Page NO.

1. 1 to 53 – Mid Term
2. 54 to 122- Final Term

MD IFTAKHAR KABIR SAKUR

25th BATCH

COMPUTER AND COMMUNICATION ENGINEERING

International Islamic University Chittagong

COURSE CODE: CCE-3509

COURSE TITLE: Computer Architecture And Organization

COURSE TEACHER:

Mohammad Nadib Hasan

Lecturer
Computer and Communication Engineering

CCE-3509

Computer Architecture & Organization

Computer Architecture:-

Interconnectivity of computer's hardware components as well as mode of data transfer & processing.

Example:-

~~8086, made~~

→ x86, made by Intel & AMD

→ The SPARC, made by sun microsystems and others

→ The powerpc, made by Apple, IBM & Motorola.

Computer Organization:-

The way system is structured so that all those catalogued tools can be used. The significant components of computer organization are ALU, CPU, memory & memory organization.

Differences between Computer Architecture & Computer organizations

Computer Architecture	Computer Organization
(1) It deals with the functional behavior of Computer Systems.	(1) It describes how it does it.
(2) It deals with the functional behavior of Computer systems.	(2) It deals with a structural relationship
(3) It deals with high-level design issues. example:- (i) von-Neumann Architecture (ii) Harvard Architecture (iii) Instruction Set Architecture (iv) Micro-architecture	(3) It deals with low-level design issues (1) organization of Angle accumulator (2) Organization of general registers (3) Stack organization

Computer Architecture	Computer Organization
(4) Architecture indicates its hardware.	(4) Organization indicates its performance.
(5) As a programmer, it can be viewed as a series of instructions, addressing modes and registers.	(5) The implementation of the architecture is called organization.
(6) For designing a computer, its architecture is fixed first.	(6) For designing a computer, an organization is decided after its architecture.
(7) It is called as Instruction Set Architecture. (ISA)	(7) It is frequently called microarchitecture.
(8) It makes the computer Architecture visible.	(8) It offers details on how well computer performs.

Computer Architecture	Computer Organization
(9) Architecture coordinates the hardware or and software of the system.	(9) Computer organization handles the segments of the network of it, a system.
(10) The software developer is aware of it.	(10) It escapes the software programmer's detection.
(11) Example:- Intel, AMD created the x86 processor. Sun Microsystems & Others created SPARC processor. Apple, IBM and Motorola created the powerpc.	(11) Example:- Organizational qualities include hardware elements that are invisible to the programmer, such as interfacing of computer & peripherals, memory technologies, and control signals.

F) Computer Design is the structure in which components relate to each other.

Issues in Computer Design:-

→ Not having infinite speed memory

→ " " "

→ Speed Mismatch between memory & processor.

→ Multiple processor (leads to the huge task of management & programming)

→ Multiple Threads &

→ Shared memory (Several process at a time, so it should be managed in a specific way so that the collision never happen)

→ Handling of bugs & errors:-

Bugs & errors lead to the failure of the computer system. Sometimes these errors may be more dangerous.

→ Disk access

→ Better performance.

Basic operational Concepts

⇒ The primary function of a Computer system is to execute a program, sequence of instructions. These instructions are stored in Computer memory.

⇒ These instructions are executed to process data which are already loaded in the Computer memory through some input devices.

⇒ After processing data, the result is either restored to Computer memory or, sent to the outside world through some output port.

⇨ To perform the execution of an instruction, works related to ALU, Control Unit, the processor Contains a number of registers used For temporary storage OF data.

⇒ The program Counter monitors the execution of instructions. It keeps track of which instruction is being executed and what the next instruction will be.

⇒ Instruction Register (IR) is used to hold the instruction that is cwocently being executed.

⇒ TWO register Memory Address Registers (MAR), Memory Data Registers (MDR) are used to transfer data transfer between main memory & the processor.

⇒ The special function registers include PC, IR, MAR & MDR.

⇒ MAR holds the address of the main memory to or from which data is to be transferred.

⇒ MDR contains the data which to be written into or read from the address

⇒ processor communicates with devices which is known as servicing the ~~devicing~~ devices. This can service in two ways :-

⇒ ① It can be used the polling routine, and the other way is to use an interrupt.

② Polling enables the processor software

to check each of the input and output devices frequently. During this check, the processor tests to see if any devices need servicing or not.

=> Interrupt method provides as external asynchronous input that informs the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device.

General System Architecture

⇨ Store program Control Concept -

(i) Von-Neumann Model.

(ii) General purpose System.

(iii) parallel processing.

=> Flynn's Classification of Computers :-

(i) Single Instruction Stream, Single Data Stream (SISD)

(ii) Single Instruction stream, Multiple Data stream (SIMD)

(iii) ~~Mulp~~ Multiple Instruction stream, Single Data stream (MISD)

(iv) Multiple Instruction stream, Multiple Data Stream (MIMD)

Stored program Control Concept

ENIAC (Electronic Numerical Integrator & Computer) was first computing system designed in the early 1940s. It was based on stored program concept in which machine use memory for processing data.

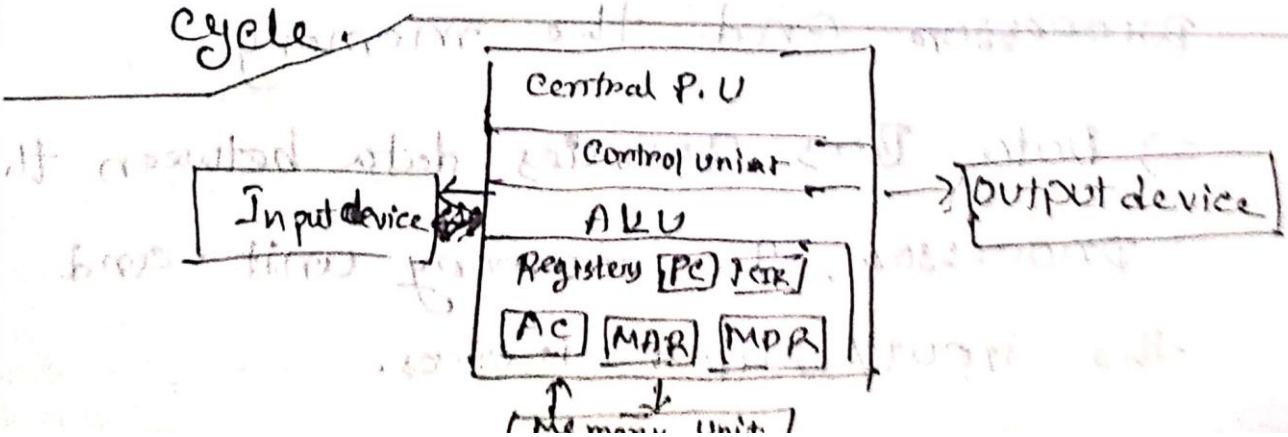
Von-Neumann Model

This is based on:-

→ A single processor

→ One memory for both instructions & data.

→ Following the fetch-decode-execute cycle.



Components of VN Model

① CPU :-

The part of the computer that performs the bulk of Data processing operations is called as the Central processing unit. And it is referred to as CPU.

⇒ It is an electric circuit responsible for executing instructions.

→ CPU :- ALU, CU (Control Unit), variety of registers.

② Buses :-

Address Bus carries the address of data (but not the data) between the processor and the memory

⇒ Data Bus carries data between the processor, the memory unit and the input/output devices.

⇒ Control Bus carries signals / commands from the CPU.

Memory - Unit:

Collection of storage cells together with associated circuits needed to transfer information in & out of the storage.

The internal storage of memory unit is specified by the number of words it contains and the number of bits in each word.

(Memory stores binary info in groups of bits called words)

→ 2 Major types of Memory:

→ ROM

→ RAM

General purpose System

Modified version of VN Model.

→ cpu (ALU, CU, Registers)

→ Memory Unit

→ Input, output

→ Connected by Bus

→ Includes data, Address, Control Status lines

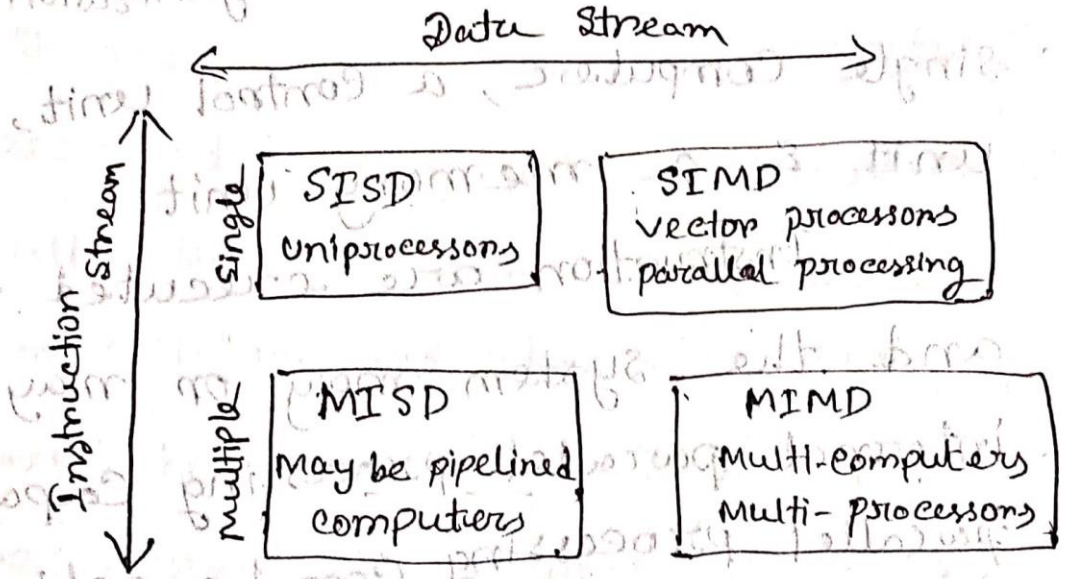
Flynn's Classification of Computers

(organisation वर्गीकरण)

⇒ proposed by MJ Flynn a classification for the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

=> The sequence of instructions read from memory constitutes an instruction stream.

=> The operations performed on the data in the processor cont constitute a data stream



→ Flynn's Classification of computers

SISD (Single Instruction & Data Stream)

CU = Control Unit
PE = processing Element
M = memory

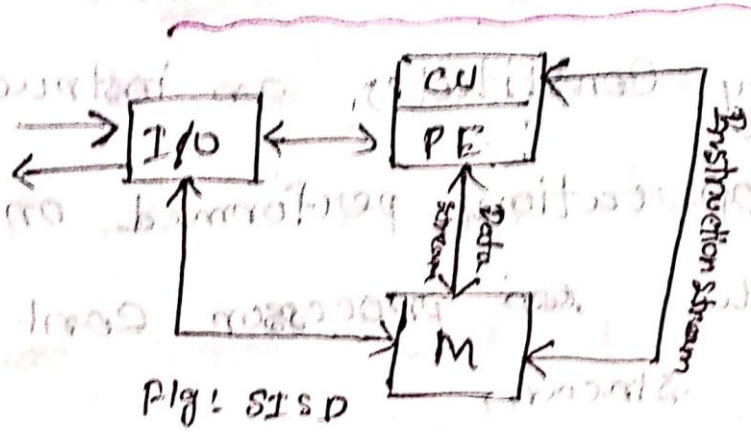


Fig: SISD

Data Stream Flows between the processors & memory is bi-directional

=> SISD represents the organization of a single computer, a control unit, a processor unit & a memory unit.

Instruction are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing can be achieved by means of multiple functional units or by pipeline processing.

SISD architecture is like von-Neumann computers.

Instructions → Decoded by CU → Sends instruction to processing Unit → Execution

SIMD (Single Instruction & Multiple Data Stream)

⇒ SIMD → Includes many processing units
→ A common control unit (CU).

⇒ Receives the same instruction from CU but operate on different types of data.

The shared memory unit must contain multiple modules so that it can communicate with all the processor simultaneously.

It is mainly dedicated to array processing machines. However, vector processors can also be seen as a part of this group.

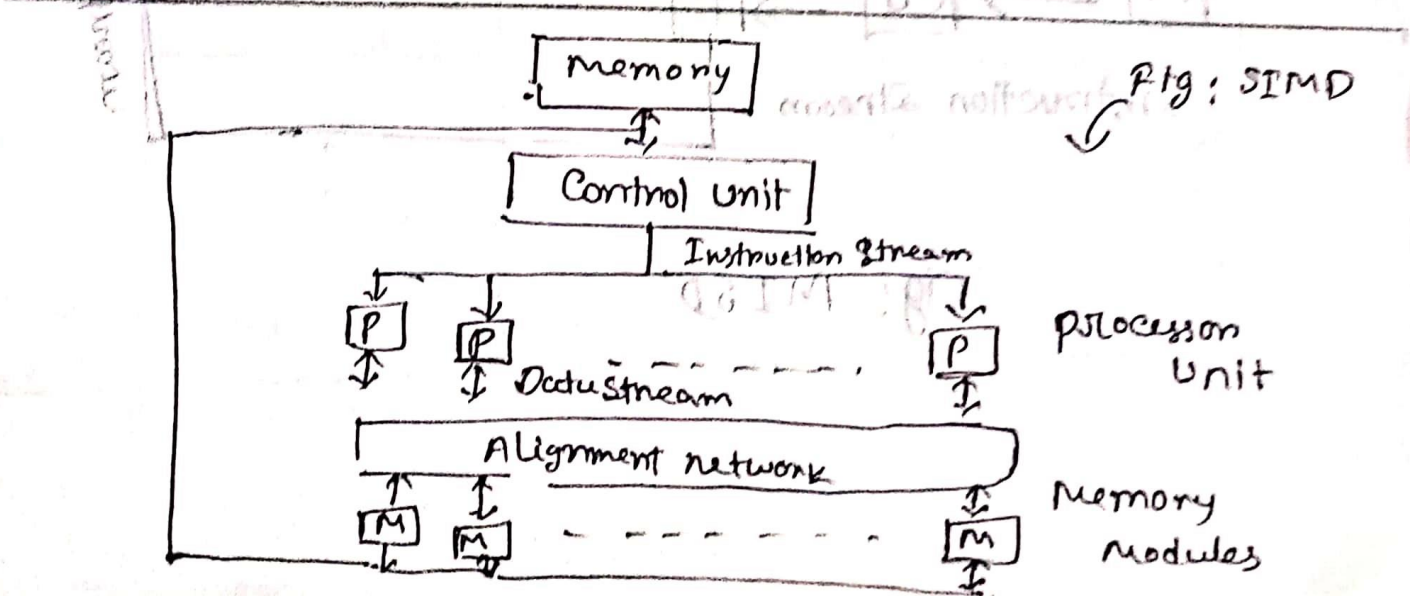


Fig: SIMD

MISD (Multiple Instruction & Data Stream)

It is only theoretical interest since no practical system has been constructed using this organization.

In this multiple processing unit operate on one single-data stream.

Experimental:-

Carnegie-Mellon C.mmp Computer (1971)

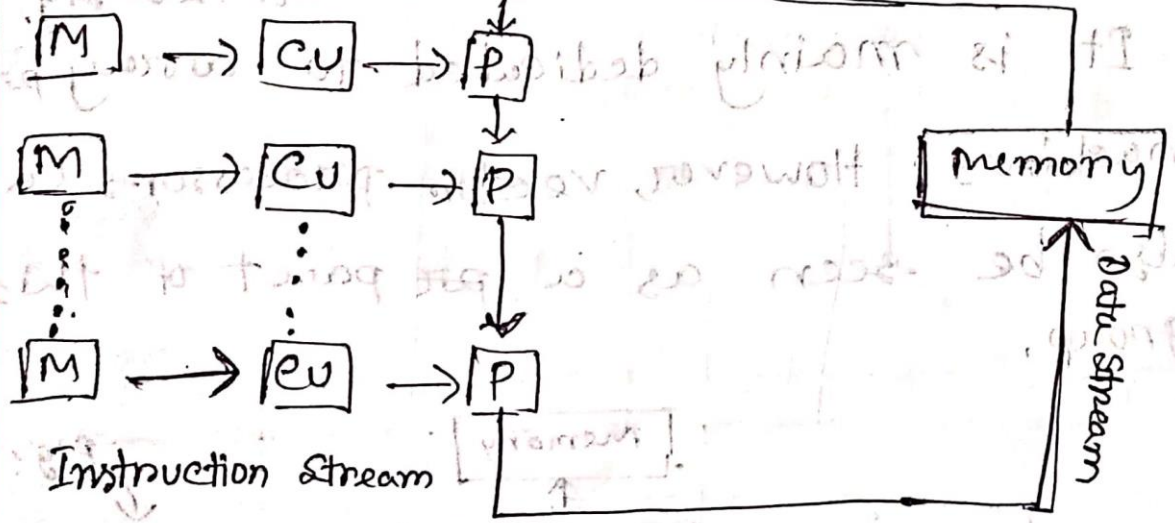


Fig: MISD

MIMD (Multiple Instruction & Multiple Data Stream)

Each processor has a separate program and and Instruction stream is generated from each program.

Ex: Cray T90, Fa·cray T3E, IBM-SP2

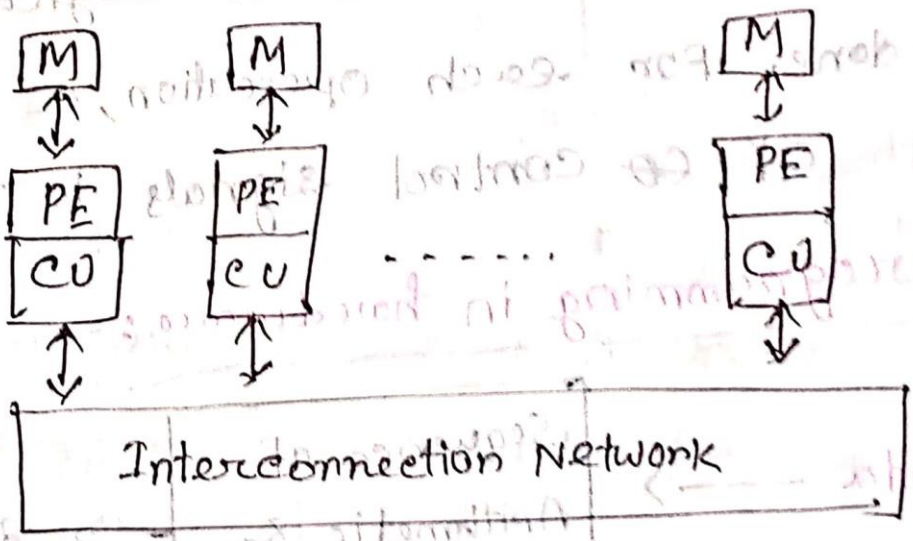


Fig: MIMD

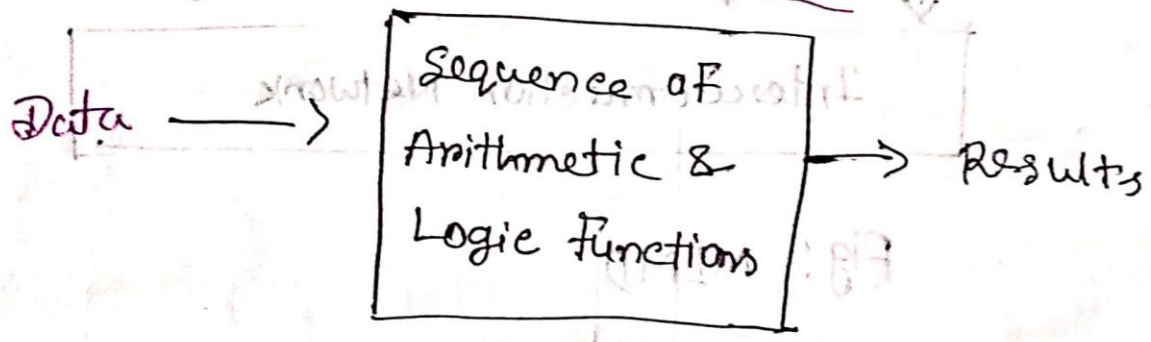
Lesson-02

Computer Function & interconnection

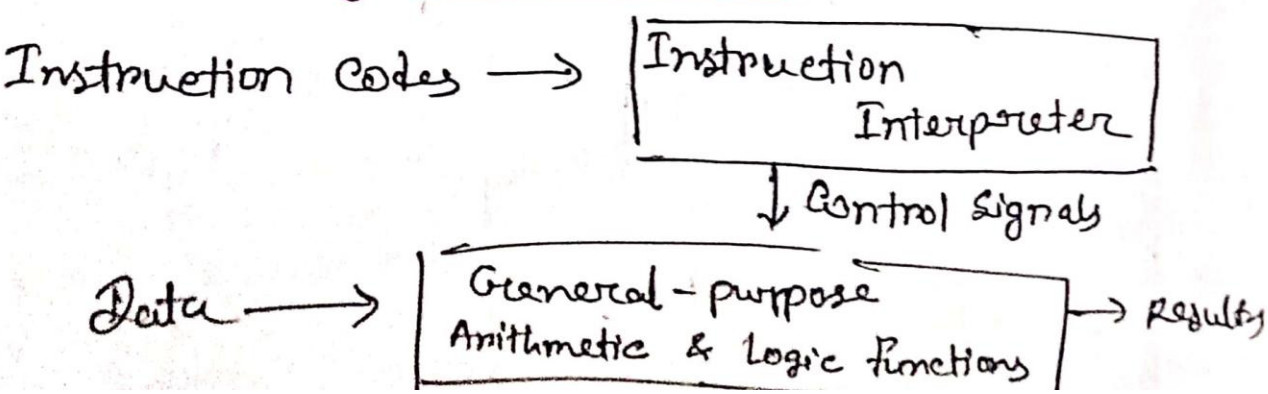
Computer Components

Program:- A sequence of steps. For each step, an arithmetic or logical operation is done. For each operation, a different set of control signals is needed.

✱ programming in hardware:-



✱ programming in Software:-



(*) The Control Unit & ALU Constitute the CPU

(*) Data & instruction get into the system & gives us the result.

Input/Output:-

→ Storage of Code (Input)
→ The result (output)

Main Memory:-

CPU exchanges data with memory, For this purpose, it typically makes use of two internal (to the CPU) ~~pest~~ registers.

