5th Semester

Advanced Computer Architecture

Objectives

- To understand the advance hardware and software issues of computer architecture
- To understand the multi-processor architecture & connection mechanism
- To understand multi-processor memory management

Module-I: (10 Hours)

Microprocessor and Microcontroller, RISC and CISC architectures, Parallelism, Pipelining fundamentals, Arithmetic and Instruction pipelining, Pipeline Hazards, Superscalar Architecture, Super Pipelined Architecture, VLIW Architecture, SPARC and ARM processors.

Module-II: (10 Hours)

Basic Multiprocessor Architecture: Flynn's Classification, UMA, NUMA, Distributed Memory Architecture, Array Processor, Vector Processors.

Module-III: (10 Hours)

Interconnection Networks: Static Networks, Network Topologies, Dynamic Networks, Cloud computing.

Module IV (10 Hours)

Memory Technology: Cache, Cache memory mapping policies, Cache updating schemes, Virtual memory, Page replacement techniques, I/O subsystems.

Outcomes

- Ability to analyze the abstraction of various advanced architecture of a computer
- Ability to analyze the multi-processor architecture & connection mechanism
- Ability to work out the tradeoffs involved in designing a modern computer system

Books:

[1] John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 6th edition, 2017

[2] Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, McGraw Hill, 5th Ed, 2014

[3] Kai Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, 3rd Ed, 2015

Digital Learning Resources:

Course Name: Advanced Computer Architecture
Course Link: https://nptel.ac.in/courses/106/103/106103206/
Course Instructor: Prof.John Jose, IIT, Guwahati

Course Name: High Performance Computer Architecture
Course Link: https://nptel.ac.in/courses/106/105/106105033/
Course Instructor: Prof.A. Pal, IIT, Kharagpur

Introduction to microprocessor and microcontroller

- A microprocessor is an IC which has only the CPU inside them, i.e. only the processing powers such as Intel's Pentium 1,2,3,4, core 2 duo, i3, i5 etc. These microprocessors don't have RAM, ROM, and other peripherals on the chip. A system designer has to add them externally to make them functional.
- Applications of microprocessor include Desktop PC's, Laptops, notepads etc.
- A microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip. At times it is also termed as a mini computer or a computer on a single chip. Today different manufacturers produce microcontrollers with a wide range of features available in different versions. Some manufacturers are ATMEL, Microchip, TI, Freescale, Philips, Motorola etc.
- Microcontrollers are designed to perform specific tasks. Specific means applications where the relationship of input and output is defined. Depending on the input, some processing needs to be done and output is delivered.
- For example, keyboards, mouse, washing machine, digicam, pendrive, remote, microwave, cars, bikes, telephone, mobiles, watches, etc. Since the applications are very specific, they need small resources like RAM, ROM, I/O ports etc and hence can be embedded on a single chip. This in turn reduces the size and the cost.
- Microprocessors find applications where tasks are unspecific like developing software, games, websites, photo editing, creating documents etc. In such cases the relationship between input and output is not defined. They need high amount of resources like RAM, ROM, I/O ports etc.
- The clock speed of the Microprocessor is quite high as compared to the microcontroller. Whereas the microcontrollers operate from a few MHz to 30 to 50 MHz, today's microprocessors operate above 1GHz as they perform complex tasks.
- Generally, a microcontroller is far cheaper than a microprocessor. However, a microcontroller cannot be used in place of microprocessor and using a microprocessor is not advised in place of a microcontroller as it makes the application quite costly.
- A microprocessor cannot be used stand alone. They need other peripherals like RAM, ROM, buffer, I/O ports etc and hence a system designed around a microprocessor is quite costly.

Evolution of Microprocessors

- Transistor was invented in 1948 (23 December 1947 in Bell lab).
- IC was invented in 1958 (Fair Child Semiconductors) By Texas Instruments J kilby.
- First microprocessor was invented by INTEL (INTegrated ELectronics).

Size of microprocessor – 4 bit

Name	Year Of Invention	Clock Speed	Number Of Transistors	Inst. Per Sec
INTEL 4004/4040	1971 by Ted Hoff and Stanley Mazor	740 KHz	2300	60,000

Size of microprocessor – 8 bit

Name	Year Of Invention	Clock Speed	Number Of Transistors	Inst. Per Sec
8008	1972	500 KHz		50,000
8080	1974	2 MHz	60,000	10 times faster than 8008
8085	1976 (16 bit address bus)	3 MHz	6500	769230

Size of microprocessor – 16 bit

Name	Year Of Invention	Clock Speed	Number Of Transistors	Inst. Per Sec
8086	1978 (multiply and divide instruction, 16 bit data bus and 20 bit address bus)	4.77 MHz, 8MHz, 10MHz	29000	2.5 Million
8088	1979 (cheaper version of 8086 and 8 bit external bus)			2.5 Million
80186/ 80188	1982 (80188 cheaper version of 80186, and addtional components like interuppt controller, clock generator, local bus controller,counters)	6 MHz		
80286	1982 (data bus 16bit and address bus 24 bit)	8 MHz	134000	4 Million

Size of microprocessor – 32 bit

Name	Year Of Invention	Clock Speed	Number Of Transistors	Inst. Per Sec
INTEL 80386	1986 (other versions 80386DX, 80386SX, 80386SL and data bus 32 bit address bus 32 bit)	16 MHz – 33 MHz	275000	
INTEL 80486	1986 (other versions 80486DX, 80486SX, 80486DX2, 80486DX4)	16 MHz – 100 MHz	1.2 Million transistors	8 KB of cache memory
PENTIUM	1993	66 MHz		Cache memory 8 bit for instructions 8 bit for data

Size of microprocessor – 64 bit

Name	Year Of Invention	Clock Speed	Number Of Transistors	Inst. Per Sec
INTEL core 2	2006 (other versions core2 duo, core2 quad, core2 extreme)	1.2 GHz to 3 GHz	291 Million transistors	64 KB of L1 cache per core 4 MB of L2 cache
i3, i5, i7	2007, 2009, 2010	2.2GHz – 3.3GHz, 2.4GHz – 3.6GHz, 2.93GHz – 3.33GHz		

Generations of microprocessor:

First generation: From 1971 to 1972 the era of the first generation came which brought microprocessors like INTEL 4004 Rockwell international PPS-4 INTEL 8008 etc.

Second generation: The second generation marked the development of 8 bit microprocessors from 1973 to 1978. Processors like INTEL 8085 Motorola 6800 and 6801 etc came into existence.

Third generation: The third generation brought forward the 16 bit processors like INTEL 8086/80186/80286 Motorola 68000 68010 etc. From 1979 to 1980 this generation used the HMOS technology.

Fourth generation: The fourth generation came into existence from 1981 to 1995. The 32 bit processors using HMOS fabrication came into existence. INTEL 80386 and Mororola 68020 are some of the popular processors of this generation.

Fifth generation: From 1995 till now we are in the fifth generation. 64 bit processors like PENTIUM, celeron, dual, quad and octa core processors came into existence.

Types of microprocessors:

Complex instruction set microprocessor: The processors are designed to minimise the number of instructions per program and ignore the number of cycles per instructions. The compiler is used to translate a high-level language to assembly level language because the length of code is relatively short and an extra RAM is used to store the instructions. These processors can do tasks like downloading, uploading and recalling data from memory. Apart from these tasks these microprocessors can perform complex mathematical calculation in a single command.

Example: IBM 370/168, VAX 11/780

Reduced instruction set microprocessor: These processor are made according to function. They are designed to reduce the execution time by using the simplified instruction set. They can carry out small things in specific commands. These processors complete commands at faster rate. They require only one clock cycle to implement a result at uniform execution time. There are number of registers and less number of transistors. To access the memory location LOAD and STORE instructions are used.

Example: Power PC 601, 604, 615, 620

Super scalar microprocessor: These processors can perform many tasks at a time. They can be used for ALUs and multiplier like array. They have multiple operation unit and perform many tasks, executing multiple commands.

Application specific integrated circuit: These processors are application specific like for personal digital assistant computers. They are designed according to proper specification.

Digital signal multiprocessor: These processors are used to convert signals like analog to digital or digital to analog. The chips of these processors are used in many devices such as RADAR SONAR home theatres etc.

Advantages of microprocessor –

- High processing speed
- Compact size
- Easy maintenance
- Can perform complex mathematics
- Flexible
- Can be improved according to requirement

Disadvantages of microprocessors –

- Overheating occurs due to overuse
- Performance depends on size of data
- Large board size than microcontrollers
- Most microprocessors do not support floating point operations

Pipelining

To improve the performance of a CPU we have two options:
1) Improve the hardware by introducing faster circuits.
2) Arrange the hardware such that more than one operation can be performed at the same time.
Since, there is a limit on the speed of hardware and the cost of faster circuits is quite high, we have to adopt the 2nd option.

Pipelining: Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.

Example: Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(**I**), Filling water in the bottle(**F**), and Sealing the bottle(**S**). Let us consider these stages as stage 1, stage 2 and stage 3 respectively. Let each stage take 1 minute to complete its operation.
Now, in a non-pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Similarly, when the bottle moves to stage 3, both stage 1 and stage 2 are idle. But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3. Hence, the average time taken to manufacture 1 bottle is:

Without pipelining = 9/3 minutes = 3m

I F S | | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)

With pipelining = 5/3 minutes = 1.67m

I F S | |
| I F S |
| | I F S (5 minutes)

Thus, pipelined operation increases the efficiency of a system.

Design of a basic pipeline

In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that output of one stage is connected to input of next stage and each stage performs a specific operation.
Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
All the stages in the pipeline along with the interface registers are controlled by a common clock.

Execution in a pipelined processor:

Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram. For example, consider a processor having 4 stages and let there be 2 instructions to be executed. We can visualize the execution sequence through the following space-time diagrams:

Non overlapped execution:

STAGE / CYCLE	1	2	3	4	5	6	7	8
S1	I_1				I_2			
S2		I_1				I_2		
S3			I_1				I_2	
S4				I_1				I_2

Total time = 8 Cycle

Overlapped execution:

STAGE / CYCLE	1	2	3	4	5
S1	I_1	I_2			
S2		I_1	I_2		
S3			I_1	I_2	
S4				I_1	I_2

Total time = 5 Cycle

Pipeline Stages

RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations:

Stage 1 (Instruction Fetch)

In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

Stage 2 (Instruction Decode)

In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

Stage 3 (Instruction Execute)

In this stage, ALU operations are performed.

Stage 4 (Memory Access)

In this stage, memory operands are read and written from/to the memory that is present in the instruction.

Stage 5 (Write Back)

In this stage, computed/fetched value is written back to the register present in the instructions.

Performance of a pipelined processor

Consider a ‘k’ segment pipeline with clock cycle time as ‘Tp’. Let there be ‘n’ tasks to be completed in the pipelined processor. Now, the first instruction is going to take ‘k’ cycles to come out of the pipeline but the other ‘n – 1’ instructions will take only ‘1’ cycle each, i.e, a total of ‘n – 1’ cycles. So, time taken to execute ‘n’ instructions in a pipelined processor:

$$ET_{pipeline} = k + n - 1 \text{ cycles}$$
$$= (k + n - 1)\, Tp$$

In the same case, for a non-pipelined processor, execution time of ‘n’ instructions will be:

$$ET_{non\text{-}pipeline} = n * k * Tp$$

So, speedup (S) of the pipelined processor over non-pipelined processor, when ‘n’ tasks are executed on the same processor is:

S = Performance of pipelined processor / Performance of Non-pipelined processor

As the performance of a processor is inversely proportional to the execution time, we have,

$$S = ET_{non\text{-}pipeline} / ET_{pipeline}$$
$$\Rightarrow S = [n * k * Tp] / [(k + n - 1) * Tp]$$
$$S = [n * k] / [k + n - 1]$$

When the number of tasks ‘n’ are significantly larger than k, that is, n >> k

$$S = n * k / n$$
$$S = k$$

where ‘k’ are the number of stages in the pipeline.

Also, **Efficiency** = Given speed up / Max speed up = S / S_{max}

We know that, Smax = k

So, **Efficiency** = S / k

Throughput = Number of instructions / Total time to complete the instructions
So, **Throughput** = n / (k + n – 1) * Tp
Note: The cycles per instruction (CPI) value of an ideal pipelined processor is 1

Problem (example):

Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 ns. Given latch delay is 10 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 1000 tasks
5. Sequential time for 1000 tasks
6. Throughput

Solution-

Given-

- Four stage pipeline is used k=4
- Delay of stages = 60, 50, 90 and 80 ns
- Latch delay or delay due to each register = 10 ns

1: Pipeline Cycle Time-

Cycle time
= Maximum delay due to any stage + Delay due to its register
= Max { 60, 50, 90, 80 } + 10 ns = 90 ns + 10 ns = 100 ns

2: Non-Pipeline Execution Time- (no latches hence latch delay=0)

Non-pipeline execution time for one instruction
= 60 ns + 50 ns + 90 ns + 80 ns = 280 ns

3: Speed Up Ratio-

Speed up
= Non-pipeline execution time / Pipeline execution time
= 280 ns / 100 ns
= 2.8

4: Pipeline Time For 1000 Tasks-

Pipeline time for 1000 tasks
= Time taken for 1st task + Time taken for remaining 999 tasks
= 1 x 4 clock cycles + 999 x 1 clock cycle
= 4 x cycle time + 999 x cycle time
= 4 x 100 ns + 999 x 100 ns
= 400 ns + 99900 ns
= 100300 ns

5: Sequential Time For 1000 Tasks-

Non-pipeline time for 1000 tasks
= 1000 x Time taken for one task
= 1000 x 280 ns
= 280000 ns

6: Throughput-

Throughput for pipelined execution
= Number of instructions executed per unit time
= 1000 tasks / 100300 ns

Dependencies in a pipelined processor
Pipeline hazards

There are mainly three types of dependencies possible in a pipelined processor. These are :
1) Structural Dependency
2) Control Dependency
3) Data Dependency

These dependencies may introduce stalls in the pipeline.

Stall : A stall is a cycle in the pipeline without new input.

1. Structural dependency

This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Example:

INSTRUCTION / CYCLE	1	2	3	4	5
I_1	IF(Mem)	ID	EX	Mem	
I_2		IF(Mem)	ID	EX	
I_3			IF(Mem)	ID	EX
I_4				IF(Mem)	ID

- In the above scenario, in cycle 4, instructions I_1 and I_4 are trying to access same resource (Memory) which introduces a resource conflict.
- To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available. This wait will introduce stalls in the pipeline as shown below:

CYCLE	1	2	3	4	5	6	7	8
I_1	IF(Mem)	ID	EX	Mem	WB			
I_2		IF(Mem)	ID	EX	Mem	WB		
I_3			IF(Mem)	ID	EX	Mem	WB	
I_4				–	–	–	IF(Mem)	

Solution for structural dependency

To minimize structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming.

Renaming: According to renaming, we divide the memory into two independent modules used to store the instruction and data separately called Code memory (CM) and Data memory (DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the instructions.

INSTRUCTION/ CYCLE	1	2	3	4	5	6	7
I_1	IF(CM)	ID	EX	DM	WB		
I_2		IF(CM)	ID	EX	DM	WB	
I_3			IF(CM)	ID	EX	DM	WB
I_4				IF(CM)	ID	EX	DM
I_5					IF(CM)	ID	EX
I_6						IF(CM)	ID
I_7							IF(CM)

2. Control Dependency (Branch Hazards)

This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.

Consider the following sequence of instructions in the program:

100: I_1

101: I_2 (JMP 250)

102: I_3

.

.

250: BI_1

Expected output: I_1 -> I_2 -> BI_1

NOTE: Generally, the target address of the JMP instruction is known after ID stage only.

INSTRUCTION/ CYCLE	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
I_3			IF	ID	EX	Mem
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> I_3 -> BI_1

So, the output sequence is not equal to the expected output, that means the pipeline is not implemented correctly.

To eliminate this problem we need to stop the Instruction fetch until we get target address of branch instruction. This can be implemented by introducing delay slot until we get the target address.

INSTRUCTION/ CYCLE	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID (PC:250)	EX	Mem	WB
Delay	–	–	–	–	–	–
BI_1				IF	ID	EX

Output Sequence: I_1 -> I_2 -> Delay (Stall) -> BI_1

As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline.

Solution for Control dependency:

Branch Prediction is the method through which stalls due to control dependency can be eliminated. In this at 1st stage prediction is done about which branch will be taken. For branch prediction Branch penalty is zero.

Branch penalty: The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

NOTE: As we see that the target address is available after the ID stage, so the number of stalls introduced in the pipeline is 1. Suppose, the branch target address would have been present after the ALU stage, there would have been 2 stalls. Generally, if the target address is present after the k^{th} stage, then there will be (k – 1) stalls in the pipeline.

Total number of stalls introduced in the pipeline due to branch instructions = **Branch frequency * Branch Penalty**

3. Data Dependency (Data Hazard)

Let us consider an ADD instruction S, such that
S: ADD R1, R2, R3
Addresses read by S = I(S) = {R2, R3}
Addresses written by S = O(S) = {R1}

Now, we say that instruction S2 depends in instruction S1, when

$$|I(S1) \cap O(S2)| \cup |O(S1) \cap I(S2)| \cup |O(S1) \cap O(S2)| \neq \phi$$

This condition is called Bernstein condition.

Three cases exist:

- **Flow (data) dependence:** O(S1) ∩ I (S2), S1 → S2 and S1 writes after something read by S2
- **Anti-dependence:** I(S1) ∩ O(S2), S1 → S2 and S1 reads something before S2 overwrites it
- **Output dependence:** O(S1) ∩ O(S2), S1 → S2 and both write the same memory location.

Example: Let there be two instructions I1 and I2 such that:

I1: ADD R1, R2, R3
I2: SUB R4, R1, R2

When the above instructions are executed in a pipelined processor, then data dependency condition will occur, which means that I_2 tries to read the data before I_1 writes it, therefore, I_2 incorrectly gets the old value from I_1.

INSTRUCTION / CYCLE	1	2	3	4
I_1	IF	ID	EX	DM
I_2		IF	ID(Old value)	EX

To minimize data dependency stalls in the pipeline, **operand forwarding** is used.
Operand Forwarding : In operand forwarding, we use the interface registers present between the stages to hold intermediate output so that dependent instruction can access new value from the interface register directly.
Considering the same example:
I1: ADD R1, R2, R3
I2: SUB R4, R1, R2

INSTRUCTION / CYCLE	1	2	3	4
I_1	IF	ID	EX	DM
I_2		IF	ID	EX

Data Hazards

Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline. Hazard cause delays in the pipeline. There are mainly three types of data hazards:

1) RAW (Read after Write) [Flow/True data dependency]
2) WAR (Write after Read) [Anti-Data dependency]
3) WAW (Write after Write) [Output data dependency]

Let there be two instructions I and J, such that J follow I. Then,
1) RAW hazard occurs when instruction J tries to read data before instruction, I writes it. Eg:
I: R2 <- R1 + R3
J: R4 <- R2 + R3

2) WAR hazard occurs when instruction J tries to write data before instruction I reads it. Eg:
I: R2 <- R1 + R3
J: R3 <- R4 + R5
3) WAW hazard occurs when instruction J tries to write output before instruction I writes it. Eg:
I: R2 <- R1 + R3
J: R2 <- R4 + R5
WAR and WAW hazards occur during the out-of-order execution of the instructions.

Pipelining: Types and Stalling

Types of pipeline

1. Uniform delay pipeline:

- In this type of pipeline, all the stages will take same time to complete an operation.
- In uniform delay pipeline, **Cycle Time (Tp) = Stage Delay**
- If buffers are included between the stages then,

  Cycle Time (Tp) = Stage Delay + Buffer Delay

2. Non-Uniform delay pipeline:

- In this type of pipeline, different stages take different time to complete an operation.
- In this type of pipeline, **Cycle Time (Tp) = Maximum (Stage Delay)**
- For example, if there are 4 stages with delays, 1 ns, 2 ns, 3 ns, and 4 ns, then

  Tp = Maximum (1 ns, 2 ns, 3 ns, 4 ns) = 4 ns
- If buffers are included between the stages,

  Tp = Maximum (Stage delay + Buffer delay)

Example: Consider a 4-segment pipeline with stage delays (2 ns, 8 ns, 3 ns, 10 ns). Find the time taken to execute 100 tasks in the above pipeline.
Solution: As the above pipeline is a non-linear pipeline,
Tp = max (2, 8, 3, 10) = 10 ns
We know that $ET_{pipeline}$ = (k + n – 1) Tp = (4 + 100 – 1) 10 ns = 1030 ns
NOTE: MIPS = Million instructions per second

Performance of pipeline with stalls
Speed Up (S) = $Performance_{pipeline}$ / $Performance_{non\text{-}pipeline}$
=> S = Average Execution $Time_{non\text{-}pipeline}$ / Average Execution $Time_{pipeline}$
=> S = $CPI_{non\text{-}pipeline}$ * Cycle $Time_{non\text{-}pipeline}$ / $CPI_{pipeline}$ * Cycle $Time_{pipeline}$
Ideal CPI of the pipelined processor is ‘1’. But due to stalls, it becomes greater than ‘1’.
=> S = $CPI_{non\text{-}pipeline}$ * Cycle $Time_{non\text{-}pipeline}$ / (1 + Number of stalls per Instruction) * Cycle $Time_{pipeline}$
As Cycle $Time_{non\text{-}pipeline}$ = Cycle $Time_{pipeline}$,

Speed Up (S) = $CPI_{non\text{-}pipeline}$ / (1 + Number of stalls per instruction)

Dynamic Instruction Scheduling: If the programmer is aware of a pipelined architecture then it may be possible to rewrite programs statically either manually or using an optimising compiler to separate data dependencies otherwise they must be detected and resolved by hardware at runtime.
Scoreboarding: A scoreboard is a centralized control logic which uses forwarding logic and register tagging to keep track of the status of registers and multiple functional units. An issued instruction whose registers are not available is forwarded to a reservation station (buffer) associated with the functional unit it will use. When functional units generate new results, some data dependencies can be resolved. When all registers have valid data the scoreboard enables the instruction execution. Similarly, when a functional unit finishes, it signals the scoreboard to release the register resources.

Arithmetic Pipeline and Instruction Pipeline

1. Arithmetic Pipeline :

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations.

Floating point addition using arithmetic pipeline :

The following sub operations are performed in this case:

- Compare the exponents.
- Align the mantissas.
- Add or subtract the mantissas.
- Normalise the result

First, the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally, the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:

Let us consider two numbers,

X=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 time to the right to give

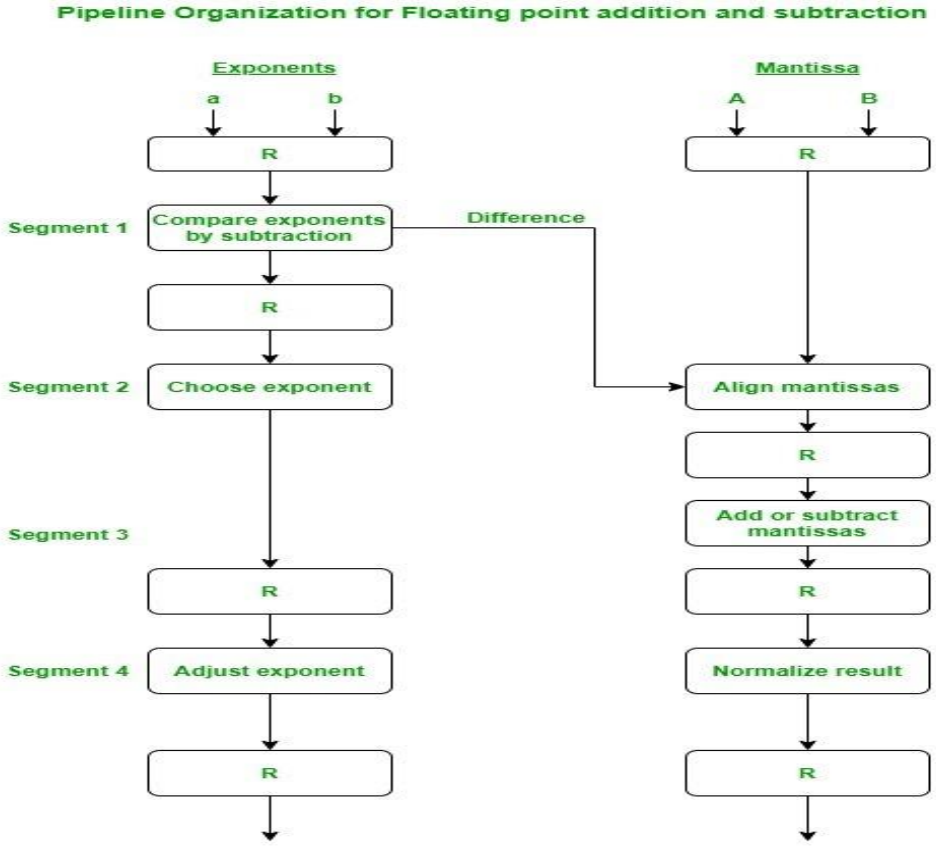
Y=0.0450*10^3

Finally, the two numbers are added to produce

Z=0.3664*10^3

As the result is already normalized the result remains the same.

