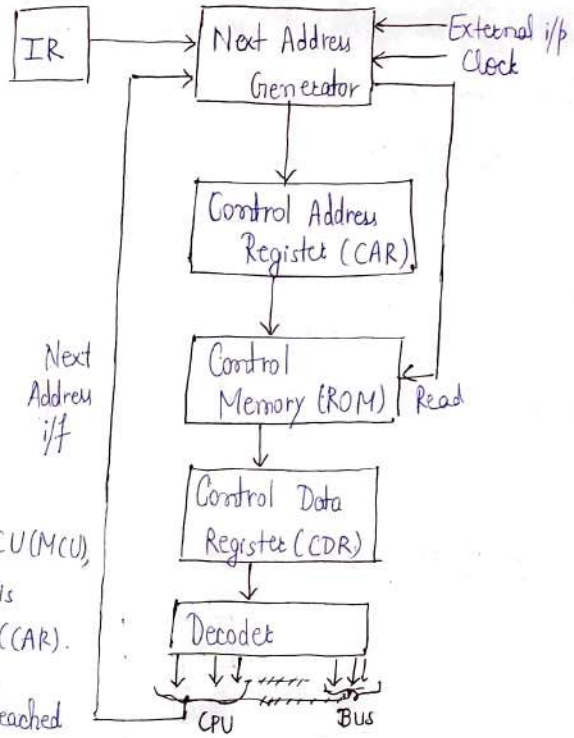
Characterization of the CU on the basis of important tasks —
(i) Defining the basic elements of the processor
(ii) Defining the micro-operations
(iii) Determining the functions that the CU must perform for executing the micro-operations

Basic functional elements of the processor:
(i) ALU — performing arithmetic & logical operations
(ii) Registers — storing the data
(iii) Internal Data Path — moving data b/w registers and b/w registers and ALU
(iv) External Data Path — links registers to the memory and I/O modules.
(v) Control Unit — generating control signals

Categorizing of the micro-operations:
(i) Transfer data from one register to another
(ii) Transferring data from a register to an external interface
(iii) Transferring data from an external interface to a register.
(iv) Performing arithmetic logic using registers for input and output.

* CU performs "sequencing" and "execution".

# Microprogrammed Control Unit




Functions:

(i) Microinstructions addressing
(ii) Microinstructions execution

Working:

(i) As we know when inst. is fetched from memory, its address is stored in MAR; similarly in microprogrammed CU (MCU), when micro inst. is fetched, it is placed in control address register (CAR).

(ii) As address of next, μ-inst. reached to CAR, μ-program sequencer will generate read signal for control memory. (next address generator)

(iii) As the read signal received from sequencer, the μ-inst. stored at particular address in control memory goes to the control data register (CDR)

(iv) The CDR ~~code~~ have the control word.

(v) Then, this control word is decoded by the decoder and the signals generator are sent to CPU and Bus,
- CPU → register-register transfer
- Bus → memory / I/O module transfer

- IR → instructions register
  - ↓ stores instructions
- External i/p → flag bits
  - ↓ determines current status of μ-inst. execution status, result of operation, ..

ect Cache Access (DCA)

A is a technique that

ta into CPU ca

, reduces late

thro

# Important Points:

- Control signals — Control information — stored in binary form (0s and 1s) in ROM
- Control memory is related to control word and it is related to μ-inst.
- Sequence of μ-inst. is determined by μ-program sequences.
- Execution of μ-inst. generates control signals
- Control memory assumed to be read-only memory (ROM) having control if that is permanently stored
- μ-program sequencer generates address of μ-inst. on the basis of inst. stored in IR.

## Addressing Modes

Instruction/Microinstruction: OP code | Operand

OP code → operations (add/sub/mul/div)

Operand → data

- In the context of instruction/μ-inst, the data isn't directly available to us, only its address is available that is present in memory/register, therefore, addressing mode basically explain operand bits, how to treat them or consider them. Here we calculate the effective address (EA).

### ① Immediate Addressing Mode

Format: OP code | Operand ← directly data

← Address →

e.g. - ADD R1, #5

# → immediate addressing

R1 ← R1 + 5

- Operand is directly provided in the address field as constant.
- No computation is req. to calculate the effective address (EA)

actual address of the operand containing its actual location

### ② Direct Addressing Mode / Absolute Addressing Mode

Page - 437 (Adv./Disadv)

Format: instruction → A → operand; memory

Here, EA = A

e.g. - ① consider an instruction -

ADD X

// X = 400

M[X]

AC ← AC + M[X]

X → operand; memory

② LD 400

// LD → load

AC ← M[400]

### ③ Indirect Addressing Mode

Format: A → B → operand; memory

Here, EA = M[A]

e.g. - OP code | X = 400 ← Address →

Here, EA = 800

actual operand = 1000

ADD X

M[X]

M[M[X]]

AC ← AC + M[M[X]]

Here, no of memory accesses = 2.

- In indirect addressing mode, address field refer to address of a word in the memory that contains full-length address of the actual operand

Cache Access (DCA)

i: a technique that ... peripherals (like NICs) to directly ... into CPU ... improves per... ter speed

Features:
- bypasses Main ... to the RAM
- reduces late...
- improves ...

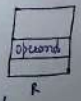
## (IV) Register Addressing Mode

Format: | OP code | Operand |

operand = register no. (R1, R2, ...)

EA = R

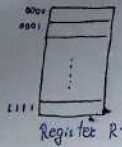
instruction: | OP code | R |




- No memory reference is present.

e.g. –


LD R1
// LD → LOAD
ACC ← R1

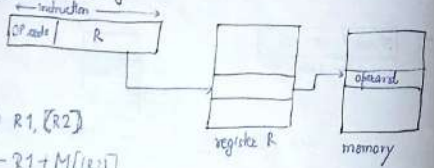



// ACC ← [R1] → going to a particular location stored at R1

- Operand is present in register.
- Register no. is written in instruction.
- Register addressing mode is similar to direct addressing, only diff. is that the address field refers to a register rather than a main memory location.

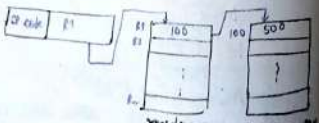
## (V) Register Indirect Addressing Mode

Format: ← instruction → | OP code | R |



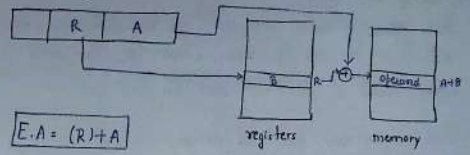

e.g. – ADD R1, [R2]
R1 ← R1 + M[[R2]]

e.g. – LD (R1)
ACC ← M[(R1)]




## (VI) Displacement Addressing Mode

Format: | R | A |




E.A = (R) + A

- It is a very powerful mode of addressing combining the capabilities of direct & indirect addressing.
- It requires an instruction to have two address components, one of which is clearly mentioned (here A)
- One address component holds a direct address 'A'.
- The content of register 'R' is added to 'A' to compute effective address.

Types:

(a) Relative Addressing — register field is program counter (PC)
- Also known as "PC Relative Addressing".

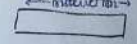
(b) Base - Register Addressing — register holds a main memory address and the instruction provides a displacement/offset from that address.
- Final memory location found by adding the displacement to the value in register.
- Displacement is stored in address field 'A' of instruction. The CPU adds displacement value to the base register's value to get final effective address.

(c) Indexing — address field 'A' holds base address and register holds the index no.

Base address + Index no. = Physical address

## (VII) Stack Addressing

Format: ← instruction →




- In stack addressing, the operand for an instruction is implicitly taken at the top of the stack, i.e the inst. doesn't need to be explicitly specify the operand's address. It assumes the top of the stack as operand.

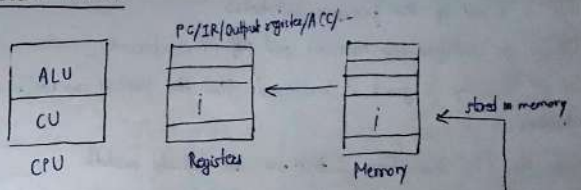
allows peripherals (like NICs) to

# Important Points:

- Stack Addressing is a way of storing and addressing data by using a stack and it is like pile of items where last item added is first item removed. Hence called as "LIFO → Last In First Out"
- Stack is reserved section of memory.
- Stack pointer keeps track of top of stack.
- When data is removed, it's taken from top

| Mode | Algorithm | Principal Advantage | Principal Disadvantage |
|---|---|---|---|
| Immediate | Operand = A | No memory reference | Limited Operand Magnitude |
| Direct | EA = A | Simple | Limited Address Space |
| Indirect | EA = M[A] | Large Address Space | Multiple Memory References |
| Register | EA = R | No memory reference | Limited Address Space |
| Register Indirect | EA = (R) | Large Address Space | Extra Memory Reference |
| Displacement | EA = A + (R) | Flexibility | Complexity |
| Stack | EA = top of stack | No Memory Reference | Limited Applicability |

## Instruction Format




Format:

| Mode | OP code | Operands (address) |
|---|---|---|

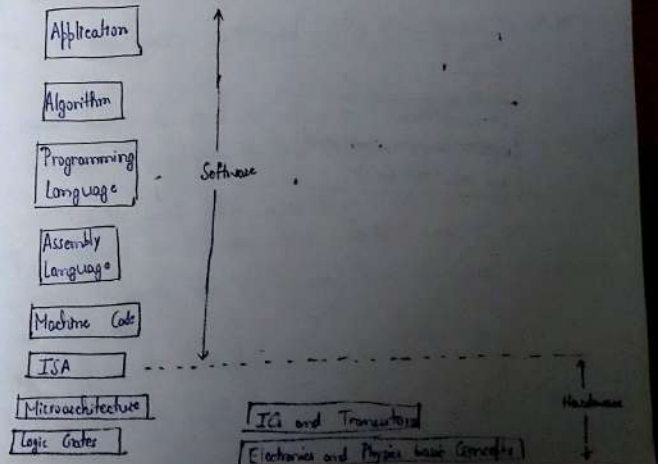
- Instruction size depends on type of organization.
  - Single accumulator → single address field
- As this increases, no. of address fields increases and thus size of IR and bus increases.

```
main(){
    int a=10;
    int b=20;        } data
    int c;
    c=a+b;
    printf(c);       } instructions
}
```

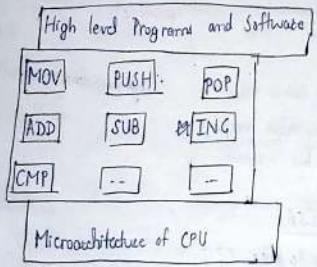
Program = data + instructions

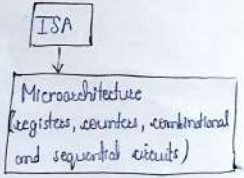
## Instruction Set Architecture (ISA)

- Set of basic instructions inside the CPU



- ISA is a part of the processor architecture
- ISA is an interface b/w hardware and software components.
- ISA is basically a group of commands that the processor can use to execute instructions.
- Inside the CPU, some basic instructions are already available
- CPU performs all major tasks by using these basic instructions
- Instructions written in high-level language are converted into basic instruction format by using these group of commands.




## Instruction Cycle State Diagram

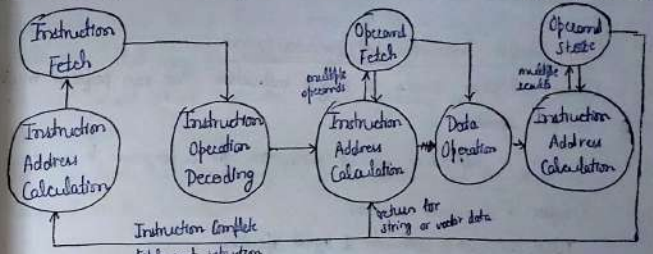



Fig: Instruction Cycle State Diagram

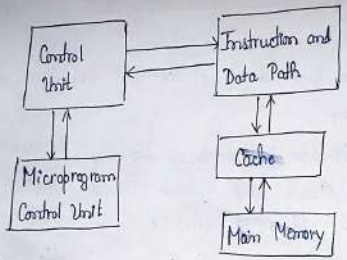
- PC register of the processor, gives address of the next instruction which needs to be fetched from memory
- Once fetched, then the OP code is decoded, on decoding the processor identifies the no. of operands. If there's any operand to be fetched from the memory, then that operand address is calculated
- Operands are fetched from the memory. If there are multiple operands, operand fetching process may be repeated.
- After this, the data operation is performed on the operands and the result is generated.
- If the result has to be stored in a register, the instruction ends here.
- If the destination is memory, the first destination address has to be calculated.
- If there are multiple results which need to be stored inside the memory, this process may be repeated, i.e. destination address calculation and storing.
- Now, the current instruction has been executed, PC is incremented to calculate address of next instruction.
- This cycle repeats for further instructions

cache, bypassing even the main memory

## CISC and RISC processors

### CISC (Complex Instruction Set Computer) System

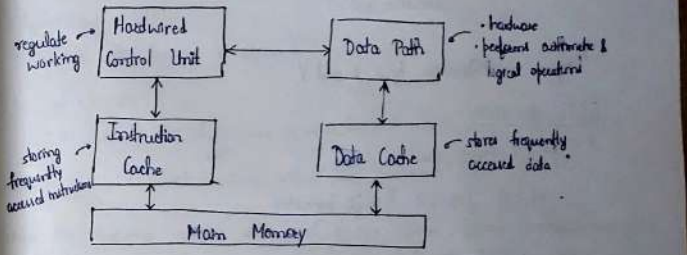
- These architecture involve a single instruction that can perform multiple low-level operations.
- Its aim is to reduce no. of instruction for a program to execute.
- Examples of low-level operations:
  PUSH, POP, LOAD, arithmetic & logical operations




- MCU – It is involved to carry out the individual steps of a complex instruction.
- CU – In this part, decoding and issuing control signals to ALU, registers, mem
- Instruction & Data Path – for fetching operands and opcodes
- * Both CU and MCU are used to manage the execution of instructions but they serve different roles.
- * This dual approach is due to the complexity of CISC architecture that aims to combine multiple operations in a single instruction.
- * CISC has to embed several low-level instructions in a single instruction set, so when decoded the instruction generates several micro-instructions to execute

### RISC (Reduced Instruction Set Computer) System

- The main idea behind RISC systems is to use a small set of simple instructions that can be executed quickly.
- It has small set of instructions which generally include register to register operation.
- Data is stored in processor (register) for computations and the results are transferred to the memory.