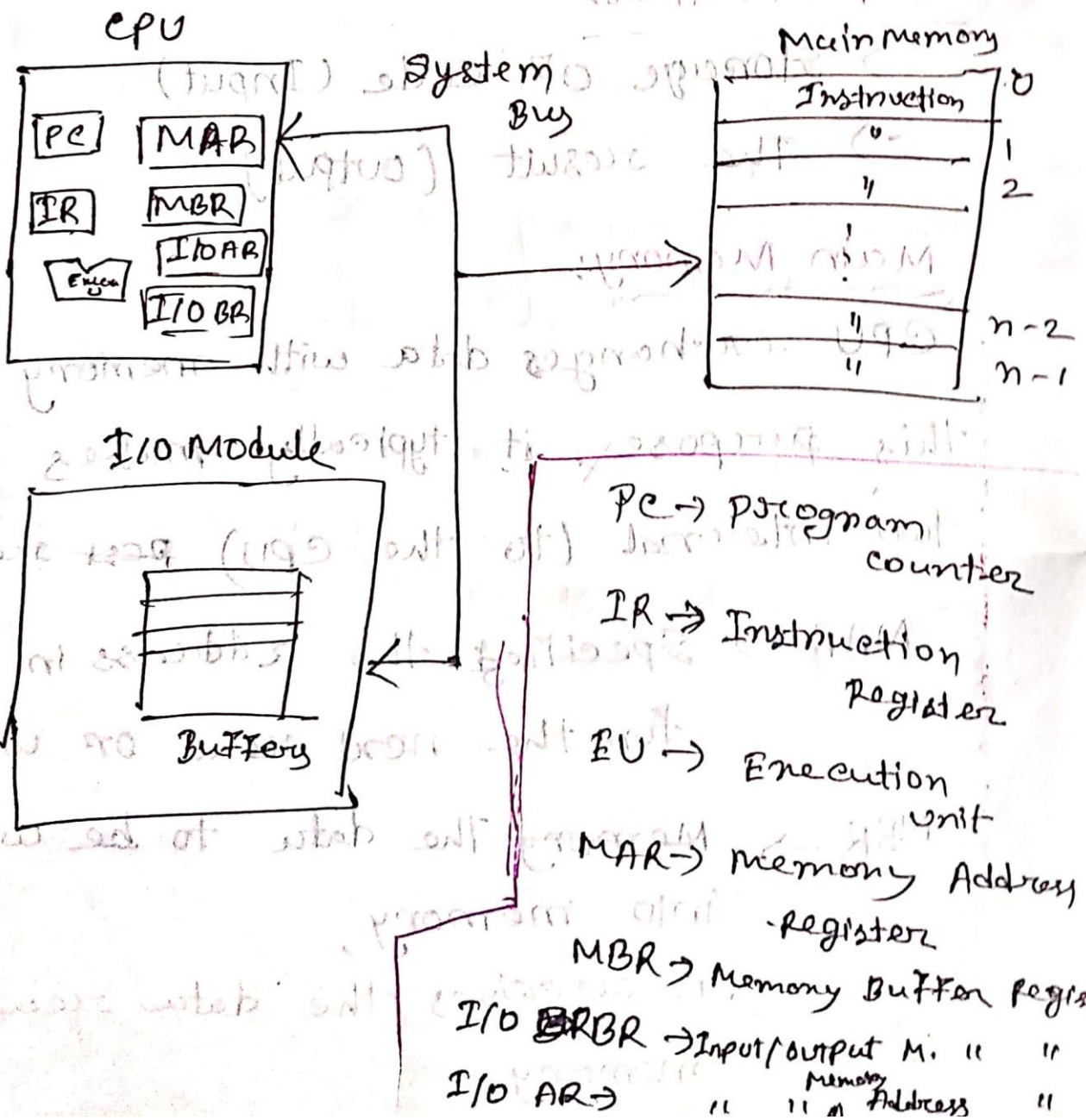
MAR → Specifies the address in memory for the next read or write

MBR → ~~Memory~~ The data to be written into memory,
Or, receives the data read from memory.

I/O AR → Specifies particular I/O device

I/O BR → exchange of data between an I/O module & the CPU.

Architecture



Instruction cycle

Instruction

=> The processing required for a single instruction is called an instruction cycle.

Two step:-

→ Fetch cycle

→ Execute cycle

Fetch cycle

→ Program Counter (PC) holds address of next instruction to fetch

→ Processor fetches instruction from memory location pointed to by PC.

→ Increment PC

→ Instruction load into Instruction Register (IR)

→ Processor interprets instruction & performs required actions.

④ ~~Interprets~~ processor interprets the instruction & performs the required action. These actions fall into four categories.

Processor-Memory:-

Data can be transferred from processor to memory or memory to processor.

Processor-I/O:-

Data may be transferred to or from a peripheral device by transferring between the processor & an I/O module.

Data processing:-

The processor may perform some arithmetic or logic operation on data

Control:-

An instruction may specify that the sequence of execution be altered.

For Example:- The processor may fetch an instruction from location 149, which specifies that the next instruction be from 182. The processor will remember this fact by setting the PC to 182. Thus on next fetch cycle, the instruction will be fetched from location 182, not from 150.

Execute cycle

* Processor – Memory :-

Data transfer between CPU & main memory.

* Processor I/O :-

Data transfer between CPU & I/O module.

(*) Data processing :-

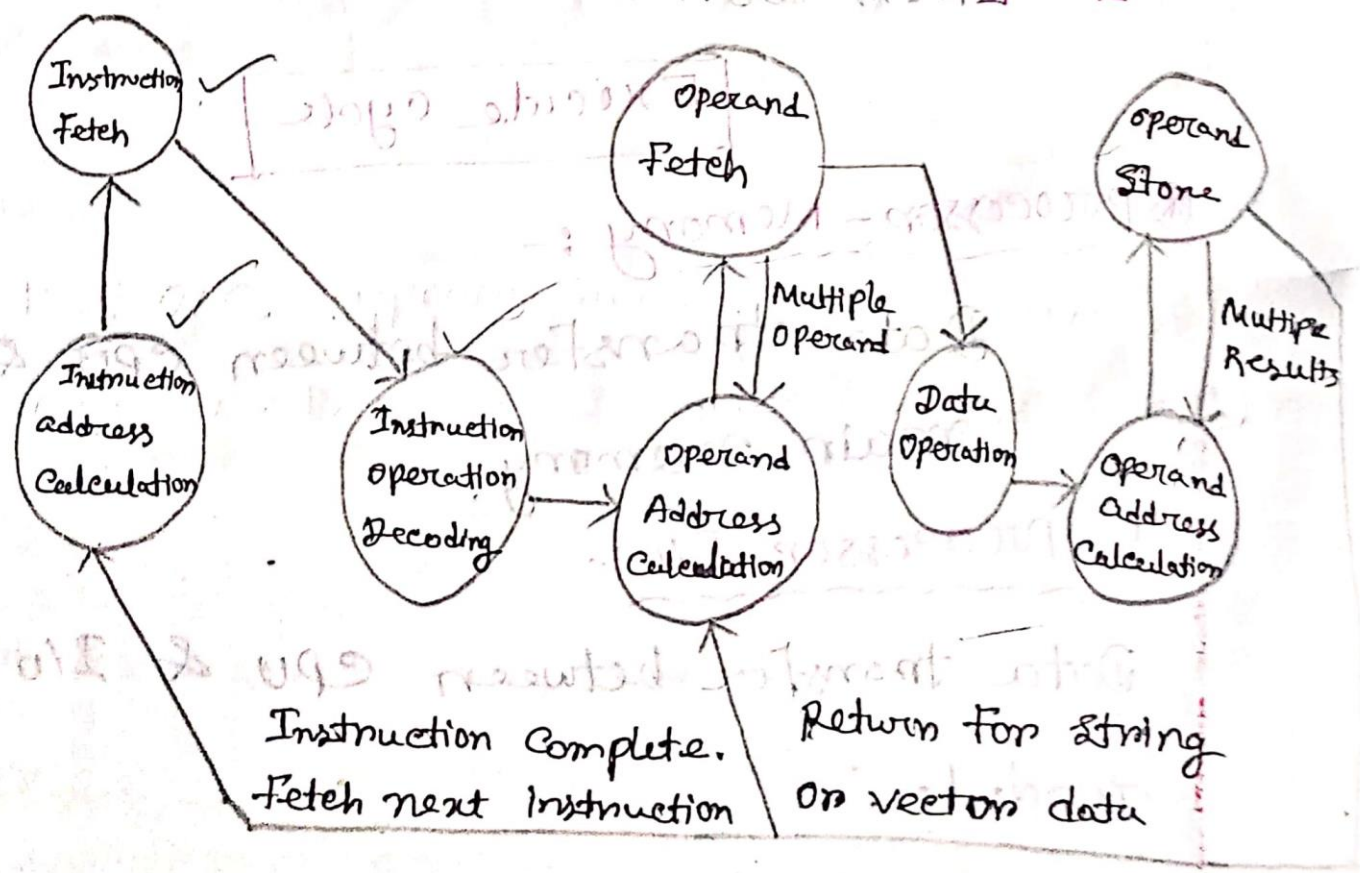
Some arithmetic or logical operation on data.

(*) Control :-

Alteration of sequence of operations

example :- Jump.

The figure is in the form of a state diagram.



Operand:- The quantity on which an operation is to be done.

Description:-

* Instruction Address Calculation (IAC):-

Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction.

* Instruction Fetch (IF):-

Read instruction from its memory location into the processor.

* Instruction operation Decoding (IOD):-

Analyze instruction to determine type of operation to be performed & operand(s) to be used.

* Operand Address Calculation (OAC):-

If the operation involves reference to an operand in memory or available via input I/O, then determine the address

of the operand.

* Operand Fetch (OF):-

Fetch the operand from memory or read it in from I/O.

* Data operation (DO):-

perform the operation indicated in the instruction.

* Operand Store (OS):-

write the result into memory or out to I/O.

NOTE:-

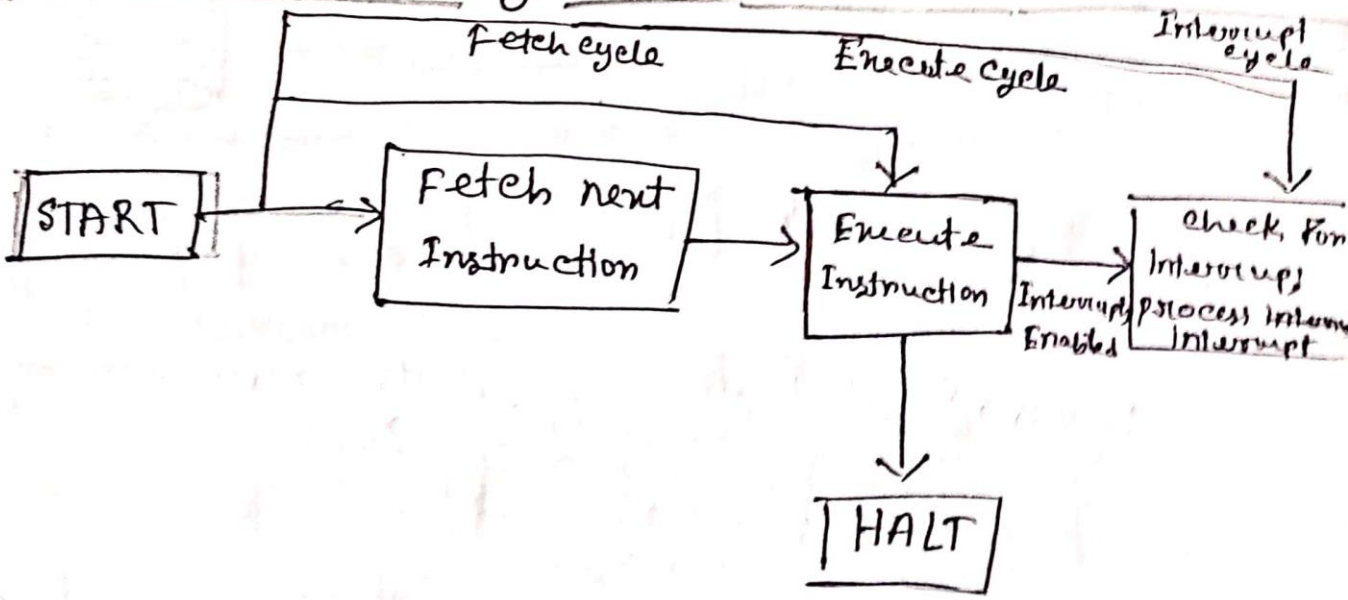
The OAC appears twice, because an instruction may involve a read, a write & both.

Instruction cycle with interrupts

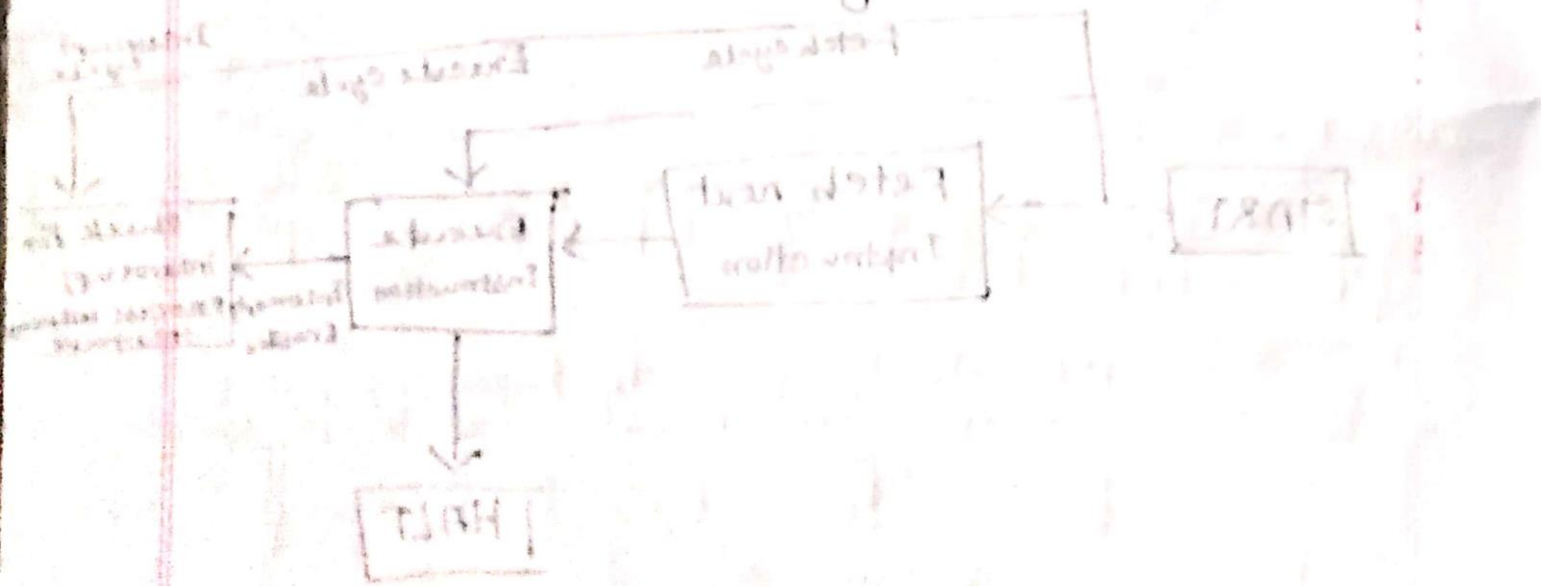
The processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.

If no interrupts are pending, the processor proceeds to the fetch cycle & fetches the next instruction of the current program.

But, If interrupt is pending, the processor does the following:-



* It suspends execution of the current program being executed and saves its content. This means, saving the address of the next instruction to be executed (current contents of the program Counter) & any other data relevant to the processor's current activity.
It sets the PC to the starting address of an interrupt handler routine.



Connecting

- 3 basic types of Components :-

→ processor

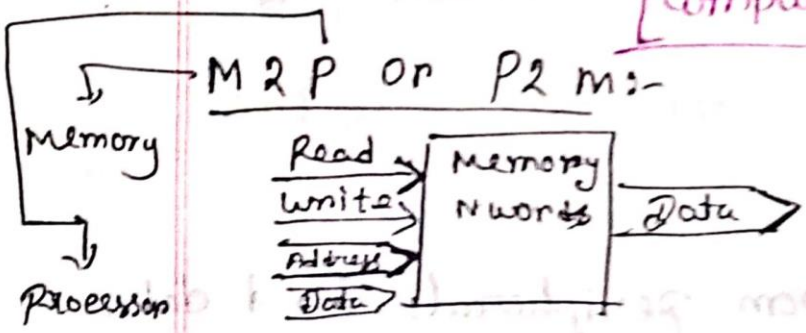
→ Memory

→ Input / ~~Out~~ output

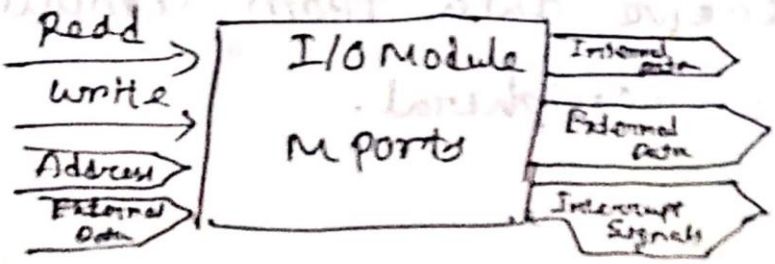
The collection of paths connecting the various modules is called the interconnection structure.

Computer Modules

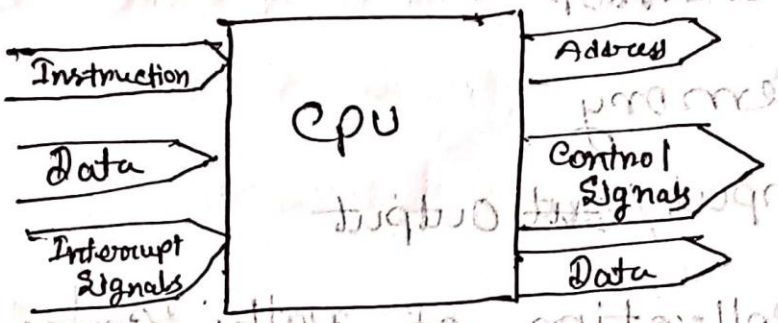
M 2 P or P 2 M :-



I/O to processor
or, processor to I/O :-



I/O to Memory
or
Memory to I/O :-



Memory Connection

→ Read
→ write
→ Timing

Input:-
Receive data from peripheral, send data to computer.

Output:-
Receive data from computer, send data to peripheral.

CPU Connection

⇒ Reads instruction & data

⇒ Writes out data (after processing)

⇒ Sends controls signals to other units.

⇒ Receives interrupts.

Buses

⇒ A bus that connects major computer components (processor, memory, I/O) is called a system bus.

This is used of one or more system buses.

* 32 bit data bus is 32 separate bit channels.

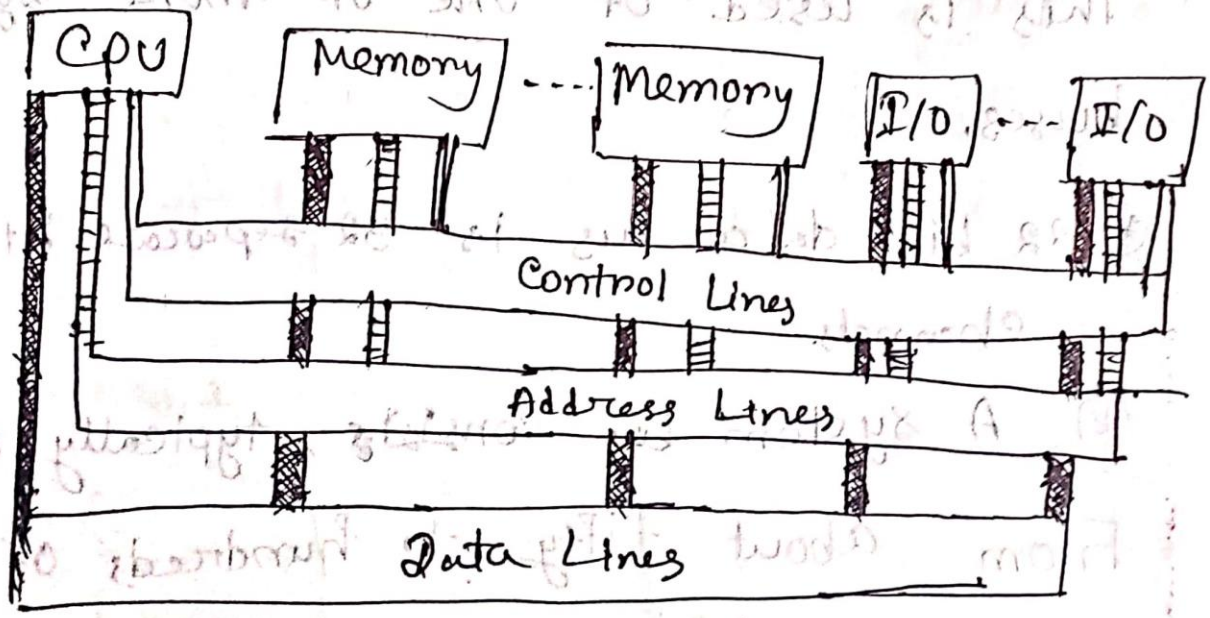
* A system bus consists, typically, of from about fifty to hundreds of separate lines.

Each line is assigned a particular meaning or function. The bus can be classified into three functional groups:-

→ data
→ address
→ Control lines

There may be a power distribution lines that supply power to the attached modules.

Bus interconnection scheme



Data Bus

Provide a path for moving data among system modules carries data.

→ Width is a key determinant of performance.

For example:-

If the data bus is 32 bits wide & each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

* It is bidirectional

  µp → Memory or µp → I/O

  and, Memory → I/O

* Write operation :- put data on the data Bus

* Read operation :- Data from specific memory & put it into the data bus.

Address Bus

* Carries Address only
→ Unidirectional
MP → memory
or, MP → I/O

Example:-

Processor wishes to read a word (8, 16 or 32 bits) of data from memory. It puts the address of the desired word on the address lines.

→ 8085 has 16 bit address but giving 64K address space.

Control Bus

→ Generate timing & control signals to control all the associated peripherals.

→ The data & address lines are shared by all components, there must be a means of controlling their use.

→ Command signals specify operations to be performed.

Some control signals are:-

→ Memory write :- Bus written into the address location

→ Memory Read :- ~~Address~~ data from address location to be placed on the bus.

→ I/O write :- Output to the addressed I/O port

or I/O Read :- Addressed I/O port to be placed on the bus.

Transfer ACK :- Data accepted from or placed on the bus.

* Bus request:- Module needs to gain control of the Bus.
* Bus grant:- Indicates that a requesting module has been granted control of the bus.
* Interrupt ACK:- Acknowledges that the pending interrupt has been recognized
* Clock:- is used to synchonize operations
* Reset:- Initializes all modules.

Single vs Multiple Bus

* The greater the bus length & hence the greater the propagation delay. This delay can noticeably affect performance.
* There will be problems if the aggregate data transfer demand.

So, to get rid from this multiple buses are used.

Bus Arbitration

⇒ Refers to the process by which the current bus & then leaves the control of the bus & passes it to another bus requesting processor unit. The controller of the bus is known as bus master.

* There are two approaches to bus arbitration :-

→ Centralized Bus Arbitration :-

A single bus arbiter performs the required arbitration.

Distributed B.A :-

All devices participating in the selection of the next Bus master.

Central Bus Arbitration (CBA)

Three bus arbitration methods—

→ Daisy chaining Methods :-

All the bus masters use the same line for making bus requests. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus.

The device with the highest priority is placed in the first position. Followed by lower priority devices up to the device with the lowest priority, which is placed last in the chain.

* Polling or Rotating Priority Method:-

The devices are assigned unique priorities & competes to access the bus, but the priorities are dynamically changed to give every device an opportunity to access the bus.

* Fixed priority or Independent Request Method:-

Each master has seperate pair of bus request & bus grant lines & each pair has a priority assigned to it.

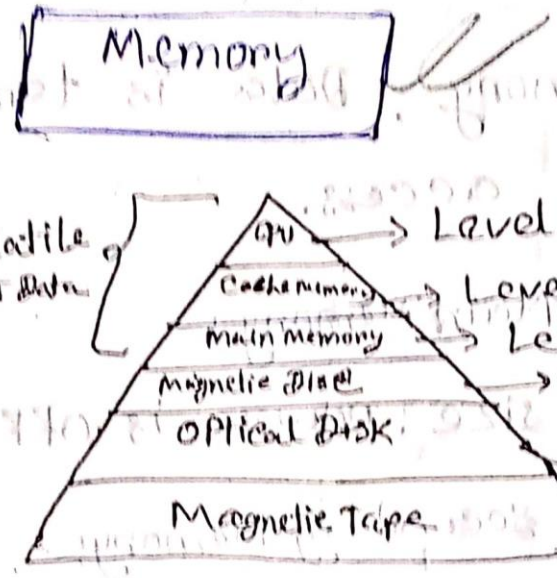
Dist

Distributed Bus Arbitration

All devices participate in the selection of the next bus master. Each device on the bus is assigned a 4 bit ID number. The priority of the device will be determined by the generated ID.

