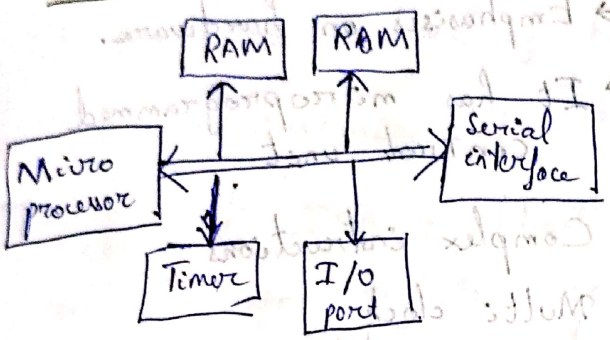
20/1/2023

Components of Processor → ALU, CU, Registers

Microprocessor



↳ Heart of Computer System

↳ Just a processor, memory and I/O components has to be connected externally

↳ Large circuit size

↳ Cost is high

↳ Not suitable to be used with devices running on stored power like batteries.

↳ Mostly don't have power features

↳ Generally in Von-Neumon architecture

Eg. Intel, Pentium, AMD

Micro controllers

Micro processor	ROM	RAM
Timer	I/O port	Serial Interface

↳ Heart of Embedded System

↳ It has external processor along with internal memory and I/O components

↳ Smaller circuit size.

↳ Cost is low.

↳ Since the power consumption is less, it can be used with devices running on stored power like batteries.

↳ Have power saving features.

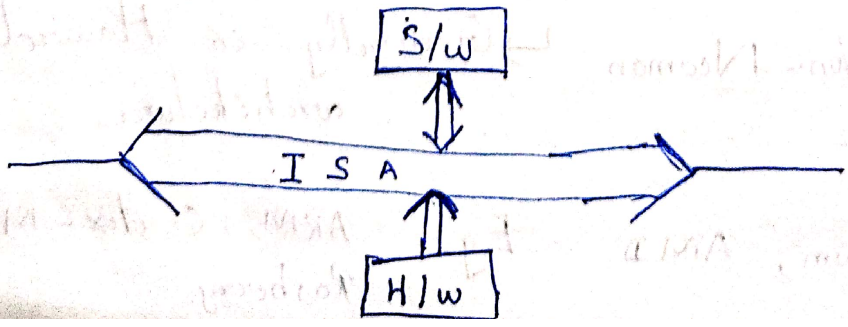
↳ Generally in Havard architecture.

Eg - ARM, Cortex-M4, Rasberry.

4/9/2024

RISC	CISC
↳ Reduced instruction Set Computers	↳ Complex Instruction Set computers.
↳ Emphasises on software	↳ Emphasises on hardware.
↳ It has hardwired control unit	↳ It has microprogrammed control unit
↳ Simple instructions	↳ Complex instructions
↳ Single clock	↳ Multi clock
↳ More no. of registers	↳ Requires Less no. of registers.
↳ Supports ~~more~~ less native data types	↳ Supports ~~less no. of~~ more native data types.
↳ Supports less instruction formats	↳ Supports more instruction format.
↳ Supports less addressing modes.	↳ Supports more addressing modes.
↳ Supports pipelining	↳ Piplining is not supported
Eg- ARM, MIPS, SPARC	Eg- Intel, AMD, IBM 360/(S)

Instruction Set Architecture (ISA)



ISA is the part of computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling and external I/O.

It serves as a boundary between software and hardware.

Seven Dimension of ISA :-

1) Class of ISA : Nearly all ISA are classified as general purpose register architecture where the operands are stored either in registers or memory locations.

Eg- 80x86 — 16 GPRS, 16 that can hold floating point data] Register memory arch.

MIPS — 32 GPRS, 32 that can hold FP data] Load-store architecture

2) Memory Addressing :-

Almost all computers use byte addressing to access the memory operands.

Eg - MIPS — Requires object must be aligned
An access to an object of size 'S' byte at byte address A is aligned if
$A \bmod S = 0$

80x86 — It doesn't require alignment but access are generally faster if operands are aligned.

3) Addressing Modes :-

It specify the address of a memory object.

Eg- MIPS :- Register, Immediate, Displacement
& 80x86

4) Types and Size of Operand :-

Eg- MIPS & 80x86 :- Operand size :-

8 bits (ASCII char)
16 bits (unicode char / half word)
32 bits (Integer / word)
64 bits (Long integer / double word)
32 bits (IEEE 754 FP single)
64 bits (IEEE 754 FP double)

* 80x86 can support 80 bits (Extended double precission)

5) Operations :-

- Data Transfer
- ALU
- Control
- FP

Eg- MIPS :- Simple and easy to pipeline ISA
Represented by 'RISC.

80x86 :- It has much richer larger set of operati

6) Control Flow Instructions :-

All ISA supports :- - Conditional Branches
- Unconditional Jumps

- procedure calls
- return from procedure

Eg- MIPS
- Conditional branches checks the content of registers.
- Procedure calls places the return address of registers.

~~8086~~ 80x85
- Conditional branches test condition code bits set on side effect of ALU operation.
- Procedure calls places the return address on a stack in memory.

7) Encoding ISA :-

There are two basic choices of encoding :-
- Fixed length encoding
- Variable length encoding

Eg- MIPS — Instructions are 32-bit long fixed.
80x86 — ~~Instructions~~ Variable length, ranges from 1-18 bytes

2/2/2023

~~Aton~~ Measuring and Reporting Performance

Performance can be measured in two ways :-

(i) Elapsed / Execution time : Finish time - start time

(ii) Throughput : No. of task executed per unit time.

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

where P → performance
T → execution time.

* More throughput and less execution time.

Basic Performance Equation :-

$$T = \frac{N \times S}{R}$$

where $T \rightarrow$ execution time of program on any HLL (High Level Language)

$N \rightarrow$ Actual no. of instruction to be executed

$S \rightarrow$ Avg. no. of basic steps

$R \rightarrow$ Clock rate $= \frac{1}{\text{clock cycle}}$

CPU Time (Amount of time the processor takes to execute a program)
- User CPU time
- System CPU time

User CPU Time : Amount of time spent by the processor to execute user instruction.

System CPU Time : Amount of time spent by the processor to execute system instruction requireed by the program.

CPI : Cycles Per Instruction

$$= \frac{\text{\# clock cycles required}}{\text{\# Instruction to be executed}}$$

IPC : Instructions Per Cycle $= \frac{\text{\# Instruction to be exeud}}{\text{\# clock cycles used}}$

* CPI should be less
* IPC should be more

Moore's Law :—

It says that no. of transistors in a chip doubles in every 2 years.

CPU Performance Equation :— (Derivation)

$$\text{CPU Time} = \text{CPU clock cycles per program} \times \text{Clock cycle Time} \quad \text{—— (i)}$$

OR

$$\text{CPU Time} = \frac{\text{CPU clock cycles per program}}{\text{clock rate}} \quad \text{—— (ii)}$$

$$\text{clock cycles} = \frac{1}{\text{clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles per program}}{\text{Instruction Count}} \quad \text{—— (iii)}$$

$$\text{CPU clock cycles per program} = \text{CPI} \times \text{Instruction Count} \quad \text{—— (iv)}$$

Substituting eqn (iv) in eqn (i) :—

$$\text{CPU time} = \text{CPI} \times \text{Instruction Count} \times \text{Clock cycle time} \quad \text{—— (v)}$$

↳ CPI depends on Organisation and Instruction set arch.
↳ Instruction count depends upon ISA and Compiler Technology
↳ Clock cycle time depends on H/w technology and Organization

$$\text{Total no. of clock cycle} = \sum_{i=1}^{n} IC_i \times CPI_i \quad \text{——— (v)}$$

where IC_i ⟶ No. of times instruction 'i' is executed on the program.

CPI_i ⟶ The avg. no. of instruction per clock cycle for instruction 'i'.

$$\text{CPU Time} = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times \text{Clock Cycle Time} \quad \text{——— (vii)}$$

$$\text{Overall CPI} = \frac{\left(\sum_{i=1}^{n} IC_i \times CPI_i\right)}{\text{Instruction Count}} \quad \text{——— (viii)}$$

Q) What is the average CPI under the following condition

Instruction Type	Cycles	Relative Frequency
A	2	40 %
B	3	30 %
C	5	25 %
D	10	5 %

Ans- Total clock cycles $= \sum_{i=1}^{n} IC_i \times CPI_i$

$= 2 \times 0.4 + 3 \times 0.3 + 5 \times 0.25 + 10 \times 0.05$

$= 0.8 + 0.9 + 1.25 + 0.5$

$= 3.45$

Average CPI = $\frac{3.45}{4}$ = 0.86

Q) A program runs in 20 secs on Machine - A with a clock speed of 200 MHz. A computer architect wants to build machine - B, which will run this program in 6 secs. The architect has determined that a substantial ~~decrease~~ increase in the clock rate is possible. But this may effect the design of the rest of the CPU, causing machine - B to require 1.2 times as many clock cycles as machine - A. What clock rate should be target for a best design ?

Ans- $CPU\ Time_A = \frac{Clock\ Cycles_A}{Clock\ Rate_A}$

$\Rightarrow$ ~~20 sec = 200 × 10^6 Hz~~

$\Rightarrow 20\ sec = \frac{Clock\ Cycles_A}{200 \times 10^6\ Hz}$

$\Rightarrow Clock\ Cycles_A = 20 \times 200 \times 10^6$

$CPU\ Time_B = \frac{Clock\ Cycles_B}{Clock\ Rate_B}$

$\Rightarrow 6\ sec = \frac{1.2 \times 20 \times 200 \times 10^6}{Clock\ Rate_B}$

$\Rightarrow Clock\ Rate_B = \frac{1.2 \times 20 \times 200 \times 10^6}{6} = 800\ MHz$

$\therefore$ Targeted clock rate should be 800 MHz.

12/02/2015

Q) Suppose we have made the following arrangement

Frequency of FP operations (other than FPSQR) = 25%
Average CPI of FP operation = 4.0
Average CPI of other construction = 1.33
CPI of FPSQR = 20
Assume two design ~~alterations~~ alternatives :-
(i) Decrease the CPI of FPSQR to 2
(ii) Decrease the CPI of FP ~~SQR~~ to 2.5

Compose the 2 design alternatives using CPU performance equation.

Ans -

Type	Frequency	CPI
FP	25%	4.0
FPSQR	2%	20
other	73%, (100 - 27)%	1.33

Now, CPU Time original = 25% × 4 + 2% × 20 + 73% × 1.33
= 2.37

(i) CPU Time FPSQR = 25% × 4 + 2% × 2 + 73% × 1.33
= 2.01

(ii) CPU Time FP = 25% × 2.5 + 2% × 20 + 73% × 1.33
= 1.63

SPEC Rating

System Performance Evaluation Corporation.

↳ A non-profit organisation ~~for~~ called SPEC selects and publishes representative application programs from different application domains.

Programs selected are :-
- Games
- Astro physics
- Quantum Chemistry
- Database
- Compilerers

↳ The same program is also compiled and run on one computer selected as reference.

$$\text{SPEC rating} = \frac{\text{Running on Reference Computer}}{\text{Running time on Computer Under Test}}$$

$$\text{Overall SPEC Rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{1/n}$$

where n ⟶ no. of programs in the suit

Q) Show that the ratio of the geometric means is equal to the geometric mean of performance ratio and that the reference computer of SPEC ratio matters not.

Ans - Let there are 2 computers A and B and SPEC ratio of each = $\frac{\text{geometric mean of A}}{\text{geometric mean of B}}$

$$= \frac{\sqrt[n]{\prod_{i=1}^{n} \text{SPEC}_{A_i}}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPEC}_{B_i}}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{SPEC}_{A_i}}{\text{SPEC}_{B_i}}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time of Reference}}{\text{Execution time of } A_i}}$$

$$\frac{\text{Execution time of Reference}}{\text{Execution time } B_i}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time } B_i}{\text{Execution time } A_i}}$$

$$= \boxed{\sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance of } A_i}{\text{Performance of } B_i}}}$$

As $\text{Performance} = \frac{1}{\text{Execution time}}$

Amdahl's Law :- 15/02/2023

It depends on 2 factors :-

(i) Fraction enhanced :- The fraction of the computation time in the original computer that can be ~~computed~~ converted to take advantage of the enchancement.

$$\boxed{\text{Fraction}_{\text{enhanced}} \leq 1}$$

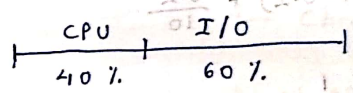
(ii) Speedup enhanced : The improvement gained by the enhanced operation mode.

$$\boxed{\text{Speedup}_{\text{enhanced}} \geq 1}$$

$$\boxed{\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}}$$

Q) The processor has to be enhanced for web serving. The new processor is 10 times faster on computation than the original one. The original processor was busy with computation 40% of time and waiting for I/O 60% of the time. What is the overall speedup gained by incorporating the enhancement?

Ans-



= 10 times faster

$\text{Fraction}_{\text{enhanced}} = 40\%$

$\text{Speedup}_{\text{enhanced}} = 10$

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}$$

$$= \frac{1}{(1 - 0.4) + \frac{0.4}{10}}$$

$$= 1.56$$

Q) Suppose FPSQR is responsible for 20% of execution time of a critical graphics benchmark, 1st proposal is to enhance the FPSQR hardware and speedup the operation by a factor of 10. Second proposal is to make all FP instructions in the graphic processor run faster by a factor of 1.6. FP instruc

are responsible for total of 50 % of the execution time for the application. Compare both design alternatives.