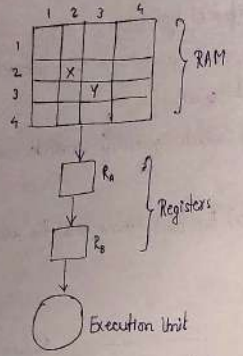

- * In this, all instructions have simple register addressing, therefore it uses less no of addressing modes.

### RISC v/s CISC

| CISC | RISC |
|---|---|
| (i) Less no. of instructions | (i) Large no of instructions are required for execution |
| (ii) Variable length instruction format | (ii) Fixed length instruction format |
| (iii) Large no. of addressing modes | (iii) Fewer no. of addressing modes (mostly 4) |
| (iv) Cost is high | (iv) Cost is less |
| (v) More powerful | (v) less powerful |
| (vi) Several cycle instructions | (vi) Single Cycle instructions |
| (vii) Manipulation directly in memory | (vii) Manipulation in registers (LOAD and STORE mainly used) |
| (viii) Microprogrammed CU based | (viii) Hardwired CU based |

CPU cache, ... by reducing memory latency and ...

Example:




Perform multiplication b/w X and Y

CISC processors

- MULT (instruction)

∴ MULT 2:2, 3:3 } Single instruction
X Y

$\boxed{X \leftarrow X * Y}$

RISC processors

- PROD (instruction)

① LOAD A, 2:2
② LOAD B, 3:3
③ PROD A, B
④ STORE 2:2, A
} multiple instructions

Instruction Set Design Factors

Importance: ① System's Performance
② Power Consumption
③ Ease of Programming

Goal: To balance b/w hardware complexity and execution speed

Factors:

① Instruction length and format
- Fixed v/s variable length
RISC CISC

② Number and type of instructions
- Data Transfer Instructions
- Arithmetic and Logical Instructions
- Branch Jumps, Control Flows, etc.

③ Addressing Modes
- specify the location of operands

④ No. of Registers
- as this ↑es, no. of memory accesses ↓s but cost ↑s

⑤ Must support direct memory access and different data types

Q) Given the following memory value and a one address machine whether accumulator, what values do the the following instructions load into the accumulator:

- word 20 → 40
- word 30 → 50
- word 40 → 60
- word 50 → 70

Find (i) Load immediate 20 (ii) Load direct 20 (iii) Load indirect 20
(iv) Load immediate 30 (v) Load direct 30 (vi) Load indirect 30

(i) 20
(ii) 40
(iii) 60
(iv) 30
(v) 50
(vi) 70

CA is a technique that allows peripheral
data into CPU cache, bypassing even the main memory
this improves ... by reducing memory latency and increasing data
transfer ...

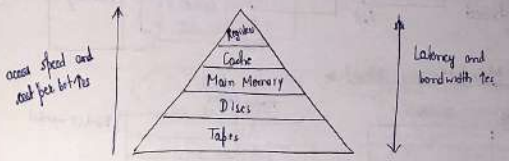
Q) let the address stored in the PC represented by symbol X1. The instruction stored in X1 has an address part (operand reference) X2. The operand needed to execute the inst. stored in the memory word with address X3. And index register contains value X4. Write the relationship b/w these quantities if addressing mode is:
(a) direct (b) indirect (c) PC relative (d) indexed

X1 → address of next inst.
X2 → address written in inst.
X3 → actual address of operand
X4 → value of index register

(a) X3 = X2
(b) X3 = M[X2] = (X2)
(c) X3 = (X1) + X2 + 1
(d) X3 = X2 + X4

# Unit - III

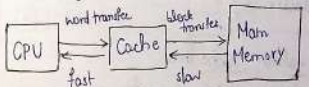
## Cache Memory




| Name | Access Type | Capacity |
|---|---|---|
| CPU registers | random | kB |
| Cache memory | random | MB |
| Main memory | random | GB |
| Disk | direct | TB |
| Tape | sequential | TBs or more |

Purpose: speed + supporting larger storage

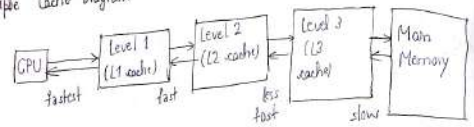
(a) Single Cache Diagram



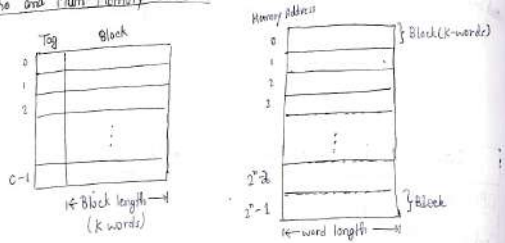

* Locality of reference: accessing the same memory location repeatedly during a short period of time.
- When processor needs to read data, first it checks the cache, if data is found it is called "cache hit" and sent to the processor.
- If not, a block of data is copied from main memory into cache and then given to processor.
- These work due to "locality of reference".
  ↳ tendency of the computer program to access the same set of memory locations over a short period of time.

CA is a technique that allows peripherals (like NICs) to directly
data into CPU cache, bypassing even the main memory
reducing memory latency and increasing data

(b) Multiple Cache Diagram




Cache and Main Memory Structure




(a) Cache (b) Main Memory

- Main memory is like a big storage made up of small parts called "words".
- Each word has a unique address.
- If memory address has n-bits, then for addressing it can hold upto $2^n$ words.

Address Range: 0 to $2^n-1$

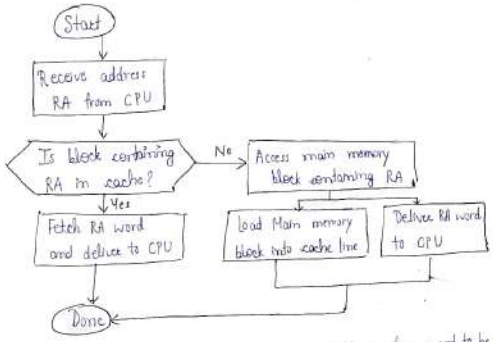
- For simplification memory is broken into words and each block has K-words therefore,

Total no. of blocks = $2^n/K$

- Cache has 'm' blocks called "cache lines".
- Each line holds 'K' words + a tag of a few bits.
- Tag is required to know or directly identify which block from memory it is storing, i.e. tag is a part of memory address.
- In cache, each line has control bits which is not shown (to identify whether the block has been modified)
- The line size holds data in the context of bytes and words

$C-1 \ll 2^n-1$

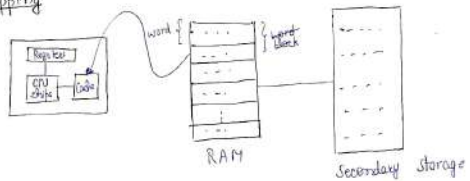
Cache Read Operation




RA: address of a word to be read

- If cache line is not available, LRU (least recently used) principle is used to replace the existing block with the new one.
- We fetch the entire block — not just the word — to increase performance of CPU (based on locality of references)

Cache Mapping




Purpose:

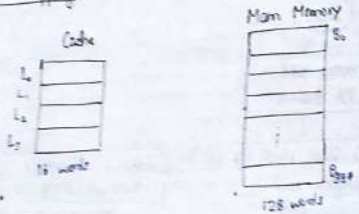
(i) in which cache line, the fetched block is in.

(ii) how CPU references it.

Direct Cache Access (DCA)

- DCA is a

Direct Mapping




- Memory size = 128 words
- Block size = 4 words
- No. of blocks = $\frac{128}{4} = 32$
- Line Size = Block Size = 4 words
- Total no. of lines = $\left(\frac{16}{4}\right) = 4$ lines

Mapping Formula: k mod n

- K → block no.
- n → no. of cache lines

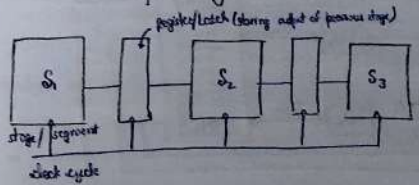
$B_0 \rightarrow 0 \bmod 4 \rightarrow L_0$

$B_4 \rightarrow 4 \bmod 4 \rightarrow L_0$

$B_2 \rightarrow 2 \bmod 4 \rightarrow L_2$

# Unit – IV

## Pipelining



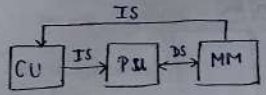

### Architectural Classification by M.J. Flynn (Flynn's Taxonomy)

– architecture of multi-processor systems (according record)

CU → Control Unit
PU → Processor Unit
MM → Memory Module
SM → Shared Memory
IS → Instruction Stream
DS → Data Stream

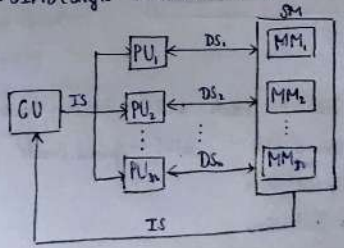
4 types:

(i) SISD (single instruction single datastream)




e.g. – IBM 701, IBM 1620

(ii) SIMD (Single Instruction Multiple Datastream)



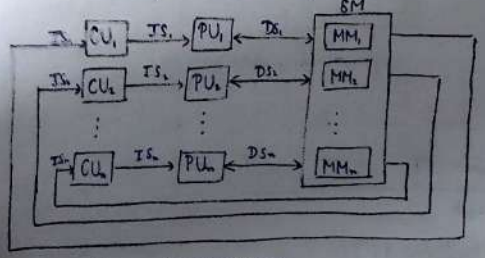

e.g. – STARAN, Illiac IV

minimizes bus cycles.

## Direct Cache Access (DCA)

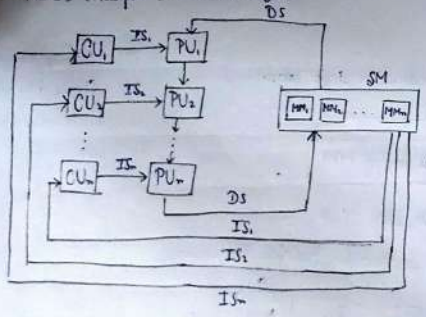
- DCA is a technique
- This
- (i)
- (ii)
- (iii)
- (iv)
- (v) G

### (iii) MIMD (Multiple Instructions Multiple Datastream)




e.g. - UNIVAC 1100, BCPB[illegible]/85

### (iv) MISD (Multiple Instructions Single Datastream)

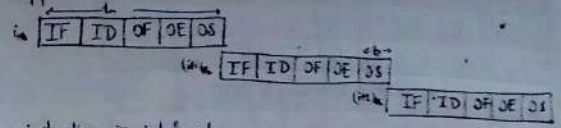



## Instruction Pipelining

Suppose Phases: (i) IF → Instruction Fetch (ii) ID → Instruction Decode
(iii) OF → Operand Fetch (iv) OE → Operand Execute
(v) OS → Operand Store

Program → Multiple Instructions

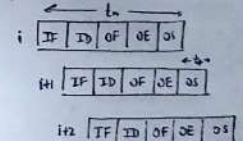
(a) Non-pipelined architecture




- instruction-wise interleaved
- No. of phases = 5×3 = 15
- No. of instructions = 3

t_p → phase duration
t_n → total non-phase execution duration

(b) Pipelined Architecture




- phase-wise interleaved
- No. of instructions = 3
- No. of phases = 7

## Performance Evaluation Matrices

(i) Speed Operation
(ii) Frequency
(iii) Pipeline Efficiency
(iv) Throughput

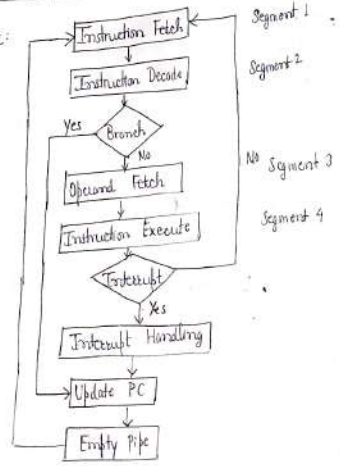
...ct Cache Access (DCA)

...A is a technique that allows peripherals (like NICs) to directly read/write ...ta into CPU cache, bypassing even the main memory

...his improves performance by reducing memory latency and increasing data ...transfe...