During any Bus cycle, the bus master may be any device — the Processor or any DMA Controller Unit, connected to the Bus.

Associative Memory (Chapter 4,5,6 Pdf)



Memory Hierarchy Design

Levels of Memory:-

Level 1 or Register:-

One kind of memory in which data is stored & accepted that are immediately stored in CPU.

Most Commonly used register is accumulator, program Counter, Address Register etc.

Level 2 or Cache Memory:-

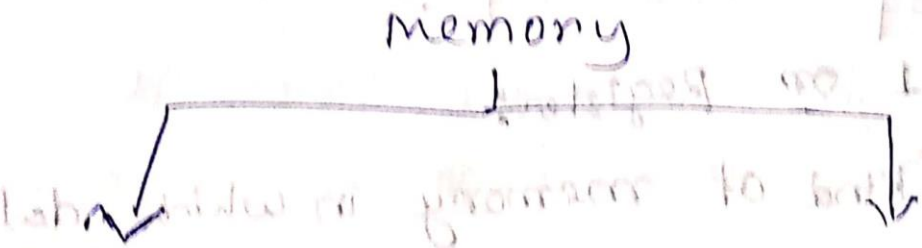
fastest memory. Data is temporarily stored for faster access.

Level 3 or Main Memory:-

Small in size. power is OFF data is vanished.

Level 4 or secondary Memory:-

Not as Fast as Main memory but data stays here permanently.



(lowest cost, highest capacity & slowest access)

provides backup to storage

→ Magnetic Discs

→ Magnetic Tapes

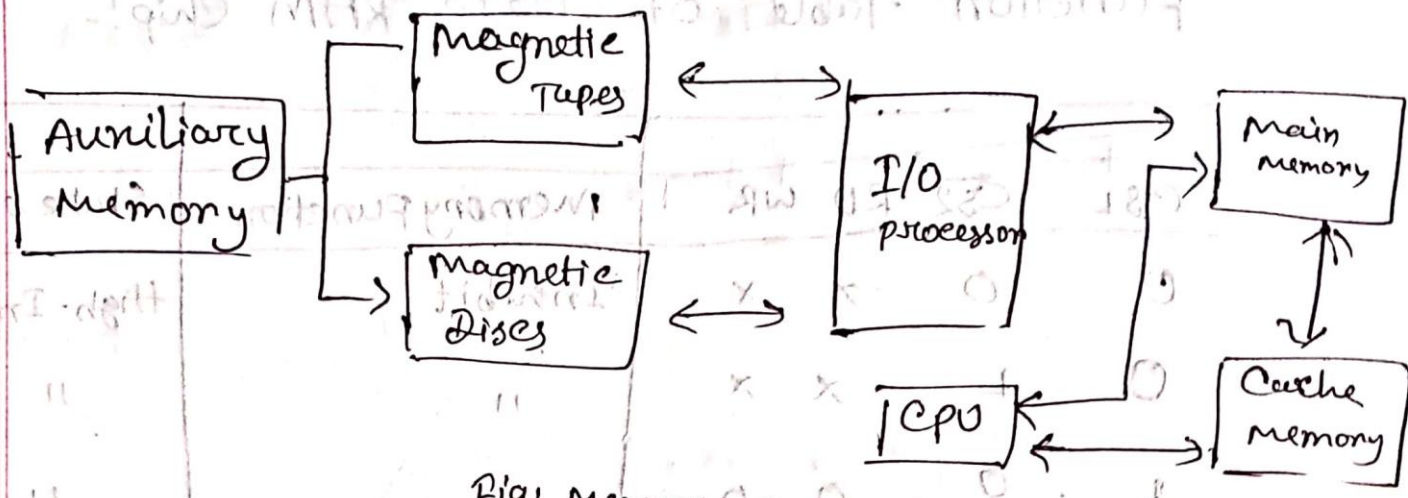


Fig: Memory Hierarchy.

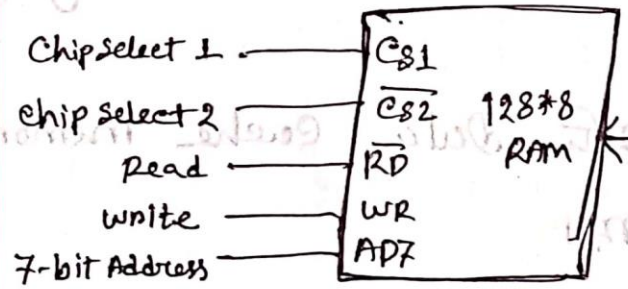
* Main memory is created with ~~Semicon~~ Semiconductor integrated circuits.

IC are classified based on Two major units:-

→ RAM

→ ROM

RAM Diagram:- 128 * 8



(128 words of 8 bits (one byte) per word → Capacity

State Buffer:- High impedence, Input Data

Data.

Function table of 128*8 RAM Chip:-

CS1	CS2	RD	WR	Memory Function	State of Data Bus
0	0	x	x	Inhibit	High-Impedance
0	1	x	x	"	"
1	0	0	0	"	"
1	0	0	1	Write	Input data to RAM
1	0	1	0	Read	Output " "
1	1	x	x	Inhibit	High Impidance

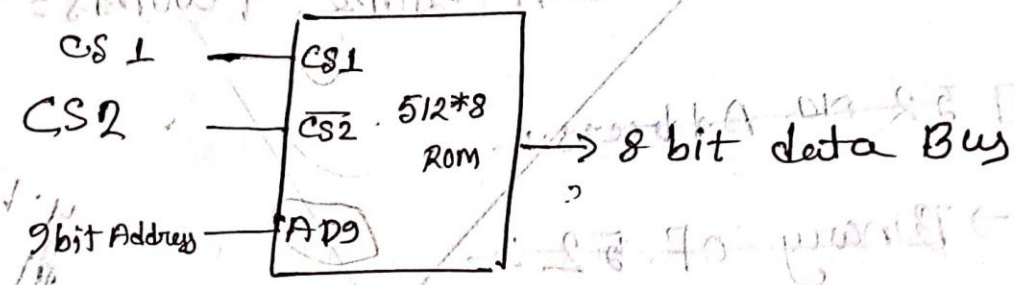
✓ Cache Hit:- CPU যদি Data cache memory তে পায়

✓ Cache Miss:- CPU যদি Data cache memory তে না পায়।

RAM → Static RAM
RAM → DRAM ⇒ (Dynamic)

ROM

16)127(7F
112
15



Decimal to Hexadecimal:-

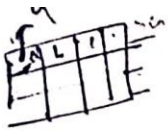
16)255(FF
16
95
80
15

16)256(10
16
96
96
0

16|209
16|13 → 1
0 → 1(D)

16)16(1
16
0

$\therefore (209)_{10} = (D1)_{16}$



64
× 8
212

Cache Mapping (Direct)

Main memory :- Size: 64 words = ~~64 byte = 64×8 = 212 bit~~

Block size : 4 words

No. of blocks in MM: 64/4 = 16 0,1,2,3,4,---15

Using 1 bit place we can address two locations.

0	m_0
1	m_1

Similarly, 2 bit place we can address 4 locations

Can be addressed

0	0	$\rightarrow m_0$
0	1	$\rightarrow m_1$
1	0	$\rightarrow m_2$
1	1	$\rightarrow m_2$

So, for 8 memory cell we will be needing

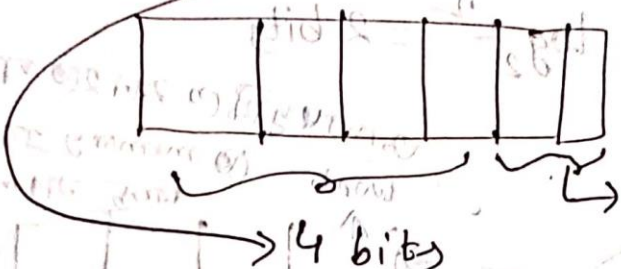
$$\log_2 8 = \log_2 2^3 = 3 \text{ bit places}$$

Similarly, for 64 bit words,

$$\log_2 64 = \log_2 2^6 = 6 \text{ bits}$$

↳ Physical Address bits

(Main memory sometimes referred to as Physical Address Space & in this physical Address space there are 16 blocks and in order to locate each one of them we will be needing $\log_2 16 = \log_2 2^4 = 4$ bits



That means:-

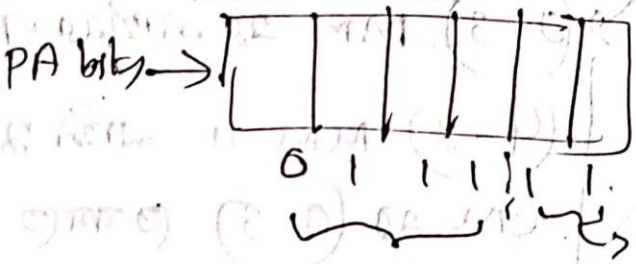
00 → 0th word

01 → 1st "

10 → 2nd "

11 → 3rd "

Examples :-



7

↓

7 no. block

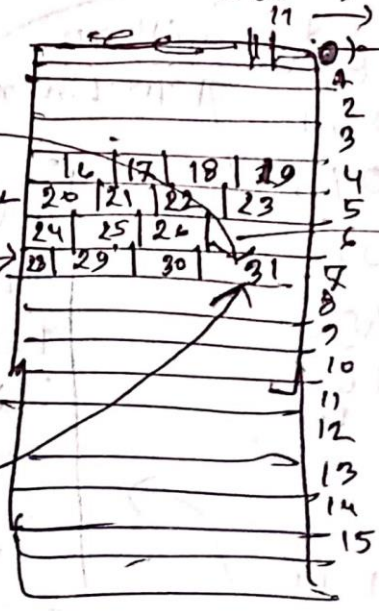
Proves

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 1 1 1

16 8 4 2 1

0 + 16 + 8 + 4 + 2 + 1 = 31



The last block

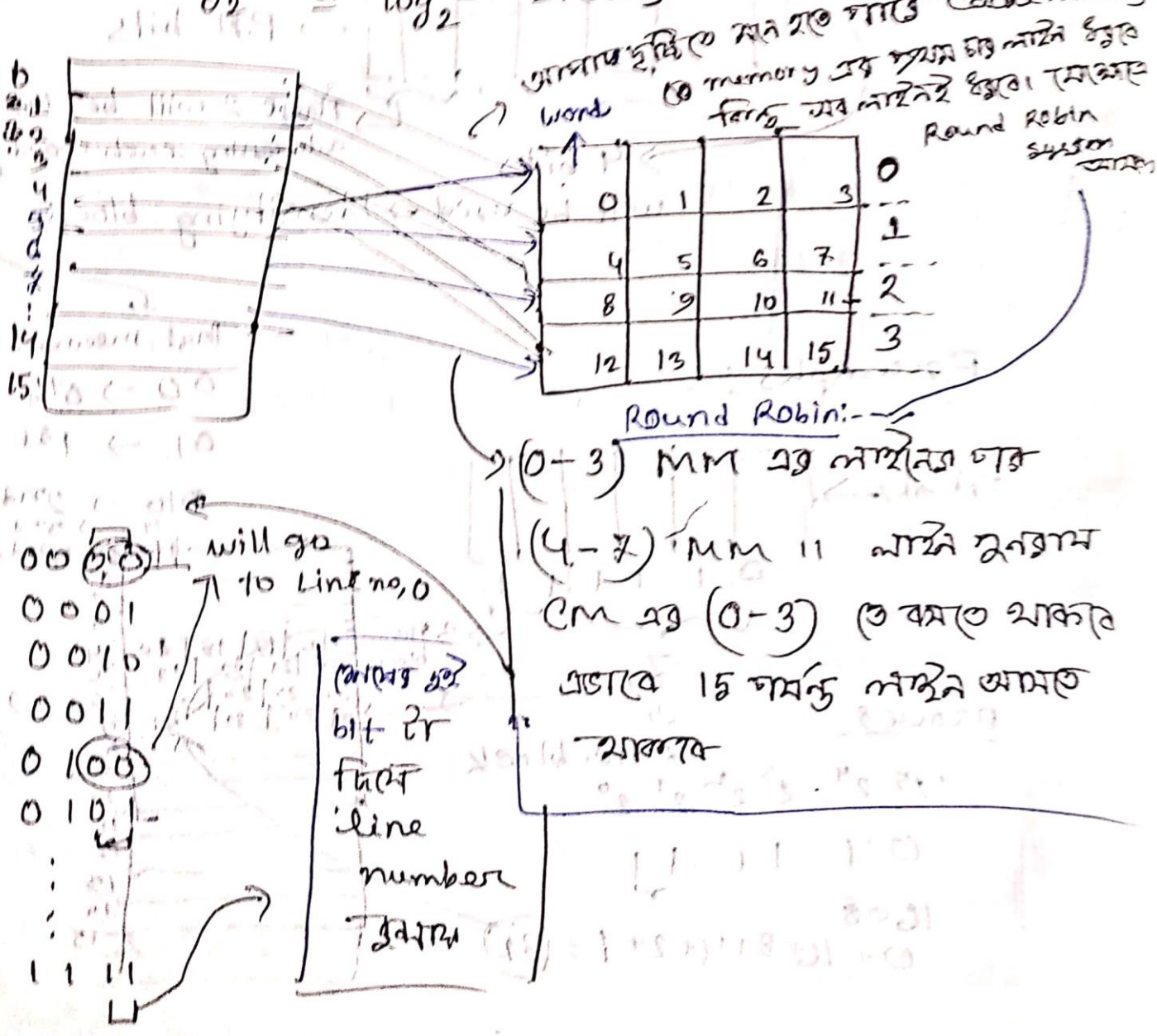
[1] We have a Cache of 16 words & Block size = 4 words.

We know, Block size = Line size = 4 words

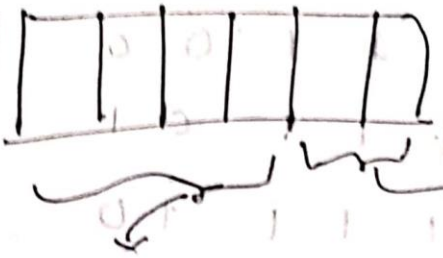
Number of Lines in Cache : 16/4 = 4

4 different lines,

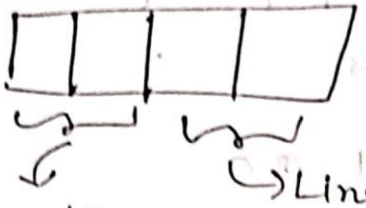
$\log_2 4 = \log_2 2^2 = 2$ bits



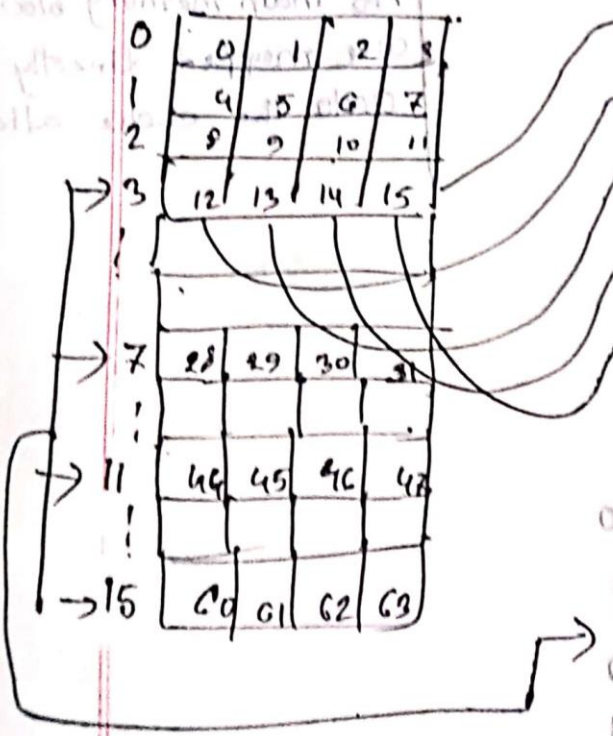
P.A bits split!-



→ Block / Line Offset (word no.)



Why it is Tag bits!- Tag bits



	2^5	2^4	2^3	2^2	2^1	2^0
→	32	16	8	4	2	1
12 →	0	0	1	1	0	0
13 →	0	0	1	1	0	1
14 →	0	0	1	1	1	0
15 →	0	0	1	1	1	1

Also we witnessed that

This block no. 3, 7, 11, 15 are also mapped onto the same cache line → which is Line no 3 [11]

Let's Observe them too.

Block NO. - 27

Tag bits

28 → 0 1 1 1 0 0

29 → 0 1 1 1 0 1

30 → 0 1 1 1 1 0

31 → 0 1 1 1 1 1

Block NO. - 3

Tag bits

12 → 0 0 1 1 0 0

13 → 0 0 1 1 0 1

14 → 0 0 1 1 1 0

15 → 0 0 1 1 1 1

Block NO - 11

Tag bits

44 → 1 0 1 1 0 0

45 → 1 0 1 1 0 1

46 → 1 0 1 1 1 0

47 → 1 0 1 1 1 1

Block NO. - 15

Tag bits

60 → 1 1 1 1 0 0

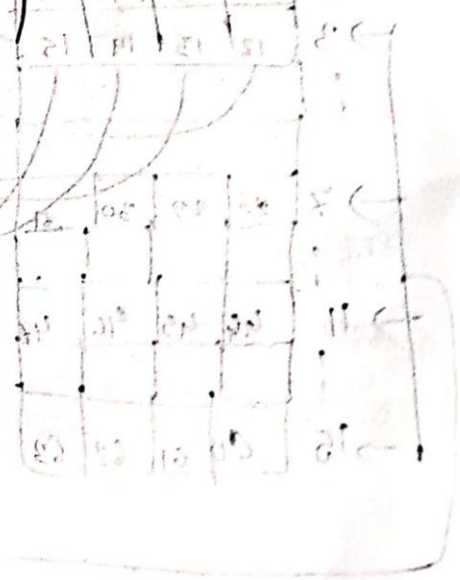
61 → 1 1 1 1 0 1

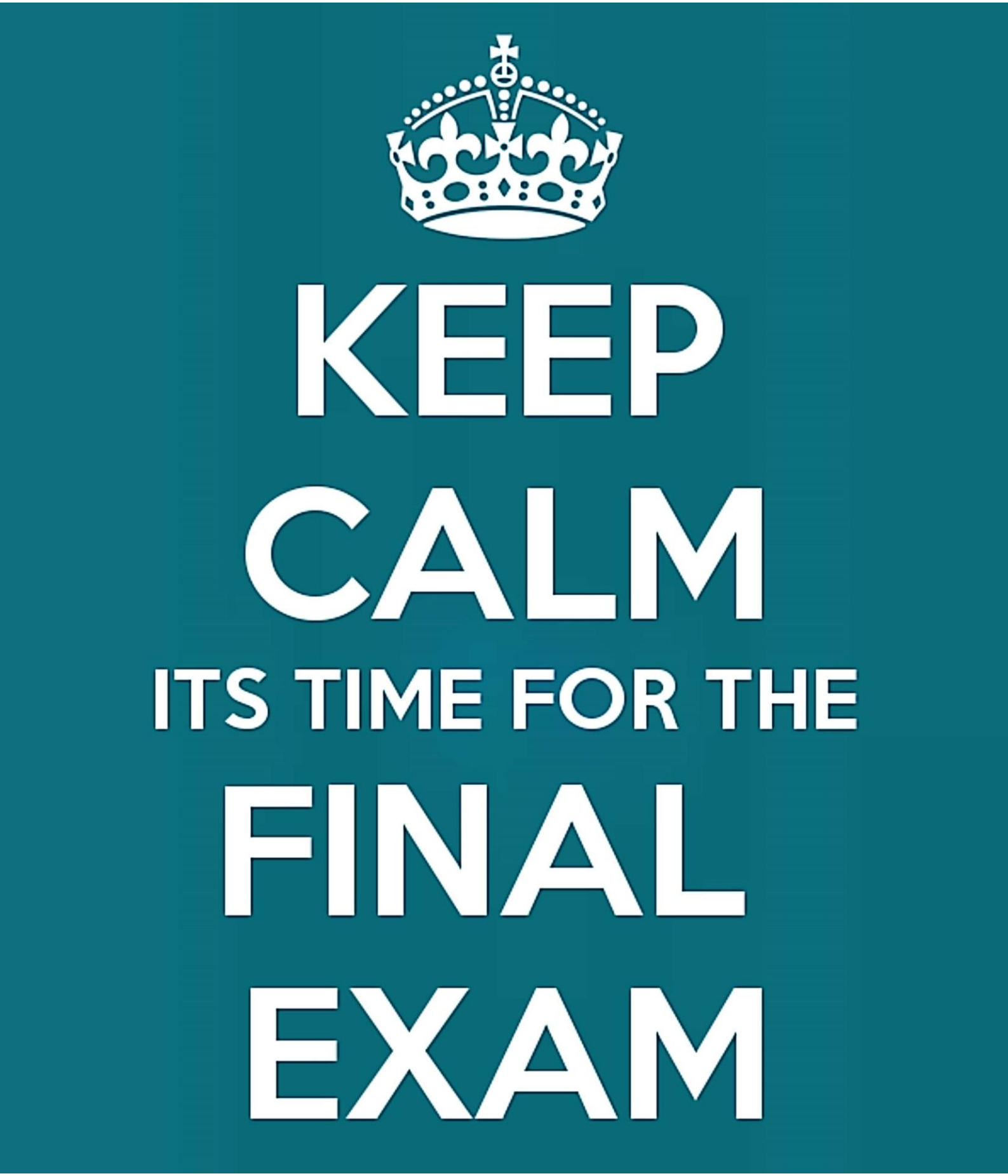
62 → 1 1 1 1 1 0

63 → 1 1 1 1 1 1

So, tag bits identify which one of the blocks is present in the caching. Means they are having a name...

This memory mapping is called direct mapping. As main memory blocks are mapped directly onto the cache alone

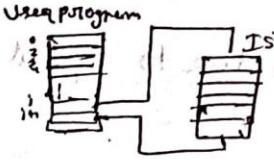



KEEP
CALM
ITS TIME FOR THE
FINAL
EXAM

Interrupt

→ High priority instruction generated by H/w or S/w to get immediate attention of CPU.

Eg: I/O device request for data transfer, power failure ~~error~~



ISR = Interrupt Service routine

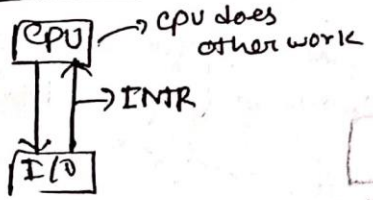
→ Interrupt Handing Mechanism:-

① ~~CPU~~ As interrupt is generated, processor execute the ISR (Interrupt Service Request) & provide the request service.

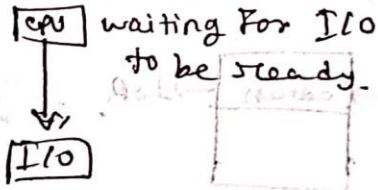
② State of processor (Content of PC, General Register, PSW (processor status word) is first saved in memory before servicing the interrupt.

③ When ISR is completed, the state of the processor is restored so that interrupted content may continue.