Ans- ⟨1⟩ Speedup FPSQR — Fraction enhanced = 20 %
Speedup enhanced = 10

⟨2⟩ Speed FP — Fraction enhanced = 50 %
Speedup enhanced = 1.6

So, $Speedup_{overall\ for\ 1} = \frac{1}{(1-0.2) + \frac{0.2}{10}} = 1.22$

$Speedup_{overall\ for\ 2} = \frac{1}{1 - 0.5 + \frac{0.5}{1.6}} = 1.23$

So, the 2nd alternative is slightly better than the 1st one

$$Speedup_{overall} = \frac{1}{\left(1 - \sum Fraction_{enhanced}\right) + \frac{\sum Fraction_{enhanced}}{\sum Speedup_{enhanced}}}$$

16/2/2023

PIPELINING

It is effective way of organising concurrent activity in a computer system.

↳ It is a technique for overlapping the execution of several instruction to reduce the execution time for a set of instruction.

DLX Pipeline

↳ It has 5 stages.

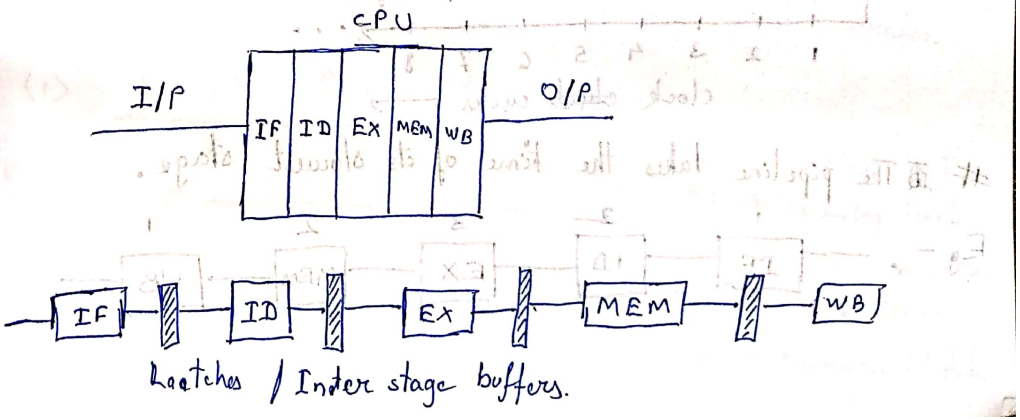
1) Instruction Fetch (IF) — Fetching the instruction; Incrementing the PC

2) Instruction Decode (ID) — Interpretation of op-code; Decode and Search where the source system are present; Read the register content; Check for branch

3) Execution (EX) — ALU operations; EA calculation (Effective Address)

4) Memory (MEM) — Load; Store

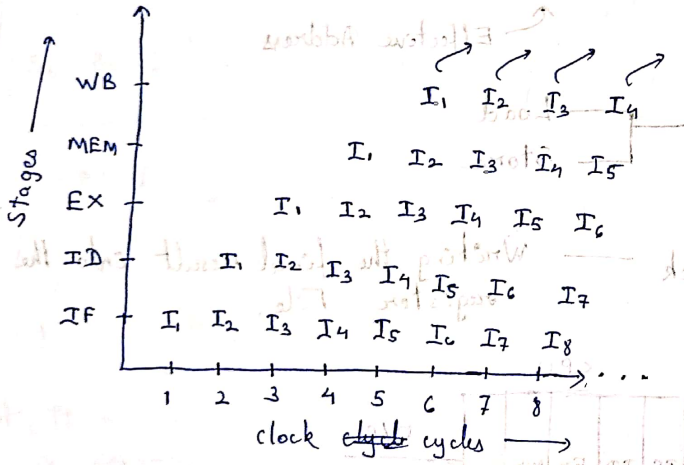
5) Write Back (WB) — Writing the final result into the register File.



An instruction can be forwarded to next stage if the buffer of the next stage is free.

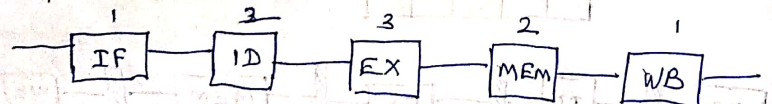
Time - Space Diagram :—

CC →	1	2	3	4	5	6	7	8	9
I_1	F_1	D_1	E_1	M_1	W_1				
I_2		F_2	D_2	E_2	M_2	W_2			
I_3			F_3	D_3	E_3	M_3	W_3		
I_4				F_4	D_4	E_4	M_4	W_4	
I_5					F_5	D_5	E_5	M_5	W_5



The pipeline takes the time of its slowest stage.

Eg -



CC →	1	2	3	4	5	6	7	8	9	10	11	12	13
I_1	F_1	D_1	D_1	E_1	E_1	E_1	M_1	M_1	W_1				
I_2		F_2		D_2	D_2		E_2	E_2	E_2	M_2	M_2	W_1	
I_3				F_3			D_3	D_3		E_3	E_3	E_3	M_3

↳ In a pipeline, a single instruction takes same or more time for execution while multiple instruction takes less time.

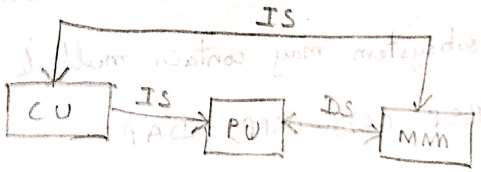
Taxonomy of Parallel Computer Architecture

Flynn's Classification :—

↳ Based on data stream and instruction stream.
↳ 4 categories of computer :-

1) SISD → Single Instruction Stream - Single Data Stream
2) SIMD → Single Instruction Stream - Multiple Data Stream
3) MISD → Multiple Instruction Stream - Single Data Stream
4) MIMD → Multiple Instruction Stream - Multiple Data Stream.

(1) SISD Organisation



CU → Control Unit
PU → Processing Unit
IS → Instruction Stream
DS → Data Stream
MM → Memory Module

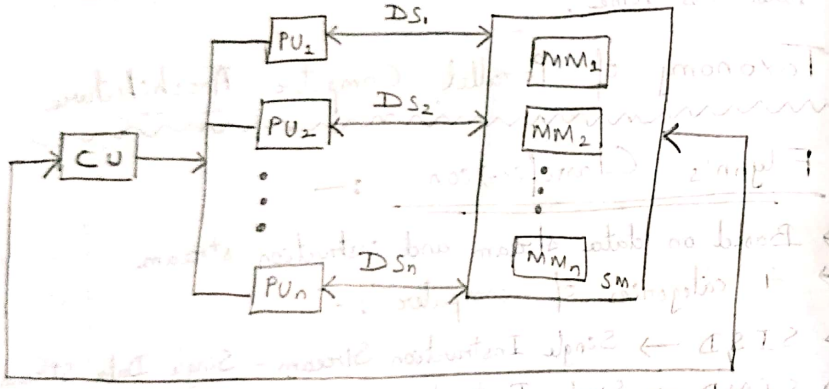
↳ Instructions are executed sequentially but may overlap in their execution stage.

↳ Most SISD uniprocessors are pipelined.

↳ It can have functional unit but one ALU.

Eg- IBM 701 (1), IBM 1620 (1), IBM 360/91, CDC 6600

(2) SIMD Organisation :-



SM → Shared Memory

↳ This class corresponds to array processor.

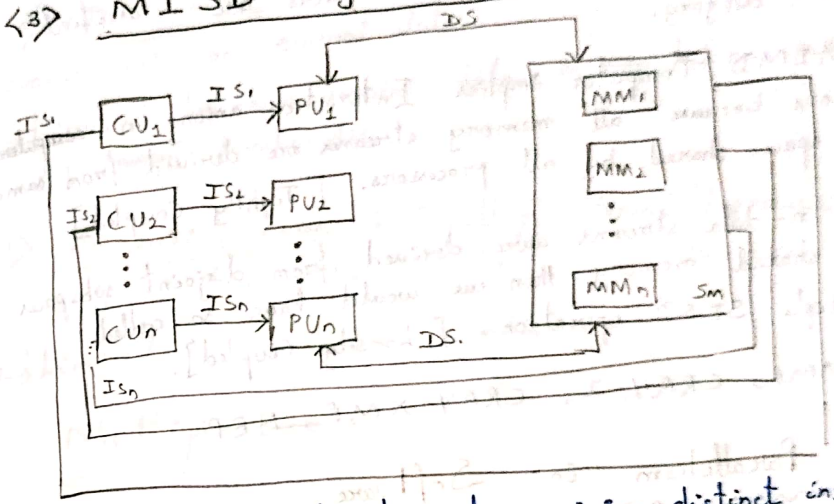
↳ There are multiple PUs supervised by the same Control Unit.

↳ All PUs receive the same instruction broadcast from the CU but operate on different datasets from distinct data streams

↳ The Shared Memory subsystem may contain multiple modules

Eg- ILLIAC-IV, BSP-(C), MPP, DAP.

(3) MISD Organisation :-

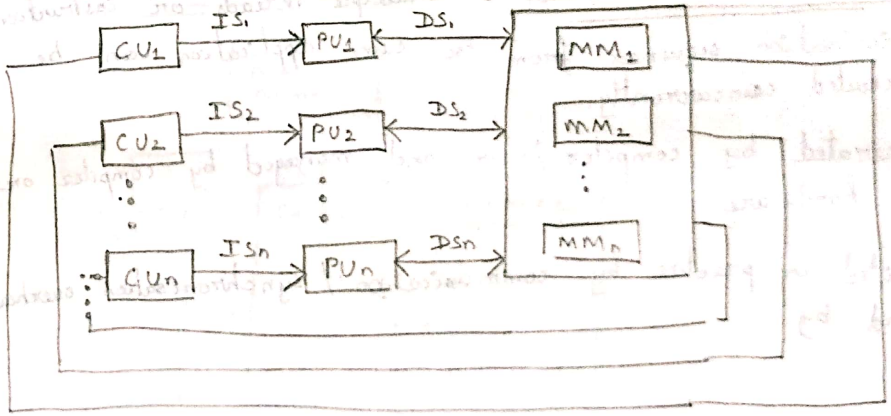


↳ Here m-processor unit ~~acts~~ each receiving distinct instruction operating over the same data stream its derivatives.

↳ The result of one processor become the input of next processor in the macropipe.

* There is no practical implementation of this organisation

(4) MIMD Organisation :-



↳ Most multiprocessors and multicomputers are classified into this category.

↳ An MIMD Computer implies Interaction among n-~~comput~~ processors because all memory streams are derived from same data space shared by all processors. [Tightly Coupled]

↳ If the n-data streams were derived from disjoint subspaces of the shared memory then we would have so called Multiple SISD operation. [Loosely Coupled].

Eg- CMMP, CRAY-2, CRAY-XMP, HEP

22/2/2023

Parallelism in Software

1) **Instruction Level** : Multiple instruction from the same instruction stream can be executed concurrently.

↳ Generated and managed by hardware (Super scalar) or by compile (VLIW).

↳ Limited in practice by data or controlled dependencies.

2) **Task Level / Thread Level** : Multiple threads or instruction or instruction sequences from the same application can be executed concurrently.

↳ Generated by compiler / user and managed by compiler or hardware

↳ Limited in practice by communication / synchronisation overhead and by

3) **Data Level** : Instructions from single stream operates concurrently on several data.

↳ Limited by non-regular data manipulation patterns and by ~~many~~ memory bandwidth.

4) **Transaction Level** : Multiple threads / processes from different transaction can be executed concurrently.

↳ Limited by concurrency overhead.

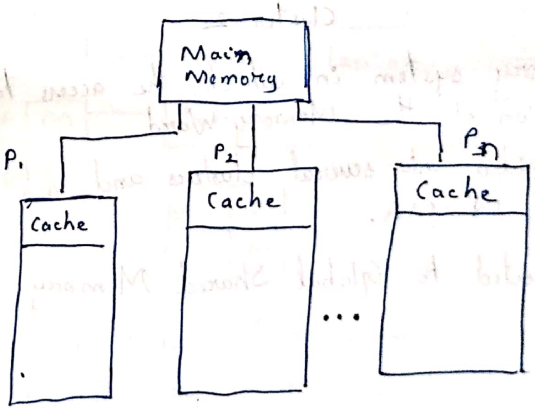
Multiprocessor

Multiple processes are

3 types :-

1) UMA (Uniform Memory Access)
2) NUMA (Non-uniform Memory Access)
3) COMA (Cache Only Memory Access)

1) Uniform Memory Access (UMA)



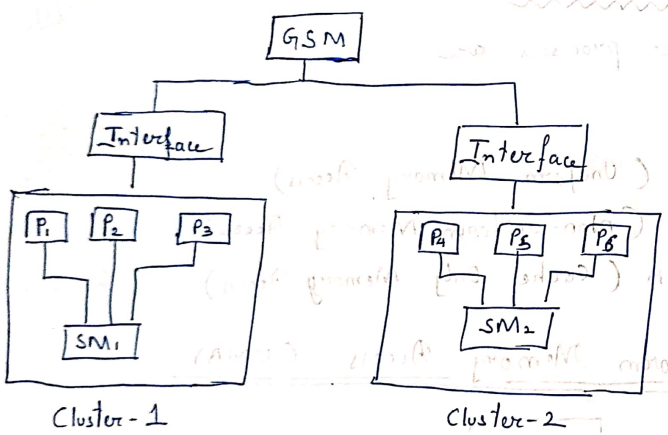
↳ All processors have equal access time to all memory

(i) Symmetric UMA : All processor have equal access to all peripheral devices.