## Pipeline Bubble/stall

Flowchart:




Clock Cycles →

| Inst. | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | IF | ID | OF | EX | | |
| 2 | | IF | ID | OF | EX | → conditional branch |
| 3 | | | IF | .. | .. | IF |
| 4 | | | | | | |

- A "pipeline stall" or "bubble" happens when the next instruction in a pipeline can't proceed because it is waiting for something that causes a delay in the flow of instructions. Therefore a pipeline inserts a delay or no operation in the stages that can't move forward.

∴ Pipeline Efficiency ↓, Throughput ↓

- Overall pipeline processor's performance decreases

- Simplified and Expanded View → Gap 14

## Pipeline Hazards

- Problems in pipeline
  - CPI ≠ 1 (Clock per instruction)
  - Pipeline processor ideal condition: CPI = 1

- Introduction of Hazard → Pipeline Stall / Bubble

3 types of Pipeline Hazard:

(i) Data Hazard

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $I_1$ | FI | DI | FO | EI | WO | | | |
| $I_2$ | | FI | DI | ~~the~~ | FO | ~~WO~~EI | WO | |
| | | | FI | .. | .. | DI | FO | EI | WO |

Fig: Timing Diagram

$I_1$: ADD EAX, EBX

$I_2$: SUB ECX, EAX

EAX ⟵ EAX+EBX

ECX ⟵ ECX−EAX

- Instructions run at same time in different stages, because of which one instruction might try to use data before the previous one finishes updating it, leading to wrong result.

3 types:

(i) Read After Write (RAW) — True Dependency
- In this, when one instruction changes the value and the next inst. wants to read that value, i.e. second inst. reads too soon, before first inst. finishes it.

(ii) Write After Read (WAR) — Anti Dependency
- In this, the first inst. reads from register/memory and next inst. writes to the same place, i.e. $I_2$ writes too soon, before $I_1$ has read it.

Direct Cache Access

- DCA is a technique that allows peripherals (like NICs) to directly [illegible] data into CPU cache, bypassing even the main memory
- This improves [illegible] by reducing memory latency and increasing data transfer

- key

(i)

(ii)

(iii)

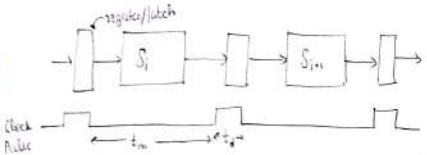
Clock Cycles →

| Phase/Segments | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | | | | |
| 2 | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | | | |
| 3 | | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | | |
| 4 | | | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | |
| 5 | | | | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ |

- $T_1 - T_6$ → total no of instructions
- No. of clock pulses / phase duration = 10
- No. of instructions = n
- No. of phases = k
- Total no of phase duration = k+n-1

$$\text{Speed Up Ratio} = \frac{\text{Non-pipelined execution time}}{\text{Pipelined execution time}}$$

$$\text{Non-pipelined execution time} = n \times t_n$$




Total no of clock cycles / pout = $\tau$

∵ All 5 phases can't have same execution time

$$\tau = t_m + t_d$$
$$= \max\left(\frac{\tau}{\tau_i}\right)^k + t_d$$

- $t_m$ → phase duration
- $t_d$ → total delay
- $S_i, S_{i+1}$ → phases

$$\text{Speed Up Ratio} = \frac{n \times t_n}{(k+n-1) \times \tau} \quad ; \quad n \gg k \Rightarrow k+n-1 \approx n$$

$$= \frac{n \times t_n}{n \times \tau}$$

$$= \frac{t_n}{\tau}$$

Pipeline Efficiency

$$E_k = \frac{S_k}{k}$$

$S_k$ → speed-up ration

$$S_k = \frac{n \times t_n}{(k+n-1) \times \tau}$$

$t_n = k \times t_p$ and $t_p = \tau$

$$= \frac{n \times k \times t_p}{(k+n-1) \times \tau}$$

$$= \frac{n k \times \tau}{(k+n-1) \times}$$

$$= \frac{nk}{k+n-1}$$

$$\therefore E_k = \frac{S_k}{k} = \frac{n}{k+n-1}$$

Throughput

$$H_k = \frac{n}{(k+n-1) \times \tau}$$

$$F = \frac{1}{\tau}$$

$$H_k = \frac{nF}{(k+n-1)}$$

Cache Access (DCA)

- DCA is a technique that allows [illegible] data into CPU cache

(iii) Write After Write (WAW) – Output Dependency

- Both instructions try to write to the same register/memory
  If the 2nd inst. writes before 1st inst, the correct data
  modified

(iv) Resource Hazard

- It happens when two or more instructions need the same H/w resource
  at the same time, because of which their execution hampers the
  performance.

(v) Control Hazard

Types:

(i) Multiple Prefetch Loop Buffer

(ii) Branch Prediction

(iii) Delayed Branch

# Memory and I/O

## Module - 1: Memory Hierarchy

### Memory System Characteristics

(i) Location

- It determines where memory resides in the system architecture
- Registers — inside CPU
- Cache Memory — b/w CPU and main memory
- Main Memory (RAM) — directly accessible by CPU

} Internal memory

- Secondary Storage — outside CPU, connected via bus (HDD/SSD)
- Tertiary Storage — external to the system (Tape / Cloud)

} External memory

(ii) Capacity

- It refers to how much data a memory unit can store.

(iii) Units of Transfer

- It is the amount of data moved in one memory operation

e.g. — words, blocks, pages, sectors, etc.

- Word — natural or smallest unit of data that a CPU processes in one operation.
- Block — smallest unit of data transfer in cache, RAM, file systems.
- Page — fixed size unit of virtual memory
- Sector — smallest storage unit on a hard disk

* Addressable Unit: It is the smallest unit of memory that can be uniquely addressed by a CPU.

- For an n-bit address, $2^n$ addressable units are possible.

(a) Byte-Addressable Memory — each byte has a unique address
- used in most modern systems as it provides flexibility

(b) Word-Addressable Memory — each word has a unique address

(iv) Access Methods

- It determines how data is retrieved and stored in memory.

Types:

(a) Sequential Access

- Data is accessed in a fixed order, one after another

Used in: magnetic tape

Advantage: simple and cost-effective

Disadvantage: slow for searching specific data

(b) Direct Access

- Data is accessed by calculating its physical location using an address

Used in: HDDs, SSDs

Advantage: faster than sequential access, suitable for large database and file storage

Disadvantage: slower than RAM, access time is still variable

(c) Random Access

- Any memory location can be selected at random and accessed directly & independently

Used in: RAM, cache memory

Advantage: fast and efficient, constant access time

Disadvantage: expensive

(d) Associative Access

- Data is accessed based on content rather than address.

Used in: Fully-associative cache

Advantage: Fastest for searching specific data

Disadvantage: expensive and requires complex hardware

(v) Performance

- It depends on speed, latency and bandwidth

(a) Access Time (Latency): time taken to perform a read/write operation

(b) Memory Cycle Time: time b/w two consecutive memory operations, i.e.
access time + any additional time required before next operation

(c) Transfer Rate: rate at which data can be transferred into/out of memory unit.

$$T_n = T_A + \frac{n}{R}$$

- $T_n$: avg. time to read/write n-bits
- $T_A$: avg. access time
- $n$: no. of bits
- $R$: rate of transfer (bps)

(vi) Physical Type

- Semiconductor memory – uses transistors & capacitors
- Magnetic memory – uses magnetic fields to store data
- Optical memory – uses laser light for read/write operations
- Magneto-optical memory – combines magnetic and optical technology

(vii) Physical Characteristics

- It determines whether the memory retains data after power-loss.

(a) Volatile memory: loses data when power is off

e.g. - RAM, cache

(b) Non-volatile memory: retains data even after power-loss

e.g. - ROM, HDDs, SSDs

(viii) Organisation

- It refers to the physical arrangements of bits to form words.

## Memory System Technologies

(i) Primary Memory Technologies

(a) Static RAM (SRAM)

Advantage: fastest memory, no refreshing required (more stable)

Disadvantage: expensive, lower cell density (low capacity)

Used in: cache

(b) Dynamic RAM (DRAM)

Advantage: higher capacity, cheaper

Disadvantage: slower, requires constant refreshing

Used in: main memory (RAM)

(ii) Secondary Storage Technologies

(a) Hard Disk Drives (HDDs)

- Larger storage capacity
- Lower cost per GB
- Used for OS, applications and user data
- Slower due to mechanical parts

(b) Solid-State Drives (SSDs)

- Uses NAND flash
- Much faster than HDDs
- Lower power consumption as no moving parts are there
- More expensive per GB

(iii) Tertiary Storage Technologies

(a) Magnetic Tape

- Used for backup and archival storage
- Very high capacity
- Lowest cost per GB
- Slow access speed (sequential access)

(b) Optical Discs

- Used for data distribution and backups
- Lower capacity and speed than SSDs.
- Uses lasers for read/write operations

## Memory Hierarchy

- Memory Hierarchy refers to the organisation of different types of memory in a computer system to balance speed, cost and capacity.

Why?

- Faster memory is expensive, slower is cheaper
- High capacity memory has smaller cost per bit
- High capacity memory is slower

(i) Capacity ↑
(ii) Cost ↓
(iii) Speed ↓
(iv) Frequency of access ↓


- Based on "locality of references".

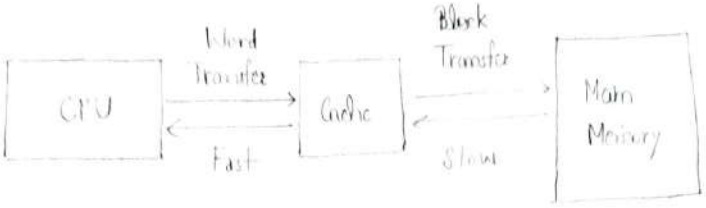
* Locality of references: It refers to the tendency of a processor to access the same or nearby locations repeatedly, i.e. memory references tend to cluster.

(a) Temporal Locality — if a memory location is accessed, it is likely to be accessed again soon.

(b) Spatial Locality — if a memory location is accessed, nearby locations are likely to be accessed as well.

# Cache Memory

- Cache is a small, expensive, high access speed memory that is used to store frequently accessed data and instructions for increasing system performance
- It is designed to combine the memory access speed of expensive, high speed memory with the large size of the less-expensive, lower-speed memory by exploiting the phenomenon of locality of references.

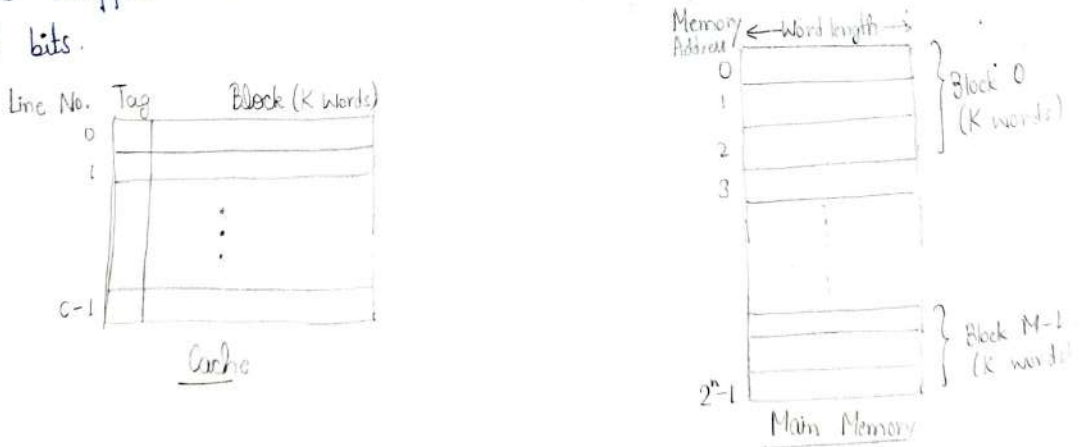



Cache and Main memory

* Cache Hit: when the processor attempts to read a word from memory, it checks whether the word is present in the cache first, if the word is found in the cache it is said to be a "cache hit"
* Cache Miss: if the word is not found in the cache, it is said to be a "cache miss"
  - In this case, a block of memory consisting of a fixed no. of words, that contains the word being accessed, is read into the cache and delivered to the processor
* Hit Ratio: it is the no. of cache hits divided by the total no. of memory references made

## Cache Memory Structure

- The main memory is divided into blocks each consisting of a fixed no. of words and each block is mapped to a cache line consisting of the words in each block plus tag and control bits.