With INTR



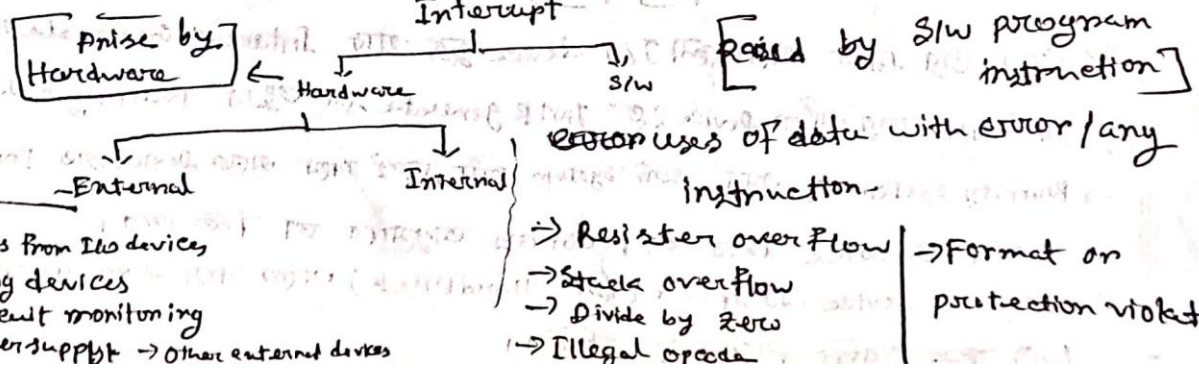
Without INTR



⇒ We use it for improving the performance of CPU.

⇒ overhead required to service the interrupt.

[Save, status, restore it] required time



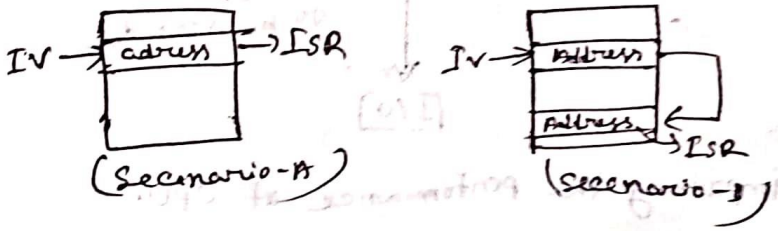
Hardware error → Data path error
→ Control error

Timeout → Endless loop of a program

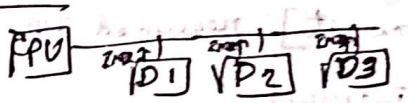
External interrupt is Asynchronous
Internal interrupt is Synchronous
(External & Internal interrupt happens due to signal occur of the CPU

Non-vectored :-
Address of ISR is stored in a fixed address/location in memory.
That means, Device/source does not provide the address of ISR.

Vectored :-
Source/device that interrupted provides the address of ISR to CPU.
And it's called as Interrupt vector.
(Adress that inform the interrupt Hardler where to find the interrupt)
It is common when CPU hey mulpile peripherals connected



Priority Interrupt :-

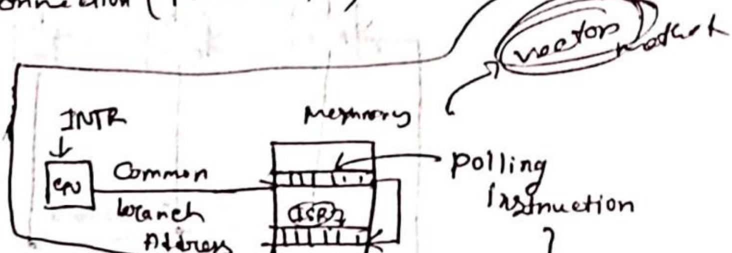


=> CPU এর সাথে অনেকগুলো I/O device যুক্ত থাকে Interrupt Generate করতে পারে use
→ একই সাথে একাধিক device থেকে INTR generate হলে তখন priority system INTR ব্যবস্থা করে।
→ Priority system হচ্ছে এমন একটি system যেটি একই সাথে একাধিক device থেকে INTR হলে কোনটির আগে Service দিবে তা priority অনুসারে তা ঠিক করে।
→ একাধিক দুই device এর speed বেশি (HardDisk) থাকে কম ~~কম~~ speed (keyboard) এর তুলনায় বেশি priority দিবে।

polling → পোলিং

PI is types
- → S/W method [polling]
- → H/W
  - Serial Connection
    - → Daisy chaining PI
    - → ~~parallel connection~~ (PPI)
  - → parallel connection (parallel PI)

Polling :-



(Program structure)

```
if(device[0].Flag)
   device[0].service();
else if(device[1].Flag)
   device[1].service();
```

→ It's a software method.

→ Processor INTR detect করলে it branches to a common ISR. যার কাজ হচ্ছে সকল I/O device এর মধ্যে poll করা যে কোন device এ INTR আছে।

→ এই method এ সবসময় common branch address থাকে সকল Interrupt এর জন্য।

→ ঐ branch address টি একটি program code যা সেটি poll করতে সহায়তা করে। এবং তাকে বলে প্রতিটি source

→ highest priority পাওয়া source সবার আগে checked হয় যে তার INTR signal ON কি না। যদি না হয় তবে পরবর্তী device এ যায়। আর এভাবেই চলতে থাকে

→ priority কোন device poll হচ্ছে তার উপর নির্ভর করে বের করা হয়।

Advantage:-
- → Flexible কারণ এটা software
- → Low cost কারণ নাই Hardware

Disadvantage
1. slow
2. অনেক interrupt এর জন্য সময় অনেক লাগবে

Priority INTR by H/W

→ এখানে priority INTR manager লাগে। যেটা interrupt request accept করে।

→ যেহেতু এটা H/W এর মাধ্যমে করা হয় তাই এটা Fast

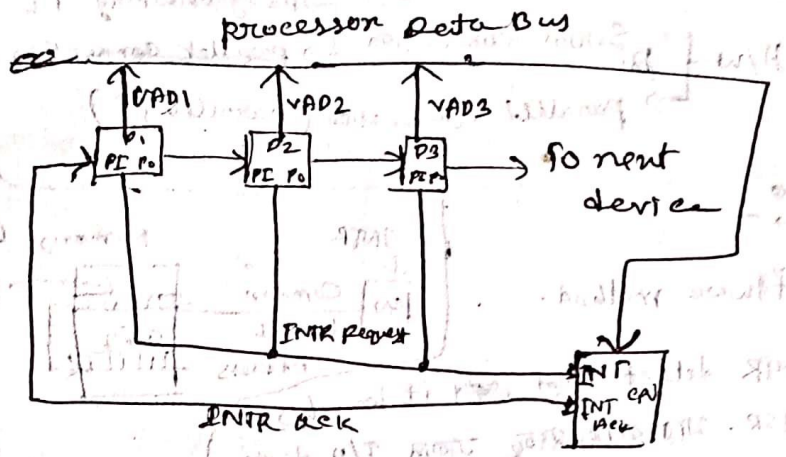
→ আর speed বাড়াতে প্রতিটি INTR source একটা interrupt vector প্রদান করে যেটা তার নিজের service Routine access করে।

→ No polling is required.

→ 2 way/methods:-
- → serial priority INTR (Daisy chaining priority method)
- → parallel priority INTR (uses priority INTR)

Daisy chaining :-

VAD = Vector Address



→ সবগুলো device serially connected. যারা INR Request দিতে পারে।

→ যার priority বেশি তাকে সবচেয়ে আগে রাখা হয়।

→ যে Attention চায়, সেটা INTR Request পাঠায় CPU কে INR R line ব্যবহার করে।

→ CPU ফিরতি INTR ack signal পাঠায় যা PI এ যায় সবচেয়ে device এর।

→ যদি ঐ device INTR Request পাঠিয়ে তবে যে priority output (PO = 0) হয়ে যায় এবং VAD কে processor Data bus এ পাঠায়। (PI = 1 & PO = 0)

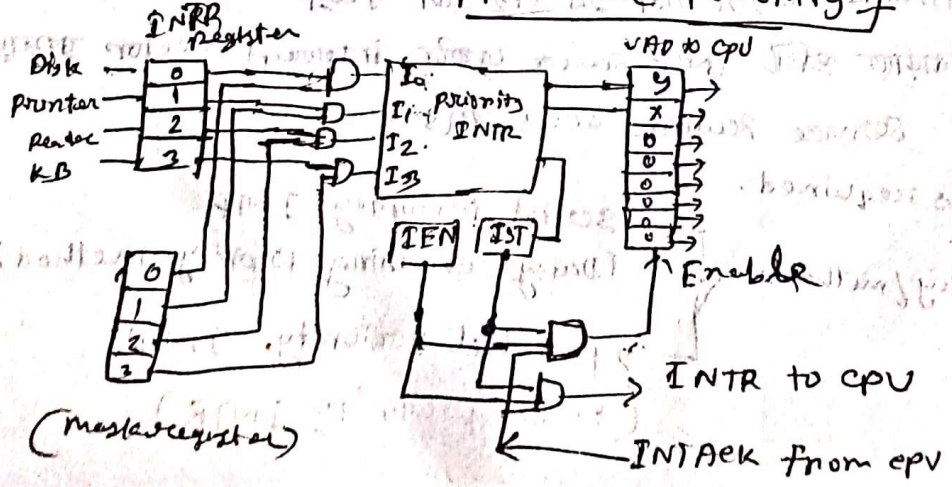
→ যদি ঐ device না পাঠায় তবে PO = 1 করে INTR Ack signal next device এ পাঠায়। (PI = 1 & PO = 1)।

→ কোনো device-ই যদি না পাঠায় INTR। তাহলে PI = 0 ও PO = 0 থাকে

→ যে device এর PI = 1 & PO = 0 সে INTR Request পাঠিয়েছে তার VAD data bus এ থাকে।

→ Device যত পাশে থাকে, তার priority ও তেই কমতে থাকে

Parallel priority :-



→ Consist: INTR Register এর bit set করে হয় separately by INTR devices.

→ Priority: bits এর position এর নির্ভর করে priority করে

→ Mask Register: use হয় যাতে Higher priority কে service দেওয়া যায়। যখন Lower priority service পায় তখন তাকে disabled করে।

→ Mask Register এর mask bit আর INTR Register এর INTR bit গুলো priority Encoder এর Input (I_0 I_1 I_2 I_3) তে চালনা করে।

→ দুইটি output হয় যা VAD to CPU তে দেয় যায়।

→ Another Output হয় IST (Interrupt Status FlipFlop এ যায়)। যখন INTR হয় তা masks হয়। (IST একটাই হয় যখন one or more Inputs are set 1)

→ The IEN (INTR Enable FlipFlop) Can be cleared by program to provide an overall controll over the the INTR system.

→ Output of IST & IEN is added along with INT Ack signal from CPU to an AND gate & provides a single signal to CPU.

→ The INT ACK enables bus buffers in O/P registers & VAD is placed into the data bus.

Priority Encoder

→ Implement priority Function. যেমন যখন দুটি INTR হলে Highest priority পাবে। (এটাই Logic)

Truth table

Inputs I_0	I_1	I_2	I_3	Output x	y	IST
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1
0	0	0	0	x	x	0

Boolean Function

$x = I_0' I_1'$

$y = I_0' I_1 + I_0' I_2'$

$IST = I_0 + I_1 + I_2 + I_3$

Maskable – Non-maskable

Maskable	Non-maskable
1] Hardware interrupt which can be disabled by the instruction of CPU	1] Can't be disabled by the instruc of CPU.
2] Can be handled after executing the current instruction.	2] The current instructions & status are stored in stack for the cpu to handle the interrupt
3] It helps to handle lower priority task	3] handle higher priority task.
4] It used to interface with peripheral device.	4] It is used during emergency purpose. like power failure, smoke detector etc.
5] It is slow	5] It is fast
6] May be vectored or non-vectored	6] vectored
7] Operation can be masked or made pending.	7] Can't be
8] Ex:- RST 6.5, 7.5, 5.5 of 8085	8] Ex:- 8085 microprocessor.

Pipeline

→ way of speeding up the execution of instruction

→ Decomposing a sequential process in sub-process

Linear pipeline:-

Cascade of processing stages which are linearly connected.

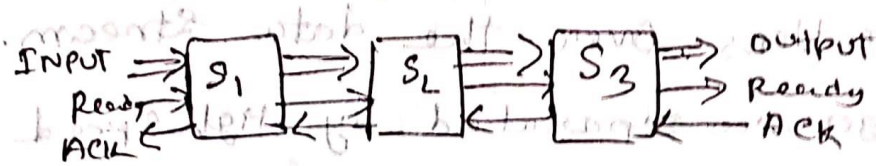
Depending on the control of data, 2 models:-

→ Asynchronous PM & (Handshaking protocol)

→ synchronous PM. (clocked latches)

Asynchronous (Handshaking):-

Data flow b/w adjacent stages controled by this



⇒ S_i sends ~~req~~ Ready ~~req~~ signal when it is ready to send data. S_{i+1} receives the ~~signal~~ data & send ACK signal back to S_i.

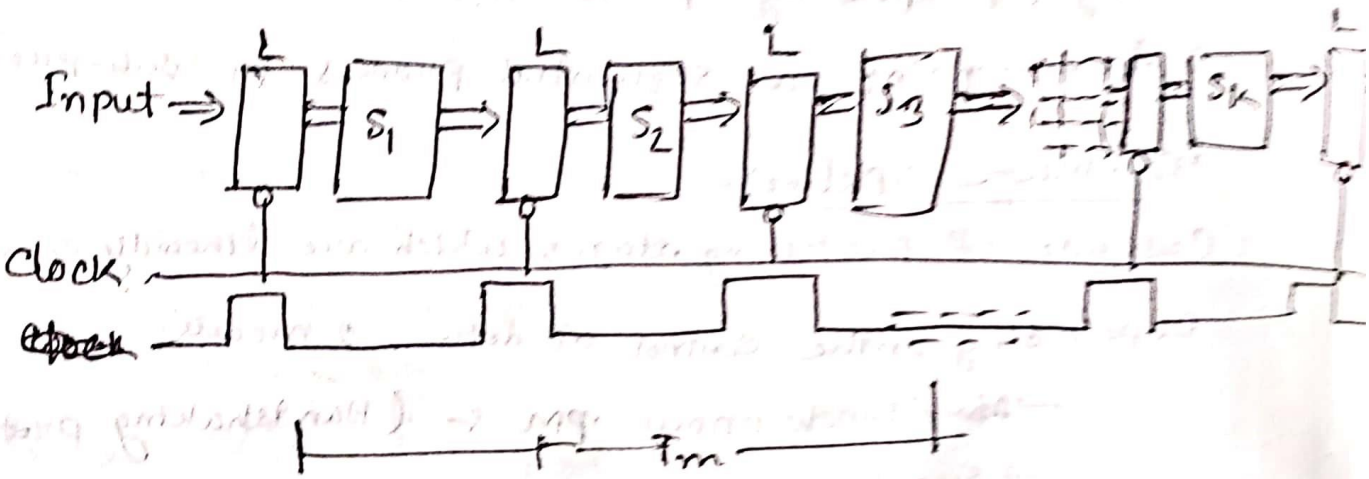
→ It is useful for designing communication channels in message passing multicomputer

→ It may have variences in performance rate because different amounts of delay may have different times.

T = Clock period
Tm = Maximum Stage delay
d = Latch delay
L = Latch
S = Stage

Synchronous Model:-

Clock & Latches

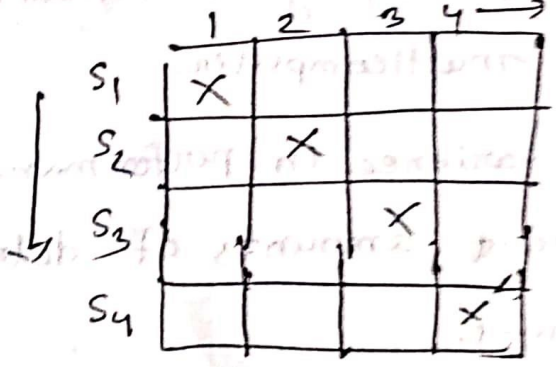


⇒ This is consists of cascade of processing stages (S_i). And these all perform arithmetic or logic operations over the data stream.

⇒ These stages are separated by High Speed latches.

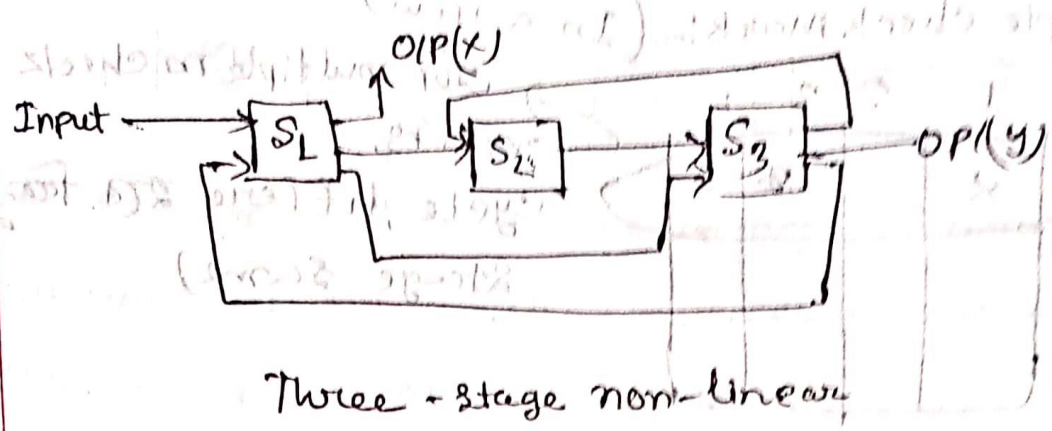
⇒ The arrival of clock pulse makes latches send the data to the next stage simultaneously.

⇒ And the pattern of successive stages specified by reservation table.



Diagonal বরে থাকা Lines
0 থেকে কোন ঘর পর্যন্ত আছে
[illegible] অনুরূপ হয়।

Non-linear pipeline processor:-



Three-stage non-linear

Three stage:- It (performs diff. functions in diff. times. It is multi-functional)

$S_1 \rightarrow S_2$
$S_2 \rightarrow S_3$ } linear connection

$S_1 \rightarrow S_3$ → Feed Forward connection

$S_3 \rightarrow S_2$
$S_3 \rightarrow S_1$ } Feed backward connection

The output don't come from the last stage only.

It has two output x & y.

Reservation table

(x) output

Sequence:- $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1$ ($S_1 \leftarrow S_3$)

	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

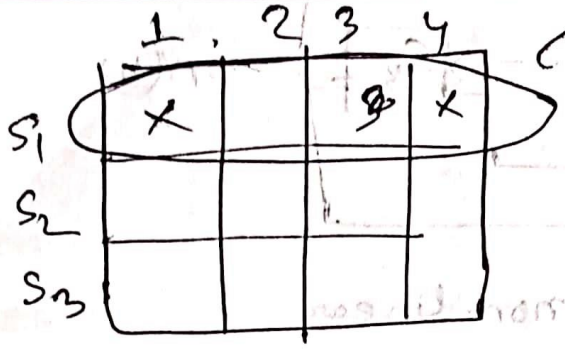
For y output:-

~~S_3~~ → ~~S_1~~ → $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1$ ($S_1 \leftarrow S_3$)

	1	2	3	4	5	6	7	8
S_1	y					y		y
S_2		y		y				
S_3			y		y		y	

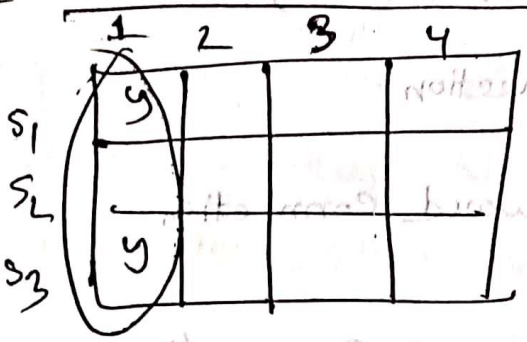
(*) Here,

Multiple check mark:- (In a row)



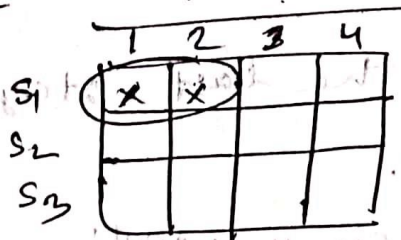
এর multiple এ check mark in a row, cycle different কিন্তু stage same)

(*) Multiple check in a column:-

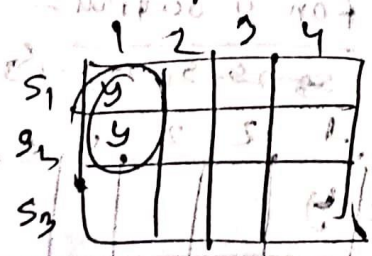


এর Multiple check mark in a column. cycle এবং stage different)

(*) Multiple check mark contiguous:-

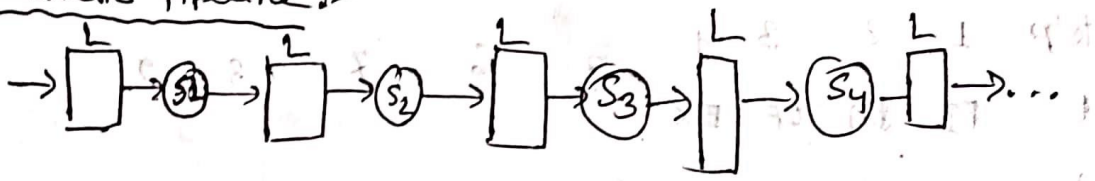


→ Multiple check mark contiguous in row ~~column~~ অর্থাৎ cycle different কিন্তু এর stage এ একই।



→ Column এ একই cycle একই stage কিন্তু এর পরপর।

Arithmetic pipeline :-



=> An arithmetic pipeline breaks any arithmetic operation into multiple arithmetic steps that can be executed one by one in segments in ALU.

=> In Arithmetic pipeline the ALU of a computer is segmented for pipeline operations in various data formats.

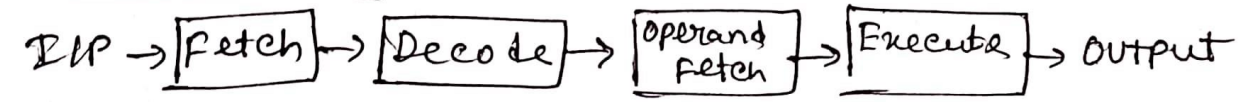
For example :-

4-stage pipeline in Star-100,

8 stage pipeline in TI-ASC,

16 stage pipeline in Cray-I.