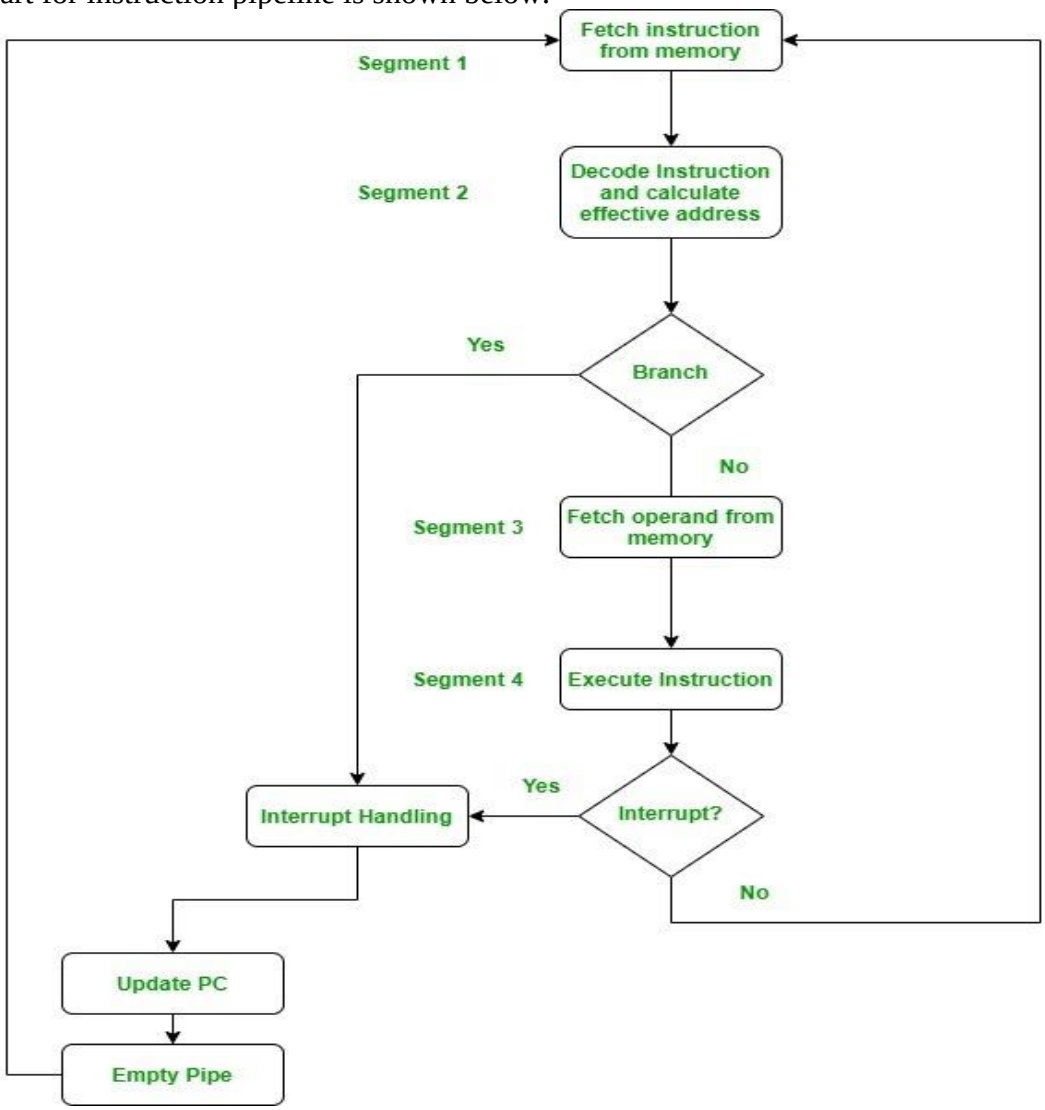
The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.



2. Instruction Pipeline:

- In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.
- In the most general case computer needs to process each instruction in following sequence of steps:
    - Fetch the instruction from memory (FI)
    - Decode the instruction (DA)
    - Calculate the effective address
    - Fetch the operands from memory (FO)
    - Execute the instruction (EX)
    - Store the result in the proper place

The flowchart for instruction pipeline is shown below.



Let us see an example of instruction pipeline.

Example:

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	---	---	FI	DA	FO	EX			
5								FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

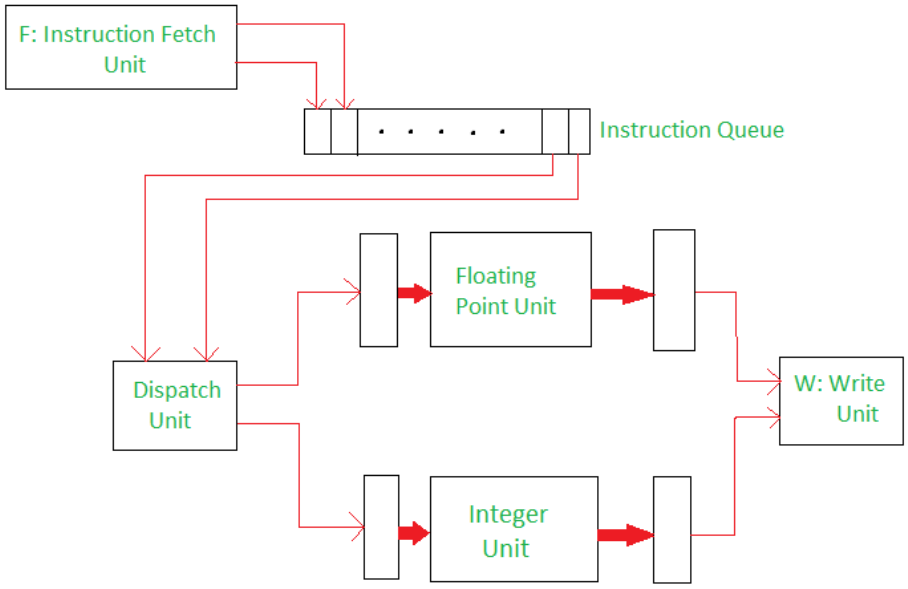
Instruction Branch

Here the instruction is fetched on first clock cycle in segment 1.

- Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.
- In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus, fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

Superscalar Architecture

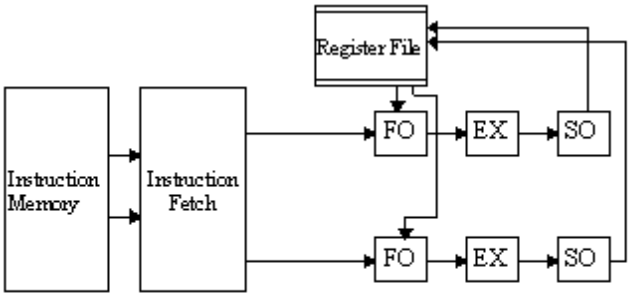
A more useful approach is to equip the processor with multiple processing units to handle several instructions in parallel in each processing stage. With this arrangement, several instructions start execution in the same clock cycle and the process is said to use multiple issue. Such processors are capable of achieving an instruction execution throughput of more than one instruction per cycle. They are known as 'Superscalar Processors'.



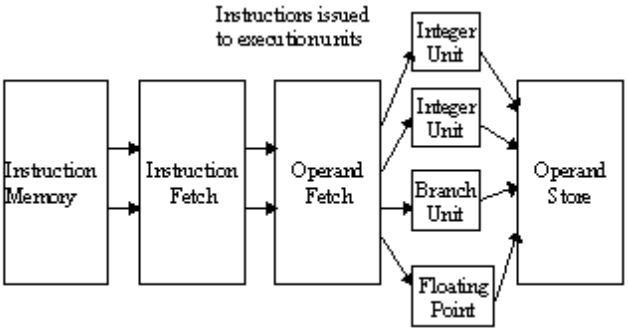
Processor with Two Execution Units

In the above diagram, there is a processor with two execution units; one for integer and one for floating point operations. The instruction fetch unit is capable of reading the instructions at a time and storing them in the instruction queue. In each cycle, the dispatch unit retrieves and decodes up to two instructions from the front of the queue. If there is one integer, one floating point instruction and no hazards, both the instructions are dispatched in the same clock cycle.

A scalar processor executes scalar instructions, that is instructions operating on single quantity operands such as integers. As we have seen, the design of such a processor may be pipelined where the staggered use of the pipeline can improve instruction throughput. A superscalar processor is one which executes more than one scalar instruction concurrently. This is achieved by having a number of independent pipelines. A limited form of superscalar operation is present in processors that have, for example, separate integer and floating point units. After the decode stage the instructions are sent to the appropriate unit and can execute in parallel. True superscalar operation can only be achieved by fetching a number of instructions simultaneously and executing them simultaneously.



Superscalar Dual Pipeline Processor



Superscalar design with multiple specialised execution units

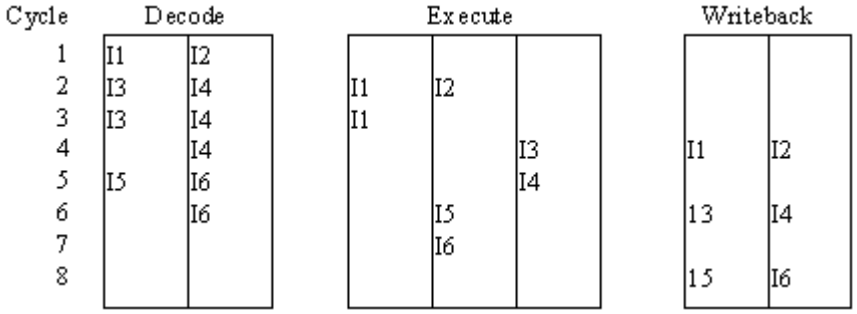
The objective of superscalar design is to improve performance by exploiting instruction level parallelism in user programs. We might expect that doubling the number of pipelines would double performance, but we have seen the problems caused by pipeline hazards in a single pipeline and can see that this problem is much more critical for superscalar processors and that some of the pipelines will be stalling. One of the characteristics of RISC designs is that the processor has a simplified instruction set architecture resulting in fewer data interdependencies. For this reason, virtually all superscalar designs have been based on a RISC architecture. The superscalar degree is low due to limited instruction parallelism that can be exploited in ordinary programs.

Instruction issue and completion policies are critical to superscalar processor performance. When instructions are issued (i.e. initiation of their execution in functional units) in program order it is called *in-order issue*. When program order is violated, *out-of-order issue* is being practiced. When instructions must be completed (i.e. have altered register and/or memory) in program order, it is called *in-order completion*, otherwise *out-of-order* completion may result. In-order issue is easier to implement but may not yield the optimal performance. Proper scheduling can avoid stalling the pipelines. A number of possible scheduling policies are possible. In the examples below, the pipeline cycle is the minimum time between consecutive exchanges between each stage of the pipeline.

1) In-Order Issue with In-Order Completion:
Assume we have a superscalar pipeline capable of fetching and decoding two instructions at a time. Let's say there are three separate functional units and that there are two instances of the writeback pipeline stage.
Let's say we have six instructions with the following constraints:

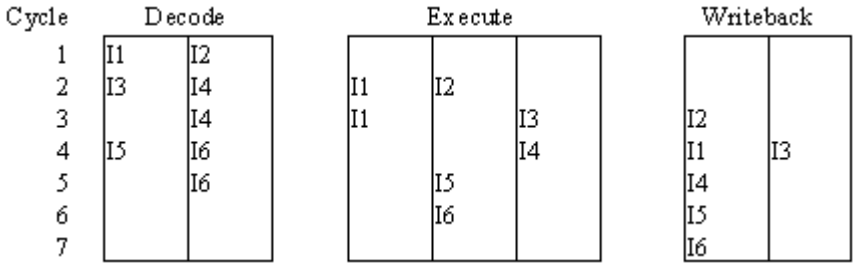
- I1 requires two cycles to execute.
- I3 and I4 conflict for the same functional unit.
- I5 depends on the value produced by I4.
- I5 and I6 conflict for a functional unit.

Instructions are only decoded up to the point of a dependency or resource conflict. No additional instructions are decoded until the conflict is resolved. This means a maximum of two instructions can be in the execute stage as later instructions have a time dependency on earlier ones executing first.



2) In-Order Issue with Out-of-Order Completion:

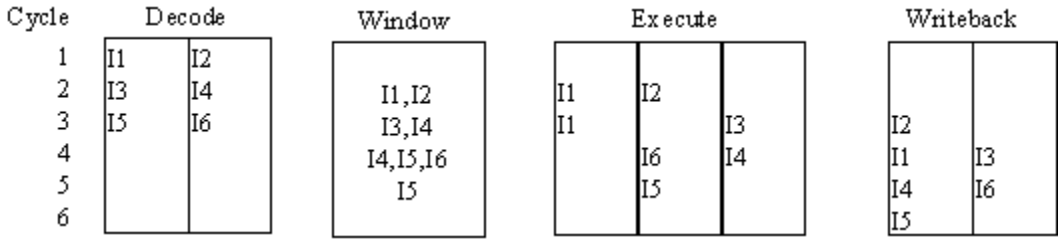
Using the same set of instructions, the next diagram illustrates the effect of allowing some instructions to complete out-of-order. With out-of-order completion, any number of instructions may be in the execution stage, limited only by the machine's parallelism. Instruction issuing in any one pipeline is stalled by a resource conflict, data dependency or procedural dependency.



- Note that I2 is allowed to complete before I1. I5 depends on the value produced by I4 and cannot be issued until cycle 5.
- Out-of-order completion requires more complex instruction-issue logic than in-order completion. It is more difficult to deal with instruction interrupts and exceptions. When an interrupt occurs, the processor must take into account that instructions ahead of the instruction that caused the interrupt may have already completed.
- The time from decoding the first instruction to writing the last is 7 cycles.

3) Out-of-Order Issue with Out-of-Order Completion:

To allow out-of-order issue, it is necessary to decouple the decode and execute stages of the pipeline. When an instruction has been decoded it is placed in a buffer known as an instruction window. As long as this buffer is not full, the processor can continue to fetch and decode new instructions. When a functional unit becomes available an instruction from the window may be issued. Any instruction that needs the functional unit and which has no conflicts or dependencies to block it may be selected.



- Note that it is possible to issue I6 before I5 as I5 has a dependency on I4. The time from decoding the first instruction to writing the last is 6 cycles.

The instruction window can be centralised or distributed. A centralised instruction window holds all instructions irrespective of their type. In the distributed approach, instruction buffers called *reservation stations* are placed in front of each functional unit. Decoded instructions are routed to the appropriate reservation station and subsequently issued to the functional unit when it is free and all operands for the instruction have been received by the reservation station.

Advantages of Superscalar Architecture :

- In a Superscalar Processor, the detrimental effect on performance of various hazards becomes even more pronounced.
- The compiler can avoid many hazards through judicious selection and ordering of instructions.
- The compiler should strive to interleave floating point and integer instructions. This would enable the dispatch unit to keep both the integer and floating point units busy most of the time.
- In general, high performance is achieved if the compiler is able to arrange program instructions to take maximum advantage of the available hardware units.

Disadvantages of Superscalar Architecture :

- Due to this type of architecture, problem in scheduling can occur.

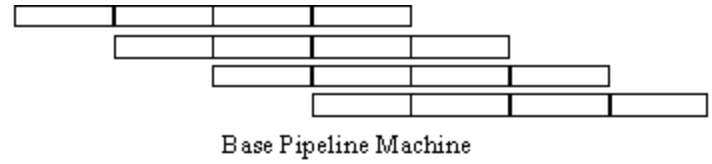
COMPARISON BETWEEN PIPELINING & SUPERSCALAR

Pipelining	Superscalar
divides an instruction into steps, and since each step is executed in a different part of the processor, multiple instructions can be in different "phases" each clock.	involves the processor being able to issue multiple instructions in a single clock with redundant facilities to execute an instruction within a single core
once one instruction was done decoding and went on towards the next execution subunit	multiple execution subunits able to do the same thing in parallel
Sequencing unrelated activities such that they use different components at the same time	Multiple sub-components capable of doing the same task simultaneously, but with the processor deciding how to do it.

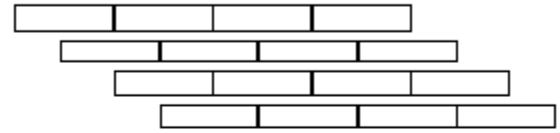
Superpipelining

An alternative approach to achieving better performance is superpipelining. Superpipelining exploits the fact that many pipeline stages perform tasks that require less than half a clock cycle. A doubled internal clock speed allows those stages to perform two tasks during one external clock cycle. In a superpipelined processor of degree n, the pipeline cycle time is 1/n of the base cycle. Stages that require the full base cycle to complete can be strung into a series of shorter stages, effectively increasing the length of the pipeline and matching the execution latency of each stage. An number of instructions may be in various parts of the execution stage. As a comparison, where an execution operation takes 1

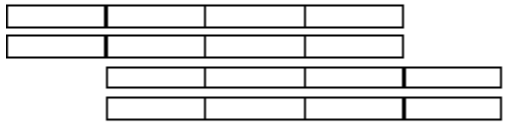
cycle in the base scalar processor, the same operation is implemented as n short cycles in a superpipelined processor with the same operation technology.



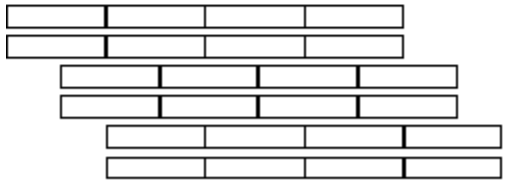
Base Pipeline Machine



Superpipeline Machine



Superscalar Machine



Superscalar Superpipelined Machine

Both the superpipeline and the superscalar implementations have the same number of instructions executing at the same time in the steady state. The superpipelined processor falls behind the superscalar processor at the start of the program and after each branch instruction.

The following terms are used for comparison.

- The *scalar base machine* is a single multi-stage pipeline processor.
- The *pipeline cycle* for the scalar base machine is assumed to be 1 time unit called the base cycle.
- The *instruction issue rate* is the number of instructions issued per cycle.
- The *instruction issue latency* is the time required between the issuing of two adjacent instructions.
- *Simple operation latency* is the time taken to execute simple operations, such as add, load, store, branch, move etc. which make up the vast majority of instructions executed by the processor.
- Complex instructions are those requiring an order of magnitude longer latency such as divides, cache misses.
- The *instruction level parallelism* is the maximum number of instructions that can be simultaneously executed in the pipeline.

Machine Type	Scalar base machine of k pipeline stages	Superscalar machine of degree m	Superpipeline machine of degree n	Superpipeline superscalar machine of degree (m, n)
Machine Pipeline Cycle	1 (base cycle)	1	1/n	1/n
Instruction Issue Rate	1	m	1	m
Instruction Issue Latency	1	1	1/n	1/n
Simple Operation Latency	1	1	1 = (n x 1/n)	1 = (n x 1/n)
Instruction Level Parallelism to fully utilise the pipeline	1	m	n	mn

Design Parameters for Pipeline Processors

- In the scalar base machine one instruction is issued per cycle, with one cycle latency for simple operations and one cycle latency between instructions. The instruction pipeline can be fully utilised if successive instructions can enter it continuously at the rate of one per cycle.
- In a superpipelined superscalar design of degree (m,n) the machine executes m instructions every cycle with a pipeline cycle 1/n of the base cycle. Simple operation latency is n pipeline cycles. The level of parallelism required to fully utilise this machine is mn instructions.
- The superscalar approach depends on the ability to execute multiple instructions in parallel. A combination of the compiler-based optimisation and various hardware techniques can be used to maximise instruction level parallelism.

COMPARISON BETWEEN SUPERPIPELINING & SUPERSCALAR

- Super-pipelining attempts to increase performance by reducing the clock cycle time. It achieves that by making each pipeline stage very shallow, resulting in a large number of pipe stages. A shorter clock cycle means a faster clock. As long as your cycles per instruction (CPI) doesn't change, a faster clock means better performance. Super-pipelining works best with code that doesn't branch often, or has easily predicted branches.
- Superscalar attempts to increase performance by executing multiple instructions in parallel. If we can issue more instructions every cycle, without decreasing clock rate, then CPI decreases, therefore increasing performance.
- Superscalar breaks into two broad categories: In-order and out-of-order.
    - In-order superscalar mainly provides benefit to code with instruction-level parallelism among a small window of consecutive instructions.
    - Out-of-order superscalar allows the pipeline to find parallelism across larger windows of code, and to hide latencies associated with long-running instructions. (Example: load instructions that miss the cache.)

NOTE:

- Super-pipelining seeks to improve the sequential instruction rate, while superscalar seeks to improve the parallel instruction rate.
- Most modern processors are both superscalar and super-pipelined. They have deep pipelines to achieve high clock rates, and wide instruction issue to make use of instruction level parallelism.

Instruction-level parallelism

- Instruction-level parallelism (ILP) is a measure of how many of the instructions in a computer program can be executed simultaneously.
- ILP must not be confused with concurrency, since **ILP** is about parallel execution of a sequence of instructions belonging to a specific thread of execution of a process (that is a running program with its set of resources - for example its address space, a set of registers, its identifiers, its state, program counter, and more). Conversely, **concurrency** regards with the threads of one or different processes being assigned to a CPU's core in a strict alternance or in true parallelism if there are enough CPU's cores, ideally one core for each runnable thread.

There are two approaches to instruction level parallelism: Hardware and Software.

- Hardware level works upon dynamic parallelism, whereas the software level works on static parallelism. Dynamic parallelism means the processor decides at run time which instructions to execute in parallel, whereas static parallelism means the compiler decides which instructions to execute in parallel. The Pentium processor works on the dynamic sequence of parallel execution, but the Itanium processor works on the static level parallelism.

Consider the following program:

```
1   e = a + b
2   f = c + d
3   m = e * f
```

- Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2(as 1 and 2 can execute concurrently and require 1unit time and instruction 3 require 1unit time).
- A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

Micro-architectural techniques that are used to exploit ILP include:

- Instruction pipelining where the execution of multiple instructions can be partially overlapped.
- Superscalar execution, VLIW, and the closely related explicitly parallel instruction computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.
- Out-of-order execution where instructions execute in any order that does not violate data dependencies. Note that this technique is independent of both pipelining and superscalar execution. Current implementations of out-of-order execution dynamically (i.e., while the program is executing and without any help from the compiler) extract ILP from ordinary programs. An alternative is to extract this parallelism at compile time and somehow convey this information to the hardware. Due to the complexity of scaling the out-of-order execution technique, the industry has re-examined instruction sets which explicitly encode multiple independent operations per instruction.
- Register renaming which refers to a technique used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations, used to enable out-of-order execution.
- Speculative execution which allows the execution of complete instructions or parts of instructions before being certain whether this execution should take place. A commonly used form of speculative execution is control flow speculation where instructions past a control flow instruction

(e.g., a branch) are executed before the target of the control flow instruction is determined. Several other forms of speculative execution have been proposed and are in use including speculative execution driven by value prediction, memory dependence prediction and cache latency prediction.

- Branch prediction which is used to avoid stalling for control dependencies to be resolved. Branch prediction is used with speculative execution.

Note: It is known that the ILP is exploited by both the compiler and hardware support but the compiler also provides inherent and implicit ILP in programs to hardware by compilation optimization. Some optimization techniques for extracting available ILP in programs would include scheduling, register allocation/renaming, and memory access optimization.

- To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks.
- The simplest and most common way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of a loop. This type of parallelism is often called loop-level parallelism.

Example 1

```
for (i=1; i<=1000; i= i+1)
  x[i] = x[i] + y[i];
```

- This is a parallel loop. Every iteration of the loop can overlap with any other iteration, although within each loop iteration there is little opportunity for overlap.

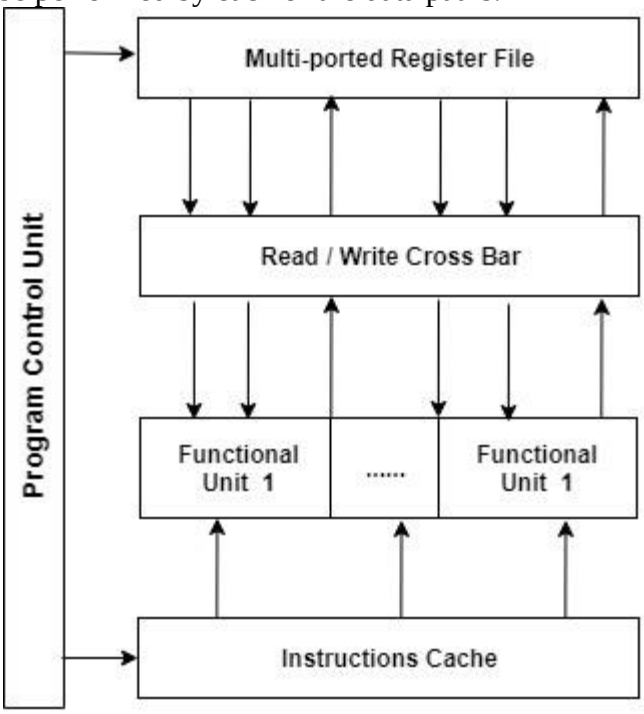
Example 2

```
for (i=1; i<=100; i= i+1){
  a[i] = a[i] + b[i];        //s1
  b[i+1] = c[i] + d[i];      //s2
}
```

Statement s1 uses the value assigned in the previous iteration by statement s2, so there is a loop-carried dependency between s1 and s2.

VLIW Architecture:

- Very Long Instruction Word (VLIW) architecture in P-DSPs (programmable DSP) increases the number of instructions that are processed per cycle. It is a concatenation of several short instructions and requires multiple execution units running in parallel, to carry out the instructions in a single cycle. A language compiler or pre-processor separates program instructions into basic operations and places them into VLIW processor which then disassembles and transfers each operation to an appropriate execution unit.
- VLIW P-DSPs have a number of processing units (data paths) i.e. they have a number of ALUs, MAC units, shifters, etc. The VLIW is accessed from memory and is used to specify the operands and operations to be performed by each of the data paths.