Cache/Main memory Structure

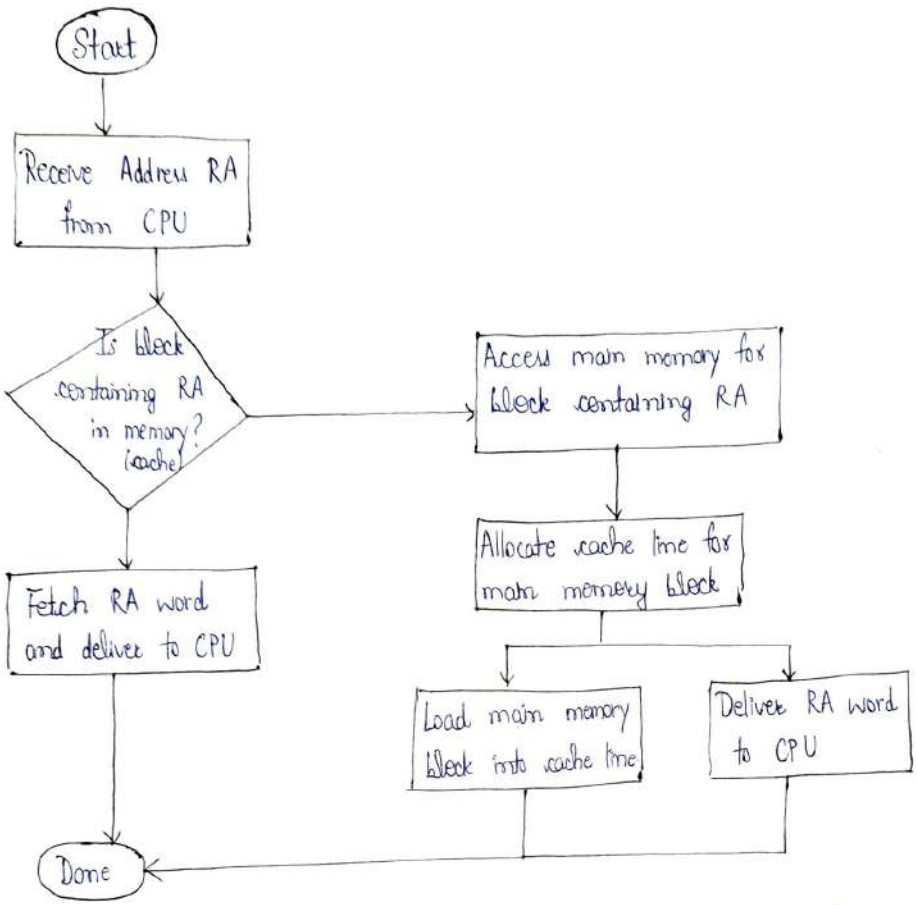
* Line Size: The length of the cache line excluding the tag and control bits is called "line size"

Line Size = No. of words in each block

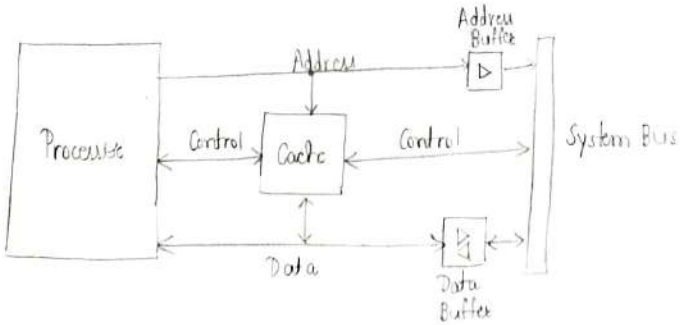
* Virtual Memory: It is a memory management technique where a processor uses secondary storage as an extension of main memory (RAM).

* Tag: Each line includes a "tag" that identifies which particular block is currently being stored as there are more blocks than lines, so an individual block can't uniquely store a block permanently.

## Cache Read Operation




(i) The processor generates read address (RA) of the word to be read.
(ii) If the word is contained in the cache, it is delivered to the processor
(iii) Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor.




Cache Organization

- When a cache hit occurs, both buffers are disabled and communication is only b/w processor & cache.
- When a cache miss occurs, the address is loaded on the address bus and data is returned to both processor and cache via the data parallely.

# Elements of Cache Design

## 8) Cache Addresses

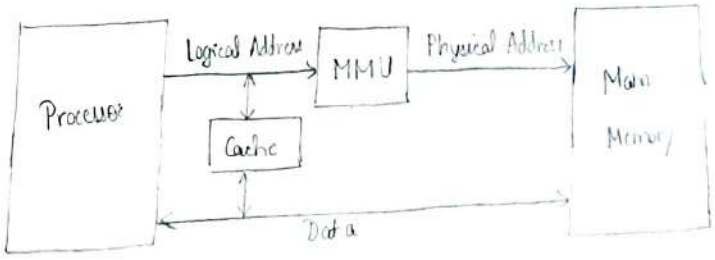
* **Virtual Memory:** It is a memory management technique where a processor uses secondary storage as an extension of main memory (RAM) to run programs that exceed the available RAM capacity.
  - A virtual address space is generated for each process which is mapped to the physical memory allowing us to address memory from a logical point of view
* **Memory Management Unit (MMU):** It is a hardware component that translates virtual addresses to physical addresses in main memory and vice-versa for read/write operations

### (a) Logical/Virtual Cache

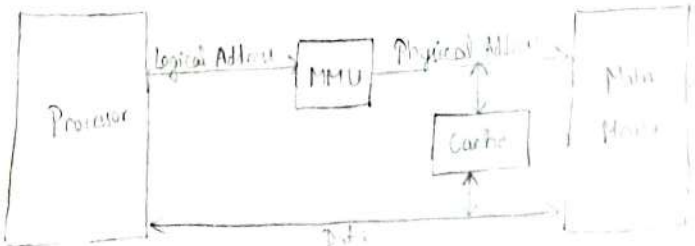
- It stores data using virtual addresses.
- It has faster access speed as the processor can access it directly without the MMU.
- It must be completely flushed with each application context, or extra bits must be added to each cache line to identify which address space it refers to as, generally each application is assigned the same virtual memory address space that starts at address 0 (therefore it is possible that the virtual address in two different applications may refer to different physical memory locations).

### (b) Physical Cache

- It stores data using main memory physical address.
- It is slower, as MMU translation is required




(a) Logical Cache




(b) Physical Cache

(ii) Cache Size

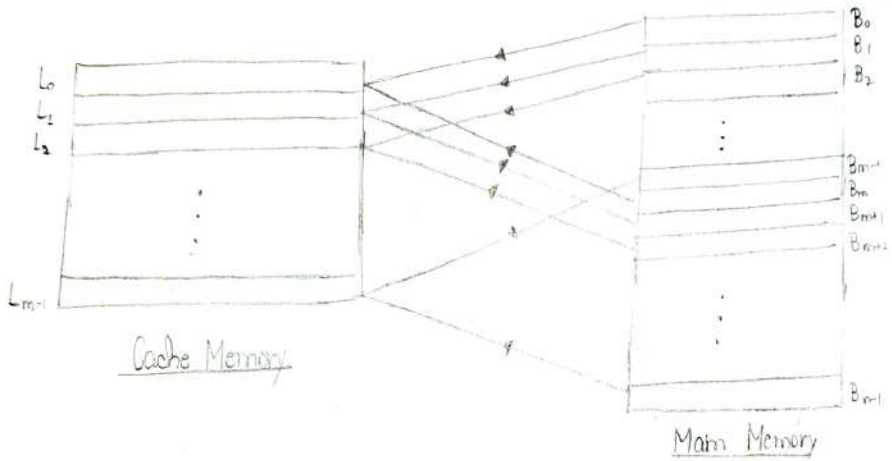
- Small enough so that the access time is close to that of the cache alone.
- Large enough so that the cost per bit is close to that of the main memory alone.
  (larger cache has higher hit rates but increased latency)

(iii) Mapping Functions

- Since no. of cache lines < no. of main memory blocks, an algorithm is needed to map main memory blocks to cache lines

(a) Direct Mapping: Each block of main memory is mapped to only one possible cache line.

- It can be expressed as: $\boxed{i = j \bmod m}$ where, $i \rightarrow$ cache line no.
  $j \rightarrow$ main memory block no.
  $m \rightarrow$ no. of cache lines



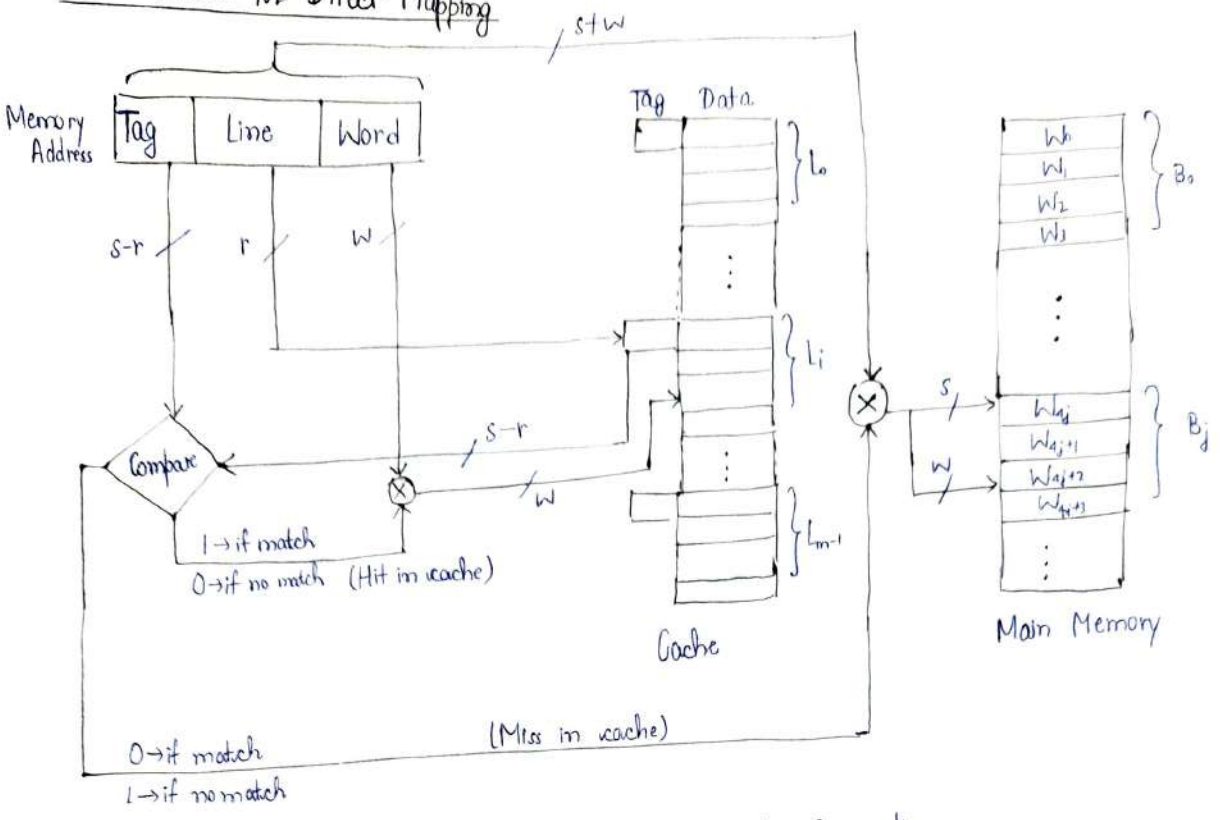

Direct Mapping

- Here, the first 'm' blocks of the main memory map to the corresponding cache lines, i.e.
  $B_0 \rightarrow L_0$, $B_1 \rightarrow L_1$, $B_2 \rightarrow L_2$, ---, $B_{m-1} \rightarrow L_{m-1}$.
- Then, block $B_m \rightarrow L_0$, $B_{m+1} \rightarrow L_1$, $B_{m+2} \rightarrow L_2$, .... and so on.

* Advantage: Simple and Inexpensive to implement
* Disadvantage: High Conflict Misses (since multiple memory blocks map to the same cache line)

- Thrashing: If a program happens to reference words repeatedly from different blocks that map into the same line, the blocks will be continually swapped and the hit ratio will be lowered, known as "thrashing".
- Victim Cache: It is a fully associative cache b/w direct mapped L1 cache and the next level of memory that stores what was discarded from the cache to reduce the miss penalty.

# Cache Access in Direct Mapping




Direct-Mapping Cache Organisation

- Each main memory address is viewed as consisting of three fields—
  - (i) Word — the least significant 'w' bits identify a unique word or byte within a block
  - (ii) Line — the next 'r' bits specify the line field, where $2^r$ = no. of cache lines
  - (iii) Tag — the most significant 's-r' bits, where $2^s$ = no. of main memory blocks

(i) The line field specifies which cache line that particular block gets mapped to.

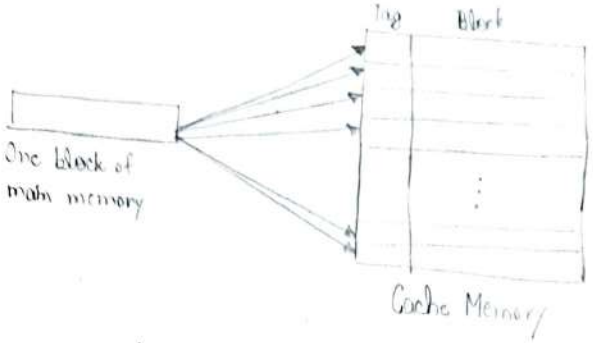
(ii) The tag of that cache line is then compared with the tag of the given memory address

(iii) If tag matches (cache hit), the word field is used to fetch the particular word from that block.