Instruction pipeline :-



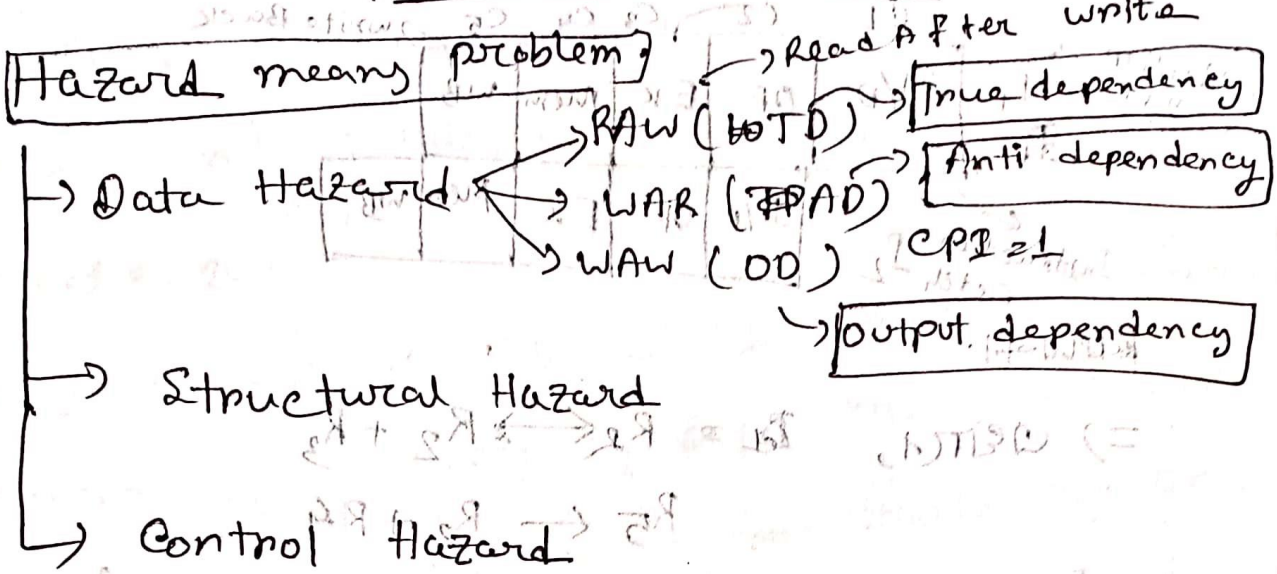
=> In instruction pipeline, the execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction.

The processor fetches the instruction from memory, decode it & determine the kind of operation it need to perform. If the operation requires operand then the processor fetches the operands from the memory. Then execute it & store it to the specified memory location

Stage	1	2	3	4	5	6	7	8	9
1	FI	DA	OF	EX					
2		FI	DA	OF	EX				
3			FI	DA	OF	EX			
4				FI	DA	OF	EX		
5					FI	DA	OF	EX	
6						FI	DA	OF	EX

Ex:- Almost All Computer use this Instruction pipeline operation

Hazard in pipelining



In pipelining clock per Instruction = 1 হবে (CPI)

হবে। অর্থাৎ প্রতি 1 clock এ 1 instruction বের হবে

হবে। আর Hazard এর জন্য delay হয়। আর delay

এর জন্য CPI = 1 হয় না।

Suppose, we have a instruction,

$R_2 \leftarrow R_2 + R_3 \longrightarrow I_1$

$R_5 \leftarrow R_2 + R_4 \longrightarrow I_2$

So, here, I_1 এবং I_2 execute হয়। I_1 এর

IF/D → fetch & decode

OF → value নেওয়া
↳ operand fetch

we have 5 stages pipeline

(Addition হয়) — Instruction fetch — write Back

	C1	C2	C3	C4	C5	
I_1	IF/D	OF	Ex	Mem	WB	
I_2		IF/D	OF	Ex	Mem	WB

~~RAW-1.~~

=> এখানে, I_1 = $R_2 \leftarrow R_2 + R_3$

$R_5 \leftarrow R_2 + R_4$

এই জায়গায় Instruction (I_1) শুরু হয় clock-1 এর সময় start হয়। Instruction Fetch করে। Clock-2 এর শেষে operand fetch হবে। তারপর value গুলো নিবো আর Execution এ Add করবো। then যদি memory তে কোনো কাজ থাকে তা হবে। Then Added value টা R_2 তে লিখে দিয়েছি।

কিন্তু I_1 এ যখন C2 চলে তখন I_2 তার Instruction Fetch start করে। সেখানে I_2 সম্পূর্ণ এর R_2 এর value updated value না হয়ে old value নিয়েই কাজ সম্পন্ন করতে থাকে। তাহলে এখানে একটা problem চলে আসে। আর এটাই Hazard।

আর এখানে write এর আগে Read করে ফেলায় এখানে Read After write Hazard দেখা দিল।

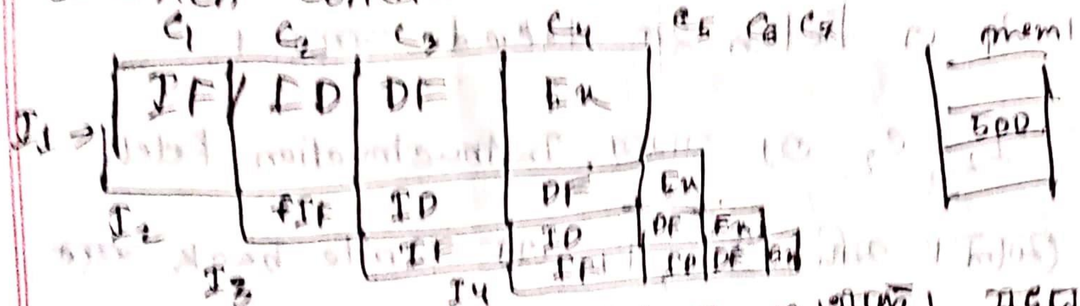
Structure Hazard:

Multiple Instruction same resource (memory, CPU, Cache) access করলে তখন Structure Hazard create হয়।

উপরের Example এ C_5 -এ I_1 Write Back করছে memory তে আবার I_2 ও memory Access করছে। তখন Hazard create হয়। যার নাম দিছি। এই ধরনের Hazard কে বলে Structural Hazard.

Control Hazard:

Branch Condition এর জন্য হয়।



=> Suppose I_1 memory-500 এ আছে। যখন Instruction Fetch করলাম তখনই PC র ও Increment হয়ে 501 হলো, আবার ও 501 এ থাকা Instruction এ I_2 কে C_2 তে time এ Fetch করা হয়েছিল। আবার, C_3 time এ I_3 (502 এ থাকা) এর Instruction Fetch হলো। আবার (503 এ থাকা) C_4 time এ I_4 instruction Fetch হলো।

কিন্তু I_1 instruction এর execution এর পর Address

যা execute করতে 2ns.

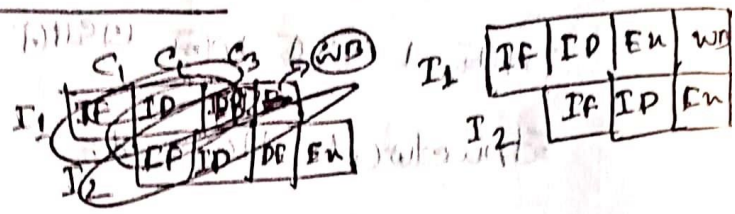
দিলো 700ns। কিন্তু I_1, I_2, I_4 একই সময়ে চলমান। তাই এই সব এখানে Hazard তৈরি হয়। এ থেকে বাঁচতে বাকি সবগুলো Instruction flush করতে হবে। যা একে Control Hazard বা branch control Hazard।

RAW

Write After Read Hazard (WAR):-

I_1 : $R_1 \leftarrow R_2 * R_3$

I_2 : $R_2 \leftarrow R_4 + R_5$



⇒ এখানে I_1 এ R_2 ও R_3 Read করার পর multiply করছে। C_1 এ fetch হয় decode করে।

কিন্তু I_2, C_2 এর সময় Instruction Fetch করে ফেলে। যদি I_2, I_1 এর write back করে R_2 এর value update করে ফেলে তাহলে I_1 ভুল value নিয়ে তার কাজ করবে।

অর্থাৎ,

I_1 $R_1 \leftarrow R_2 * R_3$

200 $\leftarrow$ 10 * 20

1400 $\leftarrow$ 70 * 20

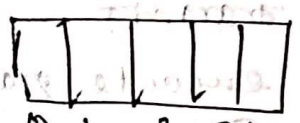
I_2 $R_2 \leftarrow R_4 + R_5$

70 $\leftarrow$ 630 + 40

এটি ভুল value নিয়ে কাজ করলো।

এটার 4 or 5 stage এ হয় না। শুধু, RAW হয় না।
এটা তখনই হয় যখন write আগে হবে।

Example:-
Autocorrection



⇒ এক্ষেত্রে আমরা Register -এ আগে I কে increment করি write then Read করি।

WAR বের করার নিয়ম:-

Domain (I_1) → R_2 & R_3

Domain (I_2) → R_4 R_5

Range (I_1) ⟶ R_1

Rag (I_2) ⟶ R_2

So, Domain (I_1) ∩ Range (I_2) ≠ φ

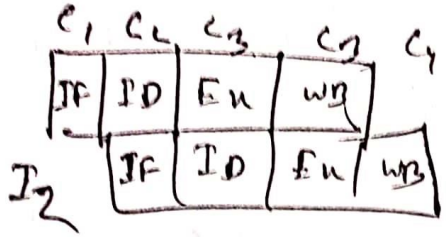
$(R_2, R_3) \cap (R_2) = R_2 \neq \phi$

So, WAR Hazard হবে।

WAW Hazard:

$I_1: R_3 \leftarrow R_1 * R_2$
$I_2: R_3 \leftarrow R_4 + R_5$

} যদি R_1 ও R_2 multiply এর পর R_3 তে save করার পর, R_4 ও R_5 Addition করে R_3 তে save করার পর, কারণ I_1 execute হবে I_2 এর আগে। তবে যদি কোনো কারণে delay হয় (multiply তে time যদি বেশি নেয়, parallel system etc এর জন্য) তখন I_1 এর R_3 এর value এর জায়গায় I_2 এর R_3 value হয়ে যাবে। এটা WAW Hazard।



~~এর সম্ভাবিক~~

WAW হয় মূলত special কোনো moment এ হয়। (যখন, যখন same register কে different Instruction Access নেয়)

কিভাবে বের করা যায়?

⇒ If,

Range $(I_1) \cap$ Range $(I_2) \neq \phi$

$R_3 \cap R_3 = R_3 \neq \phi$

Domain $(I_1) \rightarrow R_1$ & R_2
Domain $(I_2) \rightarrow R_4$ & R_5
Range $(I_1) \rightarrow R_3$
Range $(I_2) \rightarrow R_3$

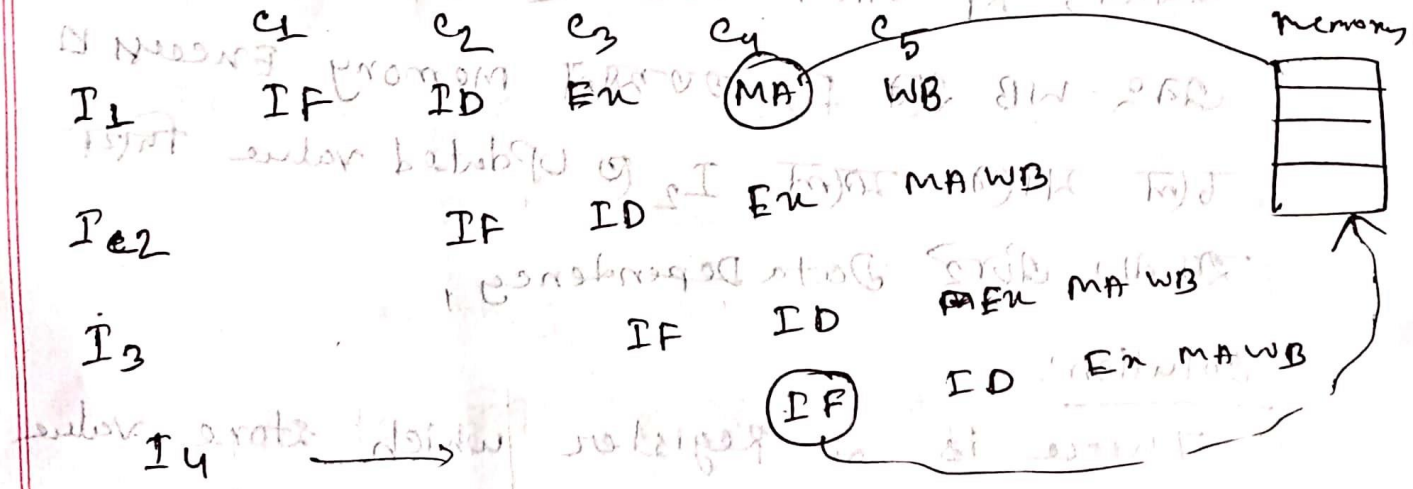
Pipelining Dependencies

The situations in which pipeline faces extra cycle without any operation.

Dependency:-

(1) Structural Dependency:-

Dependency যা structural conflict এর কারণে হয়।



এখানে, Instruction I_1 Memory Access নিচ্ছে C4 টাইমে

তিন একারণে টাইমে I_4 " " " Instruction Fetch এর জন্য।

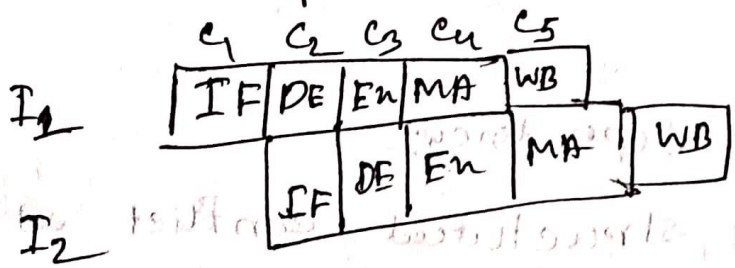
এই কারণে এখানে problem create হয় এবং I_4 C4 টাইমে Fetch করতে পারে না., তখন সে C5 টাইমে Fetch করে।

Solution:- program memory ও Data memory আলাদা করা

Data Dependency :-

$R_1 \leftarrow R_2 * + R_3$

$R_4 \leftarrow R_1 * R_3$



এখানে, R_1 আগে execute হবে ~~R_4 I_2 এগুবে~~ এবং WB হবে I_2 এগুবে memory Excess না চলে যাবে কারণ I_2 এ updated value নিবে হয় না। এটাই Data Dependency,

Solution :-

There is a Register which store value temporarily. So that register will provide updated value.

Output Dependencies:-

কোনো Instruction আগে Instruction এর আগেই instruction update করে দেয়।

এটাকে WAW Hazard ও বলে।

I_1 ⓐ ⓑ $R_0 + R_1 + R_2$

I_2 ⓐ ⓑ $R_3 * R_0 * R_2$

I_3 ⓐ ⓑ $R_0 - R_3 + R_1$

এখানে, connected value পাওয়ার জন্য I_3 ও I_2 ও I_1 এর উপর নির্ভরশীল আবার I_2 I_1 এর উপর। সেকারণে I_1 এর WB হওয়ার পর I_2 start হবে। I_2 এর উপর I_3 হবে। এটাই OD.

Anti Dependency :- (WAR)

I_1 : $R_1 \leftarrow R_2 * R_3$

I_2 : $R_2 \leftarrow R_4 + R_5$

=) এখানে, I_1 এর আগে I_2 execute হলে updated value পাবে না। তাই আগে I_1 কে আগে execute ও WB stage শেষ করতে হবে। আর এটার জন্য I_2 ে stall cycle use করতে হবে। আর এখানে

যে dependency তৈরি হয় সেটাই Anti Dependency.

Control Dependency:-

Flow Control এর কারণে যে dependency।

	C_1	C_2	C_3	C_4	C_5
I_1	IF	DE	EX	MA	WB
I_2		IF	DE	EX	MA WB
I_3			IF	DE	EX MA WB
I_4				IF	
I_{10}					

DE → Jump

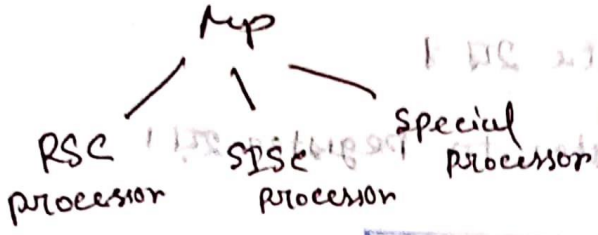
এখানে, C_4 এ I_3 Decode করার সময় বুঝানো হচ্ছে যে Jump করতে হবে I_{10} এ। কিন্তু I_4 ততক্ষণে Instruction fetch করা হয়েছে।

ফলে I_4 এ delay করানোর জন্য stall cycle চালাতে হবে।

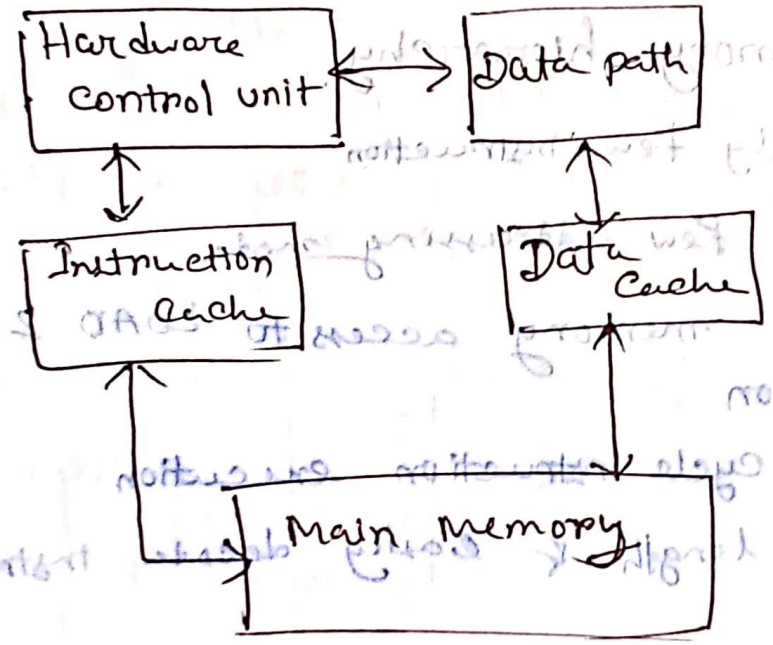
এবং এখানে যে dependency দেখা যাচ্ছে সেটা Control dependency.

RISC

(Reduced Instruction set Computer)



RISC:-



⇒ It is designed to reduce the execution time by simplifying the instruction set computer.

Each instruction requires → only one clock cycle.

One clock cycle ⇒ Fetch, decode, executing

মূলত Hardware control করে। যেখানে Instruction দ্রুত করা। Instruction মূলত uniformity maintain করে।

যেহেতু programme কে segment এ Divide করতে হয়। একে

Load store architecture বলে (যেটা কাজে execute হয় তাই),
Instruction simplified করে। pipelining বেশি হয়।
Fetch, Decode, Execute হয়।
→ Operation মূলত register to register হয়।

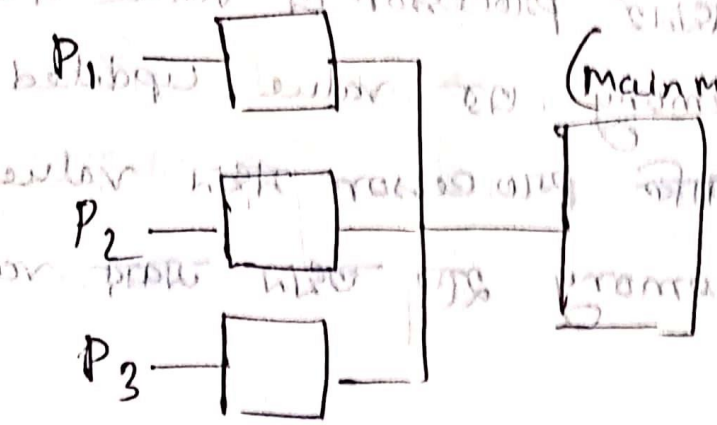
~~Cache Coherence~~
~~A situation where multiple processor cores shares the same memory hierarchy.~~

→ Relatively few instruction
→ " Few addressing mode
→ Limited memory access to LOAD & SAVE Instruction
→ Single cycle instruction execution
→ Fixed length & easily decode instruction format.

Cache coherence

A situation where multiple cores share the same memory hierarchy. Consistent data নিয়ে যে Cache তে যে problem হয় তাই Cache Coherence problem।

যেমন multi processor এর ক্ষেত্রে যে সমস্যা। process যখন এক processor থেকে অন্য processor এ যায় তখন এই problem হয়। অর্থাৎ, P_1 processor এ কাজ করার সময় কোন কিছু পরে P_2 তে shift হয়। তখন এই সমস্যা দেখা দেয়। তাছাড়া I/O module এও করতে হতে পারে।

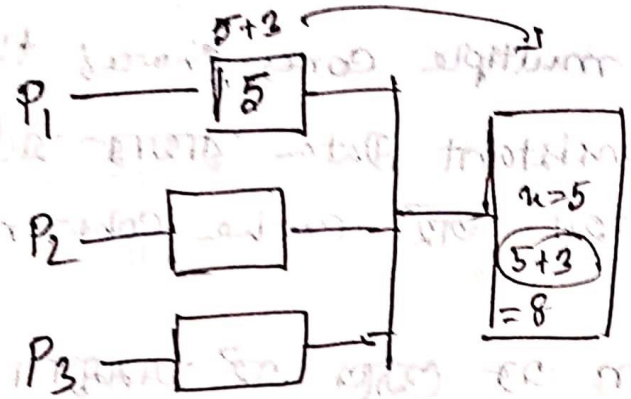


Cache coherence হওয়ার arise হয়

→ Write Through

→ Write back.