VLIW Architecture

- As shown in figure, the multiple functional units share a common multi-ported register file for fetching the operands and storing the results. Parallel random access by the functional units to the register file is facilitated by the read/write cross bar. Execution of the operations in the functional units is carried out concurrently with the load/ store operation of data between a RAM and the register file.
- The performance gains that can be achieved with VLIW architecture depends on the degree of parallelism in the algorithm selected for a DSP application and the number of functional units. The throughput will be higher only if the algorithm involves execution of independent operations. For example, in convolution by using eight functional units, the time required can be reduced by a factor of 8 compared to the case where a single functional unit is used.
- However, it may not always be possible to have independent stream of data for processing. The number of functional units is also limited by the hardware cost for the multi-ported register file and cross bar switch.

Advantages of VLIW architecture

- Increased performance.
- Potentially scalable i.e. more execution units can be added and so more instructions can be packed into the VLIW instruction.

Disadvantages of VLIW architecture

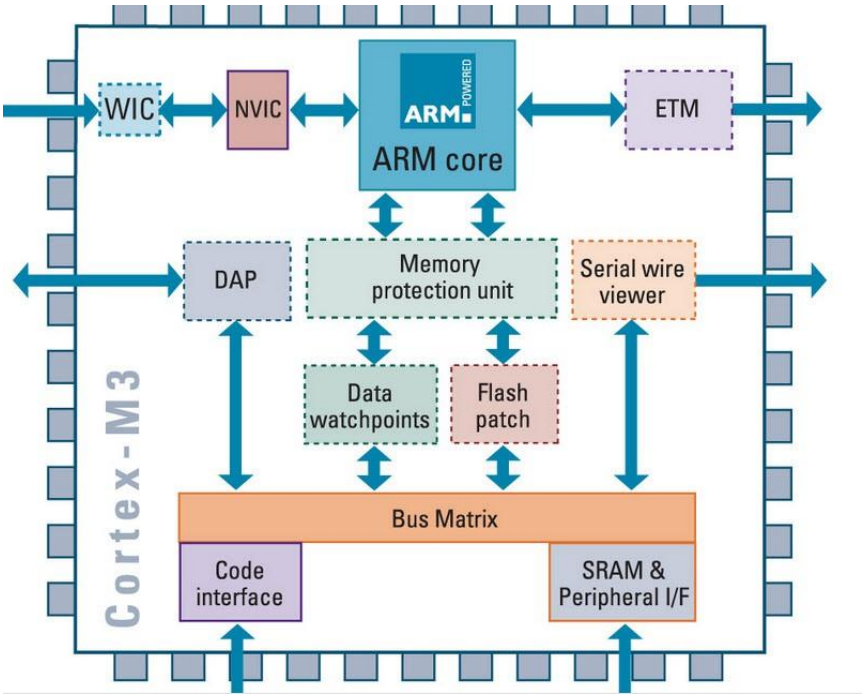
- New programmer needed.
- Program must keep track of Instruction scheduling.
- Increased memory use.
- High power consumption.

ARM Processor

- The ARM microcontroller stands for Advance Risk Machine; it is one of the extensive and most licensed processor cores in the world. The first ARM processor was developed in the year 1978 by Cambridge University, and the first ARM RISC processor was produced by the Acorn Group of Computers in the year 1985. These processors are specifically used in portable devices like digital cameras, mobile phones, home networking modules and wireless communication technologies and other embedded systems due to the benefits, such as low power consumption, reasonable performance, etc.

ARM Architecture

- The ARM architecture processor is an advanced reduced instruction set computing [RISC] machine and it's a 32bit reduced instruction set computer (RISC) microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST Microelectronics, Motorola, and so on. The ARM architecture comes with totally different versions like ARMv1, ARMv2, etc.



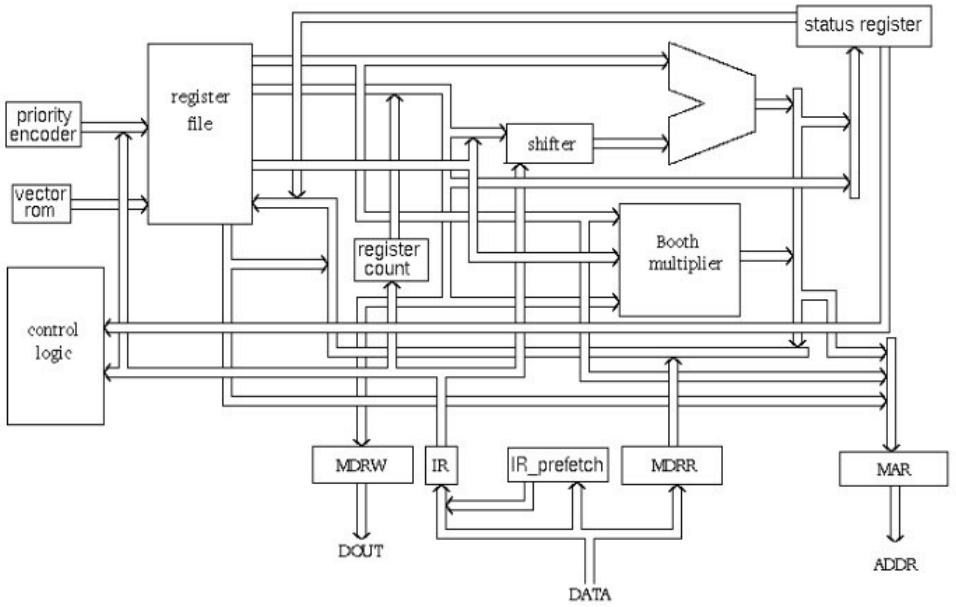
ARM Cortex:

- The ARM cortex is a complicated microcontroller within the ARM family that has ARMv7 design. There are 3 subfamilies within the ARM cortex family:
  - ARM Cortex Ax-series
  - ARM-Cortex Rx-series
  - ARM-Cortex Mx-series

ARM Block Diagram

- The ARM processor conjointly has other components like the Program status register, which contains the processor flags (Z, S, V and C). The modes bits conjointly exist within the program standing register, in addition to the interrupt and quick interrupt disable bits;

- Some special registers: Some registers are used like the instruction; memory data read and write registers and memory address register.
- Priority encoder: The encoder is used in the multiple load and store instruction to point which register within the register file to be loaded or kept.
- Multiplexers: several multiplexers are accustomed to the management operation of the processor buses. Because of the restricted project time, we tend to implement these components in a very behavioural model. Each component is described with an entity. Every entity has its own architecture, which can be optimized for certain necessities depending on its application. This creates the design easier to construct and maintain.



The ARM Architecture:

The ARM processor consists of:

- Arithmetic Logic Unit
- Booth multiplier
- Barrel shifter
- Control unit
- Register file

Arithmetic Logic Unit (ALU)

- The ALU has two 32-bits inputs. The primary comes from the register file, whereas the other comes from the shifter. Status registers flags modified by the ALU outputs. The V-bit output goes to the V flag as well as the Count goes to the C flag. Whereas the foremost significant bit really represents the S flag, the ALU output operation is done by NORed to get the Z flag. The ALU has a 4-bit function bus that permits up to 16 opcodes to be implemented.

Booth Multiplier Factor

- The multiplier factor has 3 32-bit inputs and the inputs return from the register file. The multiplier output is barely 32-Least Significant Bits of the merchandise. The multiplication starts whenever the beginning 04 input goes active. Fin of the output goes high when finishing.

Booth Algorithm

- Booth algorithm is a noteworthy multiplication algorithmic rule for 2's complement numbers. This treats positive and negative numbers uniformly. Moreover, the runs of 0's or 1's within the multiplier factor are skipped over without any addition or subtraction being performed, thereby creating possible quicker multiplication. The figure shows the simulation results for the multiplier test bench. It's clear that the multiplication finishes only in16 clock cycle.

Barrel Shifter

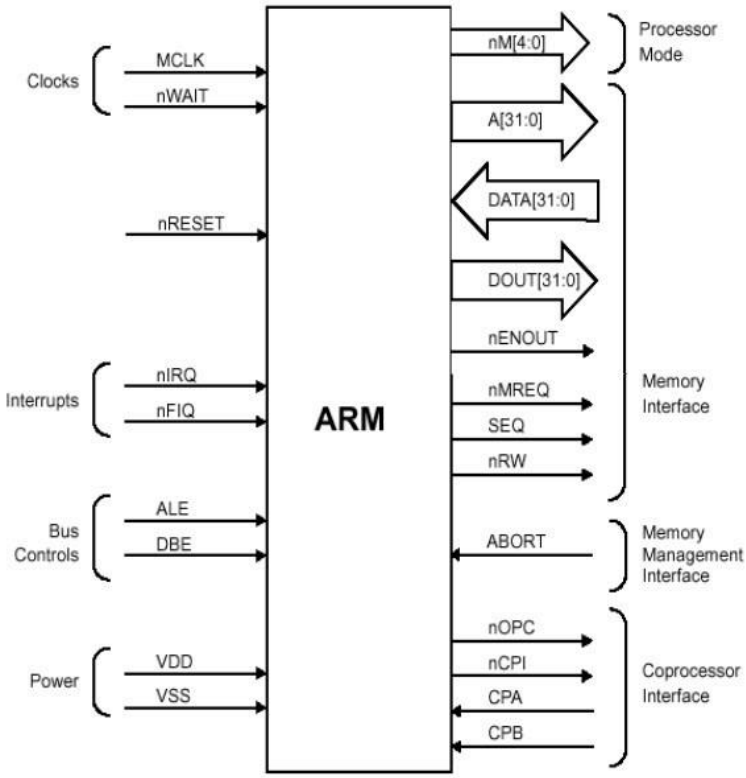
- The barrel shifter features a 32-bit input to be shifted. This input is coming back from the register file or it might be immediate data. The shifter has different control inputs coming back from the instruction register. The Shift field within the instruction controls the operation of the barrel shifter. This field indicates the kind of shift to be performed (logical left or right, arithmetic right or rotate right). The quantity by which the register ought to be shifted is contained in an immediate field within the instruction or it might be the lower 6 bits of a register within the register file.
- The shift_val input bus is 6-bits, permitting up to 32-bit shift. The shift type indicates the needed shift sort of 00, 01, 10, 11 are corresponding to shift left, shift right, an arithmetic shift right and rotate right, respectively. The barrel shifter is especially created with multiplexers.

Control Unit

- For any microprocessor, control unit is the heart of the whole process and it is responsible for the system operation, so the control unit design is the most important part within the whole design. The control unit is sometimes a pure combinational circuit design. Here, the control unit is implemented by easy state machine. The processor timing is additionally included within the control unit. Signals from the control unit are connected to each component within the processor to supervise its operation.

ARM7 Functional Diagram

The final thing that must be explained is how the ARM will be used and the way in which the chip appear. The various signals that interface with the processor are input, output or supervisory signals which will be used to control the ARM operation.



ARM Functional Diagram

Additional Uses of the Cortex Processor

- It is a reduced instruction set computing Controller having 32-bit high performance central processing unit and 3-stage pipeline and compact one.
- It has THUMB-2 technology
- Merges optimally with 16/32-bit instructions

- High performance
- It supports tools and RTOS and its core Sight debug and trace
- Support for multiple processors
- Low power Modes
- It supports sleep modes
- Control the software package
- Multiple power domains
- Nested vectored interrupt controller (NVIC)
- Low latency, low noise interrupts response
- No need for assembly programming

SPARC processor

Introduction:

- SPARC stands for Scalable Processor Architecture.
- It is developed by Sun Microsystems in the 1980s.
- It is based on the RISC structure designed at the University of California at Berkeley in early 1980s.
- The SPARC architecture is a non-proprietary architecture that any person or company can license and use to develop microprocessors and other semiconductor devices based on published industry standards.
- In 1989, Sun Microsystems transferred ownership of the SPARC specifications to an independent, non-profit organization, SPARC International, which administers and licenses the technology and provides conformance testing and other services for its members.

Design Goals

- SPARC was designed as a target for optimizing compilers and easily pipelined hardware implementations.
- SPARC implementations provide exceptionally high execution rates(MIPS) and short time-to-market development schedules.
- It provide the scalability of the cost/performance ratio of successive implementations with the current improvements in circuit technology.
- The "Scalable" in SPARC comes from the fact that the SPARC specification allows implementations to scale from processors required in embedded systems to processors used for servers.

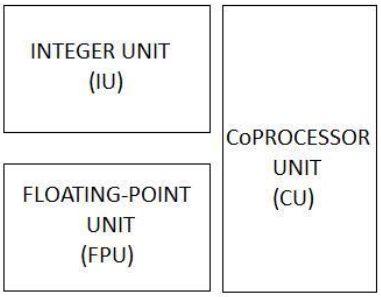
Brief History: 3 major revisions to the SPARC architecture

- SPARC-V7, 32bit, 1986
- SPARC-V8, 32bit, 1990
- SPARC-V9, 64bit, 1993
- In early 2006, Sun released an extended architecture specification, UltraSPARC Architecture 2005.

The SPARC Architecture

- It is a Load and store architecture. Operations are always done over registers.
- Uses “register window” concept thus offering a large number of registers.
- Uses delay slot to optimize branch instruction.
- Passes arguments using registers and the stack.

The Modules:

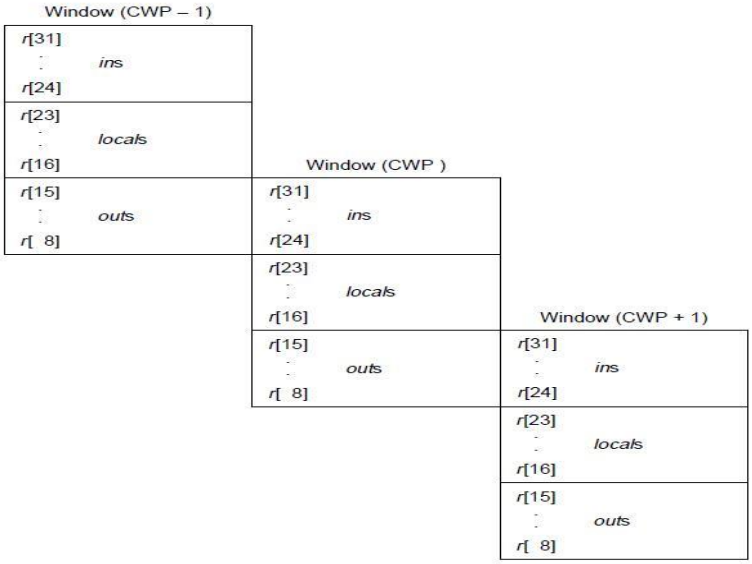


1) The Integer Unit (IU)

- Contains the general-purpose registers and controls the overall operation of the processor.
- It may contain from 64 to 528 general-purpose 64-bit r registers. They are partitioned into 8 global registers, 8 alternate global registers, plus a circular stack of from 3 to 32 sets of 16 registers each, known as register windows.
- Executes the integer arithmetic instructions and computes memory addresses for loads and stores.
- Maintains the program counters and controls instruction execution for the FPU.

2) The Register Window

- At any time, an instruction can access the 8 global registers and a 24-register window
- A register window comprises a 16-register set- divided into 8 in and 8 local registers- together with the 8 in registers of an adjacent register set, addressable from the current window as its out registers.
- When a procedure is called, the register window shifts by sixteen registers, hiding the old input registers and old local registers and making the old output registers the new input registers.
    - Input registers: arguments are passed to a function
    - Local registers: to store any local data.
    - Output registers: When calling a function, the programmer puts his argument in these registers.
- The current window into the *r registers is given by the current window pointer (CWP) register.*



3) The Floating-point Unit (FPU)

- The FPU has 32 32-bit (single-precision) floating-point registers, 32 64-bit (double-precision) floating-point registers, and 16 128-bit (quad-precision) floating-point registers.
- Double-precision values occupy an even-odd pair of single-precision registers.
- Quad-precision values occupy an odd-even number of pair of double precision registers.
- Floating-point load/store instructions are used to move data between the FPU and memory.
- The memory address is calculated by the IU.
- Floating-Point operate (FPop) instructions perform the floating-point arithmetic operations and comparisons.

4) Coprocessor Unit (CU)

- The instruction set includes support for a single, implementation-dependent coprocessor. The coprocessor has its own set of registers.
- Coprocessor load/store instructions are used to move data between the coprocessor registers and memory.

5) Instructions

Instructions can fall into following basic categories:

- Load/store
- Arithmetic/logical/shift
- Control transfer
- Read/write control register
- Floating-point/Coprocessor operate

SPARC v9 features

- 64-bit Data and Addresses as compared to 32-bit Data and Addresses of SPARC V8.
- 32 double-precision floating-point registers,
- Software-settable branch prediction
- 64-bit integer multiply and divide instructions
- load/store floating-point quad word instructions
- Branches on register value (eliminating the need to compare)
- The V9 remains binary compatible with all previous SPARC architecture.

Department of Computer Science and Engineering
Subject Name: Advanced Computer Architecture
UNIT-II

Flynn's Classification

Flynn's classification distinguishes multi-processor computer architectures according to two independent dimensions of Instruction stream and Data stream. An instruction stream is sequence of instructions executed by machine. And a data stream is a sequence of data including input, partial or temporary results used by instruction stream. Each of these dimensions can have only one of two possible states: Single or Multiple. Flynn's classification depends on the distinction between the performance of control unit and the data processing unit rather than its' operational and structural interconnections.

Following are the four category of Flynn classification and characteristic feature of each of them.

1. Single Instruction Stream, Single Data Stream (SISD)

The following figure represents an organization of simple SISD computer having one control unit, one processor unit and single memory unit.

SISD processor organizations

- They are also called scalar processor i.e., one instruction at a time and each instruction have only one set of operands.
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle.

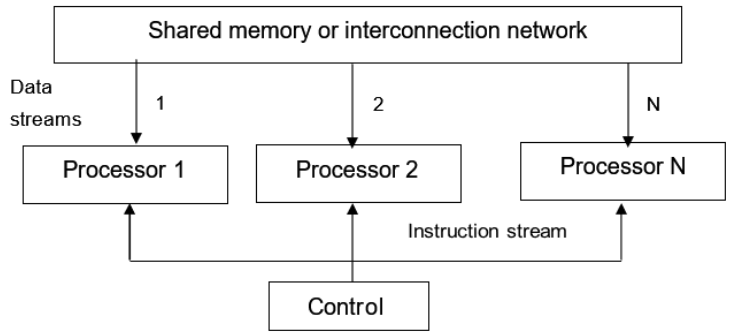
Characteristics:

- Single data: only one data stream is being used as input during any one clock cycle.
- Deterministic execution.
- Instructions are executed sequentially.
- This is the oldest and until recently, the most prevalent form of computer.

Example: most PCs, single CPU workstations and mainframes.

2. Single Instruction Stream, Multiple Data Stream (SIMD) processors

- This is a type of parallel computer.
- Single instruction: All processing units execute the same instruction issued by the control unit at any given clock cycle as shown in figure where there are multiple processors executing instruct ion given by one control unit.
- Multiple data: Each processing unit can operate on a different data element a s shown if figure below the processor are connected to shared memory or interconnection network providing multiple data to processing unit.
- This type of machine typically has an instruct ion dispatcher, a very high- bandwidth internal network, and a very large array of very small-capacity instruction units. Thus, single instruction is executed by different processing unit on different set of data.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing and vector computation.
- Synchronous (lockstep) and deterministic execution.



SIMD processor organizations

3. Multiple Instruction Stream, Single Data Stream (MISD)

- Here, a single data stream is feed into multiple processing units.
- Each processing unit operates on the data independently through independent instruction streams as shown in following figure a single data stream is forwarded to different processing unit which are connected to different control unit and execute instruction given to it by control unit to which it is attached.

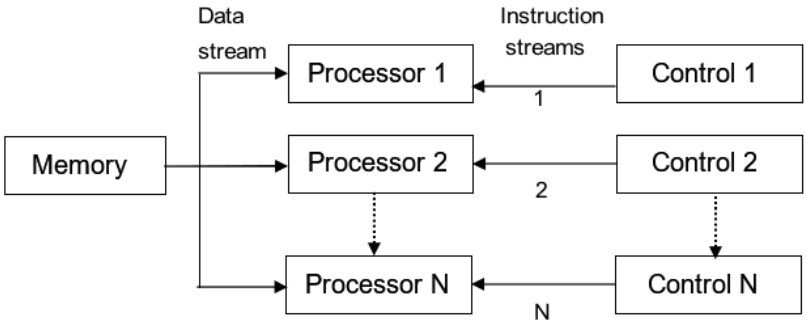
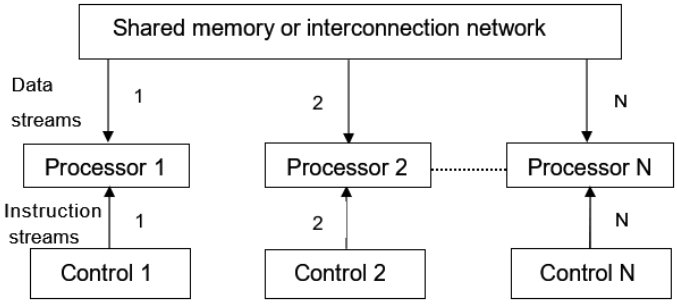


Figure: MISD processor organizations

- Thus, in these computers same data flow through a linear array of processors executing different instruction streams.
- This architecture is also known as systolic arrays for pipelined execution of specific instructions.

4. Multiple Instruction Stream, Multiple Data Stream (MIMD)

- Multiple Instructions: Every Processor may be executing a different instruction stream.
- Multiple Data: every processor may be working with a different data stream as shown in the figure multiple data stream is provided by shared memory.
- Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control.
- Execution can be synchronous or asynchronous, deterministic or non- deterministic.
- Examples: most current supercomputers, networked parallel computer " grids" and multi-processor SMP computers - including some types of PCs.



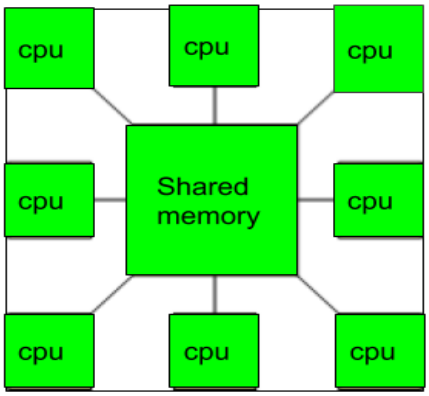
MIMD processor organizations System

Parallel Computer Models

Multiprocessor and Multicomputer

1. Multiprocessor:

- A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
- There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



Applications of Multiprocessor –

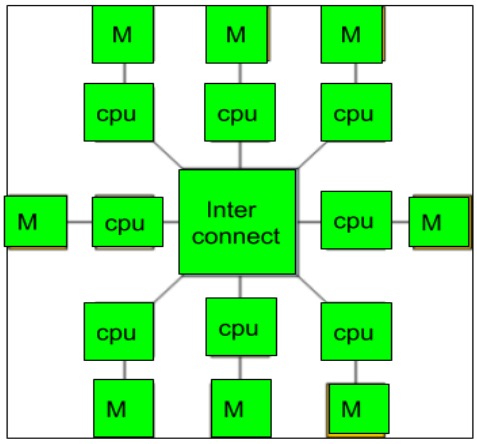
- As a uniprocessor, such as single instruction, single data stream (SISD).
- As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
- Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.
- Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

Benefits of using a Multiprocessor –

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

2. Multicomputer:

- A multicomputer system is a computer system with multiple processors that are connected together to solve a problem. Each processor has its own memory and it is accessible by that particular processor and those processors can communicate with each other via an interconnection network.
- As the multicomputer is capable of messages passing between the processors, it is possible to divide the task between the processors to complete the task. Hence, a multicomputer can be used for distributed computing. It is cost effective and easier to build a multicomputer than a multiprocessor.



Architecture of multicomputer

Difference between multiprocessor and Multicomputer:

- Multiprocessor is a system with two or more central processing units (CPUs) that is capable of performing multiple tasks where as a multicomputer is a system with multiple processors that are attached via an interconnection network to perform a computation task.
- A multiprocessor system is a single computer that operates with multiple CPUs where as a multicomputer system is a cluster of computers that operate as a singular computer.
- Construction of multicomputer is easier and cost effective than a multiprocessor.
- In multiprocessor system, program tends to be easier where as in multicomputer system, program tends to be more difficult.
- Multiprocessor supports parallel computing, whereas, Multicomputer supports distributed computing.

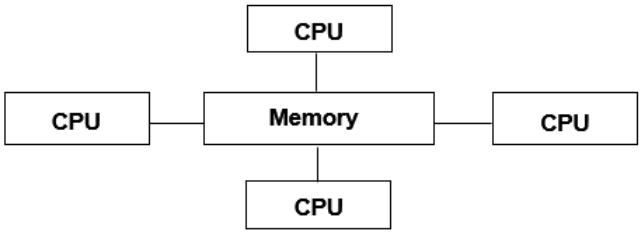
Different categories of multiprocessor:

1. Shared Memory Multiprocessor

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into three categories based upon memory access times: UMA, NUMA and COMA.

a. Uniform Memory Access (UMA):

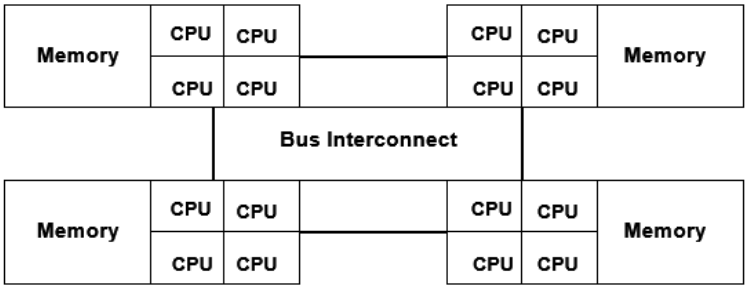
- Most commonly represented today by Symmetric Multiprocessor (SMP) machines.
- Identical processors.
- Equal access and access times to memory.
- Sometimes called CC-UMA - Cache Coherent UMA.
- Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.