(iv) If tag doesn't match (cache miss), the address is used to fetch the word from the memory and store its corresponding block in that cache line.

lines consists of a number of sets, each of which consists of a no. of cache

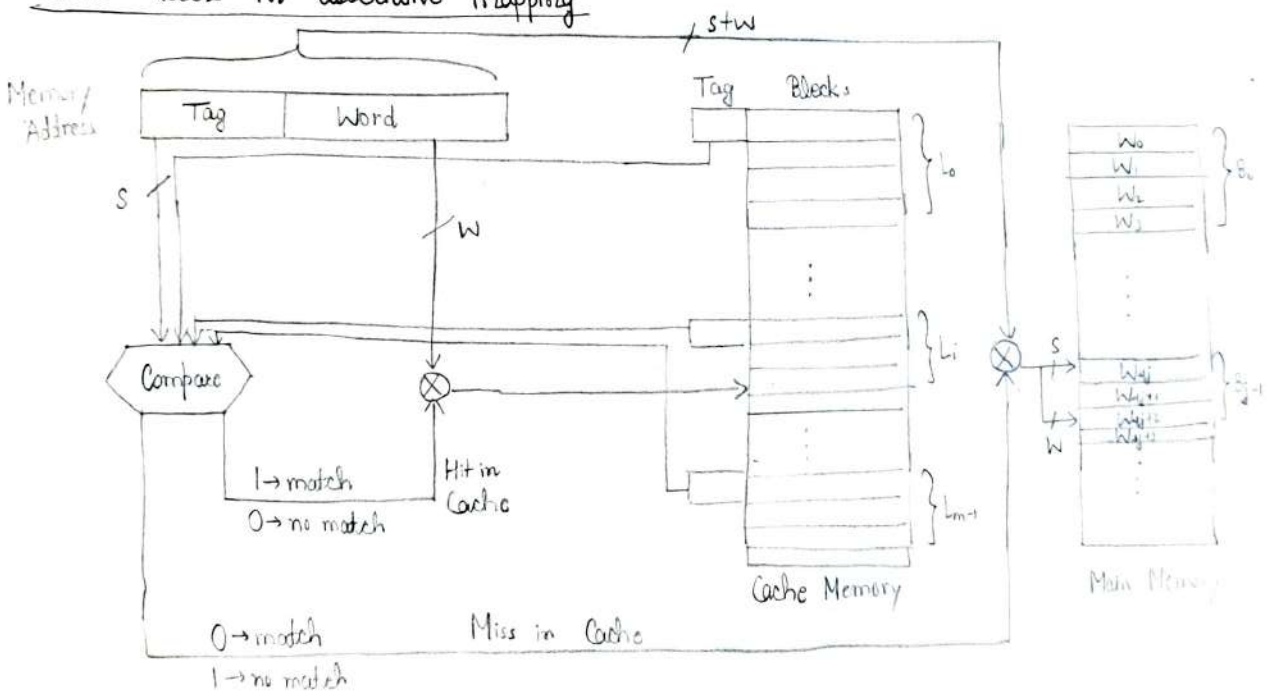
(b) Associative / Fully-associative Mapping: Each block of main memory can be loaded into any cache line




Associative Mapping

* Advantage: Flexibility as to which block to replace when a new block is read.
* Disadvantage: Complex circuitry required to examine tags of all lines in parallel.

Cache Access in associative mapping




· Each memory address is interpreted as — Tag and Word field
(most significant s-bits) (least significant w-bits)

(i) The tag of the address is compared with all the tags of the cache lines in parallel.

(ii) If tag of any line matches, the word field is used to determine the exact word to be fetched.

(iii) If tag of no line matches, the address is used to retrieve the word from the main memory and store its corresponding block in a cache line.

(c) Set-Associative Mapping: It is a compromise between direct and associative mapping.

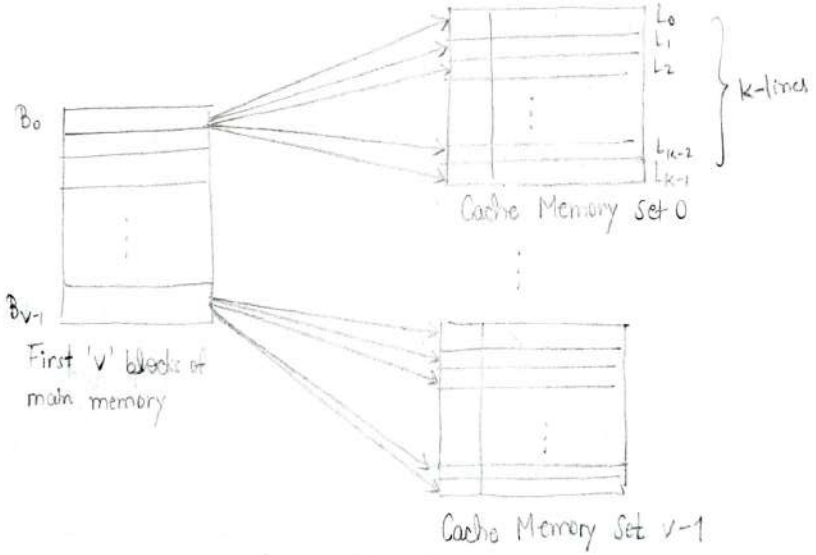
- The cache consists of a number of sets, each of which consists of a no. of cache lines

$$m = v \times k$$
$$i = j \bmod v$$

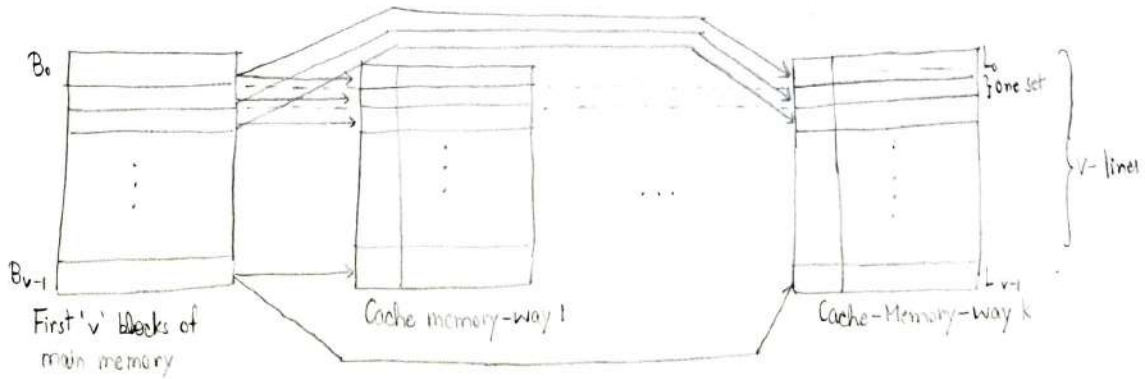
, where
- $m \rightarrow$ no. of cache lines
- $v \rightarrow$ no. of sets
- $k \rightarrow$ no. of cache lines per set
- $i \rightarrow$ cache set no.
- $j \rightarrow$ main memory block no.

* k-way set associative mapping: each set consists of 'k' cache lines

- In this mapping, each block of memory can only map to a particular set (like in direct mapping), but to any cache line within that set (like in associative mapping)
  - Advantage: reduces conflict misses than direct mapping and more efficient than associative mapping.
  - Disadvantage: more complex and expensive
- It can be physically implemented as 'v' associative caches or 'k' direct mapping caches



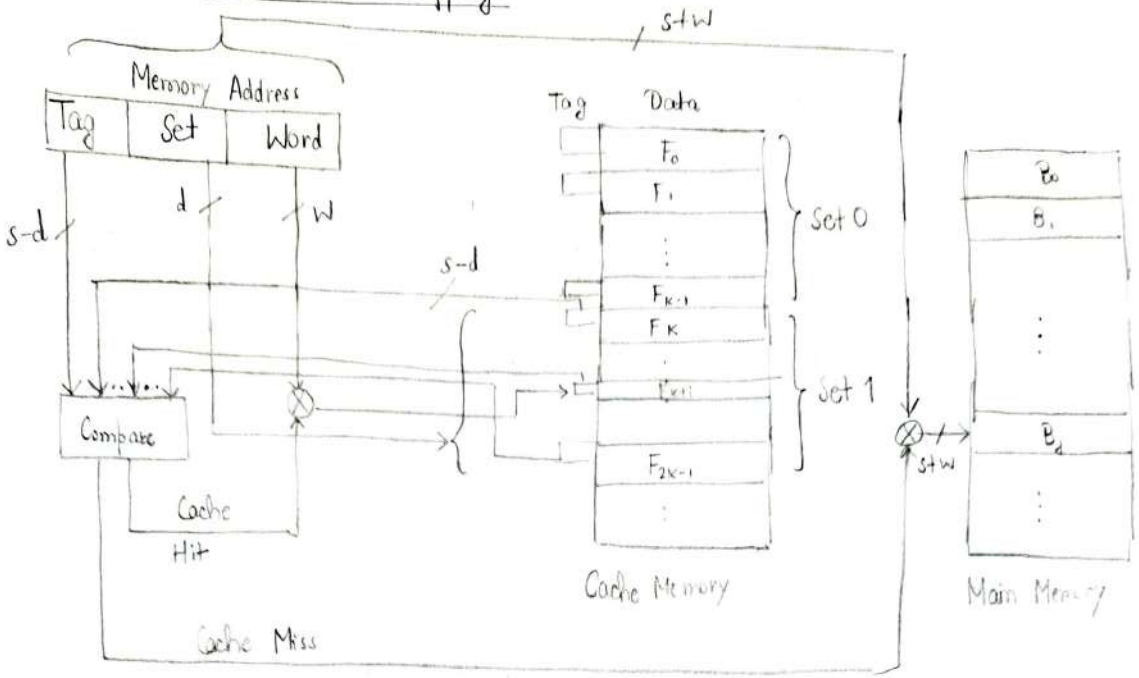

V-associative-mapped caches




k- direct mapped caches

(b) First-In-Fa...l. n. l. ... algorithm

## Cache Access in Set-Associative Mapping




k-way set associative organization

- Each memory address is interpreted as —
  - (a) Tag : most significant (s-d) bits, $2^s$ → no. of main memory blocks
  - (b) Set : next d bits, $2^d$ → no. of sets
  - (c) Word : least significant w bits, $2^w$ → block/line size

(i) The Set field determines which set the block containing the words gets mapped to.

(ii) The tags of all cache lines within that set are compared with the tag field of the address in parallel.

(iii) If any of the tag fields match, the word field is used to determine which word of the block stored in the cache line is required.

(iv) If none of the tag fields match, the address is used to retrieve the block containing the word to be fetched and store it in any cache line of its corresponding set.

* When associative and set-associative caches are full and a new block is to be added, replacement algorithms are used.

(iv) Replacement Algorithms

(a) Least Recently Used: replaces the least recently accessed block in the cache
- it is the most popular algorithm

(b) First-In-First Out: replaces the oldest block in the cache
- implemented using round-robin or circular buffer technique

(c) Least Frequently Used: replaces the block that has received the fewest references
- implemented by associating a counter with each cache line

(d) Random Replacement: randomly replaces a block

(v) Write Policy

- When a block in a cache is to be replaced and at least one write operation has been performed on it, i.e. it has been altered, the main memory must be updated before replacing it.

(a) Write-Through
- Data is updated in both the cache and main memory at the same time
- Maintains Consistency but slower (generates substantial memory traffic and may create a bottleneck)

(b) Write-back
- Data is only updated in cache, and updated in memory later while being replaced.
- Implemented by assigning a "USE" or "dirty" bit with each cache line, which signals if the data has been modified.
- Faster but requires extra logic for consistency

Cache Coherency

- Required for multiprocessor systems to ensure consistent data

(a) Write Invalidate: invalidates cache lines when a processor writes to shared memory.

(b) Write Update: updates all copies of data in caches

(c) Non-cacheable Memory: only a portion of main memory is shared among processors and it is non-cacheable.

(vi) Line Size

- Larger blocks reduce the no. of blocks that fit into the cache. Because each block fetch overwrites older cache block, a smaller no. of blocks results in data being overwritten shortly after being fetched.
- As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future.

(vii) No. of Caches

(a) Multilevel Caches

- L1 Cache: smallest, fastest, closest to CPU
- L2 Cache: larger but slower than L1
- L3 Cache: largest, slowest, shared among cores

(b) Unified and Split Caches

- Unified Cache: stores both data and instructions
- Split Cache: separates caches for data and instructions

* Split Caches are mainly used at L1 level for faster execution by having parallel access to instructions and data but it is less flexible
* Unified Caches are mainly used at L2, L3 levels as it is more flexible (efficient use of cache space) but instructions and data compete for cache

## Main Memory

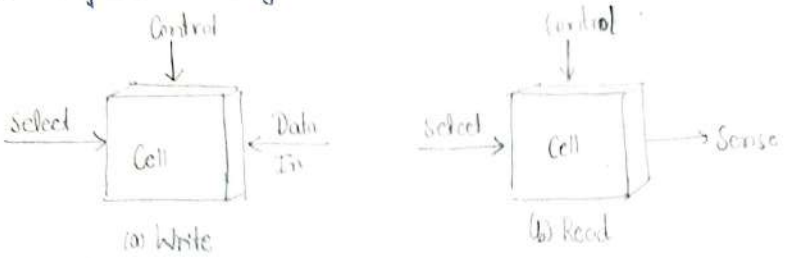
- Main Memory is the primary volatile storage in a computer that stores data and programs currently being used by the CPU.