(ii) Assymmetric UMA : Only one or a subset of processor can execute OS and handle I/O. The remaining processors have no I/O capability and are called attached processors.

26/2/2023

2) Non- Uniform Memory Access

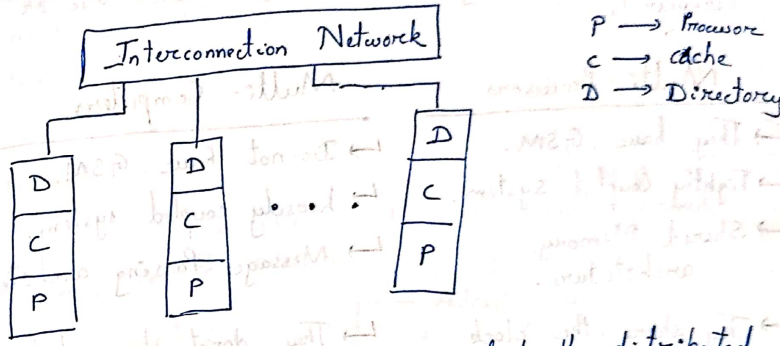


↳ It is a shared memory system in which, the access time varies with the location of the Memory Word.

↳ The processors are divided into several clusters and each cluster is a UMA or NUMA.

↳ The clusters are connected to Global Shared Memory through an interface.

3) Cache Only Memory Access (COMA)



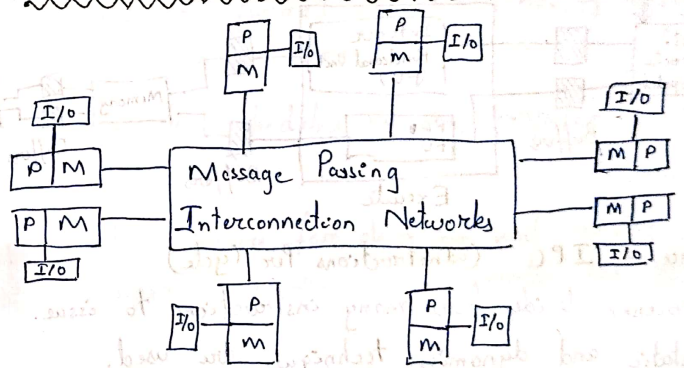
P → Processor
C → cache
D → Directory

↳ It is a special case of NUMA in which the distributed Main Memories are converted into cache.

↳ All the caches form a Global Address Space.

↳ Remote cache access is achieved by the distributed cache directories.

Distributed Memory (Multi Computer)



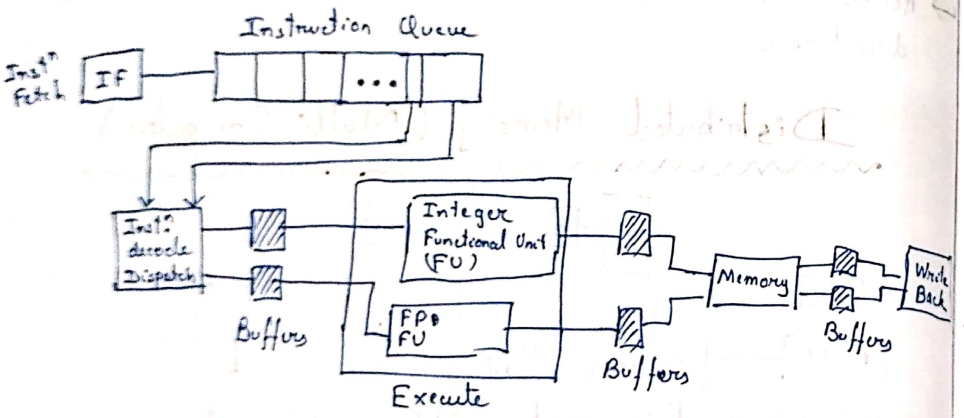
↳ The system consist of multiple computers called nodes interconnected by a message passing network.

↳ Each node is an autonomous computer consisting of a processor, local memory and sometimes attached disc or I/O peripherals.

Multi-Processors	Multi-Computers
↳ They have GSM.	↳ Do-not have GSM.
↳ Tightly Coupled system.	↳ Loosely coupled system.
↳ Shared Memory architecture.	↳ Message Passing architecture.
↳ They share the clock and I/O.	↳ They donot share clock or I/O.

Super Scalar Processors :-

29/2/2023



Performance = IPC (Instructions Per Cycle)

↳ The processor decides how many instructions to issue.

↳ Both static and dynamics techniques are used.

H/w Requirements :-

- Instruction allignment
- Arithmetic units and pipeline
- Interaction among ~~functional~~ integer and FP units

Advantages :-

↳ Smaller code size.
↳ Binary compatibility across generation of hardware.
↳ Faster computation.
↳ More throughput.

Limitations :-

↳ It limits of available ILP (Instruction Level Parallesim) in the program.
↳ Hardware complexity.
- Added FU
- Added BandWidth
- More registers and ports for
- Large cache

VLIW

Very long Instruction Word.

↳ It issues a fix number of instructions per cycle.
↳ This is done by formatting a fixed no. of instructions as one Long Instruction Word.
↳ This is statically scheduled by the compiler.

Advantages :-

- Simplified hardware for decoding and issuing instruction
- No interlocked hardware. (The compiler has already checked it)
- More registers but simplified hardware for register ports

Limitations :-

- Technical Problems
  - wasted ~~Unused~~ bits for unused FUs
  - Increase bandwidth

- A cache miss in any FU causes the entire processor to stall because of block step operation of VLIW.
- Primarey compatibility problems since machines with diff. no. of issues and FU latencies requires diff. versions of machine code.
- Cannot be proposed in assembly language because of high task ~~compatibility issue~~. complexity issue.

Vector Processing

1/03/2023

It is a co-processor specially designed to ~~compute~~ perform vector computation.

↳ A vector instruction involves a large array of operands

↳ The same computation will be performed over a array of data

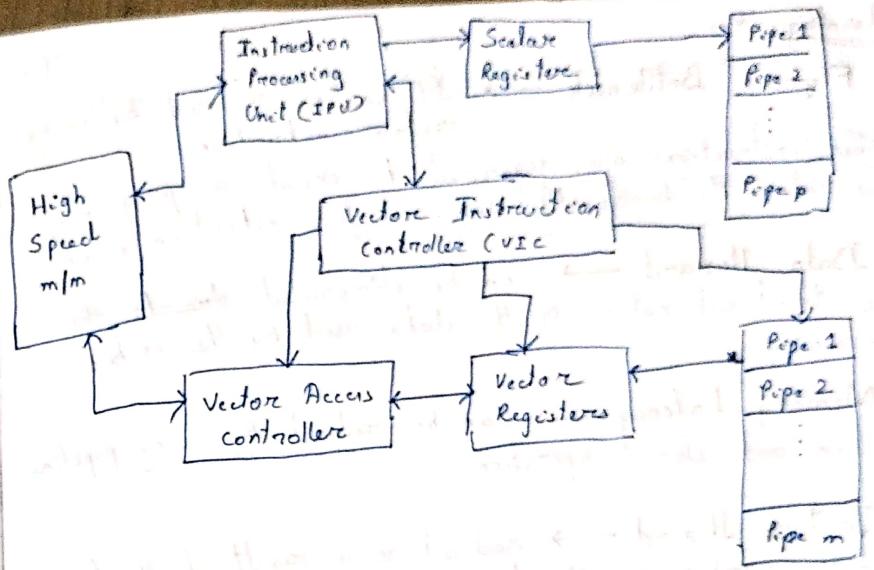
↳ These are often used in multi-pipelined super computer.

Formed of Vector Instruction :-

Op Code	Base addr. Source 1	Base Addr. Source 2	Base Addr. Destination	Vector Length

↳ The vector processing can be successfully implemented by using the following approaches :-

- Enrich vector instruction set
- Combine scalar instruction
- Select a suitable algorithm.
- Use of vectorizing compiler.



IPU : Fetches and decodes both scalar and vector instruction

- The scalar instructions are dispatched to scalar processors through scalar registers
- The vector instructions are dispatched to VIC.

VIC :-
- Decoding of vector instruction.
- Calculating effective address of vector operands
- Setting of vector access controller and vector processor
- Monitoring the execution of vector instructions.

VAC : Responsible for fetching the vector operands by a series of access from the main memory.

- More vector registers are provided to bridge the speed gap between the main memory and vector processor to avoid a stall

Advantages :-

↳ Flynn's Bottleneck ⟶ It can be reduced by using vector instructions.

Fewer instructions are required to execute a program. This reduces the bandwidth required for instruction fetch.

↳ Data Hazard ⟶ Can be eliminated ~~due to the~~ by the structured nature of the data used by the vector machine.

↳ Memory Latency ⟶ Can be reduced by using pipeline 'load and store' operation

↳ Control Hazard ⟶ Reduced as a result of specifying a large number of iterations in a single vector instructions.

↳ Pipeline ⟶ Can be exploited to the maximum extent This is facilitated by the absence of data and control hazard.

Array Processor

↳ It can be considered as a vector or a set of vectors.

↳ A processor capable of processing arrays is called array processing

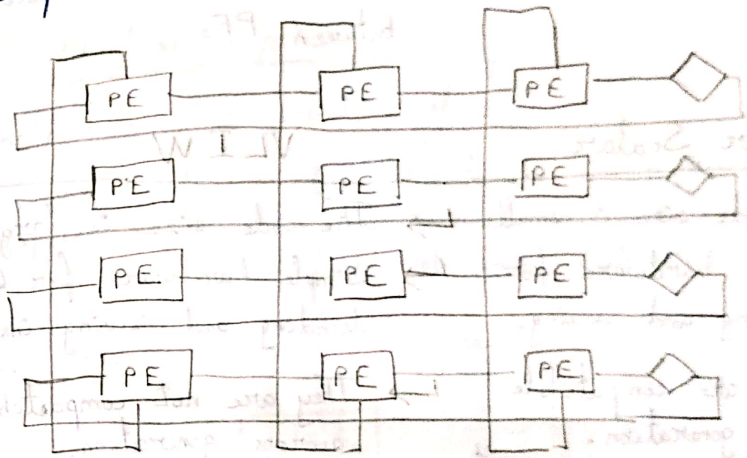
↳ It performs a single instruction on multiple execution unit in the same clock cycle.

↳ The different execution units process instruction using same set of vectors in their array.

↳ The operations are divided into several stream operands and run for the streams in the parallel processing unit.

↳ It is special case of pipelining where the parallel pipelines operate on different stream of operands

↳ The control unit of an array processors fetches and decodes instruction and broadcast the appropriate control signals to the ~~dividend~~ individual data points within all the processors.



Vector Processing

↳ Parallelism is achieved by a multiplicity of concurrently executables distinct and specialized functional units that collectively constitute the overall ALU of the process

Array Processing

↳ Parallelism is achieved by a multiplicity of processing elements (PEs) each of the each of which constructs a distinct ALU in its own right.

↳ Parallelism is also achieved by pipelining the individual functional unit

↳ Data communication is achieved through memory functional units.

↳ Several current stream data traffic are at the process memory interface parallelism is achieved.

↳ Data communication is through shared memory or through direct transmission between PFs

Super Scalar

↳ The code size is small.

↳ Complex hardware for decoding and issuing.

↳ They are compatible across generation.

↳ No change in between is required

VLIW

↳ The code size is large

↳ Simple hardware for decoding and issuing instru

↳ They are not compatible across generation.

↳ Requires more registers for simplified hardware ~~required~~ register points.

MODULE - II

7/02/2023

CLASSIFICATION OF PIPELINE

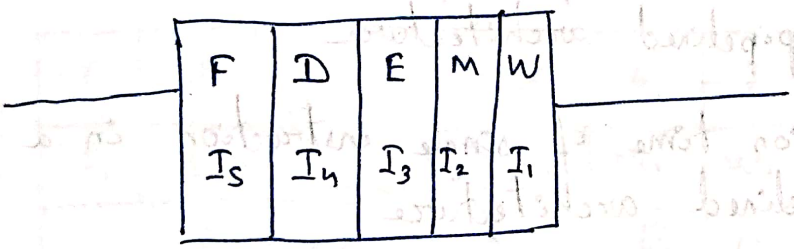
Handler's Classification - 1977
Based on Level of Processing.