Shared Memory (UMA)

b. Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower.
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA



Shared Memory (NUMA)

c. The COMA model (Cache only Memory Access):

- The COMA model is a special case of NUMA machine in which the distributed main memories are converted to caches. All caches form a global address space and there is no memory hierarchy at each processor node.

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that insure " correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter- processor memory.

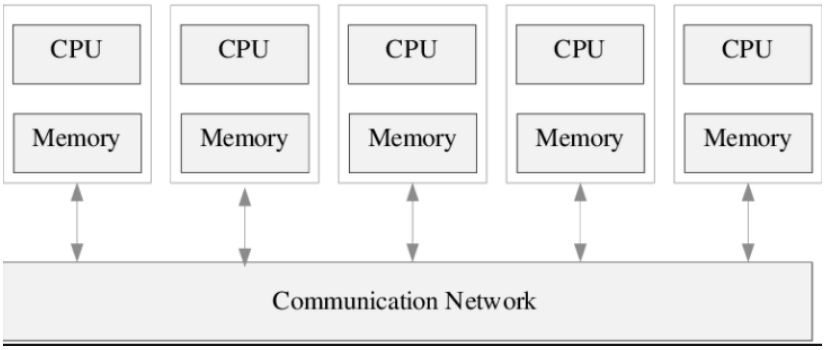


Figure: Distributed Memory Systems

- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently.
- Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- Modern multicomputer use hardware routers to pass message.

Advantages:

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.

Multi-vector and SIMD Computers:

- A vector operand contains an ordered set of n elements, where n is called the length of the vector. Each element in a vector is a scalar quantity, which may be a floating-point number, an integer, a logical value or a character.
- A vector processor consists of a scalar processor and a vector unit, which could be thought of as an independent functional unit capable of efficient vector operations.

Types of Array Processor

Array Processor performs computations on large array of data. These are two types of Array Processors: Attached Array Processor, and SIMD Array Processor. These are explained as following below.

1. Attached Array Processor:

To improve the performance of the host computer in numerical computational tasks auxiliary processor is attached to it.

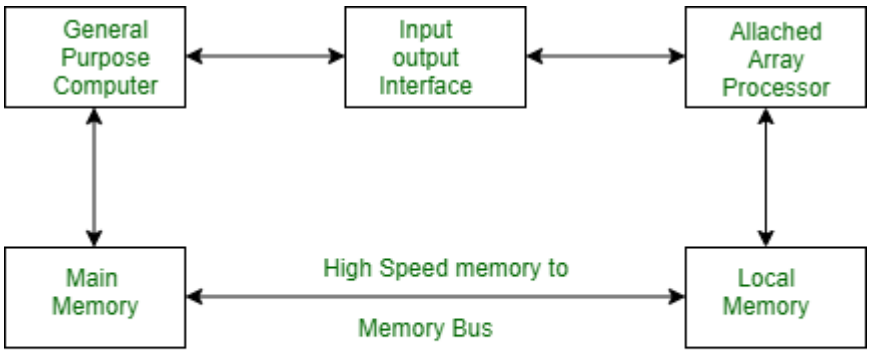


Figure - Interconnection of an attached array Processor to a host computer

Attached array processor has two interfaces:

1. Input output interface to a common processor.
2. Interface with a local memory.
   - Here local memory interconnects main memory. Host computer is general purpose computer. Attached processor is back end machine driven by the host computer.
   - The array processor is connected through an I/O controller to the computer & the computer treats it as an external interface.

2. SIMD array processor:
This is computer with multiple process unit operating in parallel Both types of array processors, manipulate vectors but their internal organization is different.

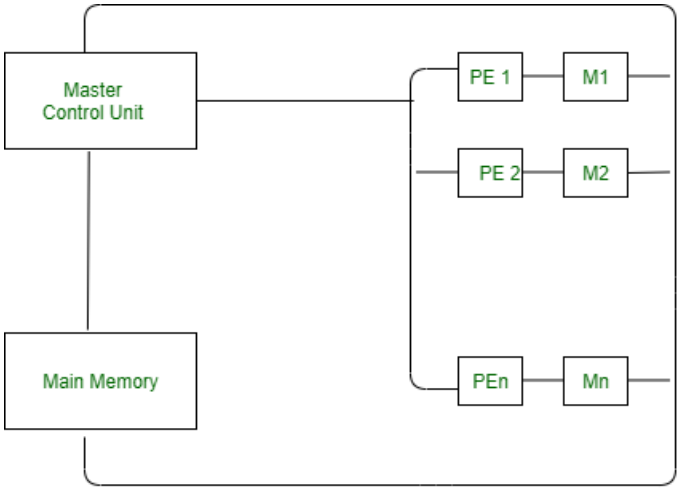


Figure - SIMD Array Processor Organization

- SIMD is a computer with multiple processing units operating in parallel.
- The processing units are synchronized to perform the same operation under the control of a common control unit. Thus, providing a single instruction stream, multiple data stream (SIMD) organization. As shown in figure, SIMD contains a set of identical processing elements (PES) each having a local memory M.

Each PE includes –

- ALU
- Floating point arithmetic unit
- Working registers

Master control unit controls the operation in the PEs. The function of master control unit is to decode the instruction and determine how the instruction to be executed. If the instruction is scalar or program control instruction then it is directly executed within the master control unit.

Main memory is used for storage of the program while each PE uses operands stored in its local memory.

The classical structure of a SIMD array architecture is conceptually simple. In such architectures a program consists of a mixture of scalar and array instructions. The scalar instructions are sent to the scalar processor and the array instructions are broadcast to all array elements in parallel. Array elements are incapable of operating autonomously, and must be driven by the control unit.

There are two important control mechanisms: a *local control* mechanism by which array elements use local state information to determine whether they should execute a broadcast instruction or ignore it, and a *global control* mechanism by which the control unit extracts global information from the array elements to determine the outcome of a conditional control transfer within the user's program. Global information can be extracted in one of two ways. Either the control unit reads state information from one, or a group, of array elements, or it senses a Boolean control line representing the logical OR (or possibly the logical AND) of a particular local state variable from *every* array element.

The three major components of an array structure are the array units, the memory they access, and the connections between the two.

- If all memory is shared then the switch network connecting the array units to the memory must be capable of sustaining a high rate of data transfer, since *every* instruction will require massive movement of data between these two components.
- Alternatively, if the memory is distributed then the majority of operands will hopefully reside within the local memory of each processing element (where processing element = arithmetic unit + memory module), and a much lower performance from the switch network can be tolerated. The design of the switch network is of central importance, a topic is covered in the section on **Networks**.

Figure 1. Classical SIMD Array Architecture

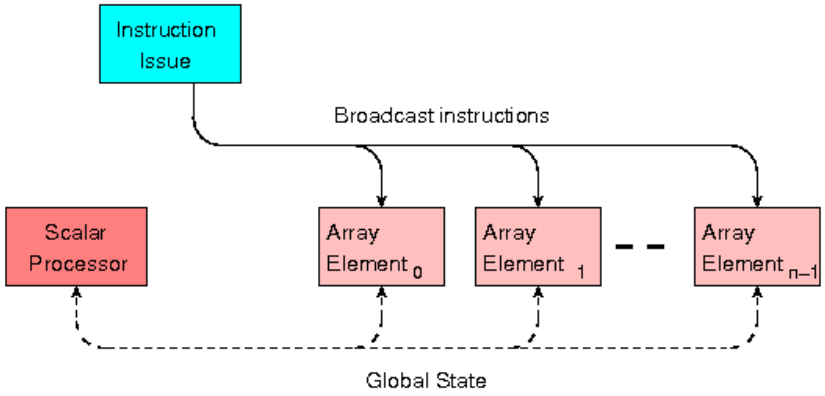


Figure 2. Array Processor with Shared Memory

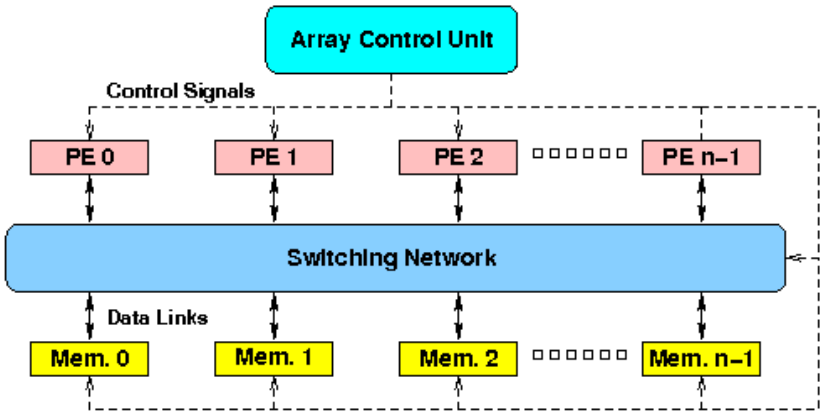
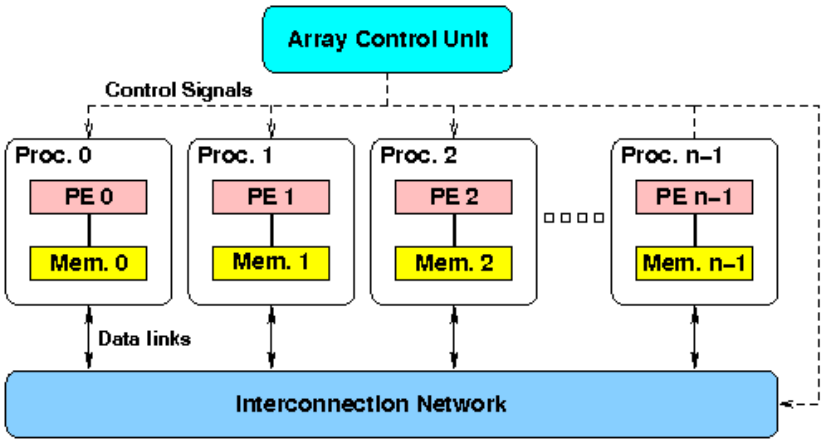


Figure 3. Array Processor with Distributed Memory



Examples of these two styles of array processor architecture were the highly influential ILLIAC IV machine, which had a fully distributed memory, and the ill-fated Burroughs Scientific Processor (BSP), which had a shared memory.

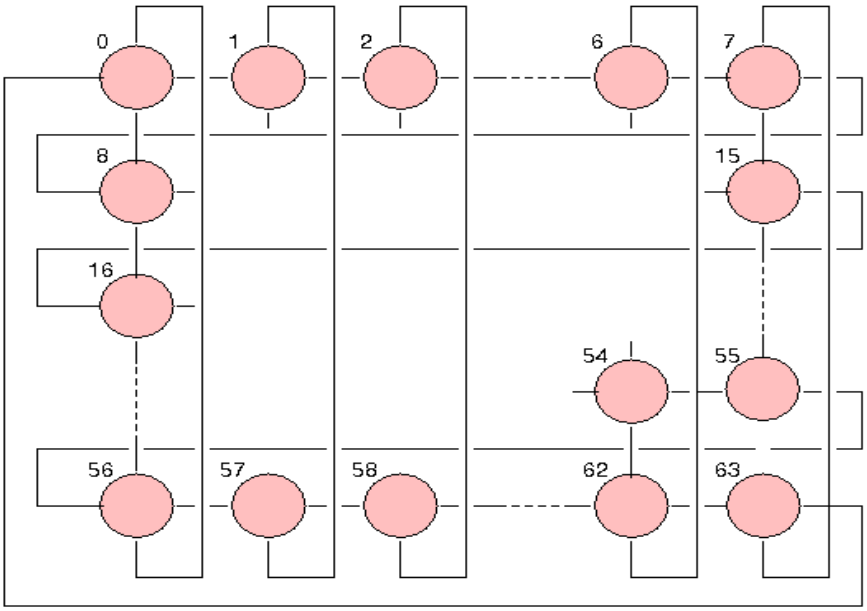
ILLIAC IV

- The ILLIAC IV system was the first real attempt to construct a large-scale parallel machine, and in its time it was the most powerful computing machine in the world. It was designed and constructed by academics and scientists from the University of Illinois and the Burroughs Corporation. A significant amount of software, including sophisticated compilers, was developed for ILLIAC IV, and many researchers were able to develop parallel application software.
- ILLIAC IV grew from a series of ILLIAC machines. Work on ILLIAC IV began in the 1960s, and the machine became operational in 1972. The original aim was to produce a 1 GFLOP machine using an SIMD array architecture comprising 256 processors partitioned into four quadrants, each controlled by an independent control unit. Unfortunately, as is often the case with such ambitious projects, escalating costs and unforeseen engineering problems resulted in just a single quadrant being built. The clock speed of the machine was intended to be 25 MHz but this too had to be reduced to 10 MHz, due partly to signal transmission delays resulting from the machine's large physical dimensions.
- The processors in each quadrant were connected in the topology shown in the figure. Although this looks superficially rather like a square grid of connections it is in fact known as a *chordal ring* (see under Interconnection Networks/Static Networks), due to the shifted wrap-around of the boundary connections. Each inter-processor link consisted of a bi-directional 64-bit wide channel.
- The control unit of ILLIAC IV was responsible for performing scalar operations and issuing SIMD instructions to an array of 64 processing elements. These elements executed instructions in lockstep, although each processing element had the ability to execute instructions conditionally using *local* condition variables. This mechanism whereby processing elements selectively "sit out" instructions makes the machine particularly flexible, and is a feature that has been included in all subsequent SIMD machines. It can even be seen in some vector machines in the form of control vectors.
- Instructions for both the scalar section and the ILLIAC IV array were stored in the 2048 x 64-bit local memories associated with each processing element. These memories were constructed using thin-film storage devices and had access and cycle times of 120 and 240ns respectively. The control unit (CU) interface to these memories was a further example of array parallelism in operation; the data pathway between the CU and the memories was 512 bits wide permitting the

CU to access one 64-bit word from each memory module in one *row* of processing elements concurrently (and at a common address), thus achieving an effective peak memory bandwidth of 1 word every 30 ns.

- Although the actual performance of ILLIAC IV on real applications was only 2 to 4 times that of a CDC 7600, the machine is of significant historical value since it is arguably the origin of all subsequent parallel machines.

ILLIAC-IV processor interconnection topology



Vector Supercomputer

- Vector computers have hardware to perform the vector operations efficiently. Operands cannot be used directly from memory but rather are loaded int o registers and are put back in registers after the operation. Vector hardware has the special ability to overlap or pipeline operand processing. Vector functional unit s pipelined, fully segmented each stage of the pipeline performs a step of the function on different operand(s) once pipeline is full; a new result is produced each clock period (cp).

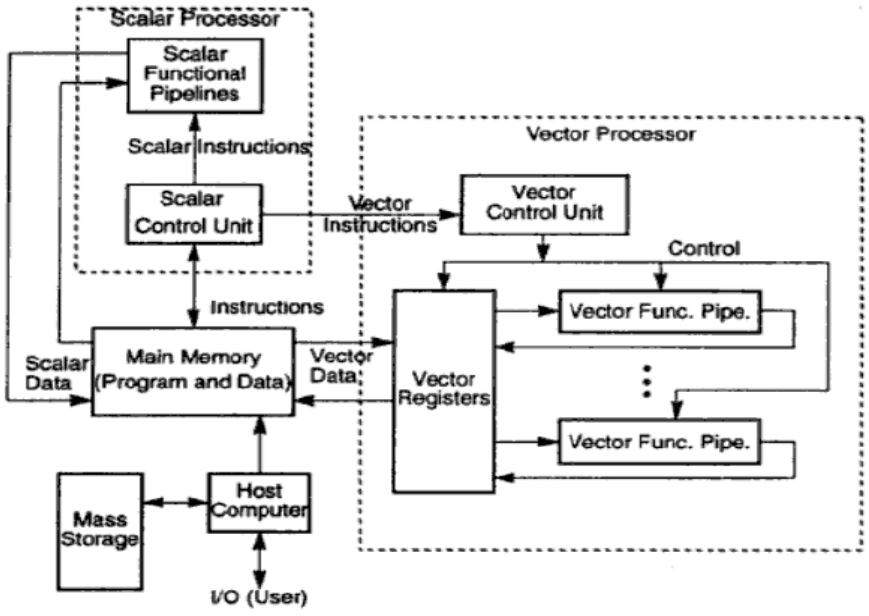


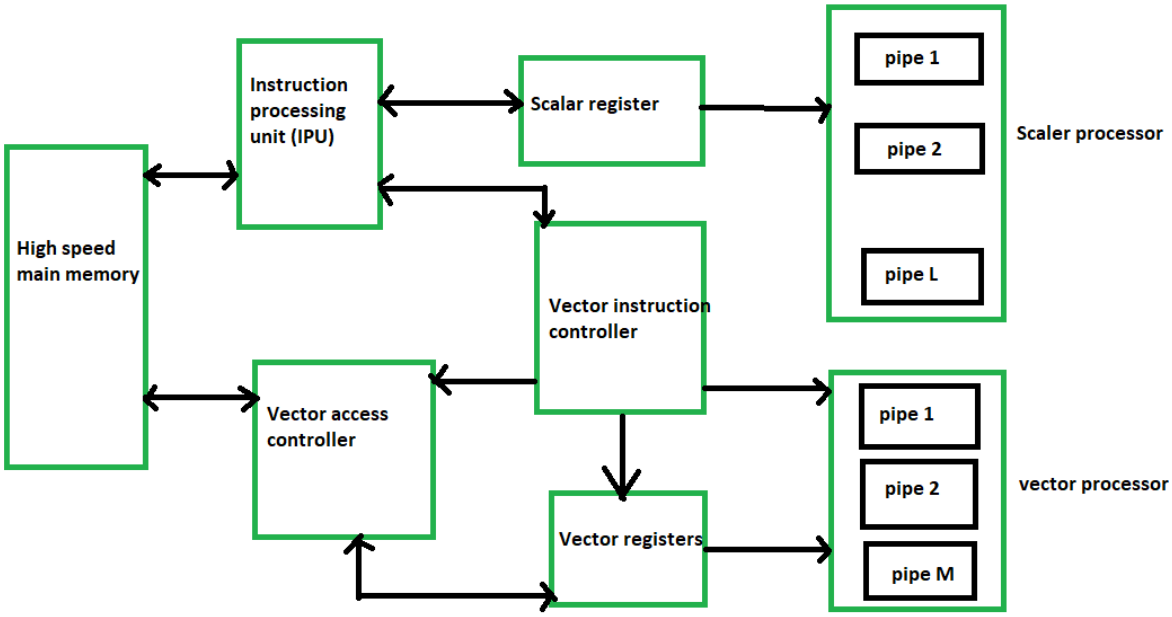
Figure: Architecture of Vector Supercomputer

Vector processor classification

According to from where the operands are retrieved in a vector processor, pipe lined vector computers are classified into two architectural configurations:

1. **Memory to memory architecture:**
   In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from the main memory. For memory to memory vector instructions, the information of the base address, the offset, the increment, and the vector length must be specified in order to enable streams of data transfers between the main memory and pipelines. The processors like *TI-ASC, CDC STAR-100, and Cyber-205* have vector instructions in memory to memory formats. The main points about memory to memory architecture are:
   - There is no limitation of size
   - Speed is comparatively slow in this architecture
2. **Register to register architecture:**
   In register to register architecture, operands and results are retrieved indirectly from the main memory through the use of large number of vector registers or scalar registers. The processors like *Cray-1 and the Fujitsu VP-200* use vector instructions in register to register formats. The main points about register to register architecture are:
   - Register to register architecture has limited size.
   - Speed is very high as compared to the memory to memory architecture.
   - The hardware cost is high in this architecture.

A block diagram of a modern multiple pipeline vector computer is shown below:



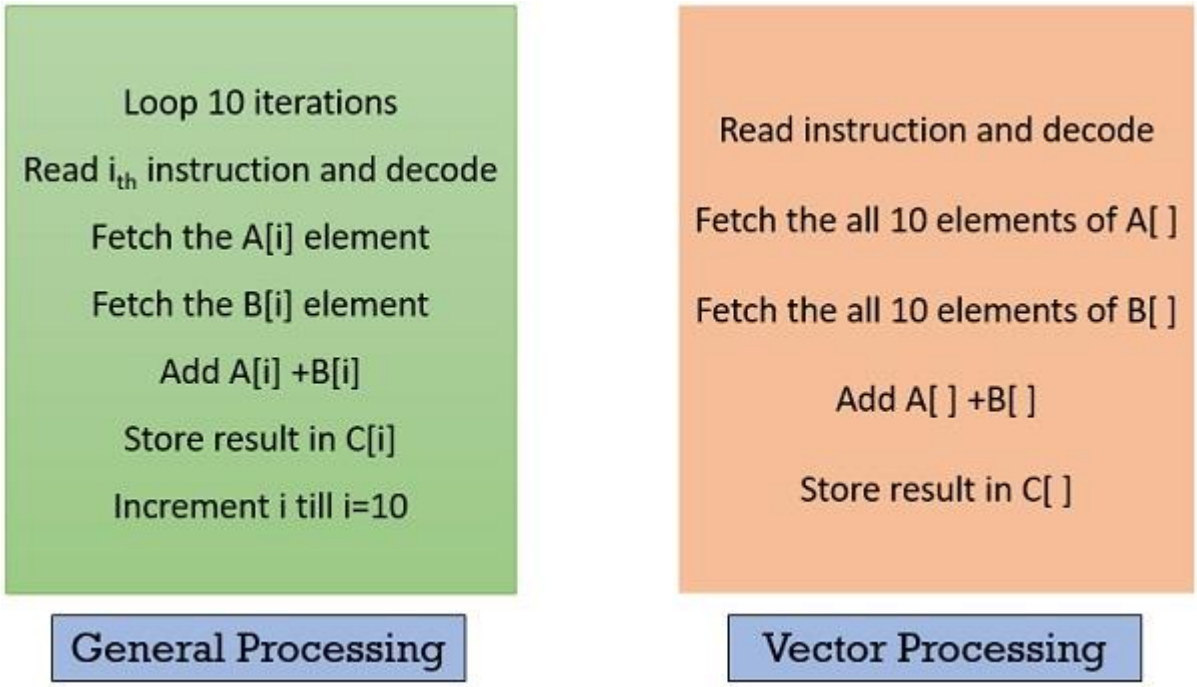
A typical pipe lined vector processor.

VECTOR PROCESSING:

- **Vector processing** performs the arithmetic operation on the large array of integers or floating-point number. Vector processing operates on all the elements of the array in parallel providing each pass is independent of the other.
- Vector processing avoids the **overhead** of the loop control mechanism that occurs in general-purpose computers.
- We need computers that can solve mathematical problems for us which include, arithmetic operations on the large arrays of integers or floating-point numbers quickly. The general-purpose

computer would use loops to operate on an array of integers or floating-point numbers. But, for large array using loop would cause overhead to the processor.

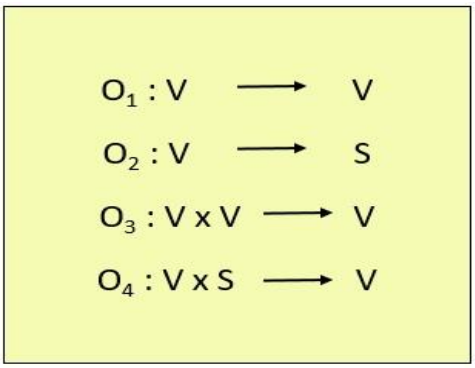
- To avoid the overhead of processing loops and fasten the computation, some kind of parallelism must be introduced. **Vector processing** operates on the entire array in just one operation i.e. it operates on elements of the array in **parallel**. But vector processing is possible only if the operations performed in parallel are **independent**.
- Look at the figure below. Below, instructions in both the blocks are set to add two arrays and store the result in the third array. Vector processing adds both the array in parallel by avoiding the use of the loop.



- Operating on multiple data in just one instruction is also called **Single Instruction Multiple Data** (SIMD) or they are also termed as **Vector instructions**. Now, the data for vector instruction are stored in **vector registers**.
- Each vector register is capable of storing several data elements at a time. These several data elements in a vector register is termed as a **vector operand**. So, if there are n number of elements in a vector operand then n is the **length of the vector**.
- Supercomputers were evolved to deal with billions of floating-point operations/second. Supercomputer optimizes numerical computations (vector computations).
- But, along with vector processing supercomputers are also capable of doing scalar processing. Later, **Array processor** was introduced which particularly deals with vector processing, they do not indulge in scalar processing.

Characteristics of Vector Processing

- Each element of the vector operand is a **scalar quantity** which can either be an integer, floating-point number, logical value or a character. Below we have classified the vector instructions in four types.
- Here, V is representing the vector operands and S represents the scalar operands. In the figure below, O1 and O2 are the unary operations and O3 and O4 are the binary operations.