Characteristics:

(i) Volatility — data is lost when power is turned off
(ii) Direct Access — CPU can directly access main memory
(iii) Speed — faster access speed than secondary storage
(iv) Capacity — smaller in size than secondary storage

## Organisation of main memory

- The basic element of a semiconductor memory is the "memory cell".
- All semiconductor memories show the following memories:
  (i) They exhibit two stable (or semistable) states, which can be used to represent binary '1' & '0'.
  (ii) They are being capable of being written (at least once) to set state.
  (iii) They are capable of being read to sense the data




(a) Write (b) Read

Memory Cell Operation

* The cell has three functional terminals:
  (i) Select Terminal — selects a memory cell for read/write operation
  (ii) Control Terminal — indicates whether to perform read or write operation
  (iii) Data Terminal — for writing, it provides an electrical signal to set the state to '0' or '1'. for reading, it outputs the cell's state

## Random Access Memory (RAM)

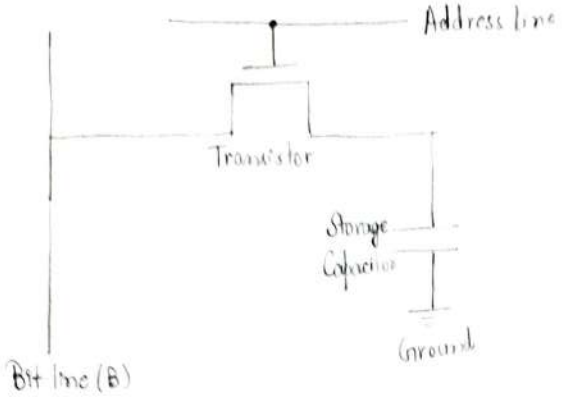
- RAM is the primary type of main memory that stores data temporarily to be used by the processor.
- It is volatile — data is lost when power is turned off.

Types:

(i) Dynamic RAM (DRAM)

- DRAM stores data as an electrical charge in a capacitor.
- The presence or absence of charge represents binary '1' or '0'.
- It needs to be continuously refreshed as capacitors lose charge over time even when power is constantly supplied.

# DRAM Structure




DRAM Cell

- Consists of a single transistor and capacitor per bit (memory cell)
- Bit line: Used to store and sense 1s and 0s in capacitors
- Address line: Used to select a particular cell for read/write operations (transistor acts a switch)

## Write Operation

i) A voltage signal is applied to the bit line ( 1 → high voltage / 0 → low voltage ) to be stored.

ii) A signal is then applied to the address line, allowing the charge to be transferred to the capacitor.

## Read Operation

i) When the address line is selected, the transistors turns on and the charge stored on the capacitor is fed to the bit line and to a sense amplifier.

ii) The sense amplifies compares the capacitor voltage to a reference value and determines if the cell contains a logic '0' or logic '1'.

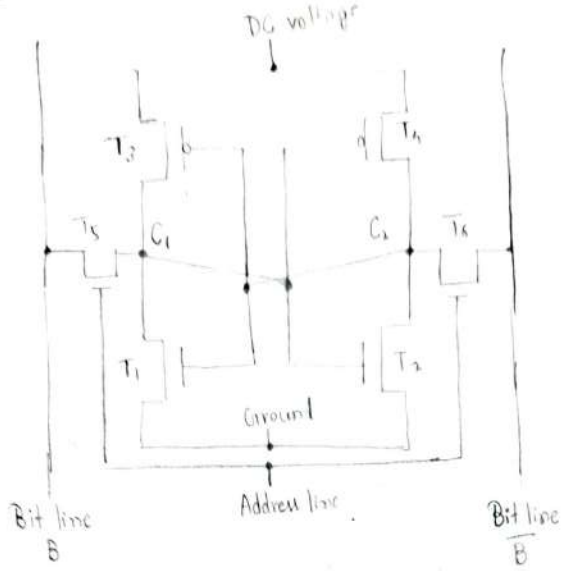
iii) The readout from the cell discharges the capacitor, which must be restored to complete the operation.

* Advantages: higher cell density, lower cost, used for main memory
* Disadvantages: requires periodic refreshing, slower than SRAM

## i) Static RAM (SRAM)

- It uses flip-flop circuits to store data
- As long as power is supplied, data remains stable (no refreshing required)

### SRAM Structure




SRAM cell

- Each bit is stored using flip-flop logic gates (6 transistors per cell)
- Transistors $T_1, T_2, T_3, T_4$ form a stable bistable circuit:
  - Logic 1: C1 → High, C2 → Low ($T_1, T_4$ → OFF and $T_2, T_3$ → ON)
  - Logic 0: C1 → Low, C2 → high ($T_1, T_4$ → ON and $T_2, T_3$ → OFF)
- Transistors $T_5, T_6$ act as switches controlled by the address line for read/write operation.

### Write Operation

(i) The address line is activated, enabling access to the cell ($T_5, T_6$ → ON)
(ii) The desired bit value is applied to bit line B and its complement to bit line $\overline{B}$.
(iii) This forces the four transistors ($T_1 - T_4$) into the correct state, storing the value.

### Read Operation

(i) The address line is activated ($T_5, T_6$ → ON)
(ii) The stored bit is read from bit line B.

* Advantages: faster than DRAM, no refreshing required, used in cache.
* Disadvantages: larger cell size so lower cell density, higher cost

| DDR1 | 2-bit | 1.. sheet of SDRAM |
|---|---|---|

(iii) Synchronous DRAM (SDRAM)

- It is a type of DRAM that operates in sync with the system clock.
- Eliminates timing mismatches by synchronizing operations with the CPU bus clock.
- Faster than normal DRAM due to controlled read/write cycles

* Features:
  - Burst Mode – allows rapid sequential data access, reducing latency
  - Multiple Bank Architecture – improves on-chip parallelism, enabling multiple memory banks to operate independently.
  - Mode Register – used to configure burst length, latency and operational settings.
  - Pipelined Operation – overlaps read/write cycles to increase throughput.

* Advantages:
  (i) Faster than asynchronous DRAM
  (ii) Supports burst transfers for efficiency
  (iii) Reduces CPU wait times

* Disadvantages:
  (i) Higher power consumption
  (ii) More complex control logic
  (iii) Higher latency for small, random accesses

Double Data Rate SDRAM (DDR SDRAM)

- It increases data transfer rates by transferring data on both the rising and falling edges of the system clock.

* Features:
  - Double Data Rate (DDR) – transfers twice the amount of data per clock cycle
  - Prefetch Buffer – stores multiple bits before transferring, allowing fast sequential access.
  - Higher bus clock rate – increases overall memory bandwidth
  - Lower Power Consumption – more efficient than SDRAM

## DDR SDRAM Generations

| Generation | Prefetch Buffer | Speed Improvement |
|---|---|---|
| DDR1 | 2-bit | 2x speed of SDRAM |
| DDR2 | 4-bit | 4x speed of SDRAM |
| DDR3 | 8-bit | 8x speed of SDRAM |
| DDR4 | 8-bit + Bank Groups | higher efficiency and speed |

* Bank Groups:
  - Allows independent parallel operations within different memory banks
  - Reduces interferences and increases memory access efficiency.

* Advantages:
  (i) Higher data transfer rate
  (ii) Efficient burst mode for sequential access
  (iii) Lower latency for large blocks of data

* Disadvantages:
  (i) More expensive than SDRAM
  (ii) Requires more power management features
  (iii) Complex Memory Controller needed

## Read Only Memory (ROM)

- ROM is a type of main memory that stores permanent non-volatile data that can only be read, not written to or altered easily.
- It is non-volatile — no power source is required to maintain the bit values.

Applications:
- Library sub-routines for frequently-accessed functions
- System Programs
- Function Tables

# Types of ROM

(i) Mask ROM (MROM)

- Data is hardwired into the chip during manufacturing.
- Used for mass production
- High fixed cost and no error tolerance while manufacturing

(ii) Programmable ROM (PROM)

- Can be programmed once after manufacturing using special equipment.
- Data is burned in using electrical fuses
- Once programmed, it can't be erased or modified

(iii) Read-Mostly Memory

- Used in applications where reads are more frequent than writes.

Types:

(a) Erasable PROM (EPROM)

- Can be erased and reprogrammed multiple times.
- Uses UV light for erasure. (slow process)

(b) Electrically Erasable PROM (EEPROM)

- Can be erased and reprogrammed electrically. (byte-level erasure)
- Used in BIOS firmware, embedded systems, etc.
- Slower write speed
- More expensive and less dense than EPROM

(c) Flash Memory

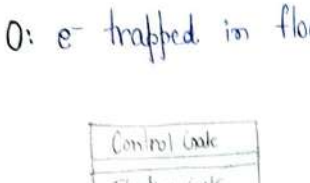
- Can erase blocks of memory instead of entire chip (no byte-level erasure)
- Faster erasure and reprogramming
- Limited write cycles
- Higher density but cheaper than EEPROM.

# Flash Memory

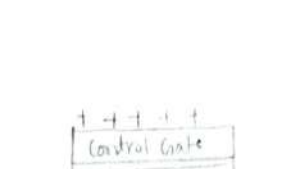
## Working

- Transistors exploit properties of semiconductors so that a small voltage applied to the gate can be used to control the flow of a large current b/w source & drain
- In flash memory cell, a second gate — floating gate, called so because it's insulated by a thin oxide layer — is added to the transistor.

Binary 1: transistor works normally
Binary 0: $e^-$ trapped in floating gate




(a) Flash Memory Cell in '1' state




(b) Flash memory cell in '0' state

Types:

(i) NOR flash

- Cells are connected in parallel
- Functions like NOR gate (each cell can be independently read/write/erased) → byte level access
- * Advantages: fast read access, byte-level access
- * Disadvantages: lower storage density, higher cost per bit

(ii) NAND flash

- Cells are connected in series
- Functions like NAND gate — bit line is active only if all transistors in a block are ON.
- * Advantages: higher storage density, cheaper cost per bit, faster write speed
- * Disadvantages: block-based access, needs controller to manage circuitry

Uses:

- NOR flash: embedded systems, firmware storage
- NAND flash: mass storage (USB, SSDs, memory cards)

(ii) Optical Storage

... (CD)

... track (not concentric)

## External Memory

- External memory refers to storage devices that provide additional capacity beyond the main memory (RAM).

(i) Magnetic Tape

- It is a sequential access storage medium using magnetizable material on a flexible polyester tape

### Storage

- Data stored in parallel tracks along the tape's length.

* Parallel Recording: data stored byte-wise or word-wise across tracks (older)
* Serial Recording: bits are stored sequentially along each track (newer)
  - Data are read and written in contiguous blocks called "physical ~~blocks~~ records".
  - Blocks on the tape are separated by gaps called "interrecord gaps".

### Serpentine Recording

- Data is written along the entire tape length for one track, then reversed for the next track.
- Process repeats back and forth until the tape is full
- Read-write heads can handle multiple tracks simultaneously to increase speed.

* Advantages:
  (i) Low Cost
  (ii) High Storage Capacity
  (iii) Durable and portable
* Disadvantages:
  (i) Slow sequential access
  (ii) Mechanical wear and tear
  (iii) Susceptible to environmental damage
* Uses:
  (i) Backups
  (ii) Archival Storage

(ii) Optical Storage

## Compact Disc (CD)

- Stores 680 MB of data using a single spiral track (not concentric)
- Data is stored as microscopic pits and lands, read by a laser beam
- Also known as CD-ROM
- Uses constant linear velocity (CLV) for even data distribution

### Working:

(i) A laser shines through the clear polycarbonate while a motor spins the disc

(ii) The intensity of the reflected laser light changes as it encounters a pit

pits → lower intensity (as light scatters)

lands → higher intensity

(iii) The change b/w pits and lands is detected by a photosensor and converted into a digital signal.

(iv) The sensor tests the surface at regular intervals

1 → beginning or end of a pit

0 → no change in elevation

* Advantages:
  - Mass replication at low cost
  - Removable storage for easy archiving

* Disadvantages:
  - Read-only (can't be modified)
  - Slow access time

## CD-Rewritable (CD-RW)

- Can be written and erased 500,000 to 1,000,000 times using phase-change technology.

## CD-Recordable (CD-R)

- Can be written to once, used for permanent archival storage

## Digital Versatile Disc (DVD)

- Higher capacity than CDs due to:
  (i) higher bit density (closer pits and tracks)
  (ii) dual-layer technology (two data layers read separately)
  (iii) double-sided storage

### Capacity:

- Single Layer, single side → 4.7 GiB
- Dual Layer, single side → 8.5 GiB
- Double-Lay sided, dual layer → 17 GiB

## High-Definition Optical Discs

- Developed for HD video storage
- Uses a blue-violet laser for smaller pits and higher storage capacity.

### Formats:

(i) HD-DVD → 15 GiB/layer (obsolete)
(ii) Blu-ray Disc → 25 GiB/layer
  - BD-ROM (read only)
  - BD-R (recordable once)
  - BD-RE (re-recordable)

## (iii) Magnetic Disks

- A magnetic disk consists of a circular platter made from a non-magnetic material (substrate) coated with a magnetizable material.

### Read-Write Mechanism

(i) Data is recorded and retrieved using a read-write head.
(ii) The write-head generates a magnetic field using an electric current, magnetizing specific areas of the disk.
(iii) The read-head uses electromagnetic induction or a magnetoresistive (MR) sensor, allowing for greater storage density and speed.