Write Through :-



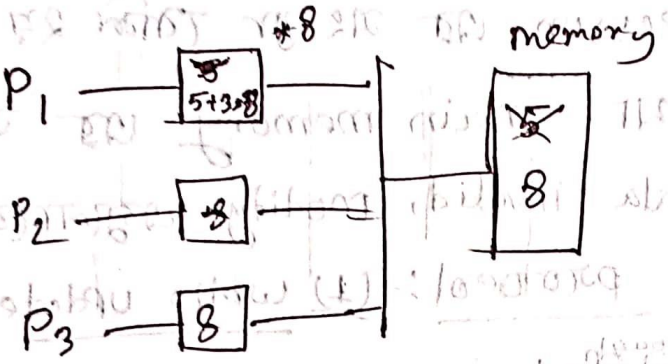
এখানে

processor P_1 → Instruction execute করার পর এখানে value updated হয়ে 5 হতে 8 হয়। কিন্তু একই সময়ে বাকি processor এর value ভিন্ন থাকে।

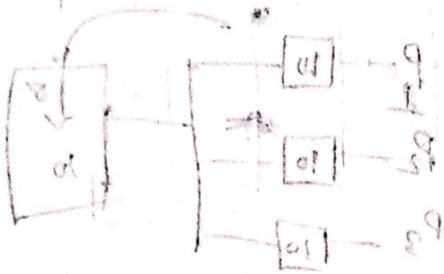
Write Through তে যখনই processor এ value updated হয়, তখনই memory এর value updated হয়ে যায়। যার ফলে বাকি processor যখন value access করে memory হতে তখন সবার value same হয়ে যায়।

Write Back:-

এই policy তে



=> এই policy তে, P_1 change হওয়ার সাথে dirty bit set থাকবে। অর্থাৎ, যে processor এ process হতে থাকবে সেই processor এ set থাকবে। যেটা সাথে Main memory তে update হবে। এবং যেখান হতে বাকি processor এর value change হবে।

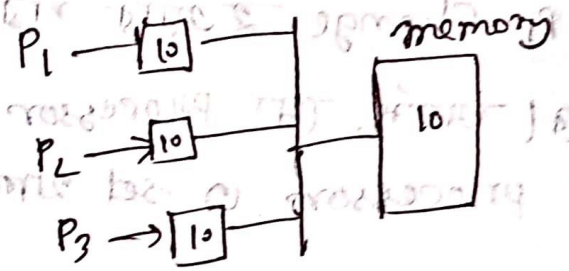


Cache coherence হতে বাঁচতে ~~So Snoopy based~~ So Snoopy Based protocol use করা হয়।

=) যখন processor এর সংখ্যা বেশি হয় এবং অনেক cache হয়, যা main memory এর সঙ্গে যুক্ত থাকে তখন Data invalid, modify এগুলোকে সমস্যা হয়।

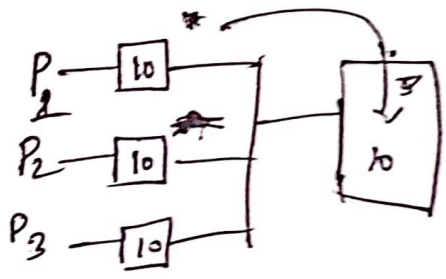
Snoopy Based protocol :- (1) Write Update :-

(i) Write Through ~~Update~~ :-



=) এক্ষেত্রে write UPDATE হলে বাকি processor গুলোও সাথে সাথে UPDATE হয়।

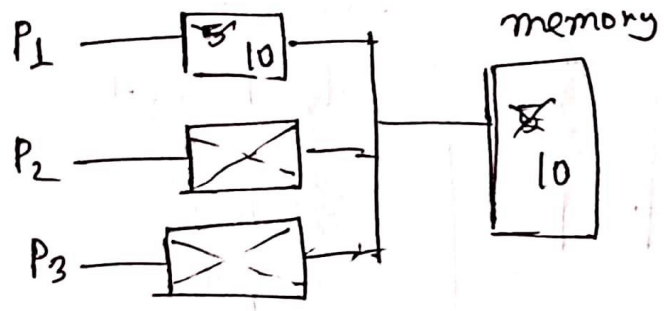
(ii) Write Back :-



=) এক্ষেত্রে যে processor process করছে সেখানে Dirty bit add হয়। পরে তা main memory তে Update হয়।
সেখান হতে বাকি processor গুলো।

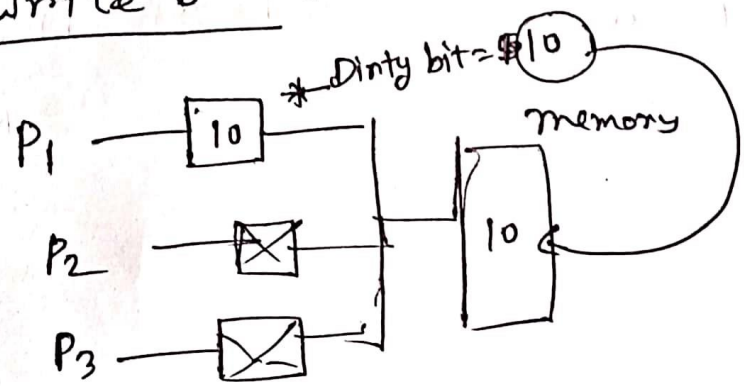
(A) Write Invalidate protocol:-

(i) Write Through:-



=) এখানে, processor P_1 instruction execute করে $P_1 = 10$ হয়েছে। এরপর, P_1 UPDATE হওয়ার পরপরই বাকি processor ও invalid করে দেয়।
এবং Main Memory ও P_1 এর value UPDATE করে দেওয়া হয়।

(ii) Write Back:-

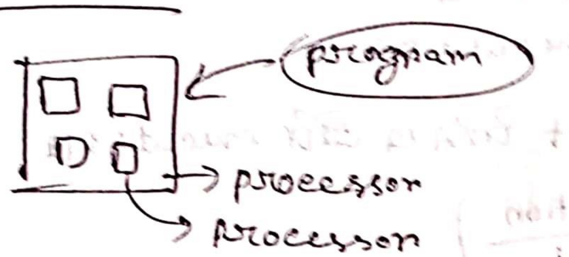


=) এখানে, P_1 এর UPDATE হলে বাকিগুলো (processor) invalid হয়ে যায়। এবং Processor এ Dirty bit add হয়।
এরপর memory এই Dirty bit এর দ্বারা UPDATE হয়।

ILP (Instruction Level parallelism)

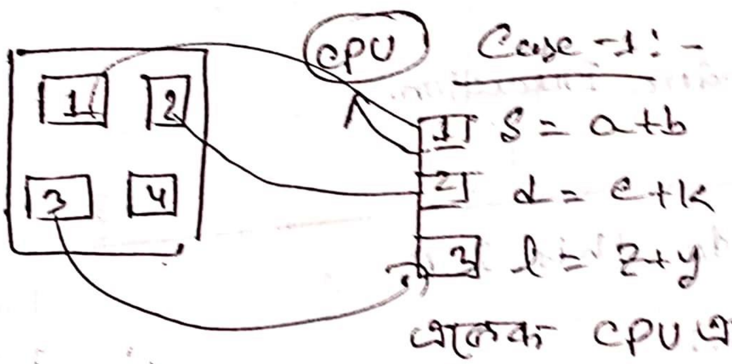
It is a measure of how many of the operations in a computer program can be performed simultaneously.

parallelism :-



যখন কোনো program দিলে processor এটি 4 ভাগ করে। এবং একসাথে execute করে একটা output দিবে। (Very Fast)

Here, ILP is a measure (1 unit time এ কতটুকু operation Computer execute করতে পারে)



প্রত্যেক CPU একেকটা execute করবে একই time এ।

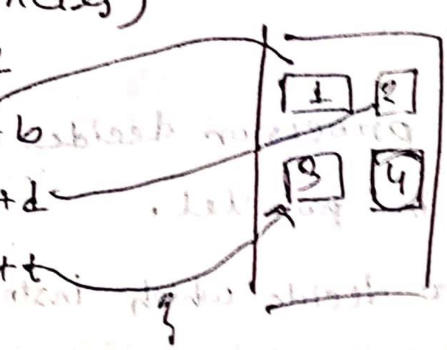
=> কিন্তু- কিছু কিছু ক্ষেত্রে একটা problem faced হয় (Shared dependencies)

Case : 2

1] S = a+b

2] t = c+d

3] U = S+t



CPU -1 : S=a+b execute করে-
CPU - 2 t = c+d " "
CPU - 3 করবে U কিন্তু এর value পাই নাই। কারণ বাকি দুইটা এখনও চলমান। তাহলে এখানে dependencies হচ্ছে।
আর এই problem টাই Shared Dependencies

② Simplify the compiler.

Superscalar:- CPU that implement a form of parallelism called ILP with in a single processor It can execute more than one instruction clearing by simultaneously disp.

ILP (Instruction Level parallelism)

Ex:- 4 operations can be carried out in single clock cycle. So there will be four Functi

3 ways:-

1] Sequential Architecture:-

program is not expected to convey any information regarding parallelism to hardware. Ex: Superscalar Archi.

2] Dependence Architecture:-

Mentions info regarding dependencies between operations like data flow architecture.

3] Independence Arch:-

prog gives info regarding which operations are independent of each other so that they can be executed instead of the nops.


LECTURE

31.10.22

Computer Architecture

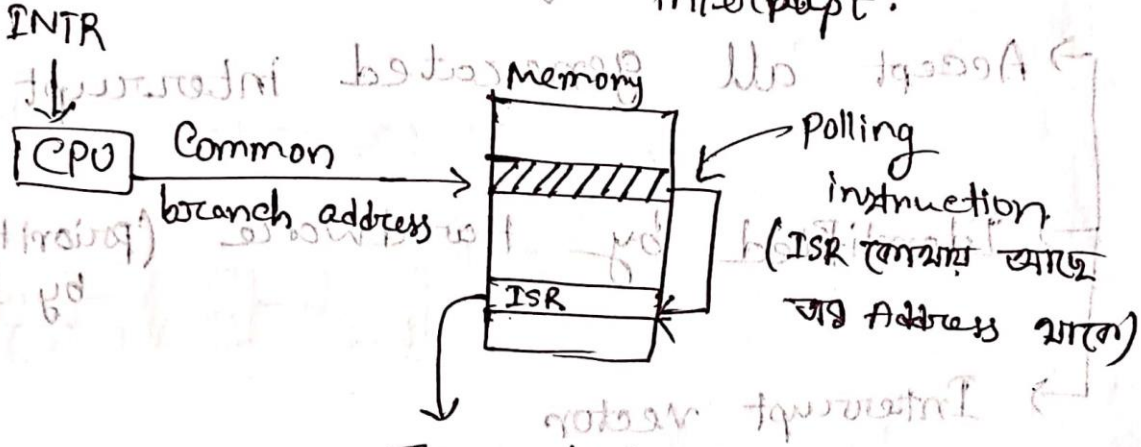
Interrupt:-

Establishing priority of Simultaneous Interrupt:-

There are two methods of it

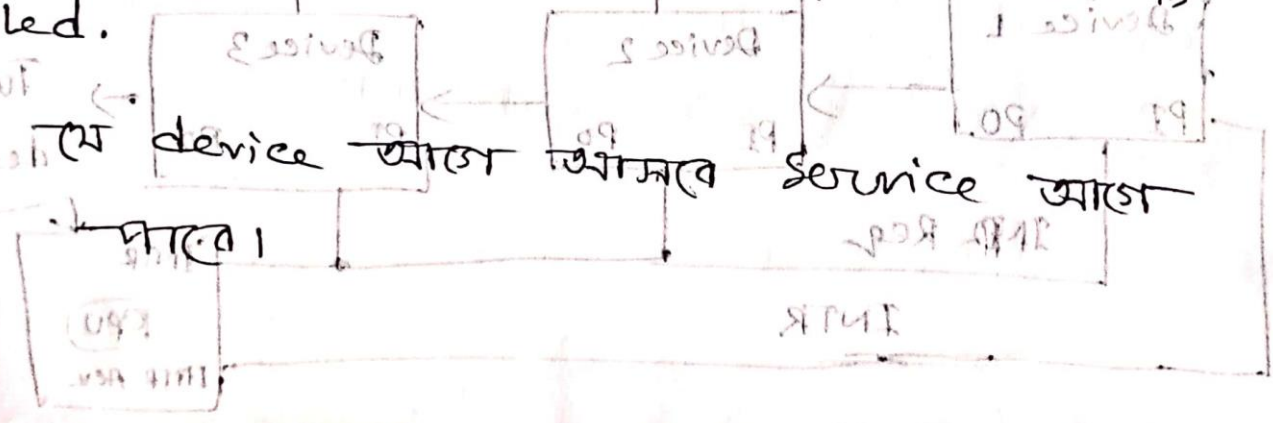
- Software Method → polling
- Hardware Method
  - Daisy chaining (Serial)
  - parallel priority interrupt.

Polling:-



A software Method by which interrupt Method is controlled.

যে device আগে যাবে সে service আগে পাবে।



Code :-

```
if (device [0]. Flag)
    device [0] . service ( );
else if ( device. [1]. Flag)
    device [1] . service 1);
```

Use :-

Benefit :- Low cost

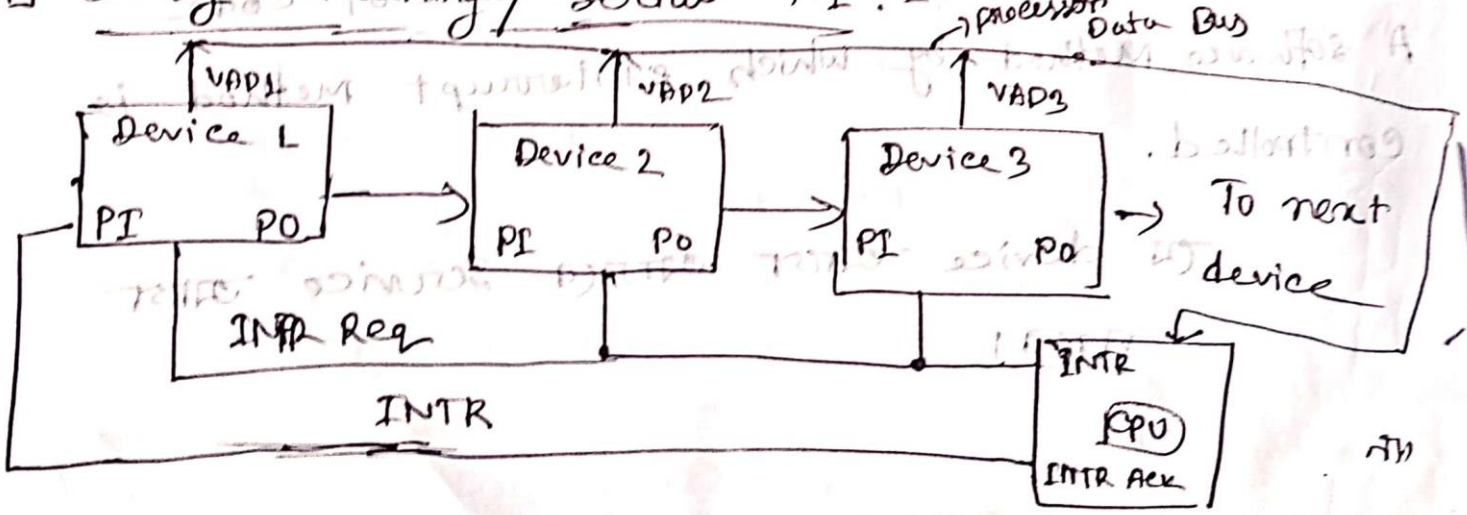
problem :- Slow, one-by-one service.

Priority Interrupt Manager :-

It works (3 types)

- Accept all generated interrupt
- Identified by Hardware (priority interrupt) by hardware
- Interrupt vector

Daisy chaining / Serial P.I :-



PI = 1 $P_0 = 1$

PI = 0 $P_0 = 0$

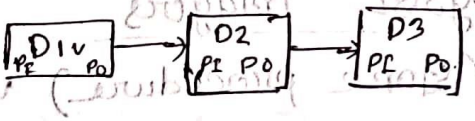
PI = 1 $P_0 = 0$

এখানে,

PI = 1 , $P_0 = 0$ করলে , suppose device -1 এর ক্ষেত্রে $P_0 = 0$ হলে device -2, device -3 তে signal বা data কিছুই যাবে না। আর এই জন্যই parallel P.I এর প্রয়োজন পড়ে।

PI = 1 , $P_0 = 1$ মানে হলো এখানে Interrupt Generate হয় নি।

parallel priority Interrupt :-



14th November, 2022

Monday

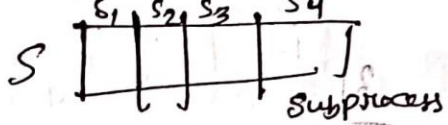
Computer Architecture

without pipeline → একসাথে যাবে

Pipelining :-

↳ way of speeding up the execution of instruction

↳ Decomposing a sequential process in sub process.



Example :-

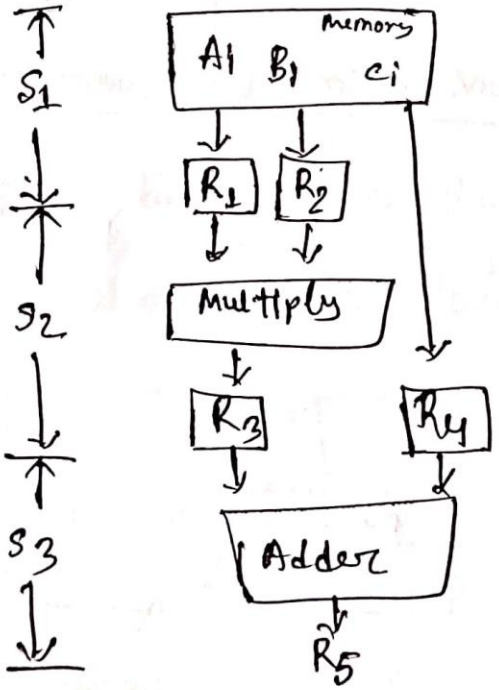
$A_i * B_i + C_i$ $i = 1, 2, 3, \ldots$

Load A_i & B_i

$R_1 \leftarrow A_i$ $R_2 \leftarrow B_i$

Multiply & Load c_i $R_3 \leftarrow R_1 * R_2$, $R_4 \leftarrow c_i$

S_3 Add ---- $R_5 \leftarrow R_3 + R_4$



S = Segment

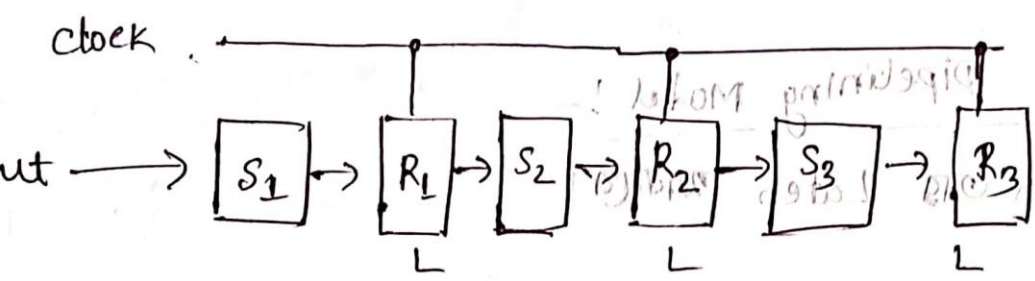
এখানে ভেঙ্গে ভেঙ্গে কাজ হয়। without pipeline এ একেবারে দুবার segment সমানভাবে হলে পরেরটা segment এর কাজ শুরু হয়। এতে সময় বেশি লাগে

With pipelining

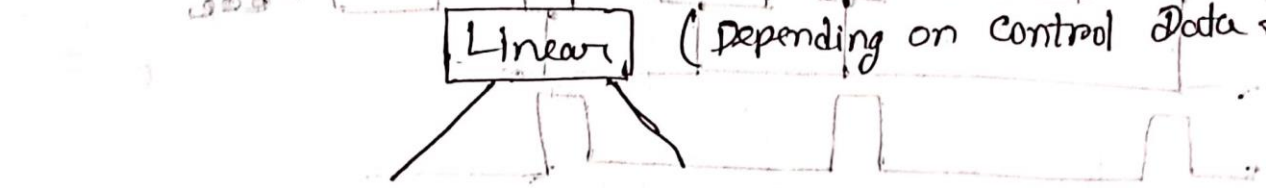
Clk pulse Number	Segment 1	Segment 2	Segment 3
	R_1 R_2	R_3 R_4	R_5
1	A_1 B_1	— —	—
2	A_2 B_2	$A_1 * B_1$	—
3	A_3 B_3	$A_2 * B_2$	$A_1 * B_1 + C_1$
4	A_4 B_4	$A_3 * B_3$	$A_2 * B_2 + C_2$
5	↓	↓	↓
6	↓	↓	↓

P.T.O

General Structure of 3 segment pipeline:-



S_1 ला output, R_1 ला store करते।



Asynchronous (Handshaking)

Synchronous (clocked Latches)

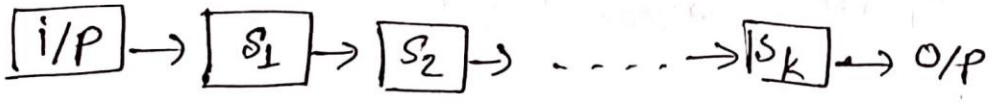
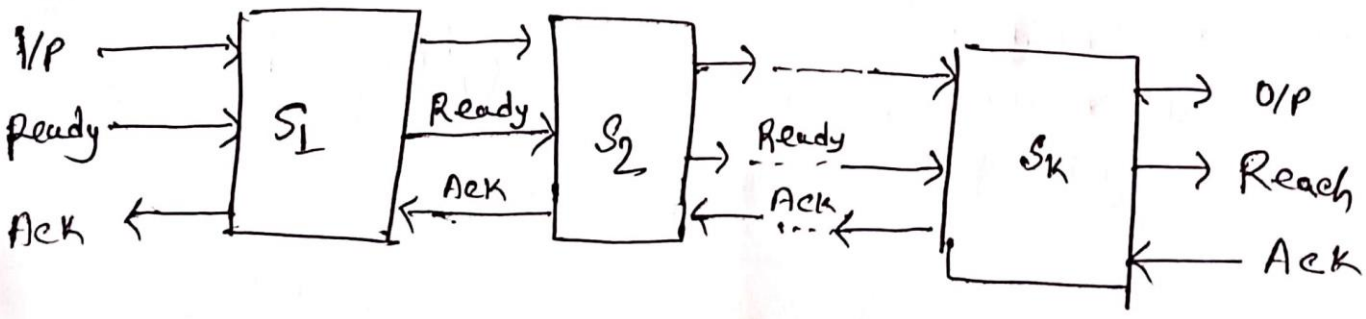


Figure → Linear pipelining (General Structure)

Asynchronous pipelining Model:-

⇒ Data Flow between adjacent ~~stges~~ stages are controlled by handshaking protocol.

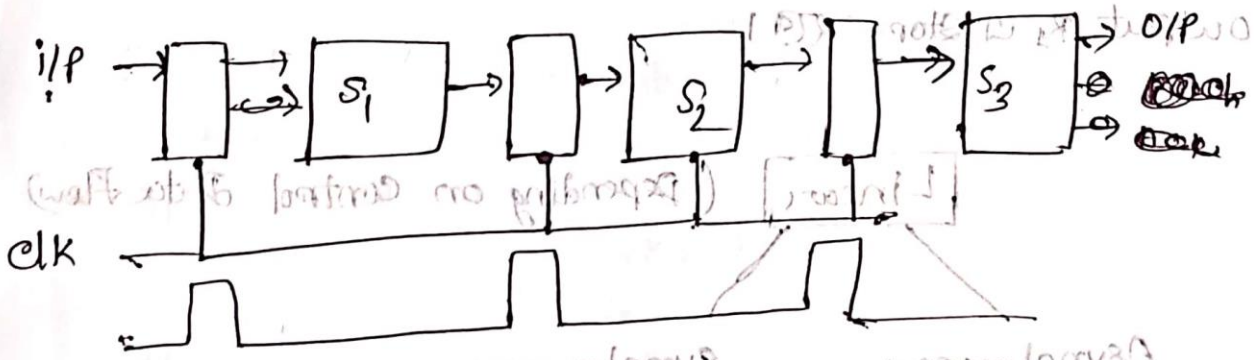


~~Clock lag~~

* Clock - latch একসাথে থাকলে সেটা হবে Synchronous.