- Most of the vector instructions are **pipelined** as vector instruction performs the same operation on the different data sets repeatedly. Now, the pipelining has start-up delay, so longer vectors would perform better here.
- The pipelined vector processors can be classified into two types based on from where the operand is being **fetched** for vector processing. The two architectural classifications are Memory-to-Memory and Register-to-Register.

i. In **Memory-to-Memory** vector processor the operands for instruction, the intermediate result and the final result all these are retrieved from the **main memory**. TI-ASC, CDC STAR-100, and Cyber-205 use memory-to-memory format for vector instructions.

ii. In **Register-to-Register** vector processor the source operands for instruction, the intermediate result, and the final result all are retrieved from **vector or scalar registers**. Cray-1 and Fujitsu VP-200 use register-to-register format for vector instructions.

Vector Instruction: A vector instruction has the following fields:

1. Operation Code: Operation code indicates the **operation that** has to be performed in the given instruction. It decides the functional unit for the specified operation or reconfigures the multifunction unit.
2. Base Address: Base address field refers to the **memory location** from where the operands are to be fetched or to where the result has to be stored. The base address is found in the memory reference instructions. In the vector instruction, the operand and the result both are stored in the vector registers. Here, the **base address** refers to the designated **vector register**.
3. Address Increment: A vector operand has several data elements and address increment specifies the **address of the next element in the operand**. Some computer stores the data element consecutively in main memory for which the increment is always 1. But, some computers that do not store the data elements consecutively requires the variable address increment.
4. Address Offset: Address Offset is always specified related to the base address. The effective **memory address** is calculated using the address offset.
5. Vector Length: Vector length specifies the **number of elements in a vector operand**. It identifies the **termination** of a vector instruction.

Improving Performance

- In vector processing, we come across two overheads setup time and flushing time. When the vector processing is pipelined, the time required to *route the vector operands* to the *functional unit* is called **Set up time**. **Flushing time** is a *time duration* that a vector instruction takes right from its *decoding* until its *first result is out* from the pipeline.
- The vector length also affects the efficiency of processing as the longer vector length would cause overhead of subdividing the long vector for processing.
- For obtaining the better performance the optimized object code must be produced in order to utilize pipeline resources to its maximum.

1. Improving the vector instruction: We can improve the vector instruction by reducing the memory access, and maximize resource utilization.
2. Integrate the scalar instruction: The scalar instruction of the same type must be integrated as a batch. As it will reduce the overhead of reconfiguring the pipeline again and again.
3. Algorithm: Choose the algorithm that would work faster for vector pipelined processing.
4. Vectorizing Compiler: A vectorizing compiler must regenerate the parallelism by using the higher-level programming language. In advance programming, the four-stage are identified in the development of the parallelism. Those are:

- Parallel Algorithm(A)
- High-level Language(L)
- Efficient object code(O)
- Target machine code (M)

You can see a parameter in the parenthesis at each stage which denotes the degree of parallelism. In the ideal situation, the parameters are expected in the order $A \geq L \geq O \geq M$.

Key Notes:

- Computers having vector instruction are vector processors.
- Vector processor have the vector instructions which operates on the large array of integer or floating-point numbers or logical values or characters, all elements in parallel. It is called **vectorization**.
- Vectorization is possible only if the operation performed in parallel are **independent** of each other.
- Operands of vector instruction are stored in the **vector register**. A vector register stores several data elements at a time which is called **vector operand**.
- A vector operand has several **scalar data elements**.
- A vector instruction needs to perform the same operation on the different data set. Hence, vector processors have a **pipelined** structure.
- Vector processing ignores the overhead caused due to the loops while operating on an array.
- So, this is how vector processing allows parallel operation on the large arrays and fasten the processing speed.

Module-3
Interconnection network
System interconnect architecture

- Various types of interconnection networks have been suggested for SIMD computers. These are basically classified have been classified on network topologies into two categories namely

i. Static Networks
ii. Dynamic Networks

- The topological structure of SIMD array processor is mainly characterized by the data routing network used in the interconnecting the processing elements.

Network properties and routing

- The goals of an interconnection network are to provide low-latency high data transfer rate wide communication bandwidth.
- Analysis includes latency, bisection bandwidth, data-routing functions and scalability of parallel architecture.
- These Network usually represented by a graph with a finite number of nodes linked by directed or undirected edges.
    - Number of nodes in graph = network size.
    - Number of edges (links or channels) incident on a node = node degree d (also note in and out degrees when edges are directed).
    - Node degree reflects number of I/ O ports associated with a node, and should ideally be small and constant.
- Network is symmetric if the topology is the same looking from any node; these are easier to implement or to program.

Other Characteristics are:

i. Diameter: The maximum distance between any two processors in the network or in other words it is the maximum number of (routing) processors through which a message must pass on its way from source to reach destination. Thus, diameter measures the maximum delay for transmitting a message from one processor to another as it determines communication time hence, smaller the diameter better will be the network topology.

ii. Connectivity: It states how many paths are possible between any two processors i.e., the multiplicity of paths between two processors. Higher connectivity is desirable as it minimizes contention.

- Arc connectivity of the network: The minimum number of arcs that must be removed for the network to break it into two disconnected networks. The arc connectivity of various network are as follows:
    - 1 for linear arrays and binary trees
    - 2 for rings and 2-d meshes
    - 4 for 2-d torus
    - d for d-dimensional hypercubes
    - Larger the arc connectivity lesser the conjunctions and better will be network topology.
- Channel width: The channel width is the number of bits that can communicated simultaneously by a interconnection bus connecting two processors.

iii. Bisection Width and Bandwidth:

- To divide the network into equal halves it is required to remove some communication links. The minimum numbers of such communication links that have to be removed are called the Bisection Width.
- Bisection width basically provide the information about the largest number of

messages which can be sent simultaneously (without needing to use the same wire or routing processor at the same time and so delaying one another), no matter which processors are sending to which other processors. Thus, larger the bisection width is the better the network topology is considered.

- Bisection Bandwidth is the minimum volume of communication allowed between two halves of the network with equal numbers of processors. This is important for the networks with weighted arcs where the weights correspond to the link width i.e., (how much data it can transfer).
- Cost the cost of networking can be estimated on variety of criteria where we consider the number of communication links or wires used to design the network as the basis of cost estimation, smaller the better the cost.

iv. Data Routing Functions: A data routing network is used for inter-PE (Processing Element) data exchange. It can be static as in case of hypercube routing network or dynamic such as multistage network. Various type of data routing functions are: Shifting, Rotating, Permutation (one to one), Broadcast (one to all), Multicast (many to many), Personalized broadcast (one to many), Shuffle, Exchange etc.

Factors Affecting Performance

- Functionality: It describes, how the network supports data routing, interrupt handling, synchronization, request/ message combining, and coherence.
- Network latency: It represents worst-case time for a unit message to be transferred.
- Bandwidth: It is the maximum data rate.
- Hardware complexity: It is determined as implementation costs for wire, logic, switches, connectors, etc.
- Scalability: It states that how easily does the scheme adapt to an increasing number of processors, memories, etc.

Static interconnection networks

- Static interconnection networks for elements of parallel systems (ex: processor, memories) are based on fixed connections that cannot be modified without a physical re-designing of a system.
- Static interconnection networks can have many structures such as a linear structure (pipeline), a matrix, a ring, a torus, a complete connection structure, a tree, a star, a hyper-cube.
- In linear and matrix structures, processors are interconnected with their neighbors in a regular structure on a plane. A torus is a matrix structure in which elements at the matrix borders are connected in the frame of the same lines and columns. In a complete connection structure, all elements (ex. processors) are directly interconnected (point-to-point)

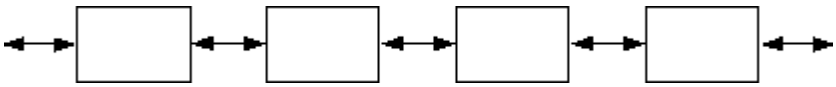


Figure: Linear structure (pipeline) of interconnections in a parallel system

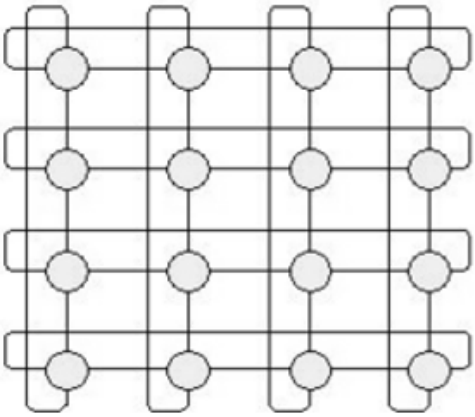


Figure: 2D Torus

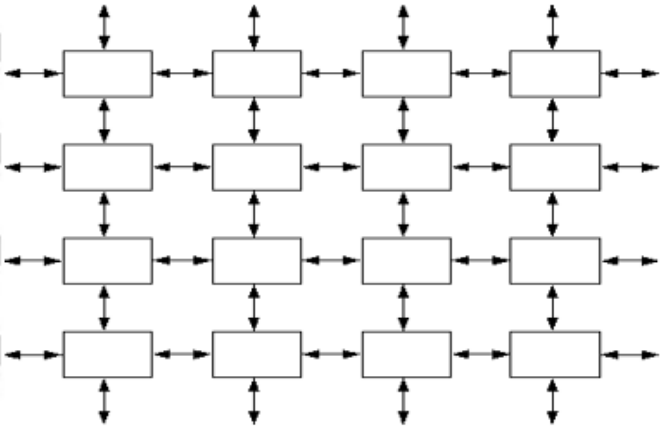
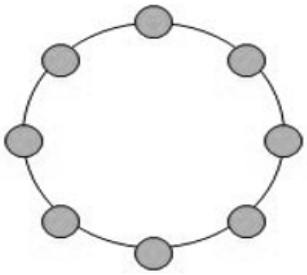
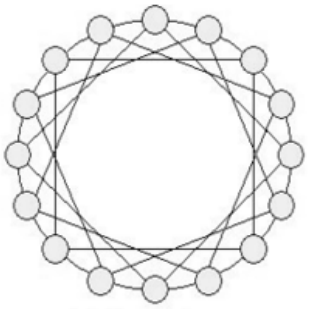


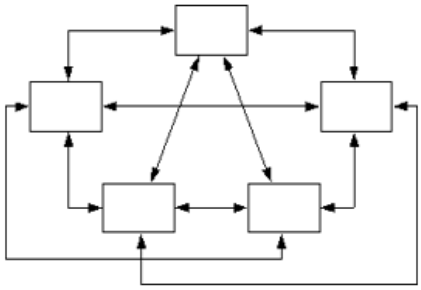
Figure: Matrix



A Ring

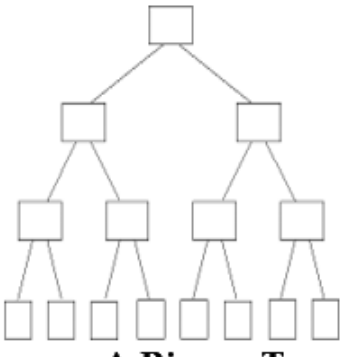


A Chordal Ring

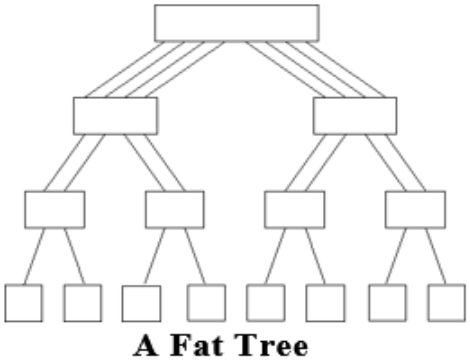


A complete interconnection

- In a tree structure, system elements are set in a hierarchical structure from the root to the leaves, as shown in the figure below. All elements of the tree (nodes) can be processors or only leaves are processors and the rest of nodes are linking elements, which intermediate in transmissions. If from one node, 2 or more connections go to different nodes towards the leaves - we say about a binary or k-nary tree. If from one node, more than one connection goes to the neighboring node, we speak about a fat tree. A binary tree, in which in the direction of the root, the number of connections between neighboring nodes increases twice, provides a uniform transmission throughput between the tree levels, a feature not available in a standard tree.



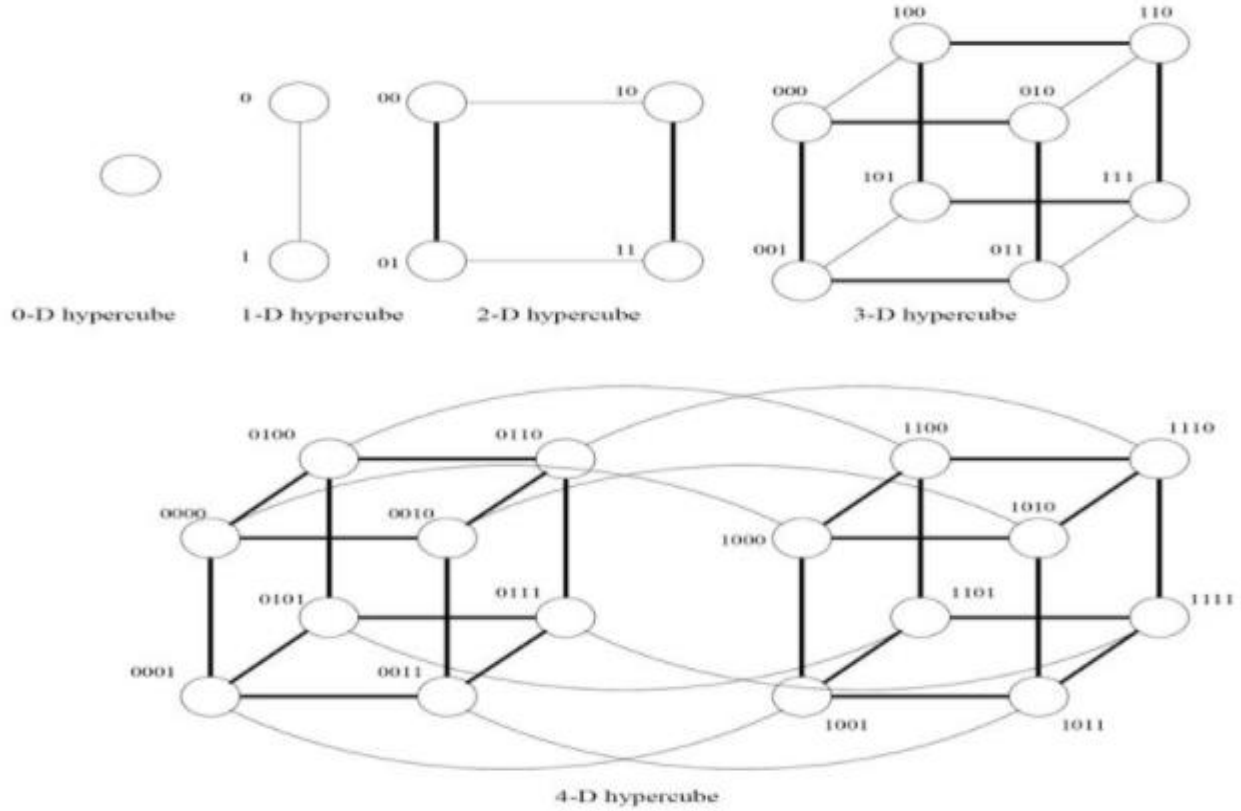
A Binary Tree



A Fat Tree

- In a hypercube structure, processors are interconnected in a network, in which connections between processors correspond to edges of an n-dimensional cube. The hypercube structure is very advantageous since it provides a low network diameter equal to the degree of the cube. The network diameter is the number of edges between the most distant nodes. The network diameter determines the number in intermediate transfers that have to be done

to send data between the most distant nodes of a network. In this respect the hypercubes have very good properties, especially for a very large number of constituent nodes. Due to this, hypercubes are popular networks in existing parallel systems.



Hypercube Interconnection Network

Dynamic interconnection networks

- Dynamic interconnection networks between processors enable changing (reconfiguring) of the connection structure in a system. It can be done before or during parallel program execution. So, we can speak about static or dynamic connection reconfiguration.
- The dynamic networks are those networks where the route through which data move from one PE to another is established at the time communication has to be performed. Usually all processing elements are equidistant and an interconnect ion path is established when two processing elements want to communicate by use of switches. Such systems are more difficult to expand as compared to static network. Examples: Bus-based, Crossbar, Multistage Networks. Here the Routing is done by comparing the bit-level representation of source and destination addresses. If there is a match goes to next stage via pass-through else in case of it mismatch goes via cross-over using the switch.
- There are two classes of dynamic networks namely
    - single stage network
    - multi stage

i. Single Stage Networks

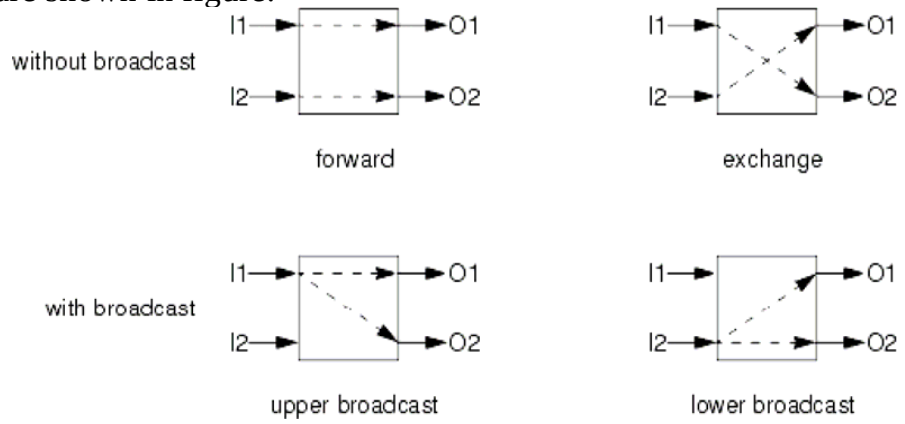
- A single stage switching network with N input selectors (IS) and N output selectors (OS).
- Here at each network stage there is a 1- to-D demultiplexer corresponding to each IS such that 1<D<N and each OS is an M -to-1 multiplexer such that 1<M <=N.
- Cross bar network is a single stage network with D=M =N. In order to establish a desired

connecting path different path control signals will be applied to all IS and OS selectors.

- The single stage network is also called as re-circulating network as in this network connection the single data items may have to re-circulate several times through the single stage before reaching their final destinations. The number of recirculation depends on the connectivity in the single stage network.
- In general, higher the hardware connectivity the lesser is the number of recirculation. In cross bar network only one circulation is needed to establish the connection path. The cost of completed connected cross bar network is O(N2) which is very high as compared to other most re-circulating networks which have cost O (N log N) or lower hence are more cost effective for large value of N.

ii. Multistage Networks

- Many stages of interconnected switches form a multistage SIMD network. It basically consists of three characteristic features
    - The switch box,
    - The network topology
    - The control structures
- Many stages of interconnected switches form a multistage SIMD network. Each box is essentially an interchange device with two inputs and two outputs. The four possible states of a switch box are straight, exchange, Upper Broadcast and Lower broadcast which are shown in figure.



A two-by-two switching box and its four interconnection states

- A two-function switch can assume only two possible state namely state or exchange states. However, a four-function switch box can be any of four possible states.
- A multistage network is capable of connecting any input terminal to any output terminal. Multi-stage networks are basically constructed by so called shuffle-exchange switching element, which is basically a 2 x 2 crossbar. Multiple layers of these elements are connected and form the network.
- A multistage network is capable of connecting an arbitrary input terminal to an arbitrary output terminal.
- Generally, it consists of n stages where $N = 2^n$ is the number of input and output lines and each stage use N/2 switch boxes. The interconnection patterns from one stage to another stage are determined by network topology. Each stage is connected to the next stage by at least N paths. The total wait time is proportional to the number stages i.e., n and the total cost depend on the tot al number of switches used and that are $N\log_2 N$.
- The control structure can be individual stage control i.e., the same control signal is used to set all switch boxes in the same stages thus we need n control signal. The second control structure is individual box control where a separate control signal is used to set the state of each switch box. This provide flexibility at the same time require n2/2

control signal which increases the complexity of the control circuit. In between path is use of partial stage control.

i. Bus networks

- A bus is the simplest type of dynamic interconnection networks. It constitutes a common data transfer path for many devices. Depending on the type of implemented transmissions we have serial busses and parallel busses. The devices connected to a bus can be processors, memories, I/ O units, as shown in the figure below.

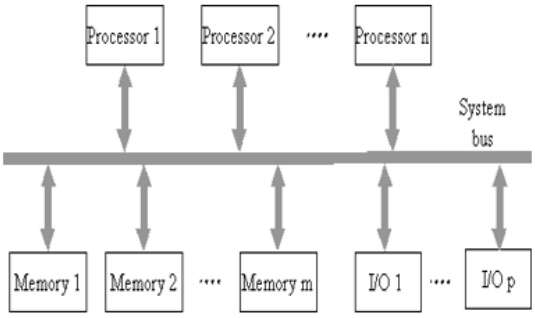


Figure: A system based on a single bus

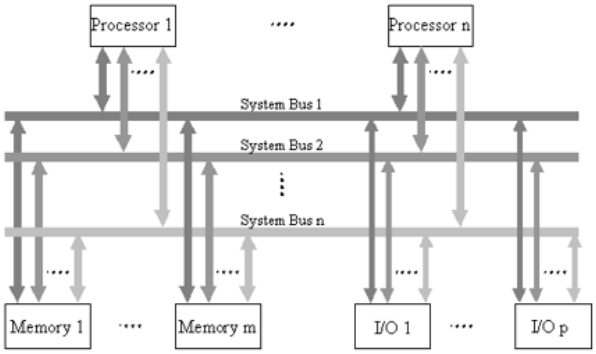


Figure: A system based on a multi-bus

- Only one device connected to a bus can transmits data. M any devices can receive data. In the last case we speak about a multicast transmission. If data are meant for all devices connected to a bus means a broadcast transmission. Accessing the bus must be synchronized. It is done with the use of two methods: a token method and a bus arbiter method. With the token method, a token (a special control message or signal) is circulating between the devices connected to a bus and it gives the right to transmit to the bus to a single device at a time. The bus arbiter receives data transmission requests from the devices connected to a bus. It selects one device according to a selected strategy (ex. using a system of assigned priorities) and sends an acknowledge message (signal) to one of the requesting devices that grants it the transmitting right. After the selected device completes the transmission, it informs the arbiter that can select another request. The receiver (s) address is usually given in the header of the message. Special header values are used for the broadcast and multicasts. All receivers read and decode headers. These devices that are specified in the header, read-in the data transmitted over the bus.
- The throughput of the network based on a bus can be increased by the use of a multi-bus network. In this network, processors connected to the busses can transmit data in parallel (one for each bus) and many processors can read data from many busses at a time.

ii. Crossbar switches

- A crossbar switch is a circuit that enables many interconnect ions between elements of a parallel system at a time. A crossbar switch has a number of input and output data pins and a number of control pins. In response to control instructions set to it's control input, the crossbar switch implements a stable connection of a determined input with a determined output.
- The diagrams of a typical crossbar switch are shown in the figure below.

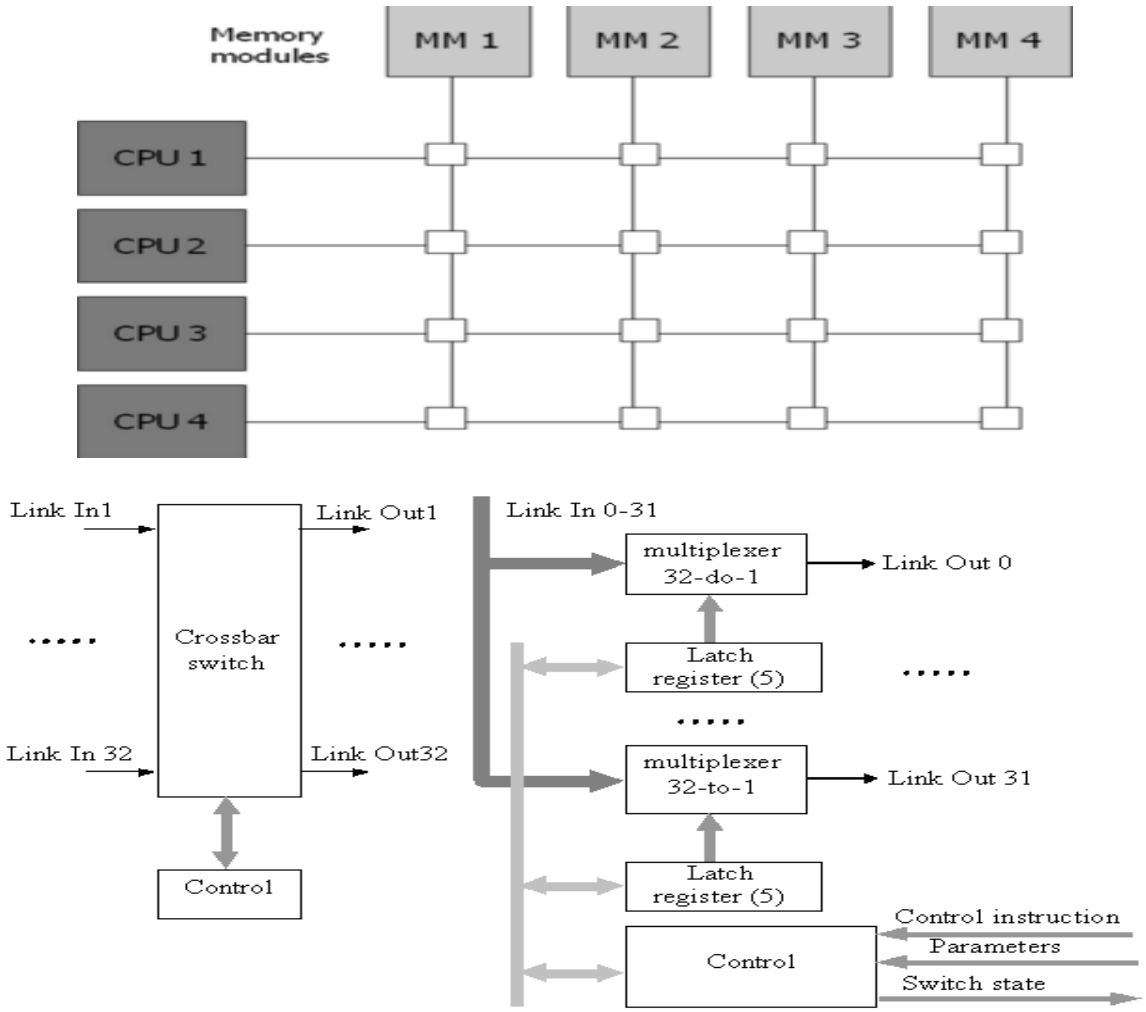


Figure: Crossbar switch a) general scheme, b) internal structure

- Control instructions can request reading the state of specified input and output pins i.e. their current connections in a crossbar switch.
- Crossbar switches are built with the use of multiplexer circuits, controlled by latch registers, which are set by control instruct ions.
- Crossbar switches implement direct, single non-blocking connections, but on the condition that the necessary input and output pins of the switch are free. The connections between free pins can always be implemented independently on the status of other connect ions. New connections can be set during data transmissions through other connections.
- The non-blocking connections are a big advantage of crossbar switches. Some crossbar switches enable broadcast transmissions but in a blocking manner for all other connections.
- The disadvantage of crossbar switches is that extending their size, in the sense of the number of input/ output pins, is costly in terms of hardware. Because of that, crossbar switches are built up to the size of 100 input/ output pins.

iii. Multiport Memory

- In the multiport memory system, different memory module and CPUs have separate buses.
- The module has internal control logic to determine port which will access to memory at any given time. Priorities are assigned to each memory port to resolve memory access conflicts.