## Data Organisation

- Data is stored in tracks (concentric circles)
- Tracks are divided into sectors.
- Intertrack gaps prevent read/write errors.
- Constant Angular Velocity (CAV) keeps disk rot" constant but limits storage capacity
- Multiple Zone Recording (MZR) increases storage by dividing the disk into zones with varying disk densities.
- Sectors contain control data for identification and error detection.

## Physical Characteristics

(a) Head Motion

(i) Fixed-head → one per track

(ii) Movable head → one per surface

(b) Disk Portability

(i) Non-removable → permanently mounted in disk drive (e.g. - hard disk)

(ii) Removable → can be removed and switched, offers portability (e.g. - floppy disks)

(c) Sides

(i) Single-sided → magnetizable coating on single side

(ii) Double-sided → magnetizable coating on both sides

(d) Platters

(i) Single Platter → only one platter

(ii) Multiple Platter → multiple platters stacked vertically with one read-write head per platter surface

* Cylinder: The set of all the tracks in the same relative position on the platter is referred to as a "cylinder".

(e) Head Mechanism

(i) Contact → read-write head physically touches the medium (floppy)

(ii) Fixed Gap → read-write head positioned a fixed distance above the platter

(iii) Aerodynamic Gap → aerodynamic foil head that rests lightly on the platter when the disk is motionless but the air pressure generated by the spinning disk is enough to make the foil rise above the surface, thus it can use narrower heads for higher data density (Winchester disks)

- NAND flash wears out after a certain no. of writes (~100,000 cycles)

## Disk Performance Parameters

(i) Seek Time: time required to move the disk arm to the required track

(ii) Rotational Delay: time required for the beginning of a sector to reach the head

(iii) Access Time: seek time + rotational delay

(iv) Transfer Time: the time required for transferring the data

$$T = \frac{b}{rN}$$

where, $T \rightarrow$ transfer time
$b \rightarrow$ no. of bytes to be transferred
$N \rightarrow$ no. of bytes on a track
$r \rightarrow$ rotational speed (revolutions per second)

$$T_{total} = T_s + \frac{1}{2r} + \frac{b}{rN}$$

where $T_{total} \rightarrow$ total avg. read-write time
$T_s \rightarrow$ avg. seek time

## (iv) Solid-State Drives (SSDs)

- SSDs are semiconductor-based storage devices that primarily uses NAND flash for data storage.

### Advantages of SSDs over HDDs

- Higher IOPS, (Input/Output Per Second) for faster data access
- Greater durability, as they are less susceptible to physical shock & vibration
- Longer lifespan, as SSDs are not susceptible to mechanical wear
- Lower power consumption than HDDs, leading to reduced energy costs
- Quieter and cooler operation, due to absence of spinning disks
- Faster access time and lower latency

### Practical Issues with SSDs

(i) Performance degradation over time — as SSDs fill up, fragmentation increases, leading to slower write speeds.

- Solutions: (a) Over-provisioning: reserves extra space to manage write operations efficiently

(b) TRIM command: allows the OS to inform the SSD about unused data blocks for internal cleanup

(ii) Limited Write Endurance

- NAND flash wears out after a certain no. of writes (~100,000 cycles)

* Techniques to prolong SSD lifespan:
  - Caching: groups and delays write operations to minimize wear
  - Wear Levelling: distributes write operations evenly across memory cells
  - Bad-block Management: identifies and avoids failing memory cells
  - RAID configurations: ensures data redundancy and reliability
  - Self Monitoring: SSDs estimate remaining lifespan to allow preemptive data backups

## Redundant Array of Independent Disks (RAID)

- It is a technology designed to improve the performance, capacity and reliability of disk storage systems.
- It involves using multiple physical disks arranged in different configurations called "RAID levels".

(i) RAID Level 0
- Characteristics: No redundancy (no parity). Data is striped across multiple disks.
- Advantages: High data transfer rates and improved performance for large I/O requests
- Disadvantages: No redundancy, so if one disk fails, all data is lost
- Application: High performance, where data loss is not a concern (supercomputers)

(ii) RAID Level 1
- Characteristics: Data is mirrored across two or more disks, providing redundancy
- Advantages: Improved read performance, simple recovery in case of disk failure
- Disadvantages: Doubles the storage cost, as each piece of data is duplicated
- Application: critical data where redundancy is needed (system files / databases) with high read requirements

(iii) RAID Level 2
- Characteristics: Data is striped at bit level, and an error-correction code (ECC) is used for error detection and correction.
- Advantages: Can correct single-bit errors
- Disadvantages: Overkill for most applications due to high cost and complexity
- Application: Environments where frequent disk errors occur (not practical)

(IV) RAID Level 3

- Characteristics: Similar to RAID 2, but uses single parity disks to store error correcting parity information.
- Advantages: High data transfer rates due to parallel data access
- Disadvantages: Only one I/O request can be handled at a time, so performance suffers in high-transaction environment.
- Application: Large file transfers where data throughput is needed.

(V) RAID Level 4

- Characteristics: Data is striped across multiple disks, and parity is stored on a dedicated disk.
- Advantages: Reduces the no. of disks that need to be accessed for each write.
- Disadvantages: Write penalty, as every write involves updating both the data and parity disk (it can become a bottleneck).
- Application: applications requiring high read speeds (file servers)

(VI) RAID Level 5

- Characteristics: Similar to RAID 4, but parity is distributed across all the disks in the array, eliminating the bottleneck issue.
- Advantages: Improved fault tolerance and read/write performance.
- Disadvantages: Write penalty due to parity calculations.
- Application: General-purpose storage systems where a balance of performance, cost and fault-tolerance is needed.

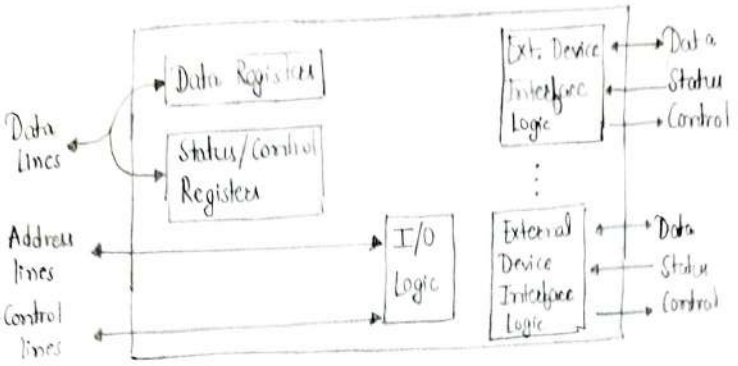
(VII) RAID Level 6

- Characteristics: Similar to RAID 5, but uses two sets of parity data for added protection.
- Advantages: Can tolerate two disk failures simultaneously without data loss.
- Disadvantages: Increased overhead for parity calculation, and slower write performance due to double parity.
- Applications: Critical applications where high availability and fault tolerance are essential

# Module 2: Input/Output Techniques

## I/O Structures and Functions

### I/O Module Structure




Block Diagram of I/O Module

- I/O module acts as an intermediatary between external devices and the computer system

Components:

(i) Status lines — Connects to the system bus

(ii) Data Registers — Stores data as buffer during transfer

(iii) Status/Control Registers — Stores device status and act as control registers

(iv) Control Lines — Allows CPU to send commands (control signals)

(v) Address Recognition logic — Identifies connected device

(vi) External Device Interface logic — Interfaces with specific peripherals.

### Functions of I/O Module

(i) Control and Timing — manage and co-ordinate data flow between internal resources and external peripherals, ensuring smooth communication

(ii) Processor Communication

- Command Decoding: Interpret/Decode commands from the CPU
- Data Exchange: Transfer of data between CPU and I/O module via the data bus.
- Status Reporting: Indicates device status (BUSY, READY, ..)
- Address Recognition: Identifies each device with a unique address.

nds ... the I/O module

(iii) Device Communication
- I/O modules handle communication between the system and ext. devices.
- It exchanges commands, status information and data with peripherals.

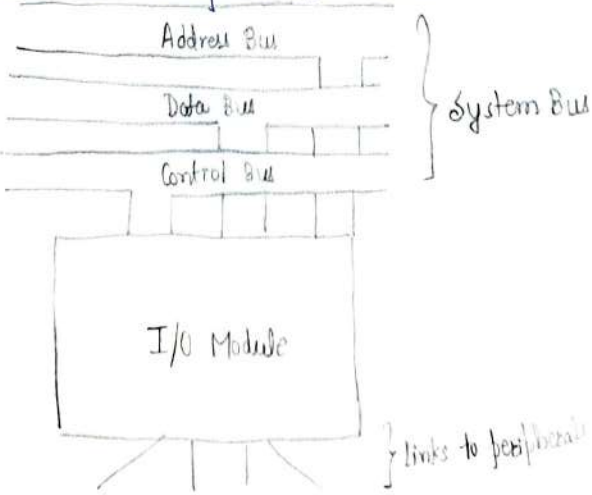
(iv) Data Buffering
- I/O modules bridges speed differences between CPU/memory (fast) and peripherals (slow).
- Stores data temporarily before forwarding it at appropriate speeds.
- Prevents CPU from being occupied by slow data transfers

(v) Error Detection
- Detects transmission errors and device malfunctions.
- Uses techniques like parity bits to identify errors in data transmission.
- Reports issues (like paper jams, bad disk tracks) to the CPU

## Types of I/O Modules

(i) I/O Controller (Device Controller)
- Simple, requires CPU control for device operations.
- Found in micro-computers (e.g. - keyboard, mouse)

(ii) I/O Channel (I/O Processor)
- More advanced, manages I/O operations with minimal CPU involvement.
- Used in mainframes for complex I/O task.




Generic Model of an I/O Module

(1) Programmed I/O

- The CPU issues an I/O command and waits for its completion.
- The I/O module performs the requested action and then sets the appropriate bits in the status register.
- The I/O module does not alert the processor, instead the processor has to periodically check the status of the I/O module, wasting processing time

## I/O Commands

- Issued by the CPU to the I/O module.

Types:

(i) Control: activates the peripheral and defines operations

(ii) Test: checks device status

(iii) Read: fetches data from device to internal buffer

(iv) Write: sends data from the CPU to the peripheral via the I/O module.

- The CPU must check the status register before each read/write operation, making the process slow.

## I/O Instructions

- Executed by the processor

(i) Each external device is given a unique identifier or address.

(ii) The CPU issues commands with this address given to the device.

(iii) The I/O module determines if the command is meant for it by checking the address.

* Key Disadvantage:

- The CPU is occupied checking device status and waiting for completion, leading to inefficient use of processing time
- Performance degrades as the processor remains idle while waiting for I/O to complete.

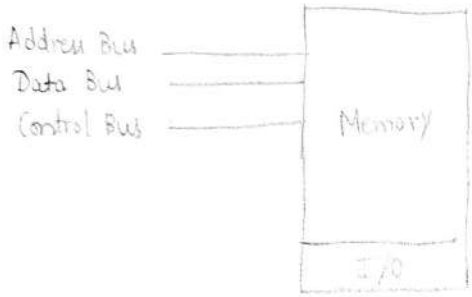
## Addressing Modes in Programmed I/O

### (i) Memory-Mapped I/O

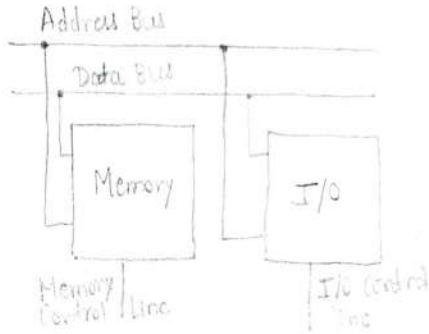
- Memory and I/O devices share the same address space.
- The CPU accesses I/O devices like memory using the same instruction.
- Requires only one set of read/write lines on the bus

### (ii) Isolated I/O

- Memory and I/O have separate address spaces.
- Uses separate control signals for Memory and I/O operations.
- Allows independent allocation of memory and I/O addresses

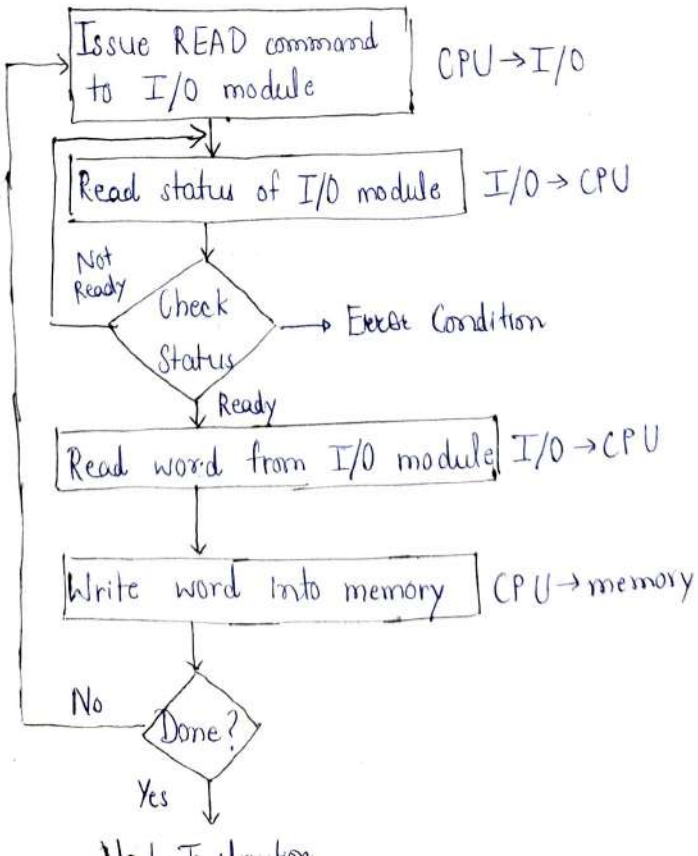



(i) Memory - Mapped I/O




(ii) Isolated I/O

Flowchart:
(for i/p of a block of data by programmed I/O)



(ii) Interrupt-Driven I/O

- Processor issues an I/O command and continues executing other tasks.
- When the I/O module is ready, it sends an interrupt to the processor.
- The processor then handles the I/O transfer and resumes its previous task.
- Reduces idle processor time compared to programmed I/O.