1) Instruction Pipeline

Instruction look-ahead pipeline

↳ Execution of stream of instructions is pipelined by overlapping the execution of current instruction with :- fetch, decode, operand fetch of other instruction

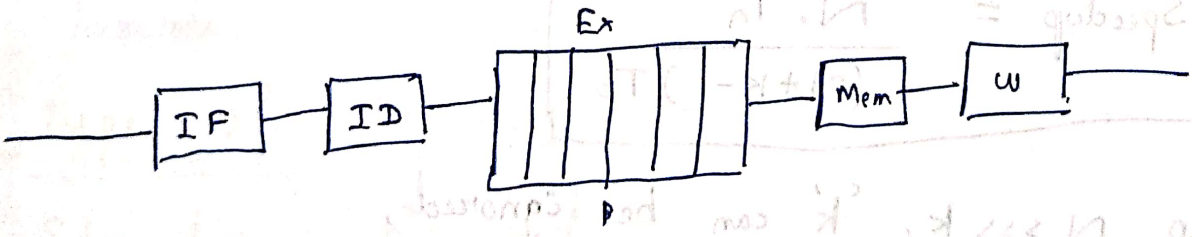
↳ It is Inter Instruction Pipeline



2) Arithmetic Pipeline

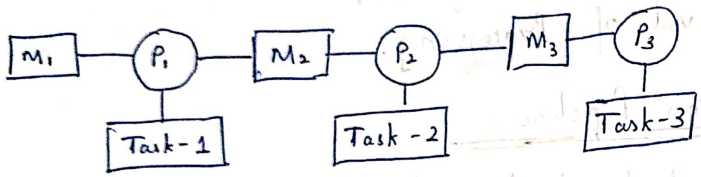
↳ The ALU of a computer are segmented into various data formats for pipeline operations.

↳ Intra-instruction pipeline.



3> Processor Pipeline

↳ Same data stream is processed by cascade of processors each processor performs a specific task.



Speedup vs No. of Stages :-

* Speedup ratio = $\frac{\text{Execution time in non-pipelined processor}}{\text{Execution time in pipelined processor}}$

Let T_n ⟶ Execution time of single instruction in a non-pipelined architecture

T_p ⟶ Execution time of single instruction in a pipelined architecture

k ⟶ No. of stages

N ⟶ No. of instructions to be executed

So, Speedup = $\frac{N \cdot T_n}{kT_p + (N-1)T_p}$

$$\text{Speedup} = \frac{N \cdot T_n}{(N+k-1)T_p}$$

when $N >>> k$, 'k' can be ignored,

Speedup = $\frac{N\,T_n}{N \cdot T_p}$

$$\Rightarrow \text{Speedup} = \frac{T_n}{T_p}$$

In an ideal case, $T_n = k \cdot T_p$

So, Speedup ratio = $\frac{k\,T_p}{T_p}$

$$\Rightarrow \text{Speedup ratio} \propto k$$ ⟶ More the no. of stages, more is the speed.

Pipeline Performance

↳ For a variety of reasons, one of the pipeline stages may not be able to complete its task for a given instruction in time allocated.

	1	2	3	4	5	6	7	8
I_1	F_1	D_1	E_1	E_1	E_1	M_1	W_1	
I_2		F_2	D_2	–	–	E_2	M_2	W_2
I_3			F_3	–	–	D_2		
I_4				–	–	E_4		
I_5					–	–	E_5	

2 clock cycle penalty

2 stalls or 2 bubbles

↳ Any reason for which a stall occurs are known as hazards

3 types of Hazards :-

(i) Structural Hazard

#The situation when 2 instructions require the use of a

given hardware resource at the same time.

Eg - • Common bus
• Multiple instruction fetch with one ALU.
• Register file not having enough read/write ports.
• Data and instructions are in the same cache.

(ii) Data Hazard

It is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline.

Eg - I_1: Add R_1, R_2, R_3 $\quad R_1 \leftarrow R_2 + R_3$
I_2 : MUL R_5, R_1, R_6 $\quad R_5 \leftarrow R_1 * R_6$

(iii) Control Hazard / Instruction Hazard

The pipeline may also be stalled because of a delay in the availability of instructions. It arises from the pipelining of branches and other instructions that changes the PC.

Eg - Cache miss

Performance of Pipeline with Stalls

$$\text{Speedup from pipeline} = \frac{\text{Average Instruction Time unpipelined}}{\text{Average Instruction Time pipelined}}$$

$$= \frac{\text{CPI unpipeline} \times \text{Clock Cycle Time unpipelined}}{\text{CPI pipelined} \times \text{Clock Cycle Time pipelined}}$$

$$= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock Cycle Time unpipelined}}{\text{Clock Cycle Time pipelined}}$$

wkt, CPI pipelined = Ideal CPI + Pipeline Stall Clock cycles per instruction.

Assuming Ideal CPI = 1

CPI pipelined = 1 + Pipeline Stall Clock Cycles per instruction

$$\text{Speedup from pipelining} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall Clock cycles per instruction}}$$

$$= \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall Clock cycles per instruction}}$$

$$\Rightarrow \boxed{\text{Speedup from pipelining} = \frac{\text{No. of stages in the pipeline}}{1 + \text{Pipeline stall Clock cycles per instruction}}}$$

Minimizing Structural Hazard

Structural hazards can be minimised by introducing additional resources.

Eg - • Introducing multi bus in place of common single bus.
• Adding multiple number of ALUs for multiple instruction fetch.
• Introducing multiple no. of R/w ports on register file
• Keeping data and instruction in separate cache.

Data Hazard Minimisation Techniques

1) Register write occurs in the 1st half of clock cycle and register read occurs in the 2nd half of clock cycle.

I1 : Add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

I2 : Mul R_5, R_1, R_6 $R_5 \leftarrow R_1 * R_6$

2) Data Forwarding / Result Forwarding / Operand Forwarding :

↳ Data are available at the output of the ALU once the execute stage completes.

↳ Hence the delay can be reduced or possibly eliminated if we arrange the result of instruction I_1 to be forwarded directly for use for Execution stage of I_2.

Data Hazards Requiring Stall :-

Unfortunately all potential Data Hazards can be handled by bypassing.

Eg- I1 : LD R_1, 0(R_2) $R_1 \leftarrow 0(R_2)$

I2 : D , $R_4, R_2 R_5$ $R_4 \leftarrow R_1 - R_5$

I3 : AND R_6, R_1, R_7 $R_6 \leftarrow R_1 \wedge R_7$

I4 : OR R_5, R_1, R_7 $R_5 \leftarrow R_1 \vee R_6$

So, forwarding path cannot operate backward in time.

3) Handling Data Hazard through Software :-

↳ We assume that data dependency is discovered by the hardware while the instruction is being decoded.

↳ The compiler can introduce some delay cycles needed between the dependent instruction by inserting NOP instruction.

↳ Being aware of the need for a delay, the compiler can attempt to reorder the instruction to perform useful task in NOP stalls and thus achieve better performance.

Eg- I1 : Add R_1, R_2, R_2

I2 : Stall Nop

I3 : Stall Nop

I4 : Stall Nop

I5 : Mul R_5, R_1, R_6

Limitation :- - Code size increases

↳ The Nop instructions inserted to satisfy the requirement of one implementation maynot be needed and hence lead to reduce performance on different implementation.

15/03/2023

Branch Hazard / Control Hazard / Instruction Hazard

↳ Branch instruction may cause pipeline to stall.

↳ The time lost as a result of branch instruction is often referred to as Branch Penalty.

↳ For a larger pipeline, the branch penalty may be higher.

Basic Mechanism for Branch Penalty -

I) The simplest scheme to handle branches is to freeze or flush the pipeline holding or deleting any instruction after the branch until the branch destination is known.

II) A higher performance but slightly more complex scheme is to treat every branch as not taken simply allowing the hardware to continue as if the branch were not executed

III) An alternative approach, to treat every branch as taken. Not so effective.

Control Hazard Minimization Techniques :-

1. Delayed Branch :-

The location following a branch instruction is called a branch delay slot.

↳ There might be more than one branch delay slot depending on the time it takes to execute a branch instruction.

↳ The instructions in the delay slots are always executed whether the branch is taken or not taken.

↳ The objective is to ~~fill~~ place useful instruction in this slot.

↳ If no useful instructions are found, these slots may be filled with NOP-instructions.

Eg- I_1 if (a == 5) {
I_2 ——
I_3 ——
I_4 }] Branch delay slot
I_5 else {
I_6 ——
I_7 ——
:
}

↳ 4 stage pipeline | F/D | Ex | Mem | WB |

Eg- LOOP : Shift Left R1
Decrement R2
Branch ≠ 0 LOOP

ADD : R5, R6 —— Branch delay slot (Instruct^n following the 'Branch' instruction)

↳ Branching takes place one instruction later than where the Branch instruction appears in the instruction sequence in the memory. Hence the name delayed branch.

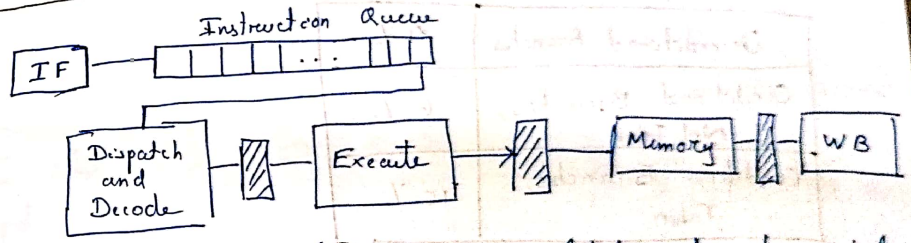
↳ Effectiveness depends on how often it is possible to re-order the instructions.

↳ More the depth of pipeline, more the branch delay slot.

↳ The compiler has to search for instruction to fill the branch delay slot.

25/03/2024

2. Instruction Queing and Prefetching :-



↳ A sophisticated fetch unit can fetch instructions before they are needed and put them in a queue.

↳ A separate unit takes instruction from the front of the queue and sends them to execution unit. It also does decoding.

↳ When hazard occurs, the fetch unit busy in fetching instructions and stores them in a queue and similarly when there is delay in fetching instruction, it continues to issue instruction from the queue.

↳ The instruction fetch unit executed the branch instruction concurrently with the execution with the execution of other instruction.

↳ The technique is also referred to as Branch Folding.

↳ Branch folding occurs if the time at which branch instruction is encountered at least one instruction must be available in the queue other than the branch instruction.

↳ Adequate supply of instruction for processing is required to make the queue full.

Q) Find the effective addition to the CPI arising from branching for this pipeline assuming the following f

Unconditional Branches	4 %
Conditional Branches, Not Taken	6 %
Condition Branches, Taken	10 %

Branch Scheme	Penalty unconditon	Penalty NT	T
Flush Pipelin	2	3	3
Predicted, T	2	3	2
Predicted, T	2	0	3

Ans-

Branch scheme	Addition to the CPI from Branch Cost: Conditional Branch	Conditional Branch NT	Conditional Branches T	All Branches
Frequency of event	4 %	6 %	10 %	20 %
Stall Pipeline	0.04 × 2 = 0.08	0.06 × 3 = 0.18	0.1 × 3 = 0.3	0.56 %
Pipeline T	0.04 × 2 = 0.08	0.06 × 3 = 0.16	0.1 × 2 = 0.2	0.46
Pipeline NT	0.04 × 2 = 0.08	0.06 × 0 = 0	0.1 × 3 = 0.3	0.38

The ideal pipeline would be 1.56 times faster than a pipeline that uses the stall pipeline.

The predicted untaken scheme would be 1.18 times better than the stall pipeline scheme under the same assumption.

28/3/2024

Instruction Level Parallelism (ILP)

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

↳ It is a potential overlap among instructions.

ILP has 2 components — Static (Itanium processor)
Dynamic (Pentium processor)

Eg-

Eg- I_1 : $e = a + b$ ILP = $\frac{3}{2}$

I_2 : $f = c - d$

I_3 : $m = e / f$

↳ More the ILP, the better.

To obtain better ILP :-

Out of Order Execution :-

Instructions execute in any order without violating the data dependency.

↳ It can be used to exploit the ILP.

↳ There are 3 different types of dependency that may arise because of out-of-order execution :-

(i) Data dependency
(ii) Name dependency
(iii) Control dependency

(i) Data Dependence :-

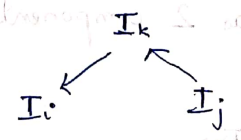
An instruction-j is data dependent on instruction-i if either of the following holds :-

↳ Instruction-i produces a result that is used by instruction-j

↳ Instruction-j is data dependent on instruction-k and instruction is data dependent on instruction-i.

Eg- I_i : $R_1 \leftarrow R_2 + R_3$

I_j : $R_5 \leftarrow R_1 + R_5$



Possible cases for out of Order Execution :-

4 possible cases :-

I) **Read after Read (RAR)** : Instruction-j tries to read a data item before I_i reads it.

Eg- I_i : Add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

I_j : Mul R_5, R_2, R_8 $R_5 \leftarrow R_2 * R_8$

} Allows out-of order execution

II) **Read After Write (RAW)** : I_j tries to read a source before it is written by I_i.

Eg- I_i : Add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

I_j : Mul R_5, R_1, R_6 $R_5 \leftarrow R_1 * R_6$

} Can't be out-of order execution

III) **Write After Read (WAR)** : I_j tries to write a destination before it is read by I_i.

Eg- I_i : Add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

I_j : Mul R_2, R_5, R_6 $R_2 \leftarrow R_5 * R_6$

} Doesn't allow out-of order

IV) **Write After Write (WAW)** : I_j tries to write an operand before it is written by I_i.