Synchronous Pipelining Model :-

Clock আর Latch থাকবে।

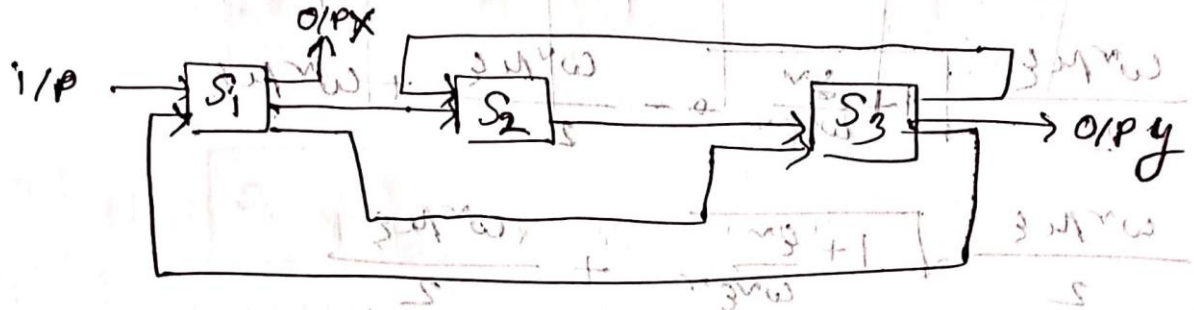


22.11.22

Tuesday

Computer Architecture

Non-Linear Pipeline processor



→ A three stage non-linear pipeline

$S_1 \rightarrow S_2$, $S_2 \rightarrow S_3$ } Streamline connection

$S_1 \rightarrow S_3$ ⇒ Feed forward connection

$S_3 \rightarrow S_2$, $S_3 \rightarrow S_1$ } Feedback

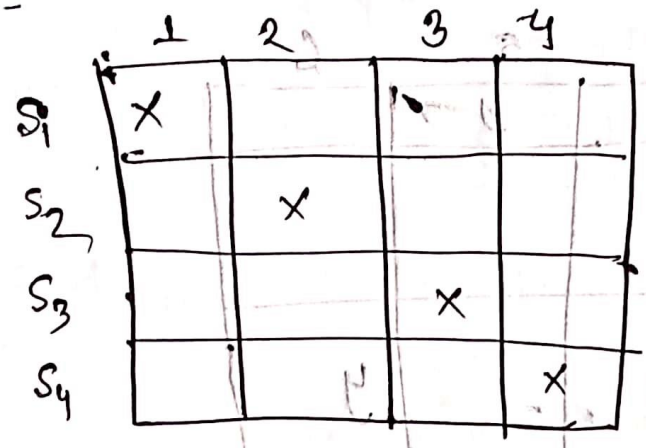
Qus!- Non Linear pipline describe করো।

→ Figure

→ কিভাবে কাজ করে।

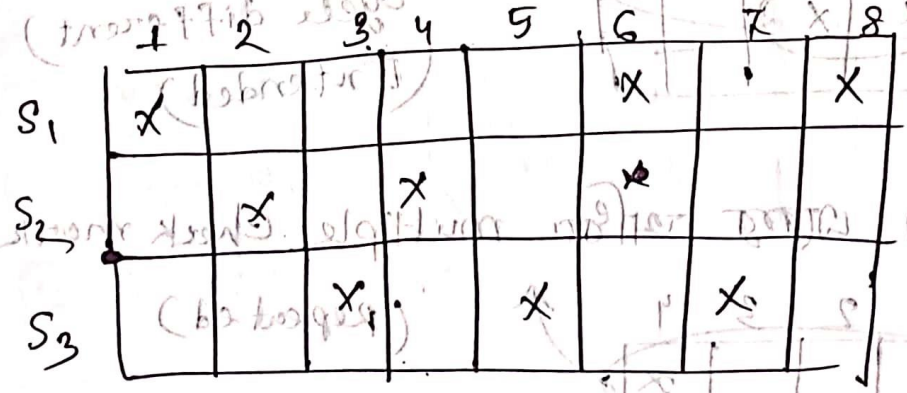
Reservation Table

Linear :-



Reservation Table Linear এর ক্ষেত্রে always Diagonal বরাবর প্রথমে 1 টি stage complete হলে পরেরটি করা উচিত।

Processing Sequence For Output X :-



Sequence :- (of output)

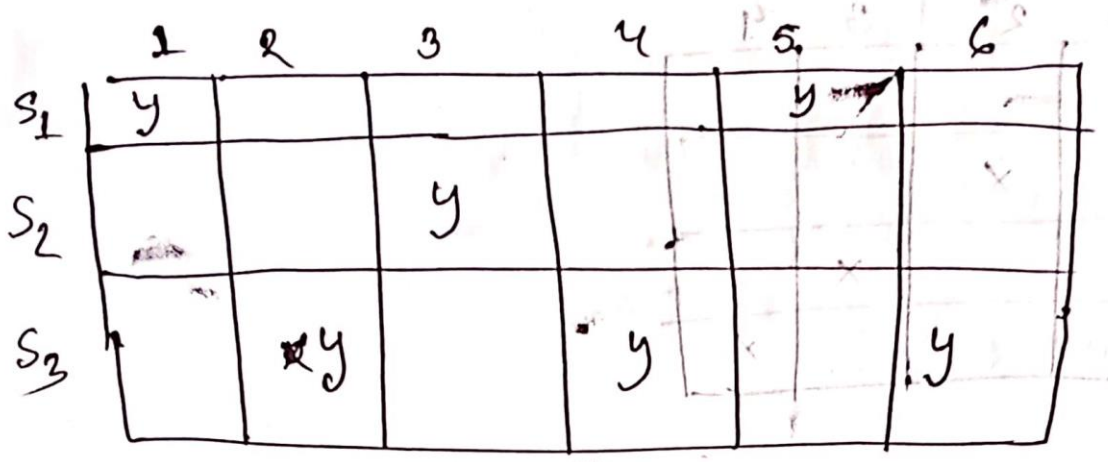
$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_3 \rightarrow$

① $S_1 \longrightarrow$ O/P X

② $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow$ O/P X

③ $S_1 \rightarrow S_3 \rightarrow S_1 \rightarrow$ O/P X

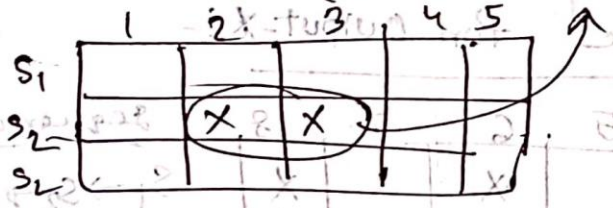
processing sequence for O/P $y \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_3$



(*) ~~Multiple usage in a row:~~ -

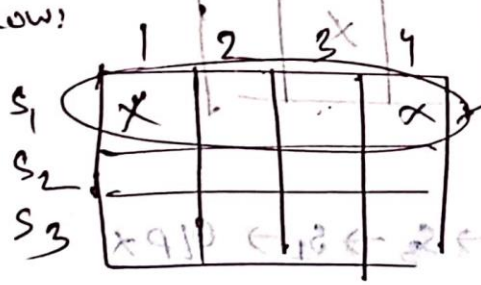
(*) ~~Continuous~~

(*) Back to back ~~दो~~ Continuous ~~रो~~



(Stage same cycle different) (Extended)

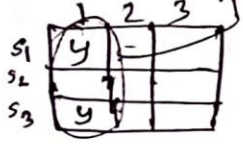
(*) एकही row में एकसे अधिक multiple check mark in a row:



(Repeated)

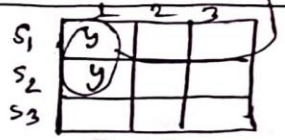
(~~Extended~~ used in a ~~or~~ different ~~use~~ cycle & same stage)

(*) Multiple check mark in a column :-



(Contiguous ~~न~~ ~~रो~~ 3 cycle same Stage different) same stage)

(*) Multiple check mark contiguous :-



(Simultaneous in a cycle Stage different

28.11.22

Computer Architecture

Dependencies:- (নির্ভরতা) → (ILP এর)

- Data dependencies → One instruction is dependent on another (একটা কাজের জন্য আরেকটির help লাগে)
- Name Dependencies → Two instructions use the same register / memory location
- Data Hazards
  - RAW → i Sub R_1, R_2, R_3 ⇒ $R_1 = R_2 - R_3$
    i+1 add R_4, R_1, R_3 ⇒ $R_4 = R_1 + R_3$
    (Instruction i+1 reads operand R_1 before instruction i writes it)
  - WAR
  - WAW
- Control Dependencies → determines the order of the instruction.

(part of the processors h/w needed by two or more instruction)

Structural Hazard ←

Control Hazard ← (due to branches) ↓

Resistor Renaming

~~Her~~ Hazard → Incorrect Result

RAW → Read After Write

WAR → Write After Read

WAW → Write After write

ILP (Instruction Level parallelism)

↳ Measure of how many operation in a computer program can be performed simultaneously.

↳ potential overlap among instruction

	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MA	WB				

IF → Inst fetch, EX → Execut, WB → writeBack, ID → Inst. Decode, MA → Memory Allocation

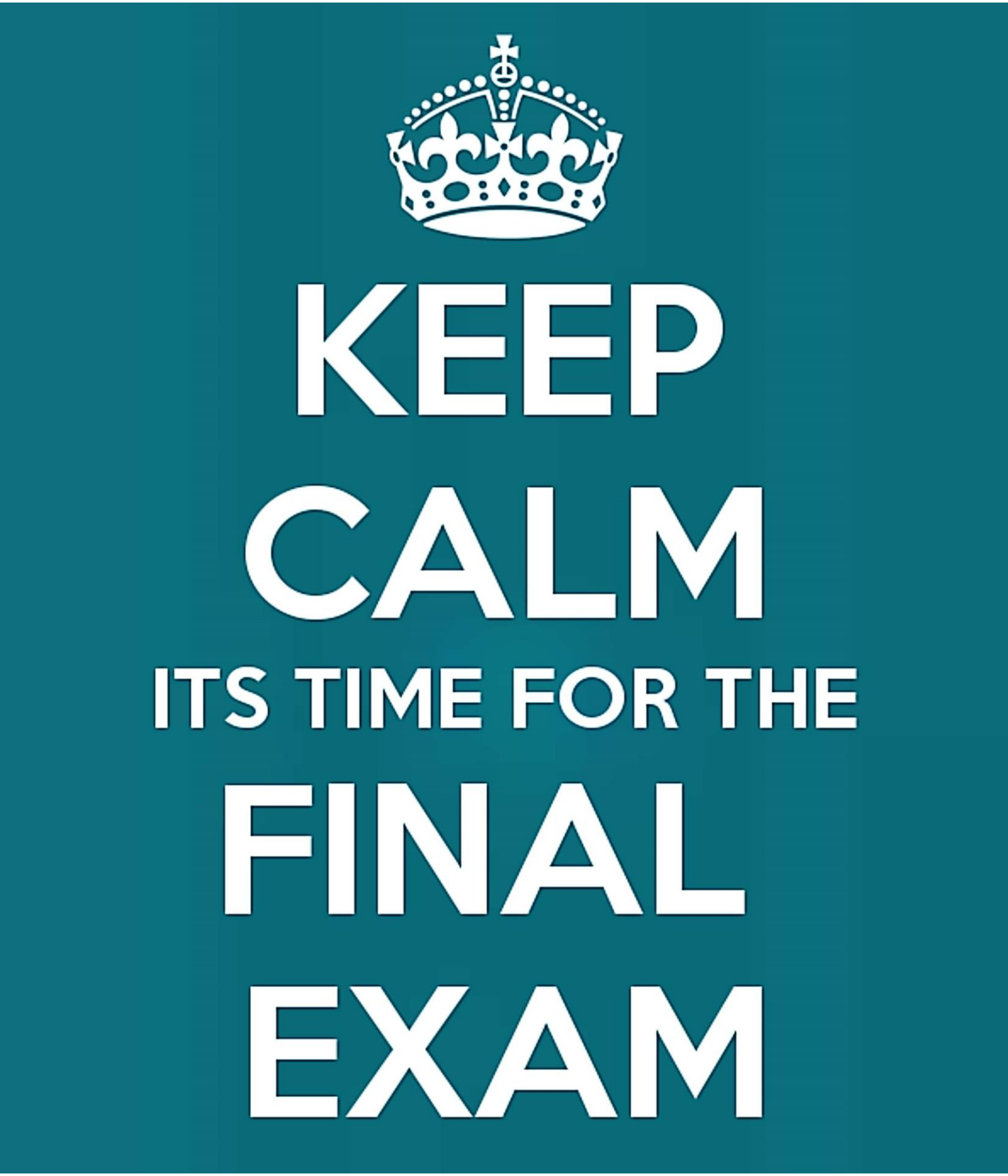
i+ → - -

WAR:→ i: sub R_4 R_5 R_6 → $R_4 = R_5 - R_6$

i+1 add R_5 R_2 R_3 → $R_5 = R_2 + R_3$

i+2 Mul R_6, R_5, R_7 → $R_6 = R_5 * R_7$

Instruction i+1 writes operand R_5 before instruction i reads it.


KEEP
CALM
ITS TIME FOR THE
FINAL
EXAM

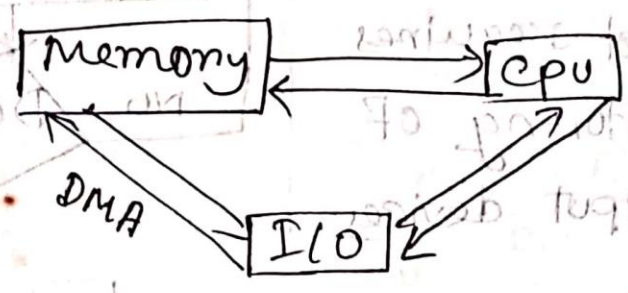
Modes of Transfer:-

=> Data Transfer between central Computer & Input Output devices may be handled in several modes, like :-

→ programmed input-output } CPU as an intermediate path

→ Interrupt initiated Input-output

→ Direct Memory Access. } Direct Transfer to memory.



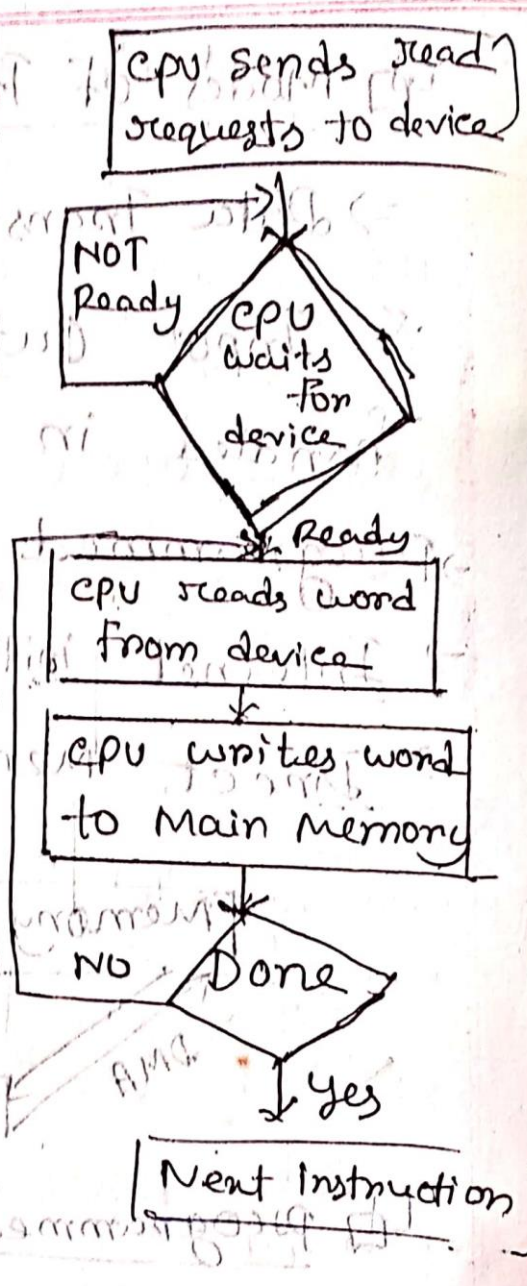
programmed Input-Output:-

→ Result of I/O instructions written in computer program.

→ Each data transfer is initiated by an I/O instruction in the program (To access register / memory on a device

→ Transferring data under program control requires constant monitoring of the input output devices by the CPU.

[In program I/O, CPU makes a request & then CPU stays in program loop (polling) until the I/O device indicates that it's ready for data transfer]



→ The input output device takes no further action to alert CPU [doesn't interrupt CPU]

Disadvantage:-

→ Time consuming process since it keeps CPU busy needlessly.

→ To avoid this problem interrupt facility can be used.

Interrupt Initiated I/O:-

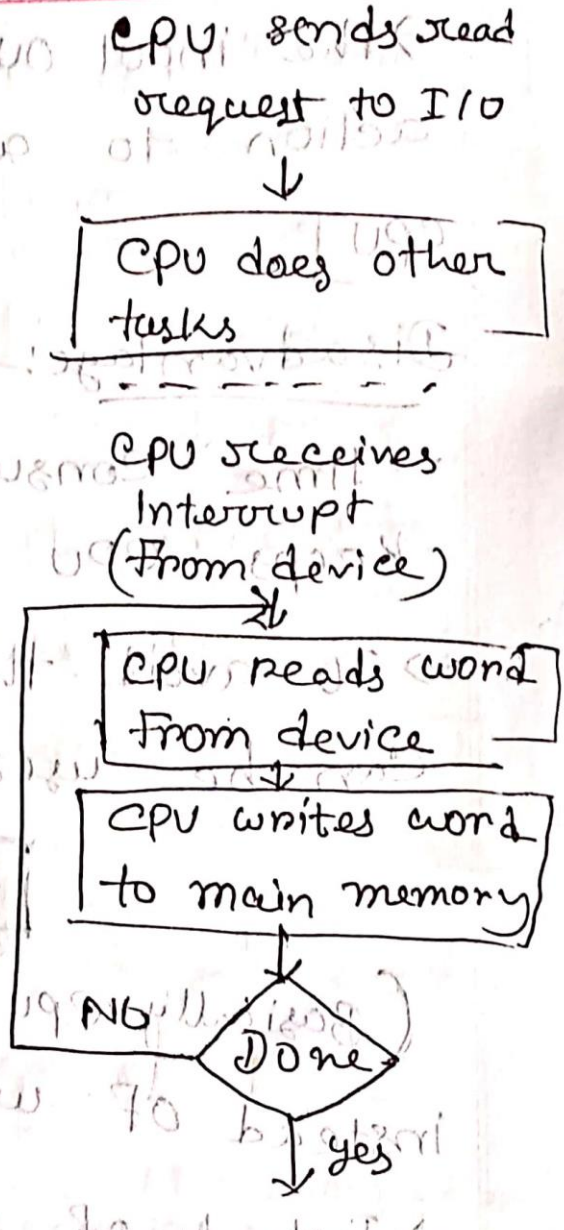
(Basically CPU keeps doing other works, instead of waiting for the I request)

→ Instead of continuous monitoring of CPU interface will be informed to issue an interrupt request signal when datas are available from the device.

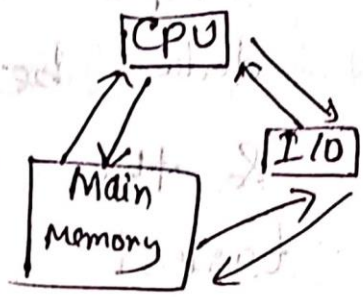
→ CPU proceeds to execute another program & interface keeps monitoring the device

→ When device is ready for data transfer it generates interrupt request.

→ Upon detecting the external interrupt signal the CPU stops the task it is performing process the I/O data transfer & then resumes the original task it was performing.



(iii) Direct Memory Access (DMA):-



To transfer large blocks of data at High-speed between external device & main memory DMA-approach is used oftenly.

→ DMA allows data transfers directly between Input output device & main memory with minimal interaction of CPU.

⇒ DMA means CPU grants I/O interface authority to read from & or write to memory without its involvement.

⇒ DMA itself controls data transfer between Main Memory & I/O device.

⇒ CPU is only involved in the begining & at the end of the transfer and interrupted only after entire block is transfered

CPU sends read request to DMA unit

↓

CPU does other tasks

CPU receives DMA interrupt (From I/O devices)

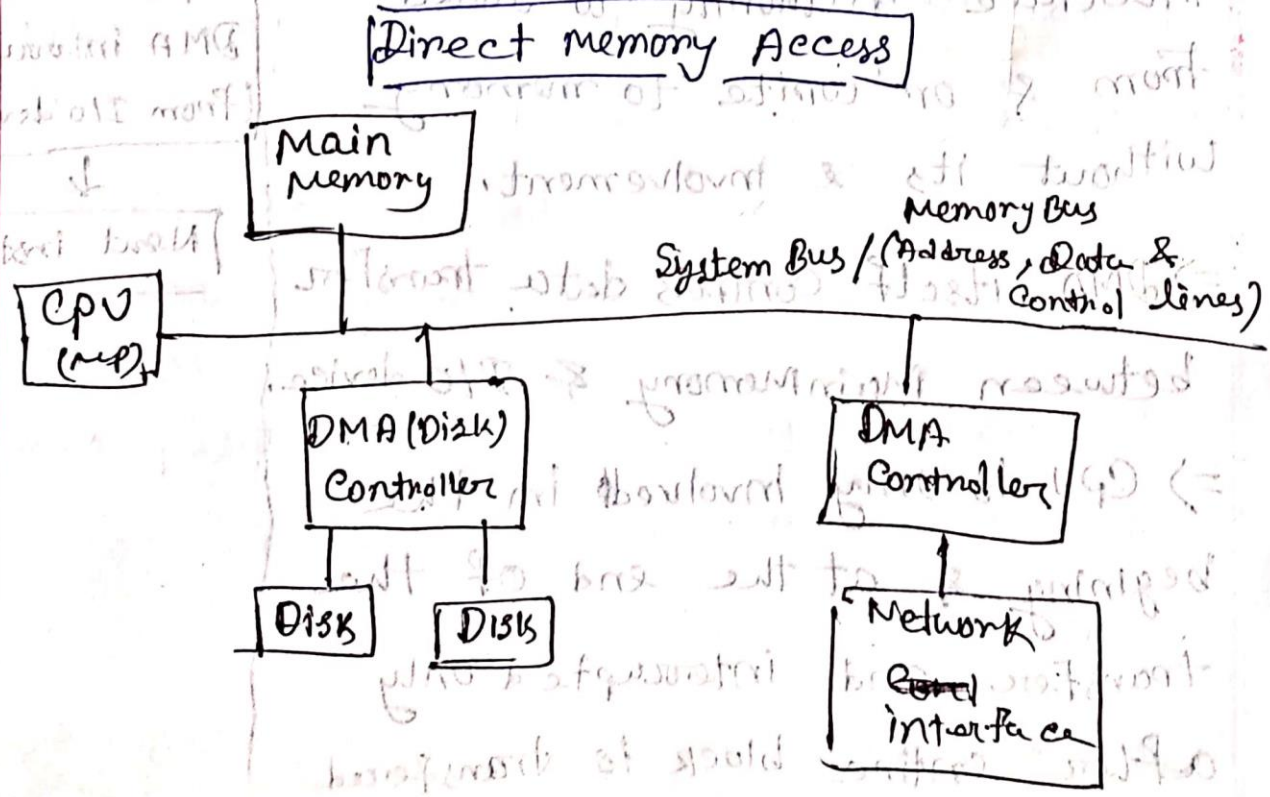
↓

Next instruction

⇒ CPU asks the DMA Controller to transfer data between a device & Main Memory & then CPU proceeds to do other tasks.

⇒ The DMA Controller issues a requests to the I/O device, waits & manages data transfer between the device & main memory.

⇒ When data transfer is finished, the DMA Controller interrupts the CPU.



=> DMA is an I/O technique that provides direct access to the MM to speed up the memory operation. while cpu is temporarily disabled.

=> The process is managed by a chip known as DMA controller (DMAC).

=> I/O devices are connected to system bus via a special interface circuit called DMAC.

=> Both CPU & DMAC have access to MM via a shared system bus having data, address & control lines.

=> During DMA transfer, the CPU is idle & has no control of the system bus.

=> DMAC temporarily borrows the address bus, data bus & control bus from the CPU & transferred data between I/O devices & MM.

-> DMA transfer is also used to do high speed memory to memory transfer. Exam:- USB drive

Making cpu Idle

One common method with two special control signals.