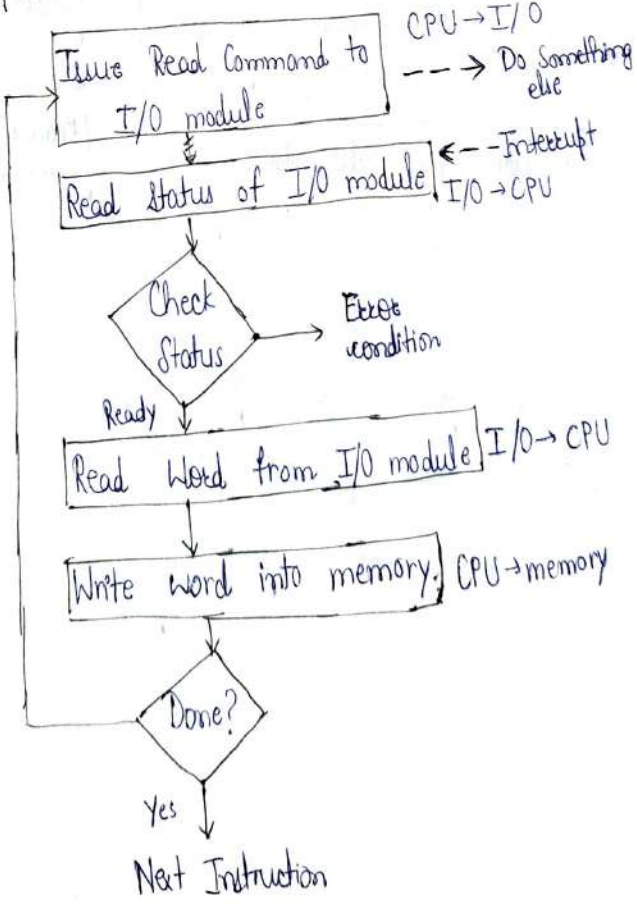
Processor Perspective

(i) Issues a READ command and continues executing other instructions.
(ii) At the end of each instruction cycle, it checks for interrupts.
(iii) If an interrupt is detected, the processor saves the current execution state, handles the I/O, and resumes the interrupted program.

I/O Module Perspective

(i) Receives a READ command from the processor.
(ii) Reads data from the peripheral into its data register.
(iii) Sends an interrupt signal to the processor.
(iv) Waits for the processor to read the data from the I/O module.

Flowchart for Input of a block of data with interrupt-driven I/O:



# Interrupt Processing

(i) The device sends an interrupt request to the processor

(ii) The processor completes the current instruction before responding

(iii) The processor acknowledges the interrupt, allowing the device to remove its request.

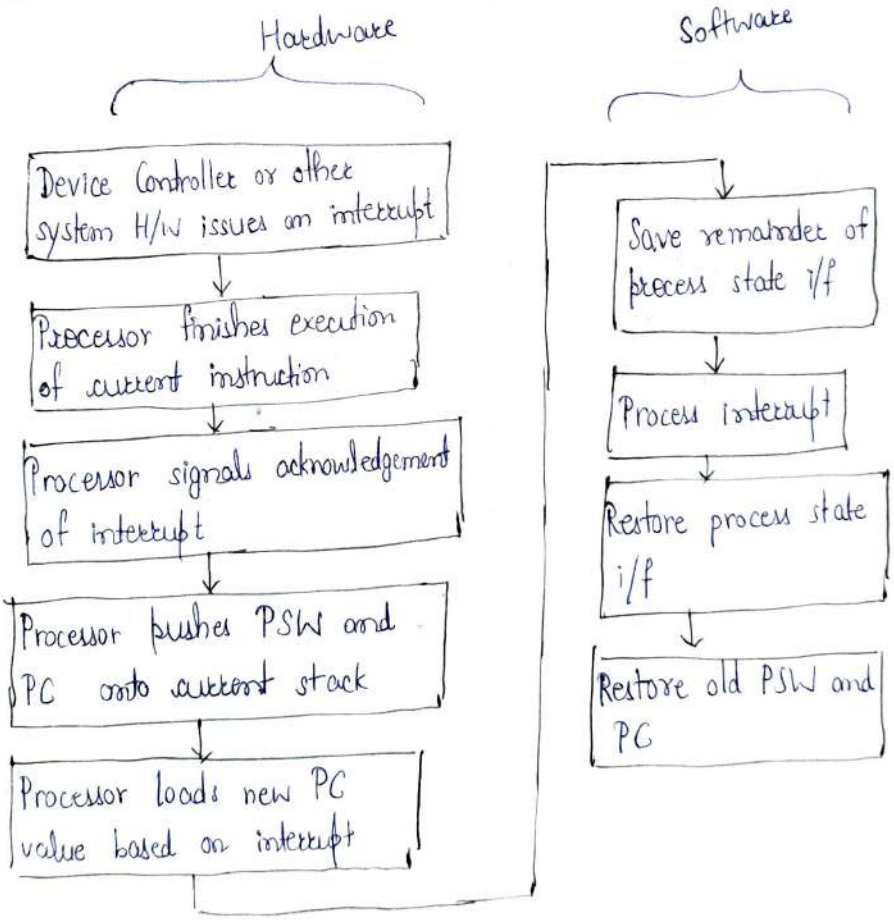
(iv) The processor saves its current execution state (Program Status Word (PSW) and Program Counter).

(v) The Program Counter is set to the address of the "interrupt handler routine."

(vi) The interrupt handler executes the necessary operations.

(vii) The processor restores the previously saved execution state.

(viii) Execution resumes from where it was interrupted.




Simple Interrupt Processing

# Design Issues in Interrupt I/O

(i) Device Identification (Which device sent the interrupt?)

- Multiple Interrupt lines: Each device has a separate interrupt line
- Software Polling: The processor queries each device sequentially to identify the source (slow)
- Daisy Chain (Hardware Polling, Vectored Interrupts):
  - All devices share a single interrupt request line
  - The interrupt acknowledge signal propagates in sequence until the requesting device responds.
  - The device sends a vector (address/identifier) to the processor for quick handling.
- Bus Arbitration (Vectored Interrupts):
  - The device must gain control of the bus before raising an interrupt.
  - The processor acknowledges and reads the interrupt vector from the requesting device.

(ii) Priority Handling (Which interrupt to process first?)

- Multiple Interrupt lines: Highest-priority line first
- Software Polling: Order of polling determines priority
- Daisy Chain: Closer the device to the processor, higher the priority
- Bus Arbitration: Follows a pre-defined priority scheme.

## Advantages

(i) Efficient Use of CPU time: The processor can perform other tasks while waiting for the I/O.

(ii) Reduces waiting time: No need for continuous status checking.

(iii) Supports multi-tasking: Processor can switch b/w multiple tasks and handle I/O asynchronously.

# Direct Memory Access (DMA)

## Drawbacks of Programmed and Interrupt-Driven I/O

(i) Processor Overhead: Processor is actively involved in transferring data b/w memory and I/O devices.

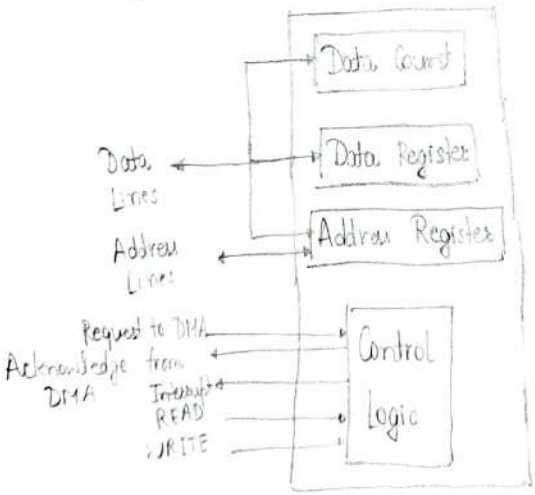
(ii) Limited I/O Transfer Rate: Speed to I/O transfer is restricted by how fast the processor can process and service I/O request.

(iii) Trade-Off:

- Programmed I/O — dedicates the processor to I/O but achieves high transfer rates
- Interrupt-Driven I/O — allows processor to perform other tasks but at the cost of slower I/O transfer

## DMA Function

- DMA Module: A separate module that takes over data transfer b/w memory and I/O devices without involving the processor.
- Cycle Stealing: The DMA module takes control of the system bus temporarily by forcefully suspending the processor operations for data transfer.




Typical DMA Block Diagram

## Steps in DMA operation

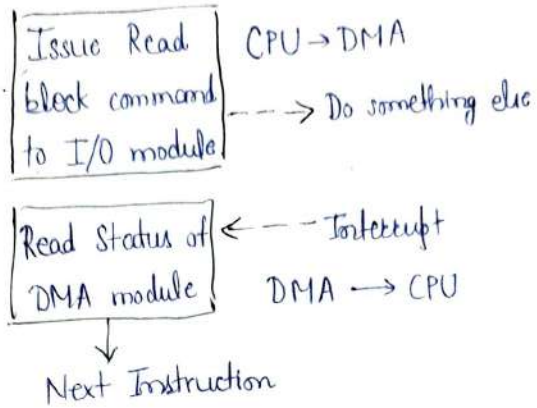
(i) Processor issues command to DMA module.

- Specifies READ/WRITE operation
- Provides device address, memory location and count of words to read/written.

(ii) DMA transfers data
- directly moves data b/w memory and I/O device without processor intervention.

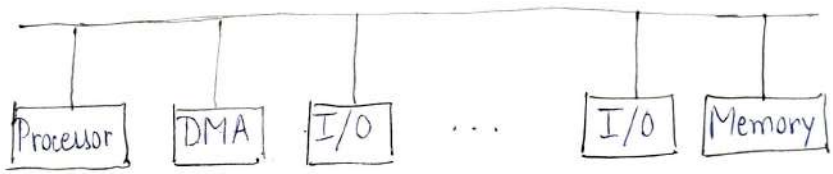
(iii) DMA sends interrupt
- notifies the processor when the transfer is complete

Flowchart:
(for input of a block of data)

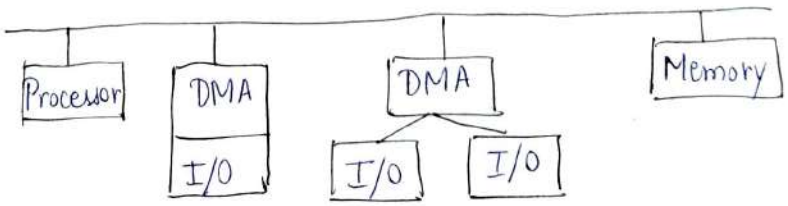



Types of DMA configuration

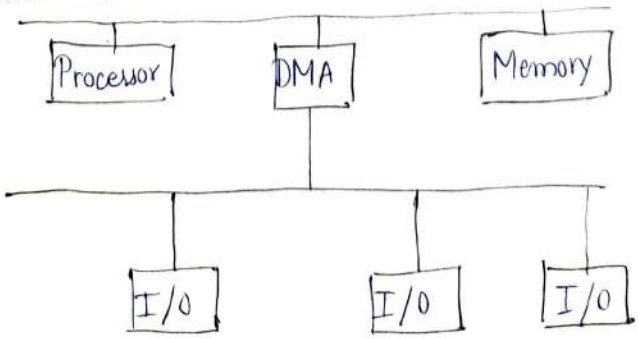
(i) Shared Bus DMA
- All modules share the same system bus.
- Inefficient as each data transfer requires two bus cycles




(ii) Integrated DMA and I/O
- DMA module has a direct connection to I/O devices, reducing bus usage




(iii) I/O bus based DMA
- A dedicated bus connects DMA to I/O modules; more efficient and expandable.



## Advantages of DMA

(i) Reduces Processor Overhead: CPU is only involved at the start and end of a transfer

(ii) Faster Data Transfers: Transfers large data blocks efficiently

(iii) Efficient Bus Utilization: Avoids unnecessary processor intervention and minimizes bus cycles.

## Direct Cache Access (DCA)

- DCA is a technique that allows peripherals (like NICs) to directly read/write data into CPU cache, bypassing even the main memory
- This improves performance by reducing memory latency and increasing data transfer speed.

* Key Features:

(i) Bypasses Main Memory: DCA eliminates the need for data to be first transferred to the RAM.

(ii) Reduces Latency: CPU can directly access cache.

(iii) Improves Performance: commonly used in high speed networking devices.

(iv) Works with DMA: generally implemented alongside DMA to optimize data transfers.

(v) Common Use Cases:

- High-speed network packet processing
- Storage device acceleration
- Real-time computing and gaming