Eg- I_i : Add R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$

I_j : Mul R_1, R_5, R_6 $R_1 \leftarrow R_5 * R_6$

} Doesn't allow out-of-order execution

WAW hazards are present only in pipelines that writes in more than one pipeline stages or allow an instruction to proceed even if the previous instruction is stalled.

(ii) Name Dependencies :-

Occurs when two instructions use the same register or memory location called 'name', but there is no flow of data between the instructions associated with that name.

Name Dependence

- WAR (Anti-dependency)
- WAW (Output-dependency)

↳ The name dependency can be resolved by using ~~Register Ren~~ Register-Renaming Techniques.

↳ This can be done statically by the compiler or dynamically through hardware.

RAR
RAW ——— True dependency as it allows out-of-order
WAR dependency ~~as~~ while other can be
WAW resolved.

29/03/2024

(iii) Control Dependency :-

It determines the ordering of an instruction-i wrt a branch instruction so that the instruction-i is executed in correct program order and only when it should be.

Eg- if (P1)
{ S1
}
if (P2) { S2

Here S1 is control dependent on P1
S2 is control dependent on P2
S2 is not control dependent on P1
S1 is not control dependent on P2

↳ The control dependencies must be preserved to preserve the program order.

↳ There are 2 constraints imposed on ~~in~~ control dependences :-

(i) An instruction that is control dependent on a branch can't be moved before the branch so that its execution is no longer controlled by the branch.

(ii) An instruction that is not control dependent on a branch can't be moved after the branch so that its execution is controlled ~~after~~ by the branch.

Branch ~~Predicte~~

MODULE - III

Branch Prediction

It is a technique to reduce branch penalty for conditional branches.

It predicts whether a particular branch will be taken or not.

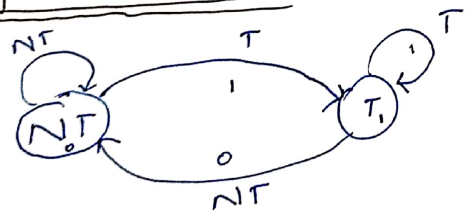
Speculative Execution : It means that instructions are executed before the processor is certain that they are in the correct execution sequence.

Care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed.

Static Branch Prediction : The prediction decission is always same everytime a given instruction is executed.

Dynamic Branch Prediction : The approach in which the prediction decission may change depending on execution history.

Single bit Prediction



Q) Code snippet

```
if (d == 0)
    d = 1;
if (d == 1)
```

Inputs are given sequentially 0, 1, 2,

Ans- Let initial assumption of b_1 & b_2 = NT

d == ?	b_1 prediction	b_1 action	New b_1 prediction	b_2 prediction	b_2 action	New b_2 prediction
0	NT	NT ✓	NT	NT	NT ✓	NT
1	NT	T ✗	T	NT	NT ✓	NT
2	T	T ✓	T	NT	T ✗	T

So, No. of correct preediction = 4
No. of miss preediction = 2

Q) Code Snipet

```
if (d == 0)
    d = 1;
if (d == 1)
```

Inputs are given sequentially 2, 2, 0 0

Ans- Assume b_1 and b_2 as NT

d == ?	b_1 prediction	b_1 action	New b_1 prediction	b_2 prediction	b_2 action	New b_2 prediction
2	NT	T	T	NT	T	T
2	T	T	T	T	T	T
0	T	NT	NT	T	NT	NT
0	NT	NT	NT	NT	NT	NT

No. of correct prediction = 4
No. of incorrect prediction = 4

Q) Inputs given sequentially 2, 0, 2, 0.
Ans- Assume b_1 and b_2 as NT

d == ?	b_1 prediction	b_1 action	New b_1 prediction	b_2 prediction	b_2 action	New b_2 prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

No. of correct prediction = 0

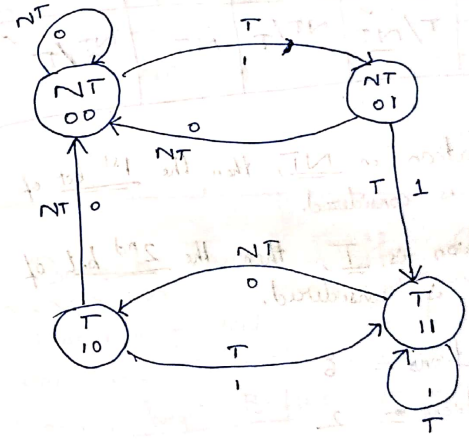
↳ The execution history used in predicting the out come of a given branch instruction is the result of most recent execution of that instruction.

↳ The processor assumes that the next time the instruction is to be executed, the result is likely to be same.

↳ Better performance can be achieved by keeping more information about execution history.

Two-bit Prediction :-

Here the bits are called correlated bits.



State transition diagram for 2-bit predictions.

Q) Code snippet :-

```
if (d == 0)
    d = 1;
if (d == 1)
```

Following inputs are given sequentially :-
2, 0, 2 0

Find the no. of correct prediction and no. of miss prediction using 2 bit branch prediction techniques.

Ans-

d == ?	b_1 prediction	b_2 Action	New b_1 prediction	b_2 prediction	b_2 Action	New b_2 prediction
2	NT/NT	T	T/NT ✗	NT/NT	T	NT/T ✗
0	T/NT	NT	T/NT ✓	NT/T	NT	NT/T ✓
2	T/NT	T	T/NT ✓	NT/T	T	NT/T ✓
0	T/NT	NT	T/NT ✓	NT/T	NT	NT/T ✓

Note :-

1> If the previous prediction is NT, then the 1st bit of the correct prediction is considered.

2> If the previous prediction is T, then the 2nd bit of the correct prediction is considered.

No. of correct predictions = 6

No. of miss predictions = 2

Q> Input 0, 1, 2, 0, 1, 2

Ans- No. of correct prediction = 8

No. of miss prediction = 9

d == ?	b_1 prediction	b_2 action	New b_1 prediction	b_2 prediction	b_2 action	New b_2 action
0	NT/NT	NT	NT/NT	NT/NT	NT	NT/NT
1	NT/NT	T	T/NT	NT/NT	NT	NT/NT
2	T/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T
1	T/NT	T	T/NT	NT/T	NT	NT/T
2	T/NT	T	T/T	NT/T	T	NT/T

Branch Target Buffer (BTB)

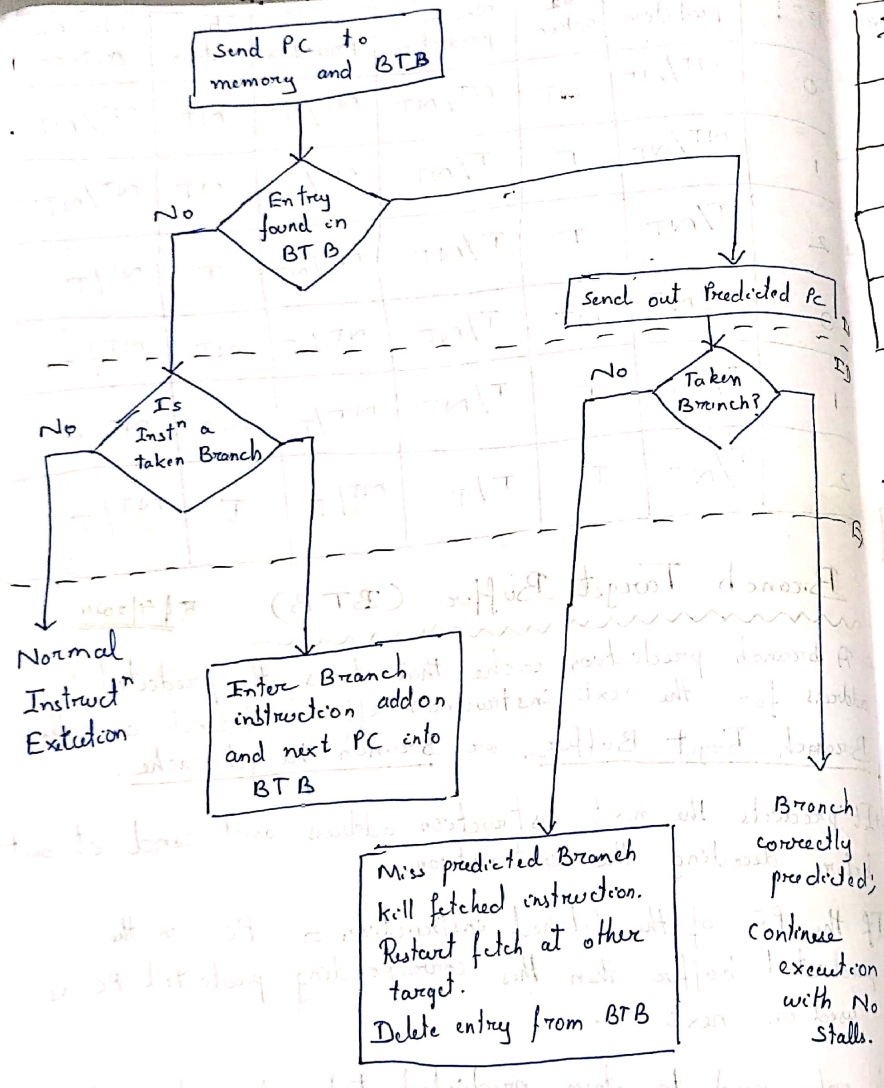
5/4/2024

A branch prediction cache that stores the predicted address for the next instruction after a branch is called Branch Target Buffer. or Branch target cache.

↳ It predicts the next instruction address and send it out before decoding the instruction.

↳ If the PC of the fetched instruction = PC in the predicted buffer then this corresponding predicted PC is used on next PC.

↳ We only need to store predicted taken branch instruction.



Instructions in Buffer	Prediction	Actual Branch	Penalty cycle
Yes	Taken	Taken	0
Yes	Taken	Not Taken	2
No		Taken	2
No		Not Taken	0

Loop Unrolling

It is a technique to reduce the branch penalty.

Eg- X : i = 1
if (i <= 100) {
A[i] = B[i] + C[i]
i++;
goto X
}

continuous branch inst^n occurs

```
for (i=1; i<=100; i++)
  A[i] = B[i] + C[i];
```

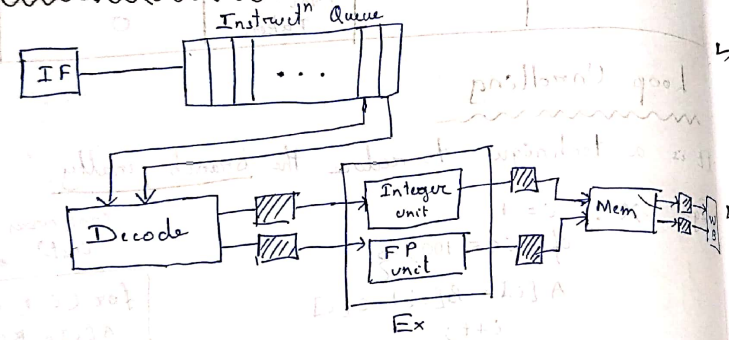
```
for (i=1; i<=100; i=i+4)
{
A[i] = B[i] + C[i];
A[i+1] = B[i+1] + C[i+1]
A[i+2] = B[i+2] + C[i+2];
A[i+3] = B[i+3] + C[i+3];
}
```

↳ It transfers the loop with N- iterations into a loop with N/M iteration where each of the iteration in the new loop does the work of M iteration of the ~~loop~~ old loop.

↳ It increases the number of ~~distract~~ instruction between branches giving the compiler and hardware more opportunity to find the instruction level parallelism.

↳ There will be reduction in the loop overhead.

Super Scalar Operation



cl	1	2	3	4	5	6	7	8	9
I_1	F_1	D_1	E_1	M_1	W_1				
I_1	F_1	D_1	E_1	E_1	E_1	M_2	W_1		
I_2		F_2	D_2	F_2	M_2	W_2			
I_2		F_2	D_2	E_2	E_2	E_2	M_2	W_2	

cl	1	2	3	4	5	6	7	8	9
I_1	F_1	D_1	E_1	M_1	W_1	-	-	W_1	

↳ An instruction may cause exception.

↳ Let I_2 depends on I_1, if an exception occurs in I_2, then I_1 will be inconsistent state.

↳ However one or more succeeding instruction have been ~~expected~~ executed to completion.

↳ If such a situation is permitted, the precision is set to have imprecise exception.

↳ If an exception occurs during an instruction, all subsequent instruction that have been partially executed are discarded. This is called precise exception.

↳ The instruction may complete execution out of order but they must in the Program order.