① Bus Request (BR)

② Bus Grant (BG)

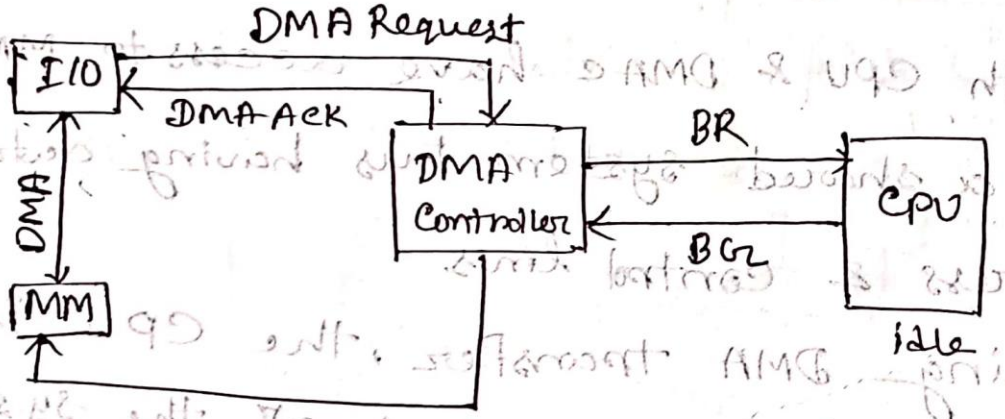


Figure:- MAking CPU Idle.

=>

① The I/O wants to transfer data with MM.

② I/O sends DMA request to DMAC.

③ DMAC sends BR signal to CPU. For the buses.

(4) DMAC waits for the BGr signal from CPU.

(5) CPU relinquishes control of buses, places ABus, DBus, RD & WR lines into a high impedance state.

(6) CPU activates BGr signal & become in idle state.

(7) DMAC takes control of buses & conduct direct memory transfers without CPU intervention.

(8) When DMA transfer terminates, it disable BR signal.

(9) Then the CPU disables BGr signal, takes control of the buses & returns to its normal operations.

DMA transfer are of 2 types:-

1] Burst Transfer (Block Transfer):-

⇨ A block of sequence consisting of a number of memory words is transfered in continuous burst while the DMA is master of the memory buses.

This mode is needed for fast devices such as magnetic disks where data transmission can't be stopped or slowed down until an entire block is transfered.

2] Cycle stealing:-

Allows DMAC to transfer one data word at a time, after which it must return control of the buses to the CPU.

The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to steal one memory cycle.

This mode is used by slow devices such as keyboard.

A device which allows "Direct Memory Access" is called DMA Controller.

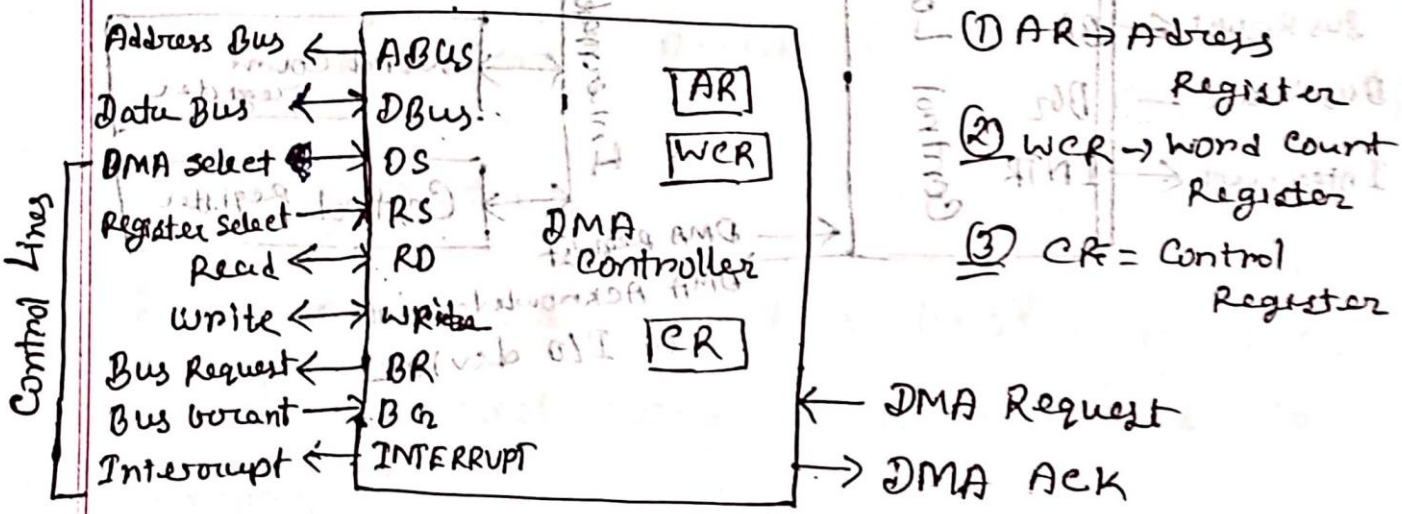
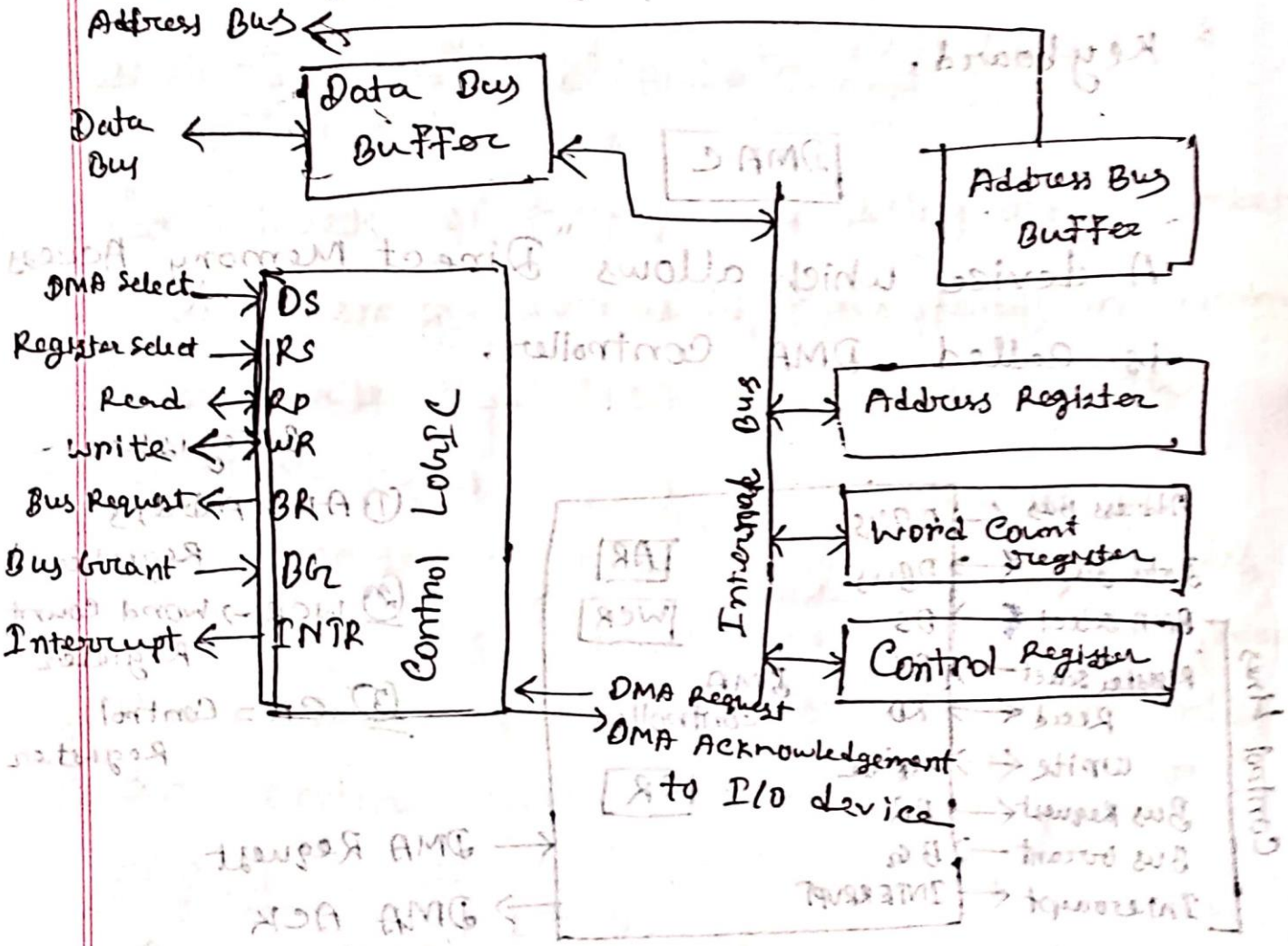


Fig:- DMAC

Block Diagram:-



⇒ The DMAC needs the usual circuits of an interface to communicate with the CPU & I/O devices.

DMac has 3 registers

→ AR (Address Register)

→ WCR (word count Register)

→ CR (Control Register)

Address Register :-

It contains address. It is incremented by 1 after each word is transferred to memory.

WCR :- It holds the number of words to be transferred and it is decremented by 1 after each word transferred and internally tested for 0 (zero).

CR :- It specifies the mode of transfer such as Read or write the memory.

Description:-

⇒ DMAC communicates with CPU via data bus and control lines.

⇒ The registers in DMA are selected by the CPU through the address bus by enabling DS & RS inputs.

⇒ The RD & WR inputs are bidirectional.

⇒ When BG is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.

⇒ When BG = 1, the CPU has relinquished the buses & the DMA can communicate directly with memory by specifying an address in the address bus & activating RD or WR control.

DMA communicates with the external I/O devices through the request & acknowledgement lines by using handshaking procedure.

=> DMA Transfer:-

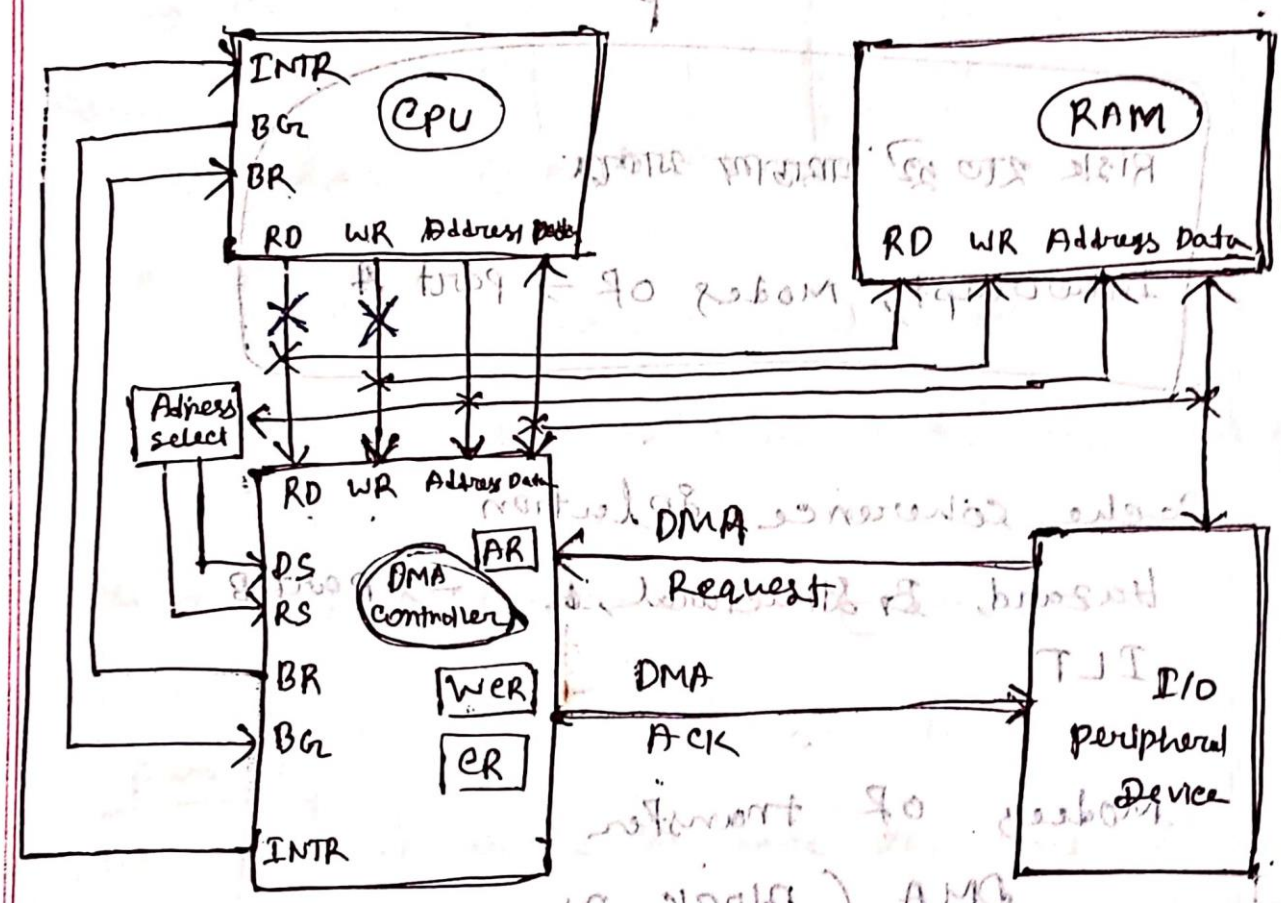


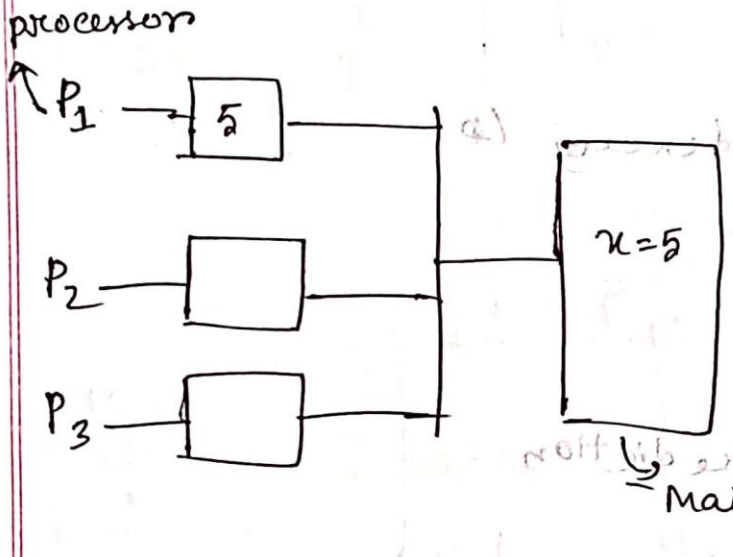
Figure:- DMA Transfer.

Qus:- Cache coherence problem কি? Discuss it. And its solution,

Ans: ~~a~~ Discussion ~~a~~ আগে Discuss করলে ভালো হয়।

Cache Coherence (VVIm)

Cache memory তুলোতে same value থাকা



processor, P-1 read request এর মাধ্যমে MM হতে value 5 নিয়ে আসছে।

তারপর Cac

Consistant Data রাখার জন্য Cache যে problem হয় তাই Cache coherence problem।

It is a challange of multiple

~~Per~~ performace বাড়াতে processor use হয়,

process mignation এর কারণে CC problem হয়,

অর্থাৎ, P_2 এর কাজ P_1 করতে। P_2 এর অনেক কাগজপত্র

কারণে

I/O module এর কারণে হতে পারে।

Cache coherence arise হয় 2 ভাবে।

(i) Write through

(ii) write back

write through:-

processor P_1 → Instruction execute

করার সময় problem হলো

P_1 এ যদি $x+3 = 5+3 = 8$ হয়

এর তখন সাথে সাথে main memory তে update করে

সাথে P_1 ও P_2 এর মধ্যে ও same value update হয়।

write back:-

এই policy তে,

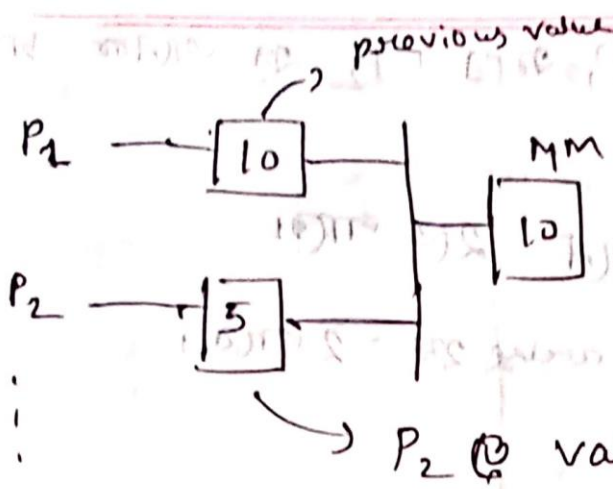
P_1 এ change হওয়ার পর P_1 এ Dirty bit

set থাকবে (অর্থাৎ, যে processor এ কাজ হবে সেই processor এ set থাকবে) । পরে শেষে MM এ

শেষে update করবে।

Snooping based protocol

→ Intel, AMD



यहां पर Cache coherence.

Solve of Cache coherence:

① Snooping based protocol (Bus based protocol)

② Directory Based protocol

Snoopy bus :- (उल्टा बस)

Write update → write through

" " → write back

Write Invalidate → write through

" → write back

Write Invalidate:-

write through হলে Invalid হবে data

•

write back

Directory protocol:-

Cache এ যে update হবে সেগুলো store করে রাখবে। এটা old protocol. এর পরেই snooping based protocol.

MESI protocol:-

M → Modified

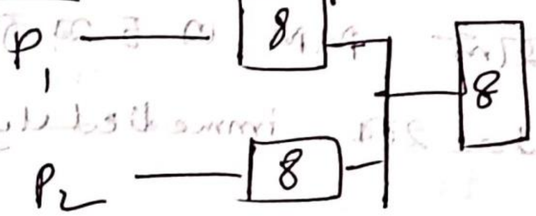
E → Exclusive

S → Shared

I → Invalid

Modified:-

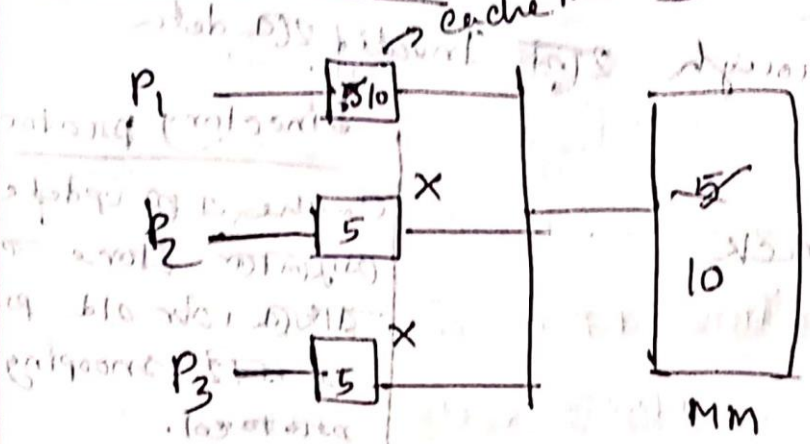
→ tag থাকবে
M → Modify।



Snooping এর problem:-

processor এর সংখ্যা বেশি হলে অনেক cache হয় MM এর সাথে যুক্ত থাকে Data, invalid, Modify এগুলোতে problem হয়

Write Update Invalidate protocol



Processor - P_1 → Instruction execute করার পর

$P_1 = 10$ হলো।

এর পর WIP তে P_1 তে update এর কারণে

Processor Invalid করে দেওয়া। এরপর:-

Write Trough:-

- এর use করলে MM তে 5 এর কারণে

10 টি update হবে immediately.

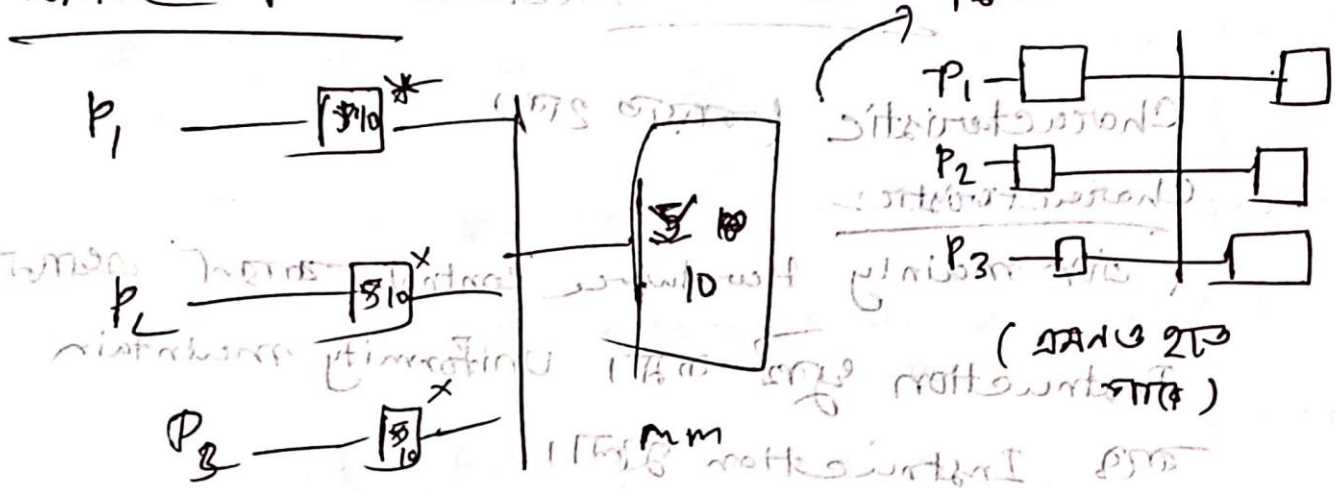
WB: (write Back)

MM পরে update হবে।

তবে আগে P_1 এ Dirty bit add হবে 10।

তারপর update।

Write update :-



Write update হলে আগে আগে থাকা processor গুলোতে update হবে।

Write Through :-

এখানে 5 হবে 10 update হওয়ার সাথে সাথে MM এও 10 আপডেট হবে।

Write Back :-

এখানে Dirty bit set হবে then update হবে MM এ।

এক্ষেত্রে যেই processor এ process হবে যেই processor এ dirty bit add হবে।

Risk Architecture

Characteristic (কাজে হবে)

Characteristic:

এটা mainly Hardware control. কারণ এখানে Instruction কম হয়। Uniformity maintain করে Instruction গুলো।

পুরো Programme কে segment এ divide হয়।

→ এটা Load Store Architecture বলা হয়। (কারণ, [illegible] মেমোরি থেকে load করে instruction execute করতে হয়)

→ Instruction simplified

→ Pipelining বেশি হয়।

→ Instruction fetch হবে decode হবে, সবগুলো Risk Arch এর part।

→ এখানকার operation register to register হয় একমাত্র শুধু Load & register

→ Addressing mode

RISK Arch
-> ~~RA~~ ~~Regis~~ load - Register কেন বলা হয় ?
-> এটা কি Hardware নাকি micro control process করে।

-> Hazard এর মধ্যে কোনটা Ask করেছে সেটা বুঝতে
-> কোনটায় কি problem arise হয় (সমূলো)
-> Data, Branch.

PPT NO-25 (Branch prediction
-> কিভাবে হয়) কী এক্সাক্টো
-> predictons কত প্রকার - -> একমাত্র নাম গুলো শিখতে হবে।

[INTR, pipelining, modes of transfer
-> ভালো করে শেখাতে হবে]

-> Cache coherence কী ? protocol গুলো ?
Solution

-> ~~Instruct~~ [ILP] -> (শেখাতে হবে)

-> MLP [শেখাতে হবে]

-> (A ~~number~~ mesure of avg number of instructions
সেটিই ILP

-> Data dependencies, Control Dependencies,
Structural Hazard.

-> ~~ILP~~