Advantages:

- Because of the multiple paths high transfer rate can be achieved.

Disadvantages:

- It requires expensive memory control logic and a large number of cables and connections.

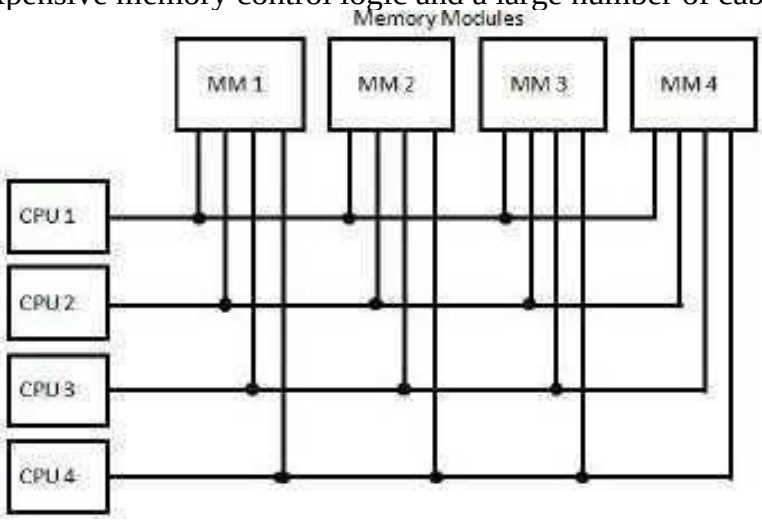


Figure: Multiport memory organization Multistage and combining networks

- Multistage connection networks are designed with the use of small elementary crossbar switches (usually they have two inputs) connected in multiple layers. The elementary crossbar switches can implement 4 types of connections: straight, crossed, upper broadcast and lower broadcast. All elementary switches are controlled simultaneously. The network like this is an alternative for crossbar switches if we have to switch a large number of connect ions, over 100. The extension cost for such a network is relatively low.
- In such networks, there is no full freedom in implementing arbitrary connections when some connections have already been set in the switch. Because of this property, these networks belong to the category of blocking networks.
- However, if we increase the number of levels of elementary crossbar switches above the number necessary to implement connections for all pairs of inputs and outputs, it is possible to implement all requested connections at the same time but statically, before any communication is started in the switch. It can be achieved at the cost of additional redundant hardware included into the switch.
- The block diagram of such a network, called the Benes network, is shown in the figure below.

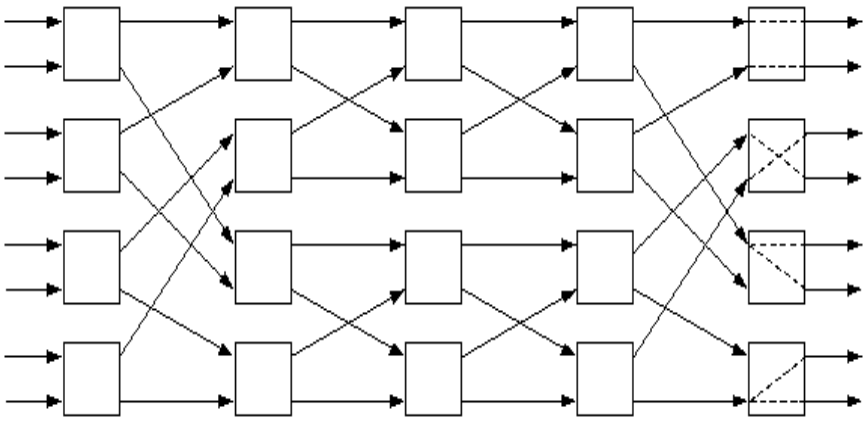


Figure: A multistage connection network for parallel systems

- To obtain nonblocking properties of the multistage connection network, the redundancy level in the circuit should be much increased.

- To build a nonblocking multistage network n x n, the elementary two-input switches have to be replaced by 3 layers of switches n x m, r x r and m x n, where $m^3=2n - 1$ and r is the number of elementary switches in the layer 1 and 3. Such a switch was designed by a French mathematician Clos and it is called the Clos network (also referred as Omega Network). This switch is commonly used to build large integrated crossbar switches.
- The block diagram of the Clos network is shown in the figure below.

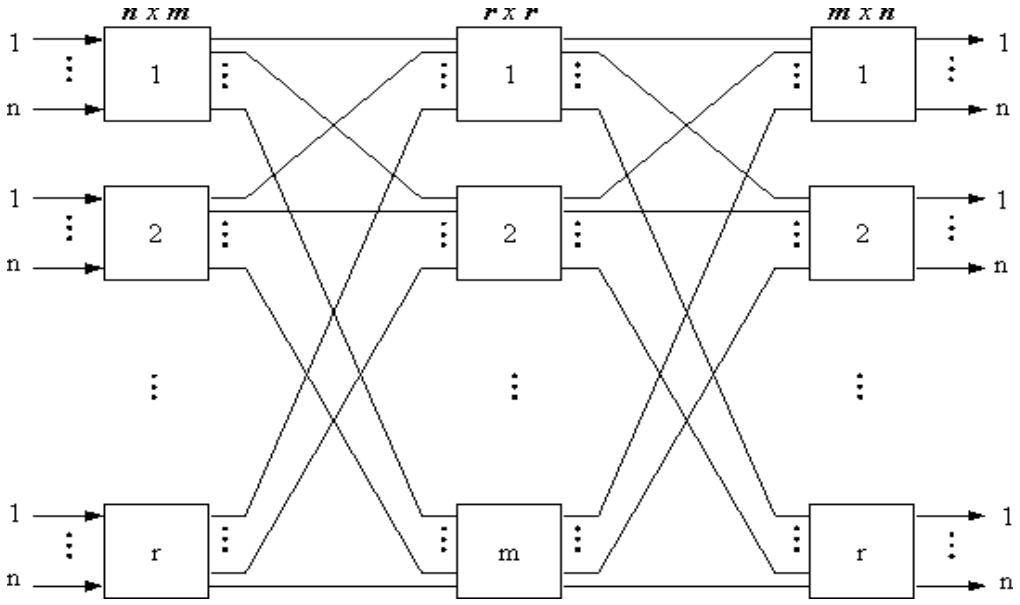


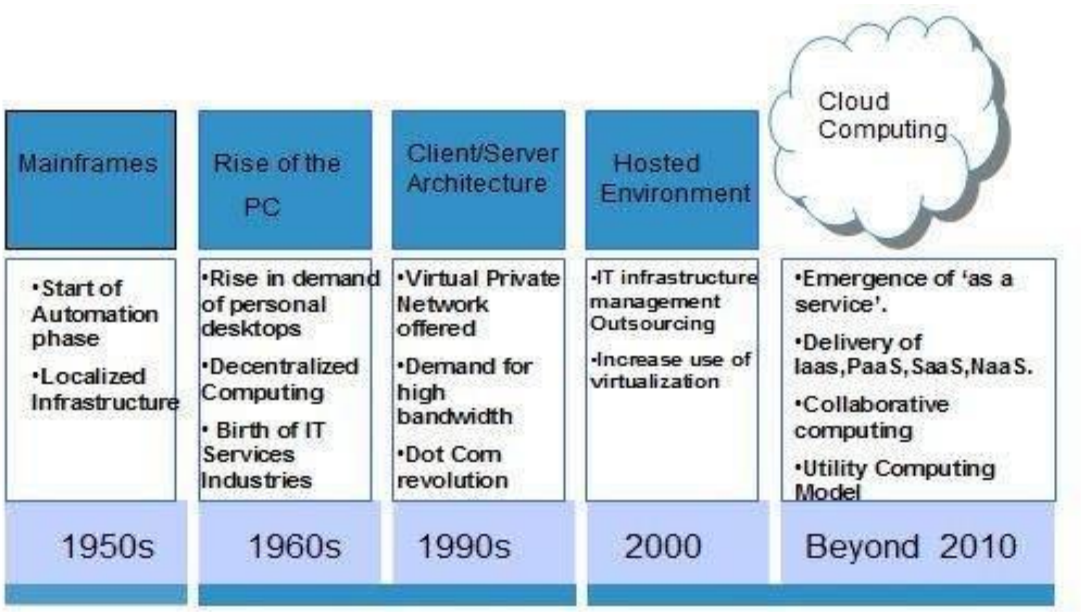
Figure: A nonblocking Clos interconnection network

Cloud Computing

- Cloud Computing provides us means by which we can access the applications as utilities over the internet. It allows us to create, configure, and customize the business applications online.
- **Cloud:** The term Cloud refers to a Network or Internet. In other words, we can say that Cloud is something, which is present at remote location. Cloud can provide services over public and private networks, i.e., WAN, LAN or VPN.
- Applications such as e-mail, web conferencing, customer relationship management (CRM) execute on cloud.
- **Cloud Computing:** Cloud Computing refers to manipulating, configuring, and accessing the hardware and software resources remotely. It offers online data storage, infrastructure, and application.

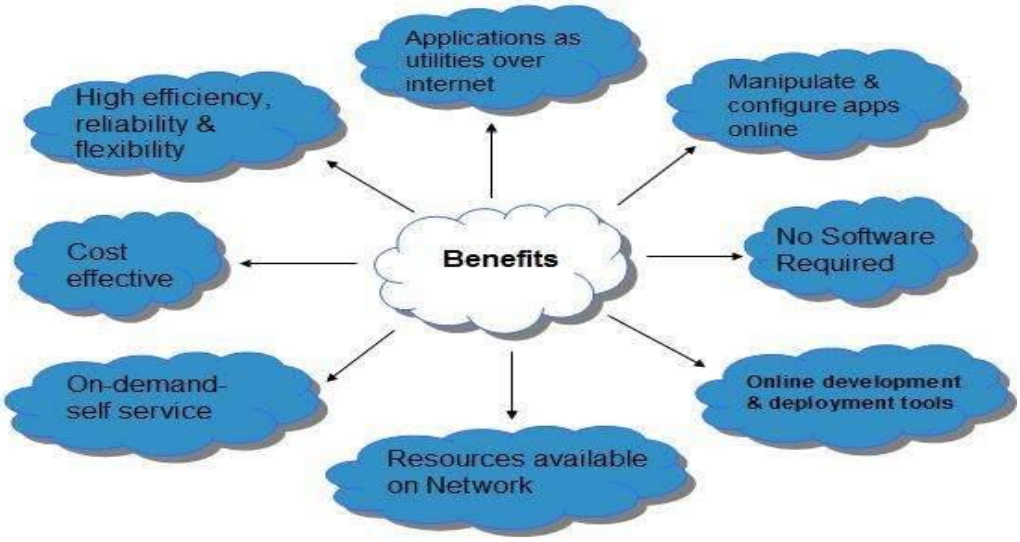
History of Cloud Computing

- The concept of **Cloud Computing** came into existence in the year 1950 with implementation of mainframe computers, accessible via **thin/static clients.** Since then, cloud computing has been evolved from static clients to dynamic ones and from software to services.
- The following diagram explains the evolution of cloud computing:



Benefits: Cloud Computing has numerous advantages. Some of them are listed below -

- One can access applications as utilities, over the Internet.
- One can manipulate and configure the applications online at any time.
- It does not require to install a software to access or manipulate cloud application.
- Cloud Computing offers online development and deployment tools, programming runtime environment through **PaaS model.**
- Cloud resources are available over the network in a manner that provide platform independent access to any type of clients.
- Cloud Computing offers **on-demand self-service.** The resources can be used without interaction with cloud service provider.
- Cloud Computing is highly cost effective because it operates at high efficiency with optimum utilization. It just requires an Internet connection
- Cloud Computing offers load balancing that makes it more reliable.



Risks related to Cloud Computing

Although cloud Computing is a promising innovation with various benefits in the world of computing, it comes with risks. Some of them are discussed below:

Security and Privacy: It is the biggest concern about cloud computing. Since data management and infrastructure management in cloud is provided by third-party, it is always a risk to handover the sensitive information to cloud service providers.

Although the cloud computing vendors ensure highly secured password protected accounts, any sign of security breach may result in loss of customers and businesses.

Lock In: It is very difficult for the customers to switch from one Cloud Service Provider (CSP) to another. It results in dependency on a particular CSP for service.

Isolation Failure: This risk involves the failure of isolation mechanism that separates storage, memory, and routing between the different tenants.

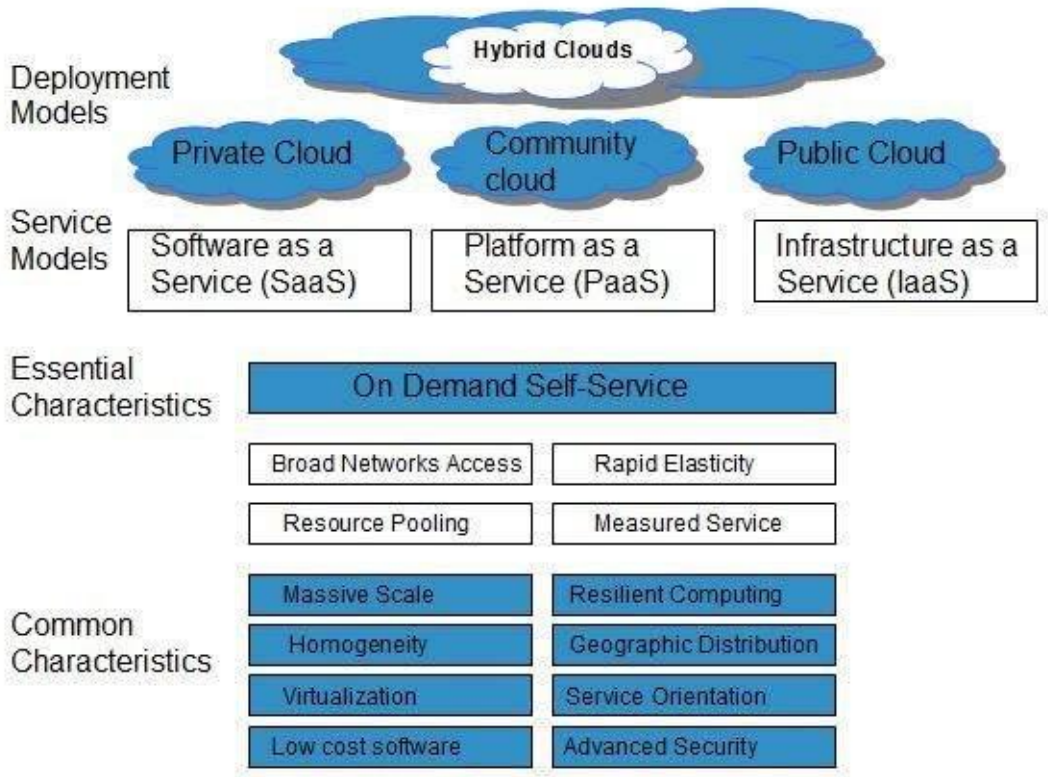
Management Interface Compromise: In case of public cloud provider, the customer management interfaces are accessible through the Internet.

Insecure or Incomplete Data Deletion: It is possible that the data requested for deletion may not get deleted. It happens because either of the following reasons

- Extra copies of data are stored but are not available at the time of deletion
- Disk that stores data of multiple tenants is destroyed.

Characteristics of Cloud Computing

There are four key characteristics of cloud computing. They are shown in the following diagram:



On Demand Self Service

- Cloud Computing allows the users to use web services and resources on demand. One can logon to a website at any time and use them.

Broad Network Access

- Since cloud computing is completely web based, it can be accessed from anywhere and at any time.

Resource Pooling

- Cloud computing allows multiple tenants to share a pool of resources. One can share single physical instance of hardware, database and basic infrastructure.

Rapid Elasticity

- It is very easy to scale the resources vertically or horizontally at any time. Scaling of resources means the ability of resources to deal with increasing or decreasing demand.
- The resources being used by customers at any given point of time are automatically monitored.

Measured Service

- In this service cloud provider controls and monitors all the aspects of cloud service. Resource optimization, billing, and capacity planning etc. depend on it.

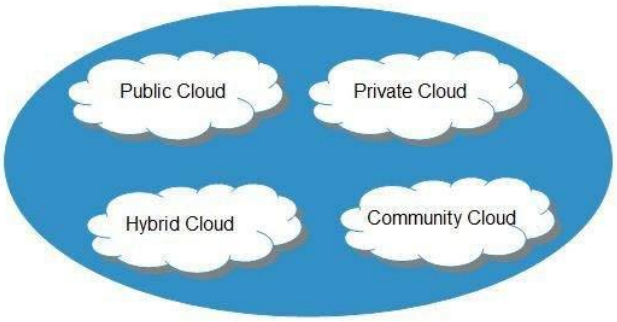
Basic working models of cloud computing

There are certain services and models working behind the scene making the cloud computing feasible and accessible to end users. Following are the working models for cloud computing:

- Deployment Models
- Service Models

Deployment Models

Deployment models define the type of access to the cloud, i.e., how the cloud is located? Cloud can have any of the four types of access: Public, Private, Hybrid, and Community.



I) Public Cloud: The public cloud allows systems and services to be easily accessible to the general public. Public cloud may be less secure because of its openness.

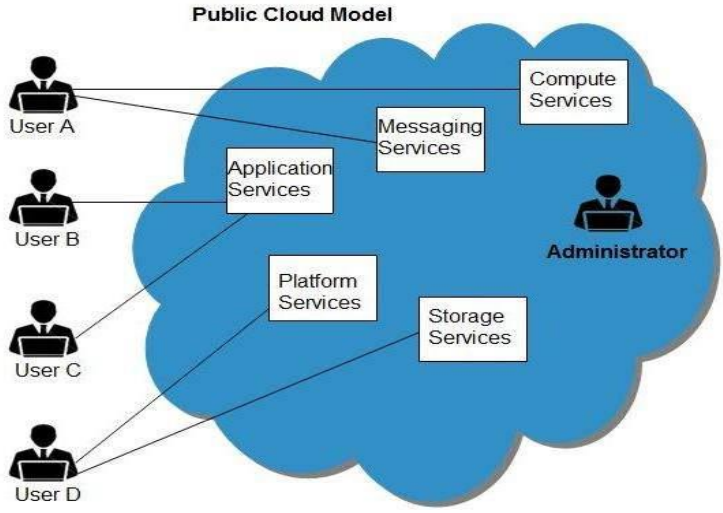
II) Private Cloud: The private cloud allows systems and services to be accessible within an organization. It is more secured because of its private nature.

III) Community Cloud: The community cloud allows systems and services to be accessible by a group of organizations.

IV) Hybrid Cloud: The hybrid cloud is a mixture of public and private cloud, in which the critical activities are performed using private cloud while the non-critical activities are performed using public cloud.

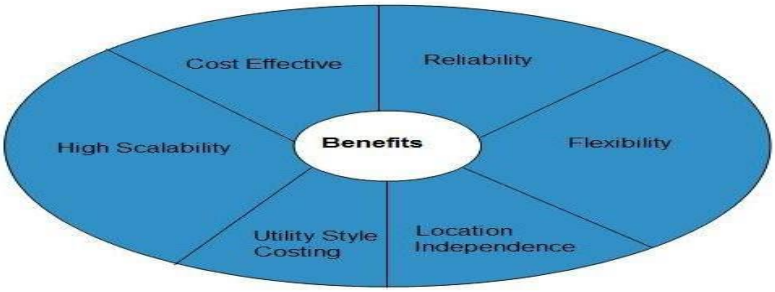
Public Cloud Model

Public Cloud allows systems and services to be easily accessible to general public. The IT giants such as **Google, Amazon** and **Microsoft** offer cloud services via Internet. The Public Cloud Model is shown in the diagram below.



Benefits

There are many benefits of deploying cloud as public cloud model. The following diagram shows some of those benefits:



Cost Effective: Since public cloud shares same resources with large number of customers it turns out inexpensive.

Reliability: The public cloud employs large number of resources from different locations. If any of the resources fails, public cloud can employ another one.

Flexibility: The public cloud can smoothly integrate with private cloud, which gives customers a flexible approach.

Location Independence: Public cloud services are delivered through Internet, ensuring location independence.

Utility Style Costing: Public cloud is also based on pay-per-use model and resources are accessible whenever customer needs them.

High Scalability: Cloud resources are made available on demand from a pool of resources, i.e., they can be scaled up or down according the requirement.

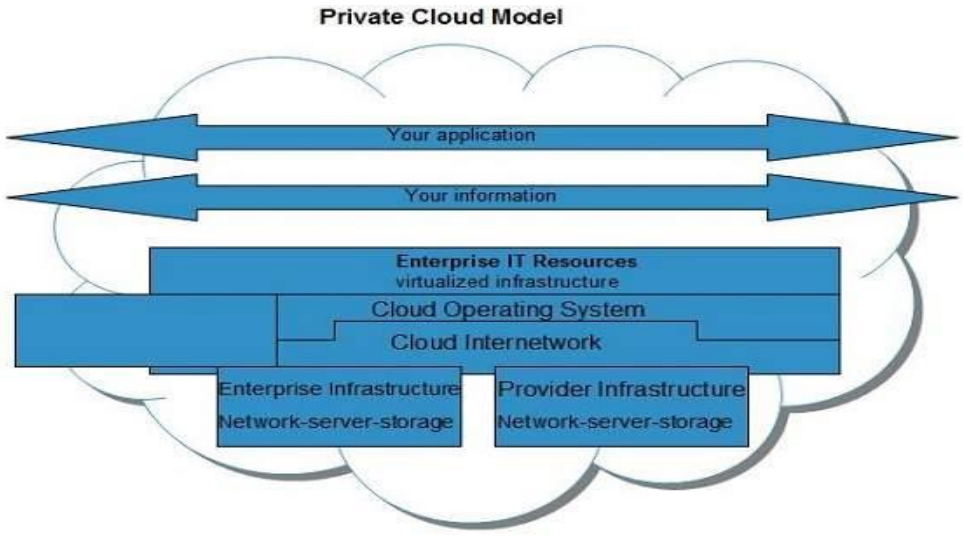
Disadvantages: Here are some disadvantages of public cloud model:

Low Security: In public cloud model, data is hosted off-site and resources are shared publicly, therefore does not ensure higher level of security.

Less Customizable: It is comparatively less customizable than private cloud.

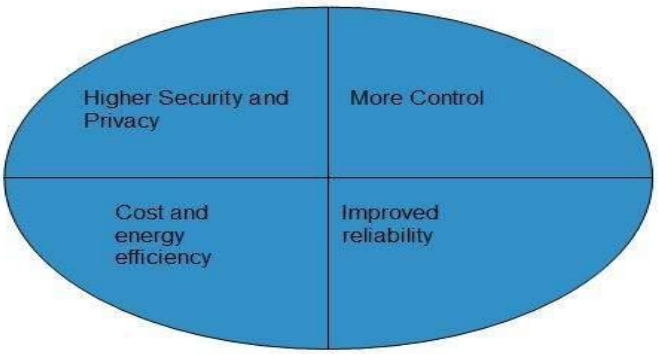
Private Cloud Model

- Private Cloud allows systems and services to be accessible within an organization. The Private Cloud is operated only within a single organization. However, it may be managed internally by the organization itself or by third-party.
- The private cloud model is shown in the diagram below.



Benefits

There are many benefits of deploying cloud as private cloud model. The following diagram shows some of those benefits:



High Security and Privacy: Private cloud operations are not available to general public and resources are shared from distinct pool of resources. Therefore, it ensures high security and privacy.

More Control: The private cloud has more control on its resources and hardware than public cloud because it is accessed only within an organization.