10/4/2023

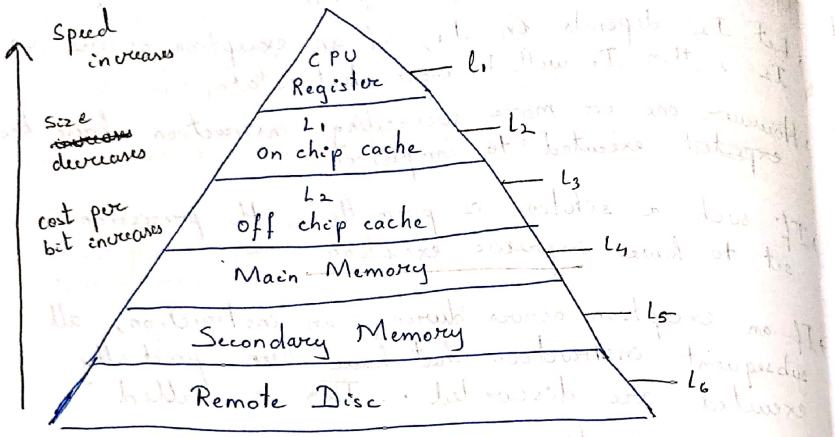
Memory Systems

Design Objectives :-
- Speed (↑) → Cache memory
- Size (↑) → Virtual memory tech.
- Cost (↓)

Memory Hierarchy

~~Cache Memory~~ Cache Memory → Static RAM

Main Memory → Dynamic RAM

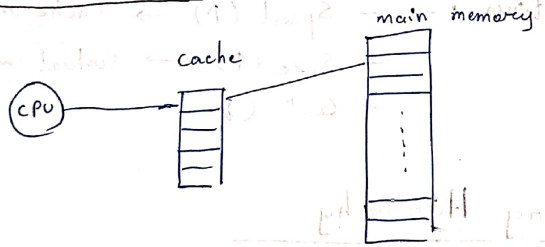


$L_1 \subseteq L_2 \subseteq L_3 \subseteq L_4 \cdots L_n$ —— inclusion property

If the data is not found in L_k —— it is not present in $L_{k-1}, L_{k-2}, \cdots L_1$

If the data is found in L_k —— it is also found in $L_{k+1}, L_{k+2}, \cdots L_n$

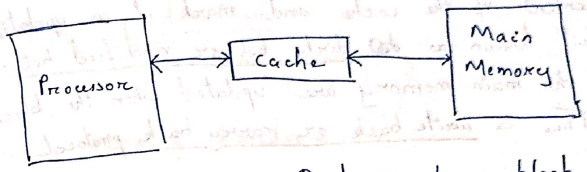
Cache Memory :—



LOR (Locality of Reference)

- Temporal LOR
- Spatial LOR

18/4/2024



↳ Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.

↳ Subsequent references to the data in this block of words are found in the cache.

↳ At any given time, only some blocks of the main memory are held in the cache which blocks in the main memory are in the cache are determined by mapping function.

↳ When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced by. This is determined by a replacement algorithm.

Existence of a cache is transparent to the processor. The processor issues Read and Write request in the same manner.

- Read
  - Cache hit
  - Cache miss
- Write
  - Cache hit
  - Cache miss

Read hit : The data is obtained from the cache.

Write hit :—

↳ Cache has a replica of contents of main memory.

↳ Contents of the cache and the main memory may be updated simultaneously called Write Through Protocol

↳ Update the contents of the cache and mark it as updated by setting a bit known as dirty bit or modified bit.

↳ The contents of the main memory are updated when the block is replaced. This is write back or copy back protocol.

Read Miss :—

↳ Block of words containing the requested word is transferred from the memory.

↳ After the block is transferred, the desired word is forwarded to the processor

↳ The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called early restart or load through.

Write Miss :—

↳ Write through protocol is used, then the contents of the main memory are updated directly.

↳ If Write-back protocol is used, the block containing the adressed word

Cache Coherence

If the data is present in main memory and the cache memory, then the data is coherent.

Mapping Functions :—

Determine how ~~many~~ memory blocks are placed in the cache.

Three mapping functions :—

(i) Direct mapping
(ii) Association mapping
(iii) set - associative mapping

Block size of main memory = Block size of cache

Q) A simple processor example :—

Ans-

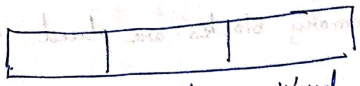
	C/M	M/M
Size	$2^7 \times 2^4 = 2^{11}$	2^{16}
No. of Block	27	$2^{16}/2^4 = 2^{12}$
Block size	2^4	2^4

(i) Direct Mapping : J^{th} block of main memory is mapped into **j mod (no. of blocks in cache memory).**

Eg-

M/M	C/M
0	0
1	1
2	2
⋮	⋮
127	127
128	0
129	1
⋮	⋮
256	0
257	1
⋮	⋮

C/M	M/M
0	0, 128, 256, 512, ...
1	1, 129, 257, ...
2	2, 130, 258, ...

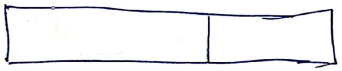


$$\text{Tag} = \frac{\text{No. of blocks in main memory}}{\text{No. of blocks in cache memory}}$$

Drawbacks :- One block of cache memory is multiple blocks of the main memory.

* Contention Problem

(ii) Associative Mapping : Any block of the main memory is mapped to any block of the cache memory.

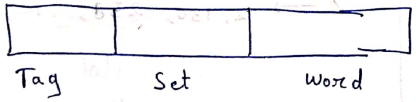


Word → No. of words in a block

↳ No. of blocks in Main Memory

Drawback :- Searching Problem.

(iii) 2-way set associative mapping :



$$\text{Set} = \frac{\text{No. of blocks in cache memory}}{\text{set size}}$$

Hit rate :-

$$\text{Hit rate} = \frac{\text{No. of hits}}{\text{Total no. of attempts}}$$

Either read hit or write hit.

$$\text{Miss rate} = \frac{\text{No. of miss}}{\text{Total no. of attempts}}$$

$$\text{Hit rate} + \text{Miss rate} = 1$$

Miss penalty : Extra time required to bring the required block from the main memory.

For single level cache :-

$$\text{Avg Access time} = hc + (1-h)M$$

where $h \longrightarrow$ hit rate

$C \longrightarrow$ cache memory access time

$M \longrightarrow$ Miss penalty

↳ Hit rate can be improved by increasing block size, while keeping cache size constant.

↳ Block sizes that are neither very small nor very large give best results.

↳ Miss penalty can be reduced if load-through approach is used when loading new blocks into cache.

For two-level cache :-

Avg. access Time, T_{av} :-

$$T_{av} = h_1 c_1 + (1-h_1) h_2 c_2 + (1-h_1)(1-h_2) M$$

where, h_1 ⟶ hit rate in L_1 cache
h_2 ⟶ hit rate in L_2 cache
c_1 ⟶ L_1 cache memory access time
c_2 ⟶ L_2 cache memory access time
M ⟶ Miss penalty.

How to reduce the Miss Penalty :-

1) Using Multilevel Cache
2) Selecting correct block size
3) Critical word first
4) Early restart
5) Giving priority to Read Miss over write miss
6) Victim cache.

Virtual Memory

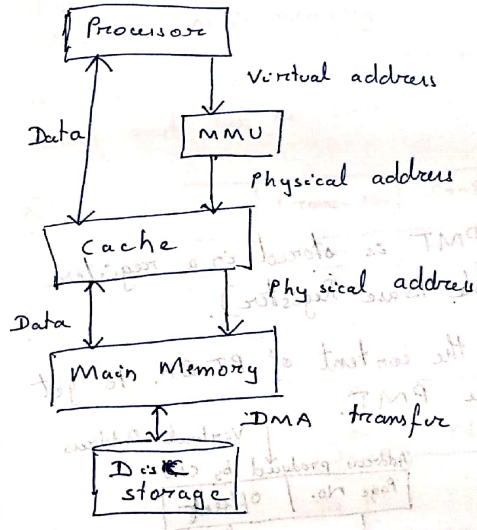
↳ The physical main memory is not as large as the address space spanned by an issued by the processor.

↳ When a program doesn't fit into the main memory, the parts of it not currently being executed are stored in secondary storage device

↳ The technique that automatically move the data blocks into physical main memory when they are required for execution is called virtual memory ~~techniq~~ technique.

↳ The processor references an instruction and data space that is independent of available physical memory.

↳ The address generated by the processor is known as virtual address which is converted to physical address through memory management unit.



Address Translation :-

↳ The program is divided into fixed sized blocks called pages and the main memory is divided into fixed size blocks called frames.

The page size = Frame size

↳ No. of pages may not be equal to no. of frames.

↳ The address generated by the processor has 2 parts :-

i) page number
ii) offset

Page No.	Offsets

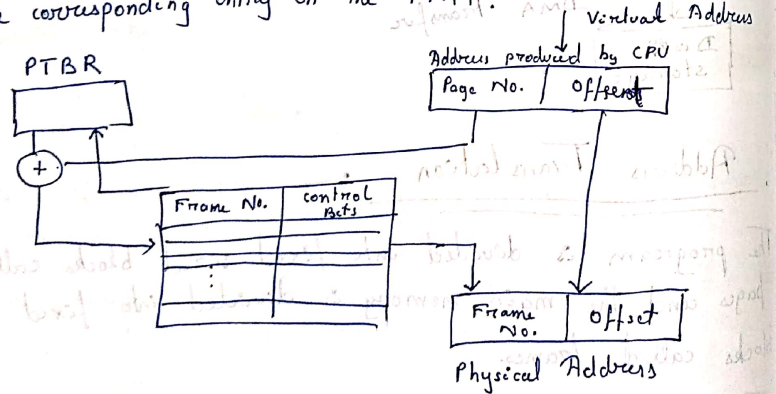
↳ There is a table called page map table which keeps track of the corresponding frame number to a corresponding page.

↳ There are 2 entries in PMT :- 1) Frame no.
2) Control bits

Frame No.	Control Bits
⋮	

↳ The starting address of the PMT is stored in a register called PTBR (Page Table Base Register).

↳ The page no. is added with the content of PTBR to get the corresponding entry in the PMT.



Control Bits : These are used to give restriction or permission to different pages.

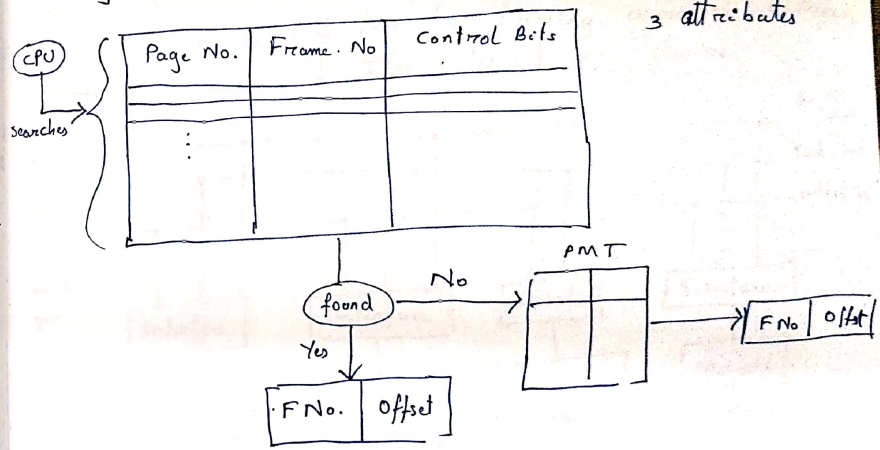
Eg- Read, Write, Execute, Modify, etc

TLB (Translation Look-Aside Buffer)

↳ Ideally the PMT should be kept in MMU which is a part of CPU

However CPU has limited no. of registers for which it is stored in main memory.

However a part of the PMT which is very frequently being used can be stored in a buffer called TLB.



Page Fault : If the requested page is not found in the main memory, a page fault is said to occur

Demand Paging : A page will not be loaded to the main memory unless and until it is being demanded.

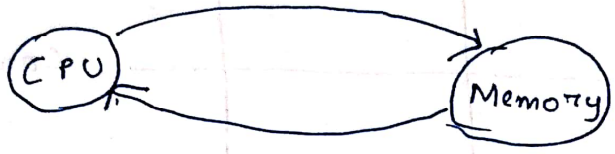
Pre-Paging : The pages will be loaded to the main memory even if it is not being demanded.

Thrashing : A high page fault may result in thrashing. The processor will be busy in swap-in and swap-out operation rather than actual execution.

MODULE - V

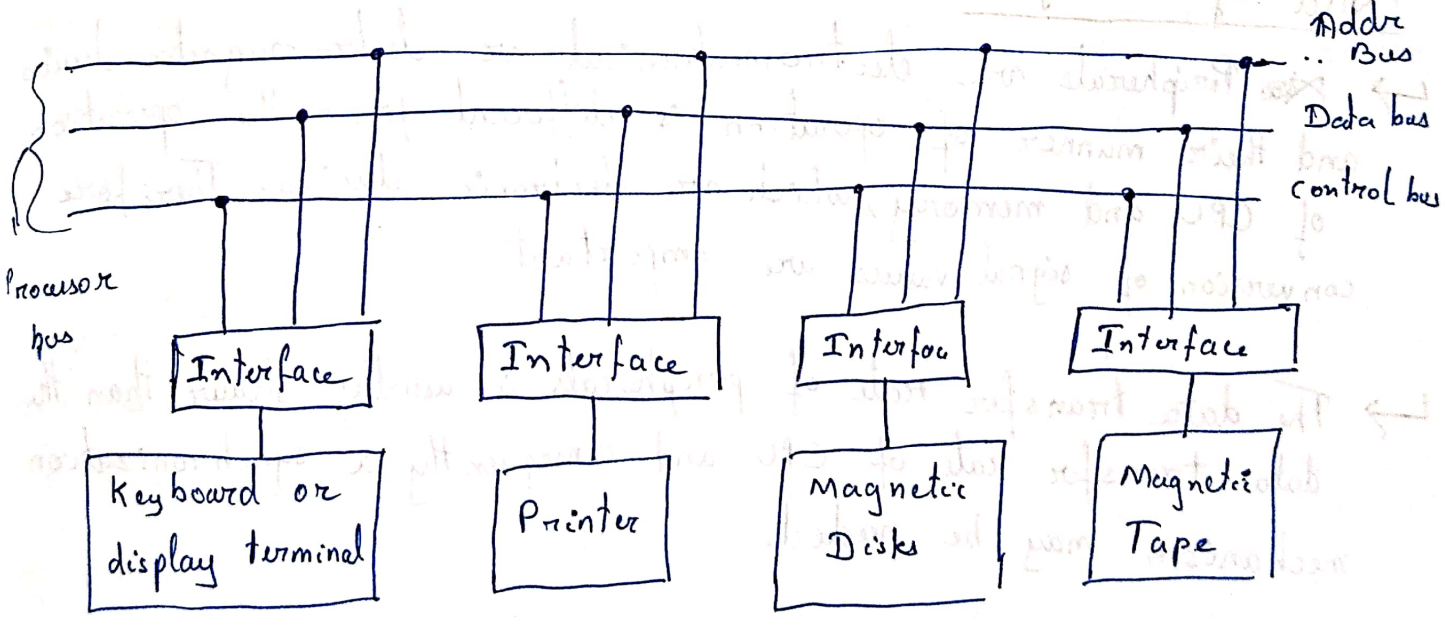
I/O

↳



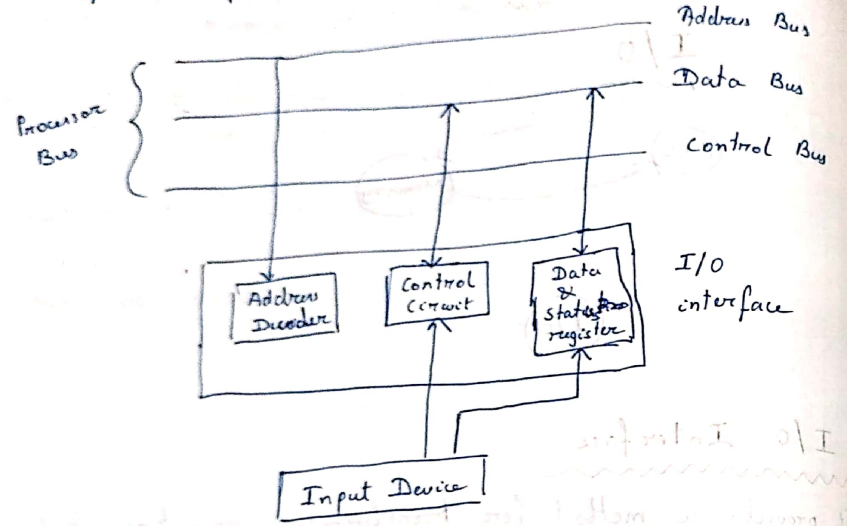
I/O Interface

It provides a method for transferring operation between internal storage and external I/O devices



Communication of I/P - O/P devices

I/O interface for an Input device :-



Need of interface :-

↳ Peripherals are electromechanical or electromagnetic devices and their manner of operation is different from the operation of CPU and memory, which are electronic devices. Therefore conversion of signal values are important

↳ The data transfer rate of peripherals is usually slower than the data transfer rate of CPU and consequently a synchronization mechanism may be needed.

↳ Data codes and formats in peripherals differ from the word format from CPU and memory.

↳ The operating methods of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

Address Decoder : Enables the device to recognise its address when this address appears in the address bus.

Data & Status Register : The data register holds the data being transferred to or from the processor.

↳ Status register contains the information relevant to the operation of I/O devices.

↳ Both data and status register are connected to the data bus and assigned with unique addresses.

Control Circuits : The control signals from the processor can be communicated to the I/O device through the control circuits.

Interface Circuit : Address decoder, data & status register, control circuits are required to coordinate that constitutes the I/O interface.

Interface Modes :-

(i) Memory Mapped I/O
(ii) I/O Mapped I/O

(i) Memory Mapped I/O

When the I/O devices and memory share the same address space, the arrangement memory mapped I/O.

Here any machine instruction that can access memory can be used to transfer data from I/O devices.

(ii) I/O mapped I/O

↳ Same processes have special input and output instructions to perform I/O transfer. (Eg - Intel).

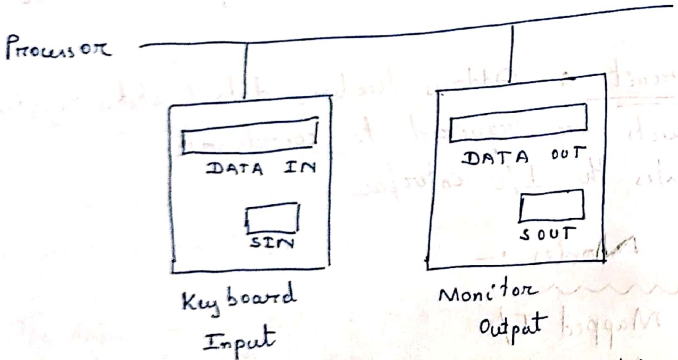
↳ Special I/O address space

↳ Deals with fewer address line.

Data Transfer

3 steps :— (i) Program controlled I/O
(ii) Interrupt Driven I/O
(iii) DMA (Direct Memory Access)

(i) Program Controlled I/O



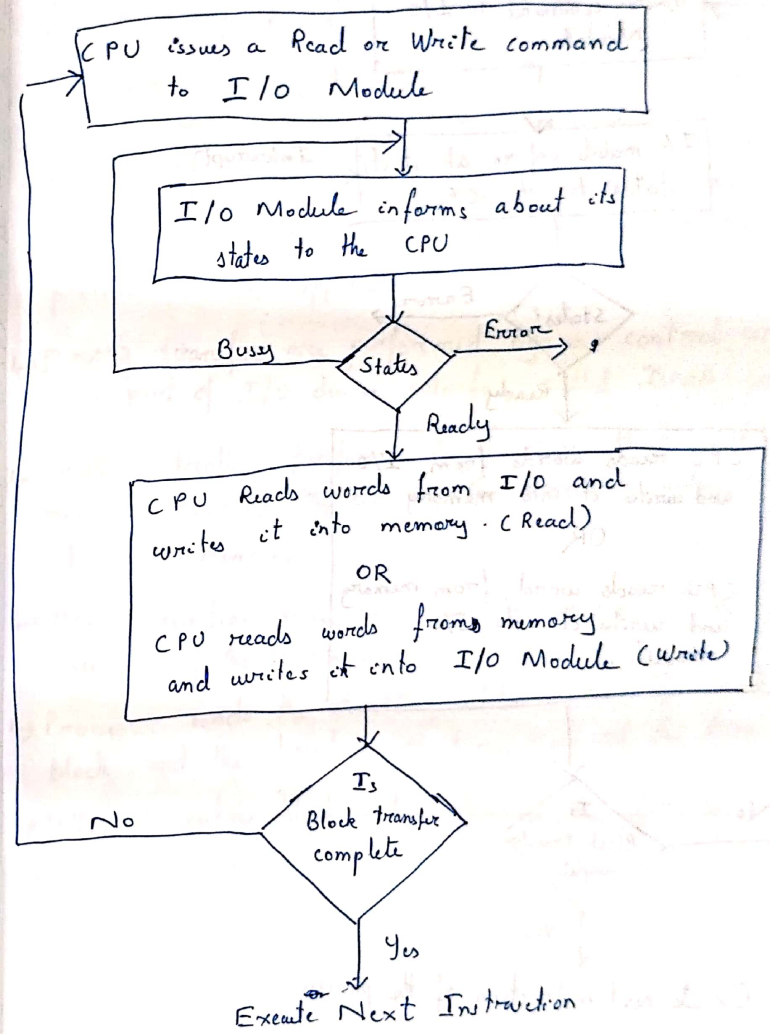
↳ In this scheme, the I/O operations are completely controlled by the CPU

↳ CPU executes the programs that directly initiates and terminates I/O operation

↳ It requires little special I/O hardware but it is quite time consuming for CPU since CPU has to wait for slower I/O operation to complete.

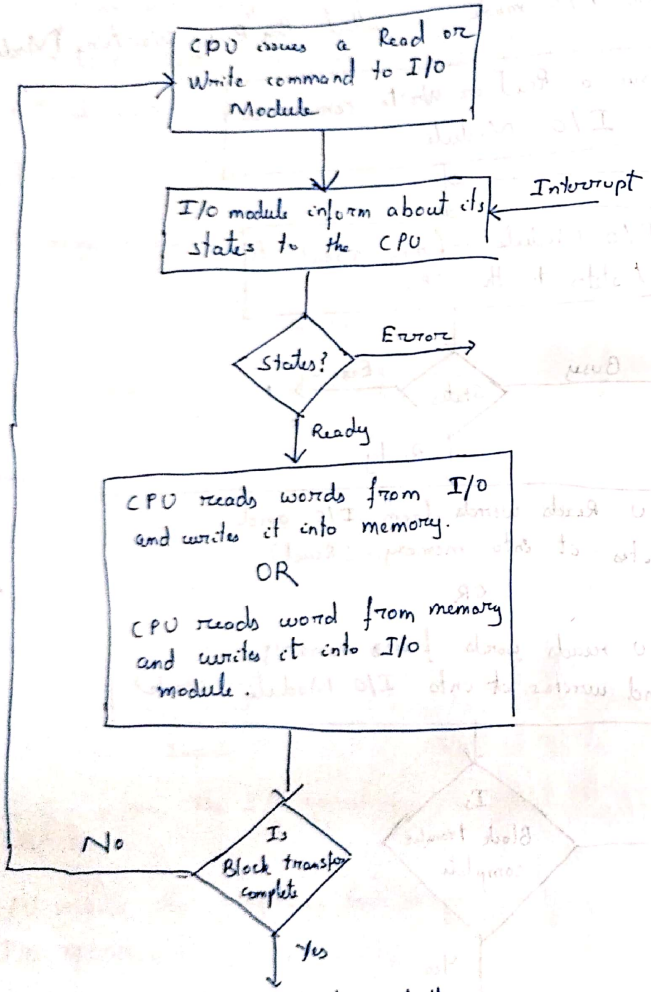
↳ The processor repeatedly checks the status of the flag to achieve the required synchronization between processor and any I/O devices.

↳ The program I/O mode is called "Busy Waiting Mode".



(ii) Interrupt Driven I/O

Interrupt : Hardware signal to which is when raised, the processor stops its normal execution to address the interrupt



↳ The CPU issues I/O command to the I/O module and starts doing other work which may be execution of some other programs.

↳ When the I/O operation completes, I/O module interrupts the CPU by informing to CPU that I/O has finished, the CPU then may proceed execution of the program.

DMA (Direct Memory Access)

↳ A special control unit may be provided to allow transfer of a block of data directly between external device and main memory without continuous intervention of the processor. This approach is called DMA.

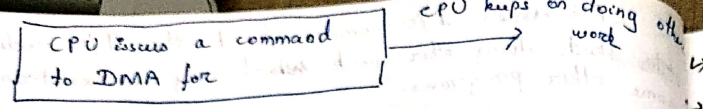
↳ DMA transfer are performed by a control circuit that is a part of I/O device interface called DMA controller.

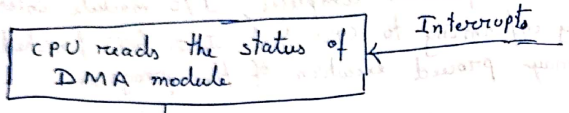
↳ DMA controller performs the function that would be normally carried out by the processor when accessing the main memory.

↳ DMA operation must be under the control of the program executed by the processor.

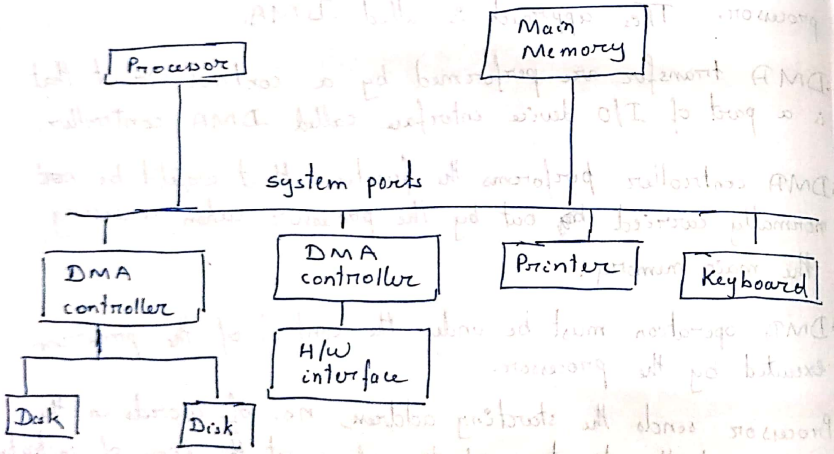
↳ Processor sends the starting address, no. of words in the block and the direction of transfer at the time of initiation.

↳ When the entire block is transferred, the controller informs the processor by raising an interrupt signal.





CPU performs the next instruction



↳ Request by DMA devices for using the bus are always given higher priority than processor

↳ Among different devices of DMA, highest priority is given to high speed peripherals.

↳ Since the processor originates most memory cycles, the DMA controller can be set to steal memory cycles from the processor.