Cost and Energy Efficiency: The private cloud resources are not as cost effective as resources in public clouds but they offer more efficiency than public cloud resources.

Disadvantages: Here are the disadvantages of using private cloud model:

Restricted Area of Operation: The private cloud is only accessible locally and is very difficult to deploy globally.

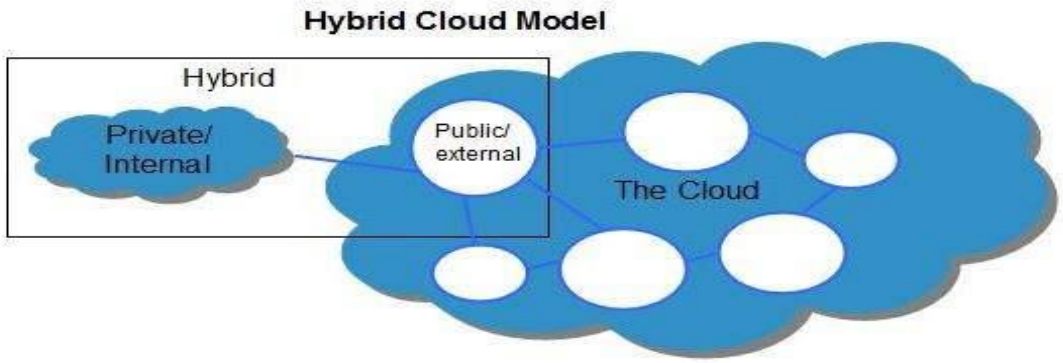
High Priced: Purchasing new hardware in order to fulfill the demand is a costly transaction.

Limited Scalability: The private cloud can be scaled only within capacity of internal hosted resources.

Additional Skills: In order to maintain cloud deployment, organization requires skilled expertise.

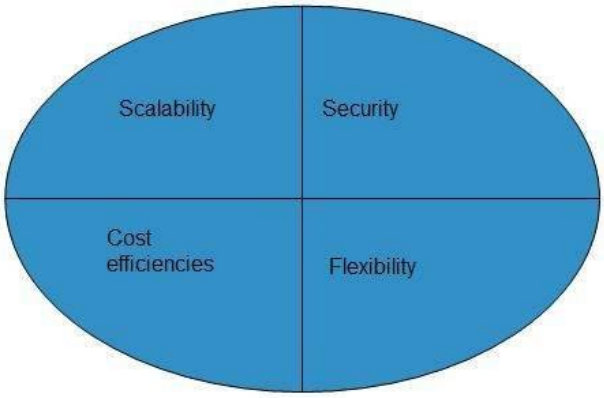
Hybrid Cloud Model

Hybrid Cloud is a mixture of **public** and **private** cloud. Non-critical activities are performed using public cloud while the critical activities are performed using private cloud. The Hybrid Cloud Model is shown in the diagram below.



Benefits

There are many benefits of deploying cloud as hybrid cloud model. The following diagram shows some of those benefits:



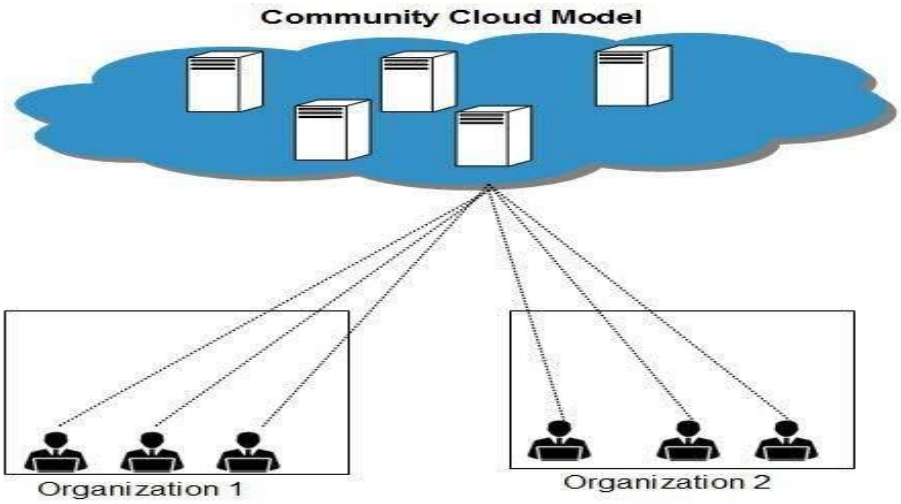
Scalability: It offers features of both, the public cloud scalability and the private cloud scalability.
Flexibility: It offers secure resources and scalable public resources.
Cost Efficiency: Public clouds are more cost effective than private ones. Therefore, hybrid clouds can be cost saving.
Security: The private cloud in hybrid cloud ensures higher degree of security.

Disadvantages

Networking Issues: Networking becomes complex due to presence of private and public cloud.
Security Compliance: It is necessary to ensure that cloud services are compliant with security policies of the organization.
Infrastructure Dependency: The hybrid cloud model is dependent on internal IT infrastructure, therefore it is necessary to ensure redundancy across data centers.

Community Cloud Model

- **Community Cloud** allows system and services to be accessible by group of organizations. It shares the infrastructure between several organizations from a specific community. It may be managed internally by organizations or by the third-party.
- The Community Cloud Model is shown in the diagram below.



Benefits

There are many benefits of deploying cloud as **community cloud model.**

Cost Effective: Community cloud offers same advantages as that of private cloud at low cost.
Sharing Among Organizations: Community cloud provides an infrastructure to share cloud resources and capabilities among several organizations.
Security: The community cloud is comparatively more secure than the public cloud but less secured than the private cloud.

Issues

- Since all data is located at one place, one must be careful in storing data in community cloud because it might be accessible to others.
- It is also challenging to allocate responsibilities of governance, security and cost among organizations.

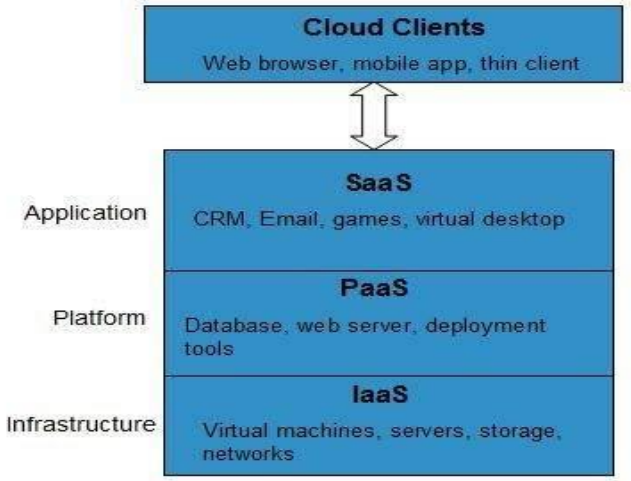
Service Models

Cloud computing is based on service models. These are categorized into three basic service models which are -

- Infrastructure-as–a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)

Anything-as-a-Service (XaaS) is yet another service model, which includes Network-as-a-Service, Business-as-a-Service, Identity-as-a-Service, Database-as-a-Service or Strategy-as-a-Service.

The **Infrastructure-as-a-Service (IaaS)** is the most basic level of service. Each of the service models inherit the security and management mechanism from the underlying model, as shown in the following diagram:



Infrastructure-as-a-Service (IaaS): *IaaS* provides access to fundamental resources such as physical machines, virtual machines, virtual storage, etc.

Platform-as-a-Service (PaaS): *PaaS* provides the runtime environment for applications, development and deployment tools, etc.

Software-as-a-Service (SaaS): *SaaS model allows to use software applications as a service to end-users.*

Top Cloud Service Providers Companies in the World

Nowadays, there are many companies which are coming with cloud services and are performing better day by day. These Cloud Service Providers, are providing: SaaS, PaaS, IaaS. So, let's discuss about some top Cloud Service Providers in the world.

1. Cloud Service Providers

Earlier the data was stored in hard drives which were not reliable and secure as the drive can access by anyone. Today the **cloud computing** services have replaced search hard drive technology and came with a new concept called cloud technology in which the data store in the cloud. There are many companies which provide Cloud computing service and they are very reliable.

2. Services Provided by Cloud Providers

These are the service, which offers by Cloud Computing Providers:

i. Software as a Service (SaaS): Software as a service, a cloud service provided by the cloud company. In SaaS, a customer provides software which can be either for a particular amount of time or for the lifetime. SaaS utilizes the internet and delivers the application to the customer. Most of the SaaS application does not require any downloads as they can use directly through the web browser.

ii. Platform as a Service (PaaS): Platform as a service is a framework for the developer where they can create an application for customizing the previously built application. This service also provided through the means of internet and here all the management is done by the enterprise or any third party provider.

iii. Infrastructure as a Service (IaaS): Infrastructure as a service**,** provided by the Cloud Service providers which help the customer to access and monitor things like computer, networking, and other services. In IaaS, the customer can purchase resources on demand rather than buying hardware which is costly and hard to maintain.

3. List of Cloud Service Providers

There are many Cloud Service providers in the market:

i. Amazon Web Service (AWS) **ii.** Microsoft Azure

iii. Google Cloud Platform
iv. IBM Cloud Services
v. Adobe Creative Cloud
vi. Kamatera
vii. VMware
viii. Rackspace
ix. Red Hat
x. Salesforce
xi. Oracle Cloud
xii. SAP
xiii. Verizon Cloud
xiv. Navisite
xv. Dropbox

Some top Cloud Service providers of the market:

i. Amazon Web Services (AWS)

- Amazon Web Services is a cloud computing platform which provides services such as compute power, database storage, content delivery and many other functions which will help to integrate a business. The Amazon Web Services is flexible, scalable, and reliable and due to this many companies are implementing it in their work. There is no upfront cost and the customer has to pay only for what they have used. It is one of the leading cloud service providers among all.
- With the help of the internet, the customer can access highly durable storage such as **Amazon Glacier**, Amazon S3, and Amazon EBS. It also has a high-performance database such as Amazon Redshift, Amazon Dynamo DB, Amazon ElastiCache and **Amazon RDS**.

Cloud Service Providers – AWS

ii. Microsoft Azure

- Microsoft Azure is a cloud computing service which is used for building testing deploying and managing the application. This process is done in a global network of the Microsoft-managed data centre. It is private as well as a **public cloud** platform. It uses virtualization which differentiates the coupling between the operating system and CPU with the help of an abstraction layer known as a hypervisor.
- This hypervisor emulates all the functionality of the physical machine such as hardware and server into a virtual one. There is numerous amount of virtual machine available and each virtual machine can run many operating systems.
- In the data centre of Microsoft, there are many servers and each server consists of a hypervisor through which multiple virtual machines can operate. With the help of Azure, it is easy for developers and IT professionals to manage and deploy their applications and services.

iii. Google Cloud Platform

- Google cloud platform is one of the leading Cloud Computing services which are offered by Google and it runs on the same infrastructure that Google uses for its end-user products. The Google cloud platform is basically used for Google search and YouTube. There are various services offered by Google Cloud such as data analysis, machine learning, and data storage.
- The data stored in Google Cloud is secure and can access easily. It offers varieties of services from infrastructure as a service to platform as a service. Google also provides a strong **commitment to security** and stability. With the help of the Google cloud platform, the user is free to think about the code and the feature which are needed to develop without worrying about the operations side.
- Here most of the services fully manage and details quite easy for the customer to concentrate on their work. In this, **machine learning** and the use of API are very easy. The API also helps in speech detection language translation very easily. So it prefers among the customers.

Cloud Service Providers – Google Cloud Platform

iv. IBM Cloud Services

- IBM cloud offers services such as platform as a service and infrastructure as a service. This cloud organization can deploy and access its resources such as storage networking and compute power with the help of internet. There are several tools which help the customer to draw on deep industry expertise.
- The speed and agility of the cloud fulfil the requirement of the customer and make them feel satisfied. A customer using IBM cloud can easily find growth opportunities, generating new revenue schemes and improving the operational efficiency. The uses of IBM cloud don't have many barriers as compared to traditional technologies.
- IBM cloud eliminates the complex problem and the problems which face by large companies. IBM Cloud computing services are also helping home appliance manufacturer, retailer, and medical supply businesses. It uses in because it offers the best services with the price as low as possible.

Cloud Service Providers – IBM Cloud Services

v. Adobe Creative Cloud

- Adobe creative cloud provides the best experience of apps services design photography and web. The Adobe cloud services provide tutorials and templates with which a beginner can easily access the cloud and can start using it. It provides many facilities to the beginner as well as professionals for easy access to the cloud.
- It consists of many **applications** and services that provide access to a collection of software which uses for video editing, web development, photography, and graphic designs. There are mobile applications as well as computer applications which can use by the customers.
- Creative Cloud allows you to work from anywhere and from any device as the files can save to the cloud and can access at any time from anywhere. Creative Cloud was the first host on **Amazon Web Services** but as per the new agreement with Microsoft, the Adobe creative cloud now hosted on Microsoft Azure.

Cloud Service Providers – Adobe Creative Cloud

Services Provided by Cloud Providers

Name of Company	IaaS	Paas	SaaS
AWS	Amazon EC2	Amazon Web Services	Amazon Web Services
Microsoft	Microsoft Private Cloud	Microsoft Azure	Microsoft Office 365
Google	–	Google App Engine	Google Applications

		(Python, Java and many)	
IBM	Smart Cloud Enterprise	Smart Cloud Application Services	SaaS Products
Adobe	–	Adobe Creative Cloud	Acrobat, Flash player, etc.

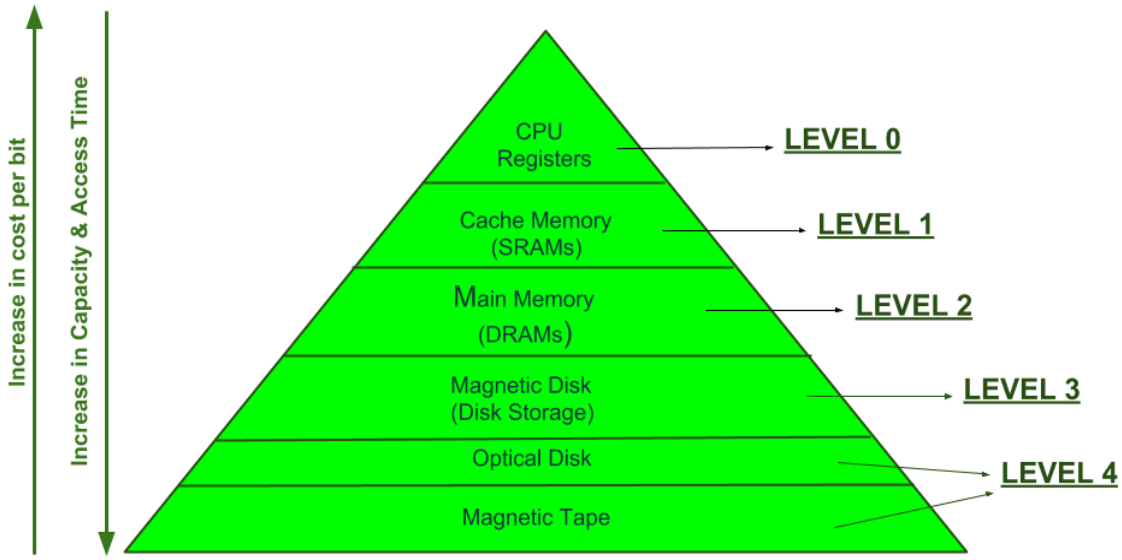
Conclusion

- Cloud Computing is helping a lot in business whether it is a small or large.
- These Cloud Service Providers companies provide storage database server networking and the software through which the business can increase. So, a customer can choose the company which is most suitable for their business and their requirement.

Unit-IV

Memory Hierarchy Design and its Characteristics

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behaviour known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy:



MEMORY HIERARCHY DESIGN

This Memory Hierarchy Design is divided into 2 main types:

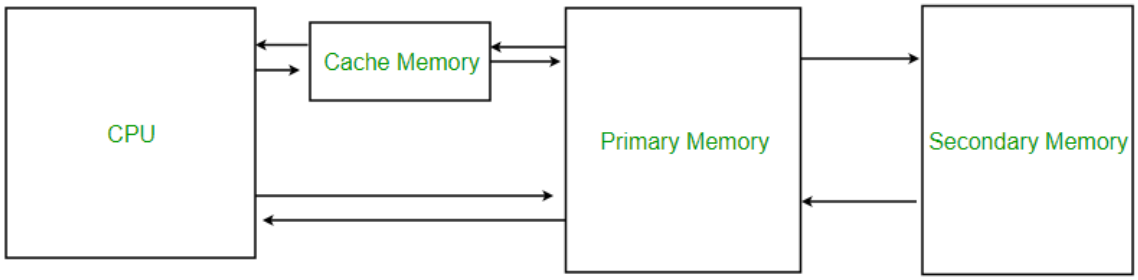
1. **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

1. **Capacity:** It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
2. **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
3. **Performance:** Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
4. **Cost per bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Cache Memory in Computer Organization

- **Cache Memory** is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register:** It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory:** It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory:** It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory:** It is external memory which is not as fast as main memory but data stays permanently in this memory.

Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

1. **Primary Cache:** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
2. **Secondary Cache:** Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred, and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Average memory access time

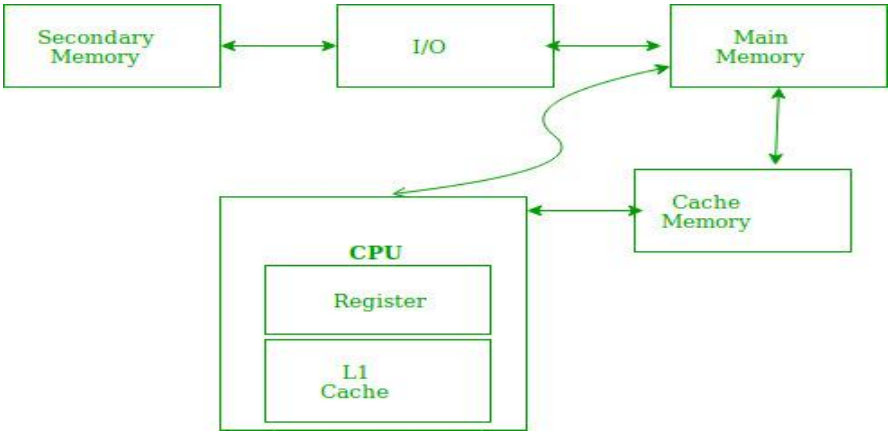
- The average memory access time, or AMAT, can then be computed.
- AMAT = Hit time + (Miss rate x Miss penalty) This is just averaging the amount of time for cache hits and the amount of time for cache misses.

How can we improve the average memory access time of a system?

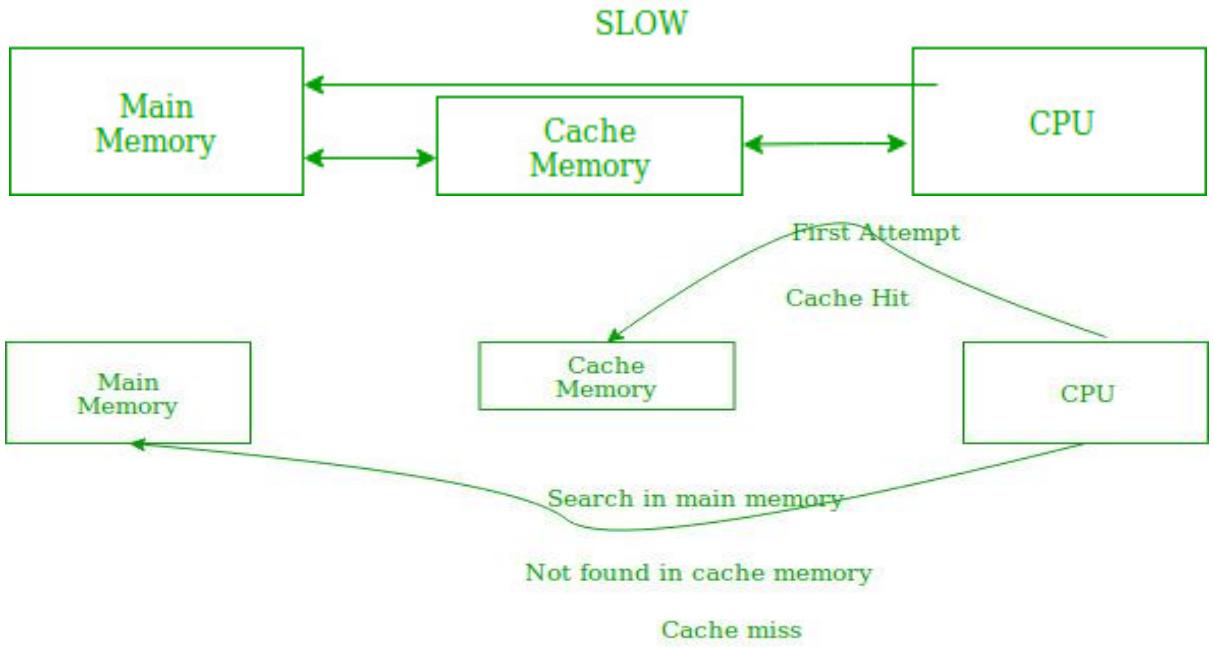
- Obviously, a lower AMAT is better.
- Miss penalties are usually much greater than hit times, so the best way to lower AMAT is to reduce the miss penalty or the miss rate.
- However, AMAT should only be used as a general guideline. Remember that execution time is still the best performance metric.

Locality of Reference and Cache Operation in Cache Memory

Locality of reference refers to a phenomenon in which a computer program tends to access same set of memory locations for a particular time period. In other words, **Locality of Reference** refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.



1. In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.
2. In case of subroutine calls, every time the set of instructions are fetched from memory.
3. References to data items also get localized that means same data item is referenced again and again.



In the above figure, we can see that the CPU wants to read or fetch the data or instruction.

- First, it will access the cache memory as it is near to it and provides very fast access. If the required data or instruction is found, it will be fetched. This situation is known as a cache hit.
- But if the required data or instruction is not found in the cache memory then this situation is known as a cache miss.

Now the main memory will be searched for the required data or instruction that was being searched and if found will go through one of the two ways:

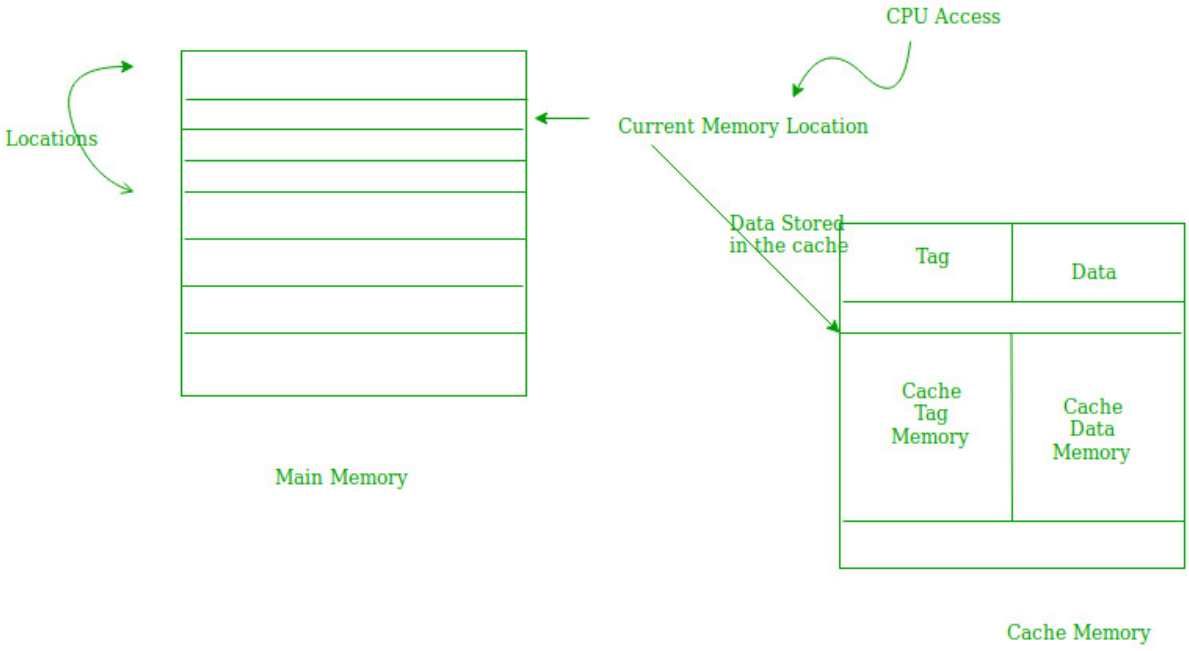
1. First way is that the CPU should fetch the required data or instruction and use it and that's it but what, when the same data or instruction is required again. CPU again has to access the same main memory location for it and we already know that main memory is the slowest to access.
2. The second way is to store the data or instruction in the cache memory so that if it is needed soon again in the near future it could be fetched in a much faster way.