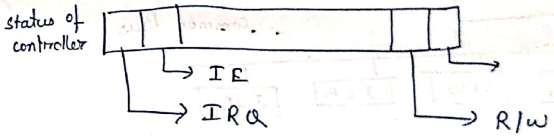
↳ This interleaved interleaving technique is called Cycle Stealing

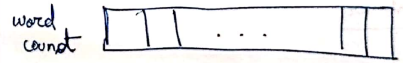
↳ DMA controller may be given exclusive access to the memory to transfer a block of data without interruption known as block / burst mode.

↳ Two registers are used for storing the starting address and the word count.

↳ Third register is used for storing the status and controll part.

Eg- R/W used for direction for transfer
I/E bit interrupt enabled or not
IRQ → used for interrupt request
→ used whether the transfer completes or not





Interconnection Networks

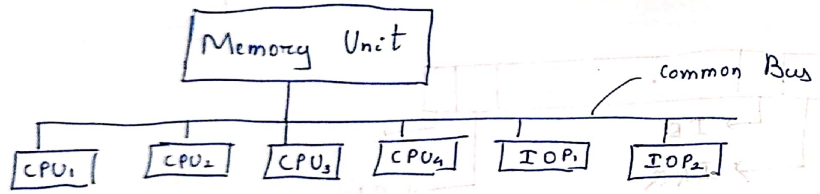
↳ It is a backbone network which facilitates efficient data transmission between processing element or between processing elements and memory module.

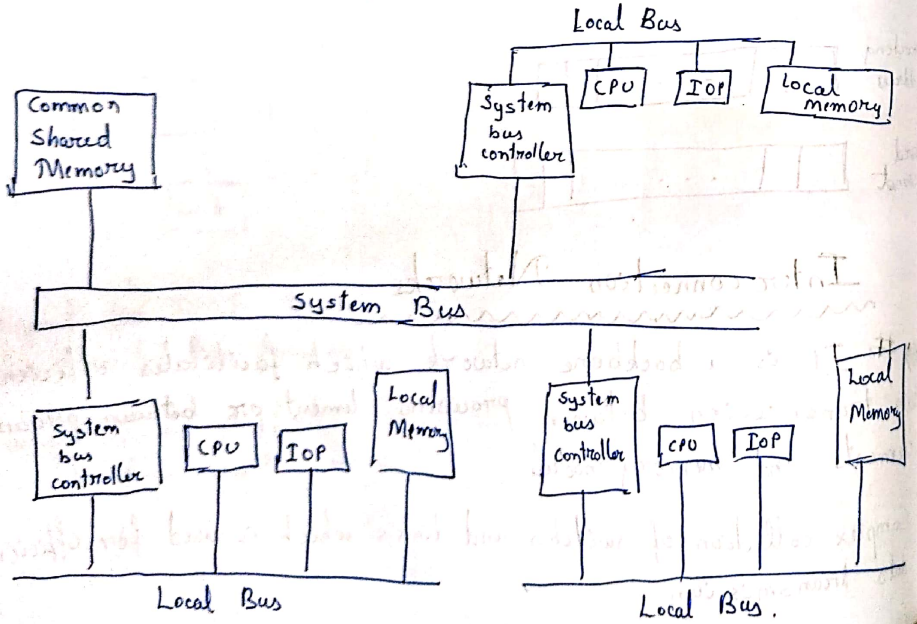
Complex collection of switches and links which is used for efficient data transmission.

There are several different physical form available for the ICN :-

(i) Time Shared Common Bus :

- The simplex ICN for multiple processor is a common communication path connecting all of the FUs.
- The common path is called time shared or common bus.
- Only one processor can communicate with memory or another processor at a given time.
- A command is issued to inform the destination unit about what operation is to be performed.





Here we have a no. of local buses each connected to its own local memory and to one or more processors.

A system bus controller links each local bus to a common system bus.

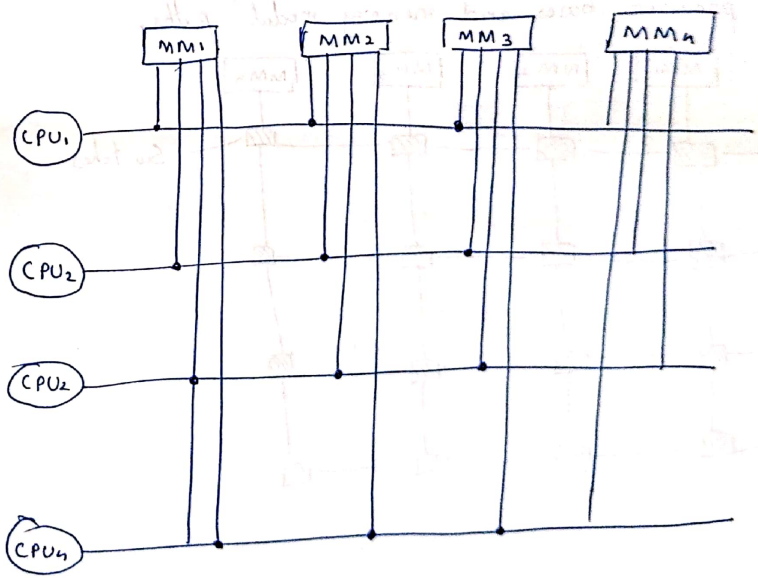
The memory connected to system bus is shared by all processors.

Drawbacks :
- Only one communication at a time
- Problem in system bus may collapse entire communication

Advantages :
- Low cost
- Attaching and detaching system bus is easy

(ii) Multiport Memory

↳ It employs separate buses for each memory module and each CPUs.



↳ Each processor bus is connected to each memory module

↳ The processor bus consist of
- Address
- Data
- Control lines

required to communicate with memory.

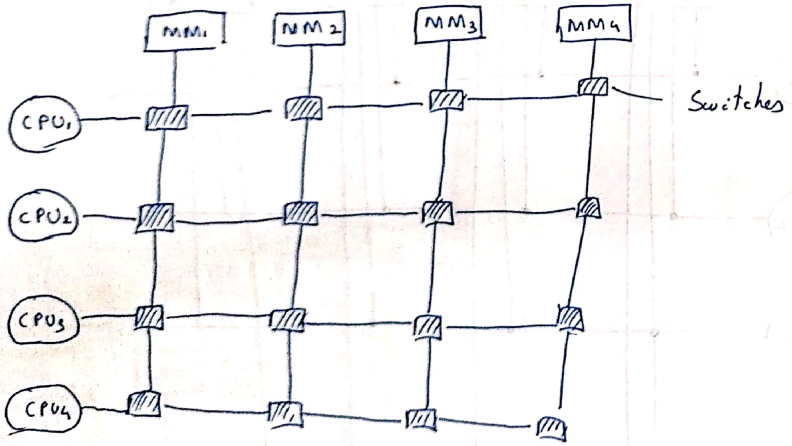
↳ Memory Access conflicts are resolved by assigning fixed priorities to each memory port.

Advantages — High transfer rate that can be achieved bus of multiple paths bet^n processor and memory

Disadvantages — Requires expensive memory control logic and large no. of cables and connectors.

(iii) Crossbar Switch

- Consists of a no. of crosspoints that are placed at intersections between processor buses and memory module paths.



↳ Each cross point is a switch that determines a path from processor to MM.

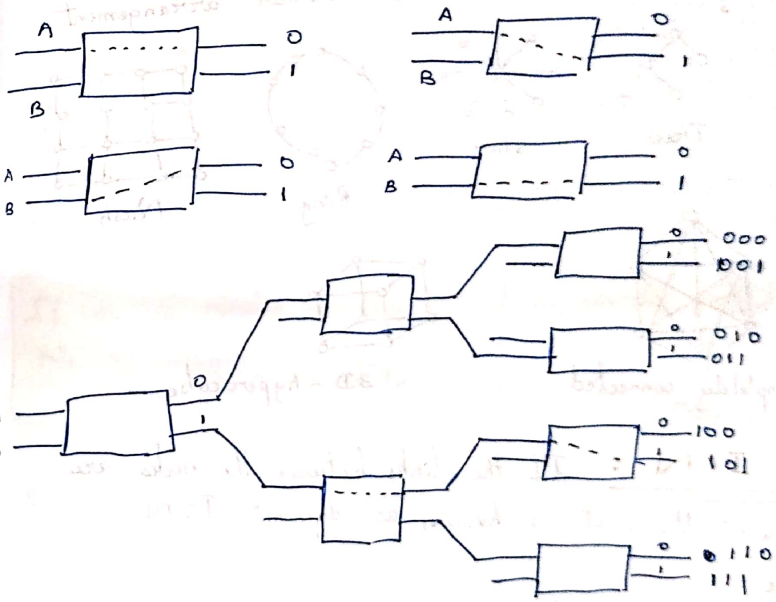
↳ Each switch has a control logic to set up the transfer path between processor and memory.

↳ Priority levels are established by the arbitration logic to select one CPU when two or more CPUs are attempt to access the same memory.

↳ The hardware required to implement the switch can become large and complex.

(iv) Multistep Switching Network

↳ Basic component is 2 input 2 output interchange switch.



↳ 2 processor P_1 and P_2 are connected through switches to 8 memory modules marked

↳ To connect P_1 to memory 101 it is necessary to form a path from P_1 to output 1 in the 1st level switch, 0 in 2nd level and 1 in 3rd level switch

↳ So, either P_1 or P_2 can be connected to any of the 8 memory modules.

(A) Based on Arrangement of Link :-

(i) Static ICN : If the links between the nodes (PE) are fixed, it is called static ICN.

Eg- 1-D → Linear arrangement

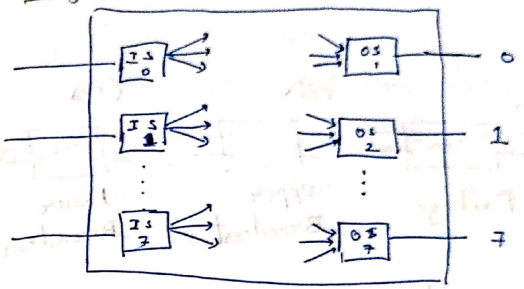
2D → Tree, Star, Ring, Mesh

3D → completely connected, 3D - hypercube

(ii) Dynamic ICN : If the links between the nodes are established dynamically, it is known as dynamic ICN.

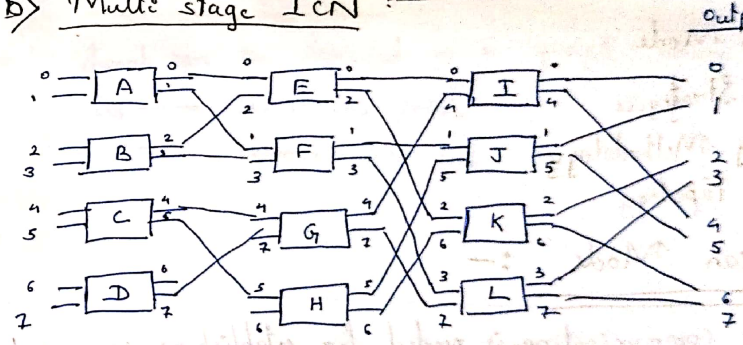
a) Single stage
b) Multi stage

a) Single stage ICN :-



IS (Input selector) : 1 → D DeMultiplexers
OS (Output selector) : D → 1 Multiplexers

b) Multi stage ICN :-

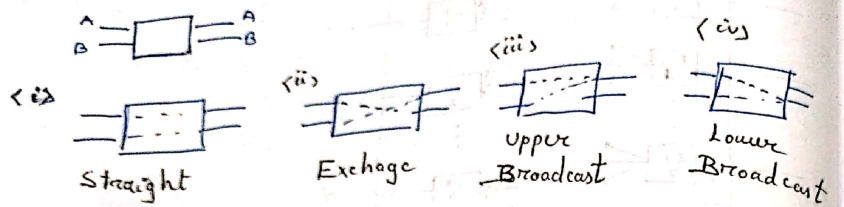


If no. of Nodes = N — usually perfect square
No. of Nodes in a stage = $\log_2 N$

Eg- No. of nodes = 16
No. of Nodes in a stage = $\log_2 4$ = 4

Eg- No. of nodes, N = 8
$\log_n N$ = 3 — No. of stages
$\frac{N}{2}$ = 4 — No. of nodes in each stage

2×2 switches



Inter PE Communication

↳ The network design for inter PE communication is based on :—

(i) Operation Mode
(ii) Control Strategies
(iii) Switching Methodology
(iv) N/W Topology

(i) Operation Mode :—

Synchronous : Communication is needed for establishing communication path synchronously either or for data instruction broadcast.

Asynchronous : Communication is needed for multiprocessing where the connection request are issued dynamically.

Combined : A system may also be designed to facilitate synchronously or asynchronously.

(ii) Control Strategies :—

Centralised Control

Distributed Control —— Used by most ICN

(iii) Switching Methodologies :—

Circuit switching : Physical path is established between source and destination and used fore bulk transfer.

Packet switching : Data is put in a packet and routed through the ICN without establishing a physical path and used fore short data transfer

Integrated : Includes the capability of circuit and packet switching

(iv) Network Topology :—

Network can be depicted in a graph where :—

Nodes ⟶ switching points
Edges ⟶ Communication links

→ Static
→ Dynamic

The Space for ICN = { operation Mode } × { switching Methodologies } × { N/W Topologies }