Cache Operation:

It is based on the principle of locality of reference. There are two ways with which data or instruction is fetched from main memory and get stored in cache memory. These two ways are the following:

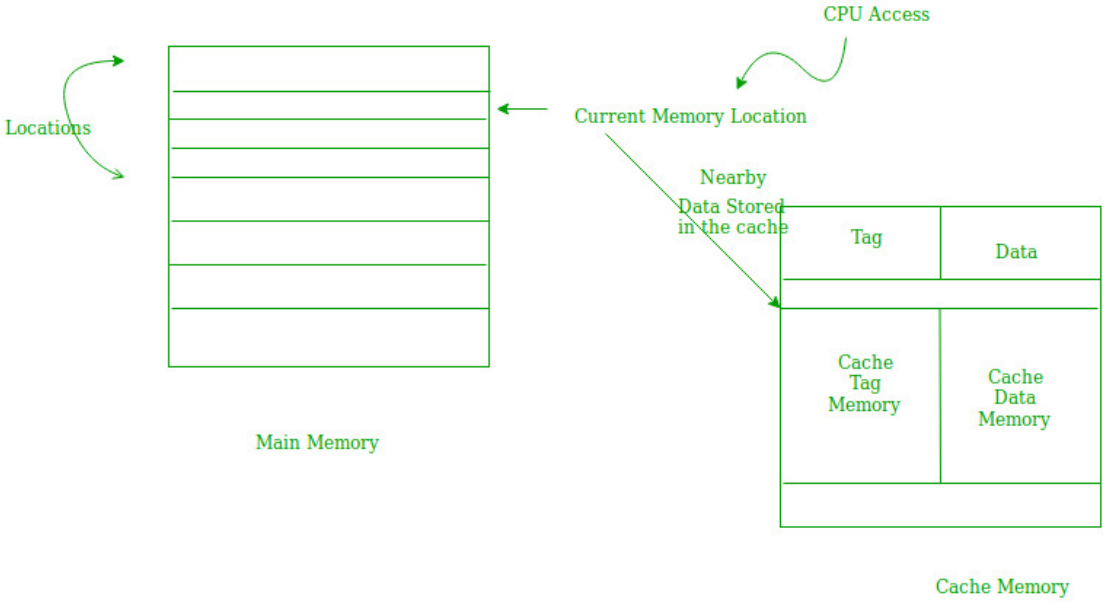
i. Temporal Locality:

- Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.
- When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future. This is known as temporal locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future.



ii. Spatial Locality:

- Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.



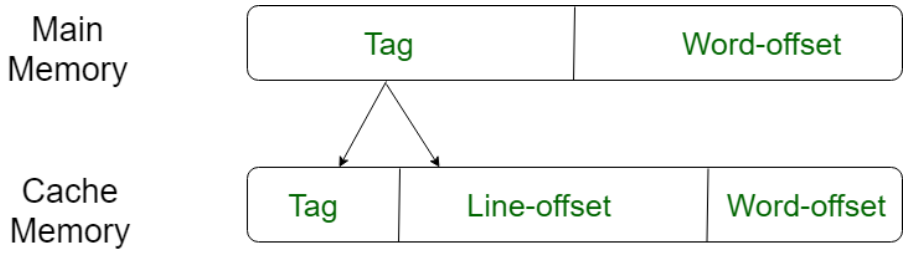
Cache Mapping

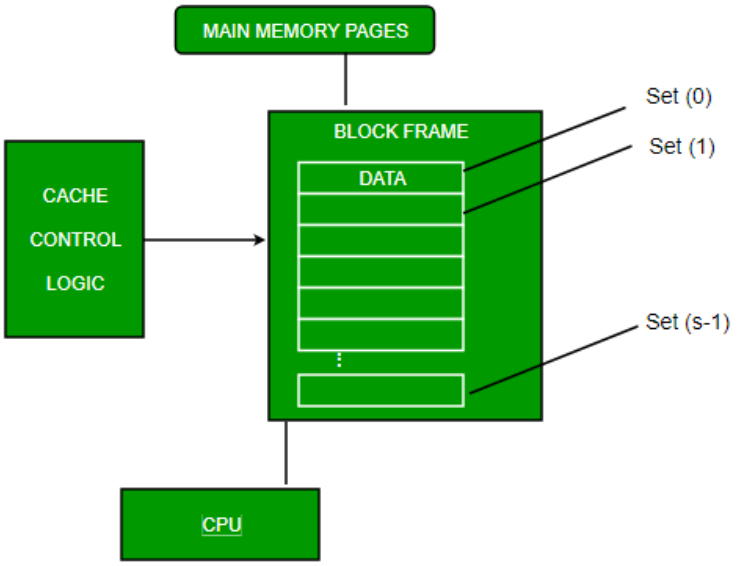
There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

1. Direct Mapping –

- The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. Or
- In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

  i = j modulo m
  where
  i=cache line number
  j= main memory block number
  m=number of lines in the cache
- For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.





Problem-01:

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

- Number of bits in tag
- Tag directory size

Solution-

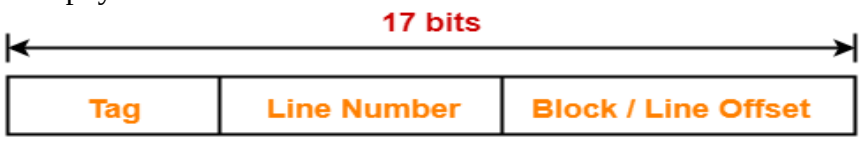
Given-

Cache memory size = 16 KB
Block size = Frame size = Line size = 256 bytes
Main memory size = 128 KB

We consider that the memory is byte addressable.

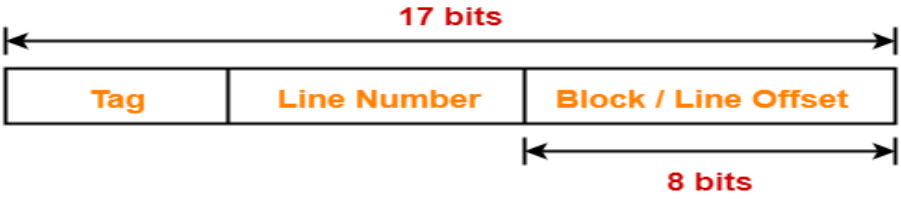
Number of Bits in Physical Address-

We have,
Size of main memory = 128 KB=128*1024B=2^7*2^10B = 2^{17} bytes
Thus, Number of bits in physical address = 17 bits



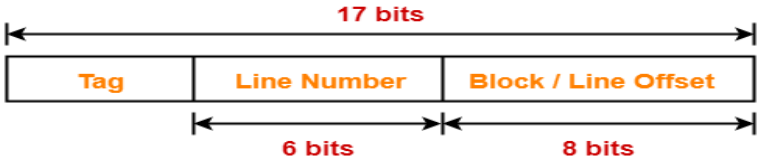
Number of Bits in Block Offset-

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



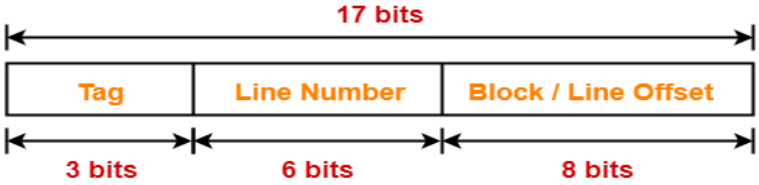
Number of Bits in Line Number-

Total number of lines in cache
= Cache size / Line size
= 16 KB / 256 bytes=16*1024B/256B=2^4*2^10B/2^8B
= 2^{14} bytes / 2^8 bytes = 2^6 lines
Thus, Number of bits in line number = 6 bits



Number of Bits in Tag-

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



Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 2^6 x 3 bits
= 192 bits=192/8 byte
= 24 bytes
Thus, size of tag directory = 24 bytes

Problem-02:

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

- Size of main memory
- Tag directory size

Solution-

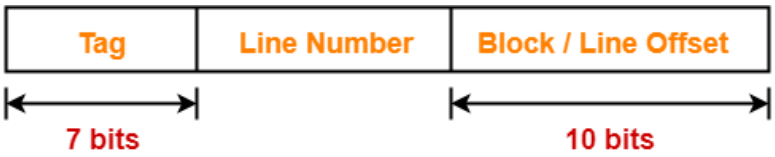
Given-

Cache memory size = 512 KB
Block size = Frame size = Line size = 1 KB
Number of bits in tag = 7 bits

We consider that the memory is byte addressable.

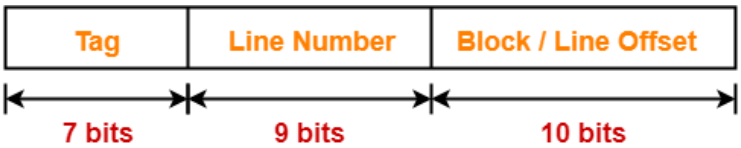
Number of Bits in Block Offset-

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



Number of Bits in Line Number-

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



Number of Bits in Physical Address-

Number of bits in physical address
= Number of bits in tag + Number of bits in line number + Number of bits in block offset
= 7 bits + 9 bits + 10 bits = 26 bits
Thus, Number of bits in physical address = 26 bits

Size of Main Memory-

We have,
Number of bits in physical address = 26 bits
Thus, Size of main memory
= 2^{26} bytes
= 64 MB

Tag Directory Size-

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

Problem-03:

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

- Size of cache memory
- Tag directory size

Solution-

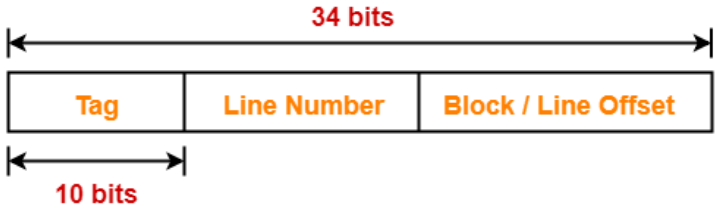
Given-

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

We consider that the memory is byte addressable.

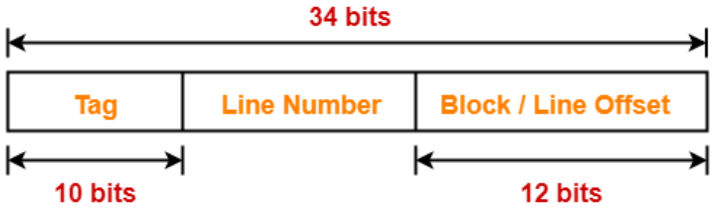
Number of Bits in Physical Address-

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



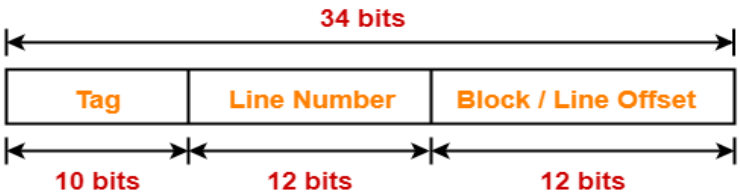
Number of Bits in Block Offset-

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



Number of Bits in Line Number-

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



Number of Lines in Cache-

We have-

Number of bits in line number = 12 bits

Thus, Total number of lines in cache = 2^{12} lines

Size of Cache Memory-

Size of cache memory

= Total number of lines in cache x Line size

= 2^{12} x 4 KB = 2^{14} KB = 16 MB

Thus, Size of cache memory = 16 MB

Tag Directory Size-

Tag directory size

= Number of tags x Tag size

= Number of lines in cache x Number of bits in tag

= 2^{12} x 10 bits = 40960 bits = 5120 bytes

Thus, size of tag directory = 5120 bytes

Problem-04:

Consider a direct mapped cache of size 32 KB with block size 32 bytes. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively-

- 10, 17
- 10, 22
- 15, 17
- 5, 17

Solution-

Given-

Cache memory size = 32 KB

Block size = Frame size = Line size = 32 bytes

Number of bits in physical address = 32 bits

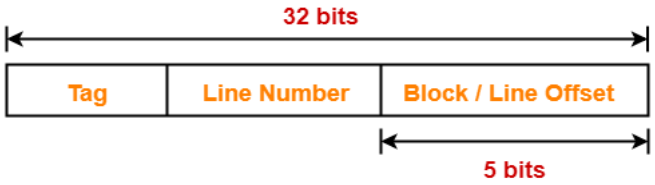
Number of Bits in Block Offset-

We have,

Block size

= 32 bytes = 2^5 bytes

Thus, Number of bits in block offset = 5 bits



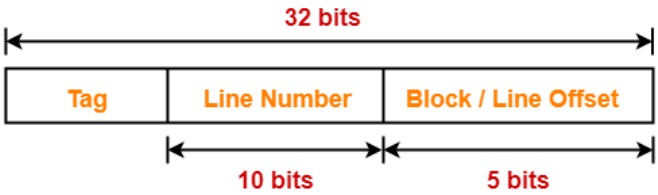
Number of Bits in Line Number-

Total number of lines in cache

= Cache size / Line size

= 32 KB / 32 bytes = 2^{10} lines

Thus, Number of bits in line number = 10 bits

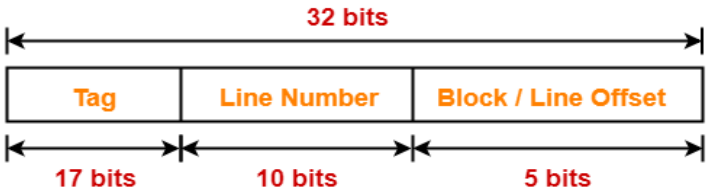


Number of Bits Required For Cache Indexing-

Number of bits required for cache indexing
= Number of bits in line number
= 10 bits

Number Of Bits in Tag-

Number of bits in tag
= Number of bits in physical address – (Number of bits in line number + Number of bits in block offset)
= 32 bits – (10 bits + 5 bits) = 32 bits – 15 bits = 17 bits
Thus, Number of bits in tag = 17 bits



Thus, Option (A) is correct.

Problem-05:

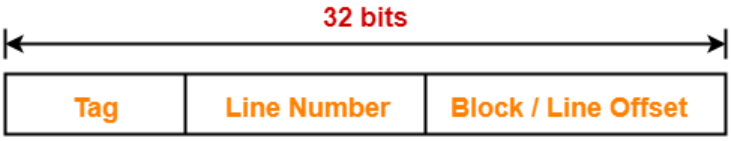
Consider a machine with a byte addressable main memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume that a direct mapped cache having 512 cache lines is used with this machine. The size of the tag field in bits is _____.

Solution-

Given-
Main memory size = 2^{32} bytes
Block size = Frame size = Line size = 32 bytes
Number of lines in cache = 512 lines

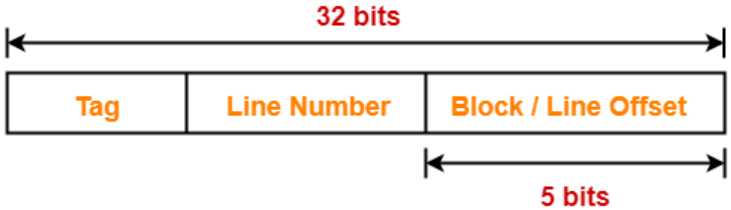
Number of Bits in Physical Address-

We have,
Size of main memory
= 2^{32} bytes
Thus, Number of bits in physical address = 32 bits



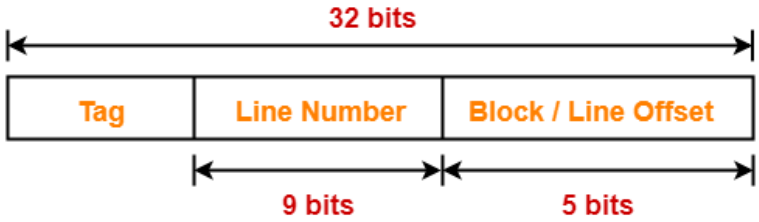
Number of Bits in Block Offset-

We have,
Block size
= 32 bytes = 2^5 bytes
Thus, Number of bits in block offset = 5 bits



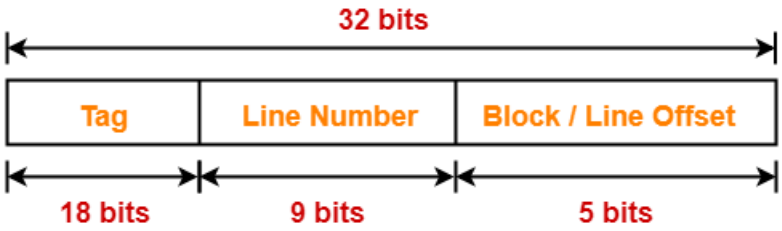
Number of Bits in Line Number-

Total number of lines in cache
= 512 lines = 2^9 lines
Thus, Number of bits in line number = 9 bits



Number Of Bits in Tag-

Number of bits in tag
= Number of bits in physical address – (Number of bits in line number + Number of bits in block offset)
= 32 bits – (9 bits + 5 bits) = 32 bits – 14 bits = 18 bits
Thus, Number of bits in tag = 18 bits



Problem-06:

An 8 KB direct-mapped write back cache is organized as multiple blocks, each of size 32 bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following-
1 valid bit
1 modified bit
As many bits as the minimum needed to identify the memory block mapped in the cache
What is the total size of memory needed at the cache controller to store meta data (tags) for the cache?

- 4864 bits
- 6144 bits
- 6656 bits
- 5376 bits

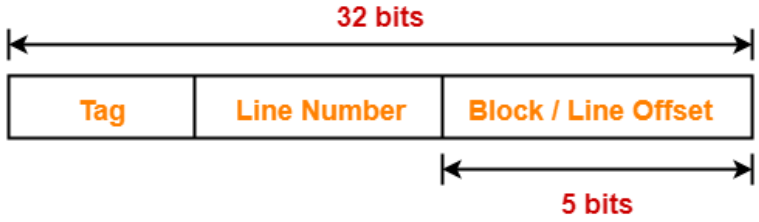
Solution-

Given-

Cache memory size = 8 KB
Block size = Frame size = Line size = 32 bytes
Number of bits in physical address = 32 bits

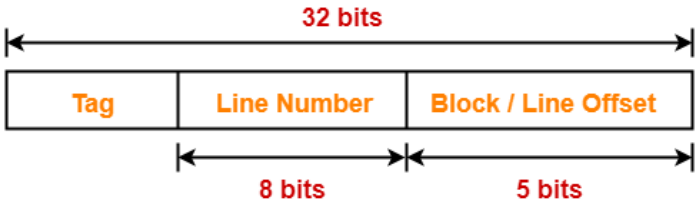
Number of Bits in Block Offset-

We have,
Block size
= 32 bytes = 2^5 bytes
Thus, Number of bits in block offset = 5 bits



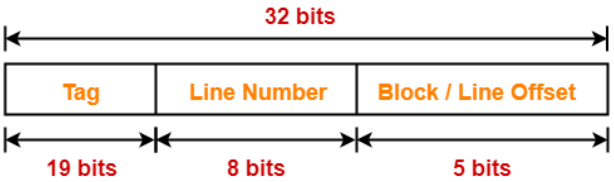
Number of Bits in Line Number-

Total number of lines in cache
= Cache memory size / Line size
= 8 KB / 32 bytes = 2^{13} bytes / 2^5 bytes = 2^8 lines
Thus, Number of bits in line number = 8 bits



Number Of Bits in Tag-

Number of bits in tag
= Number of bits in physical address – (Number of bits in line number + Number of bits in block offset)
= 32 bits – (8 bits + 5 bits) = 32 bits – 13 bits = 19 bits
Thus, Number of bits in tag = 19 bits

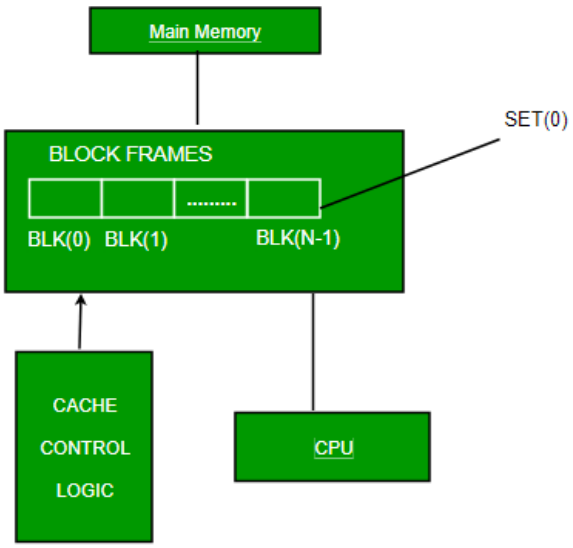


Memory Size Needed At Cache Controller-

Size of memory needed at cache controller
= Number of lines in cache x (1 valid bit + 1 modified bit + 19 bits to identify block)
= 2^8 x 21 bits
= 5376 bits

2. Associative Mapping –

- In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



Problem-01:

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

- Number of bits in tag
- Tag directory size

Solution-

Given-

Cache memory size = 16 KB

Block size = Frame size = Line size = 256 bytes

Main memory size = 128 KB

- We consider that the memory is byte addressable.

Number of Bits in Physical Address-

We have,

Size of main memory

= 128 KB=2^7*2^{10}B= 2^{17} bytes

Thus, Number of bits in physical address = 17 bits

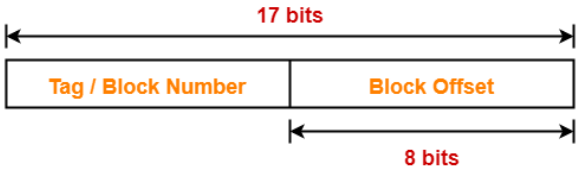
Number of Bits in Block Offset-

We have,

Block size

= 256 bytes = 2^8 bytes

Thus, Number of bits in block offset = 8 bits



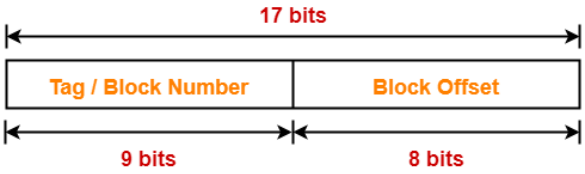
Number of Bits in Tag-

Number of bits in tag

= Number of bits in physical address – Number of bits in block offset

= 17 bits – 8 bits = 9 bits

Thus, Number of bits in tag = 9 bits



Number of Lines in Cache-

Total number of lines in cache

= Cache size / Line size

= 16 KB / 256 bytes = 2^{14} bytes / 2^8 bytes = 2^6 lines

Tag Directory Size-

Tag directory size

= Number of tags x Tag size

= Number of lines in cache x Number of bits in tag

= 2^6 x 9 bits = 576 bits = 72 bytes

Thus, size of tag directory = 72 bytes

Problem-02:

Consider a fully associative mapped cache of size 512 KB with block size 1 KB. There are 17 bits in the tag. Find-

- Size of main memory
- Tag directory size

Solution-

Given-

Cache memory size = 512 KB

Block size = Frame size = Line size = 1 KB

Number of bits in tag = 17 bits

- We consider that the memory is byte addressable.

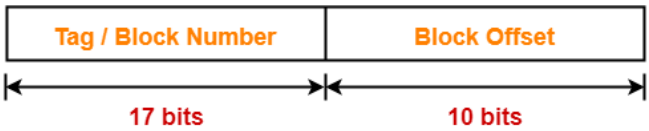
Number of Bits in Block Offset-

We have,

Block size

= 1 KB = 2^{10} bytes

Thus, Number of bits in block offset = 10 bits



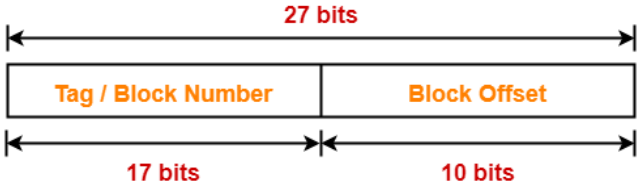
Number of Bits in Physical Address-

Number of bits in physical address

= Number of bits in tag + Number of bits in block offset

= 17 bits + 10 bits = 27 bits

Thus, Number of bits in physical address = 27 bits



Size of Main Memory-

We have,

Number of bits in physical address = 27 bits

Thus, Size of main memory

= 2^{27} bytes = 128 MB

Number of Lines in Cache-

Total number of lines in cache

= Cache size / Line size

= 512 KB / 1 KB = 512 lines = 2^9 lines

Tag Directory Size-

Tag directory size

= Number of tags x Tag size

= Number of lines in cache x Number of bits in tag

= 2^9 x 17 bits

= 8704 bits

= 1088 bytes

Thus, size of tag directory = 1088 bytes

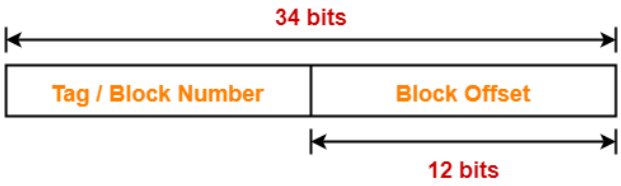
Problem-03:
Consider a fully associative mapped cache with block size 4 KB. The size of main memory is 16 GB. Find the number of bits in tag.

Solution-
Given-
Block size = Frame size = Line size = 4 KB
Size of main memory = 16 GB

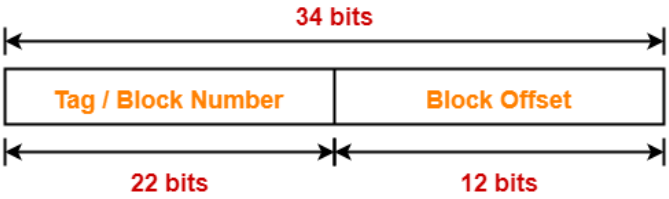
- We consider that the memory is byte addressable.

Number of Bits in Physical Address-
We have,
Size of main memory
= 16 GB = 2^{34} bytes
Thus, Number of bits in physical address = 34 bits

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



Number of Bits in Tag-
Number of bits in tag
= Number of bits in physical address – Number of bits in block offset
= 34 bits – 12 bits = 22 bits
Thus, Number of bits in tag = 22 bits



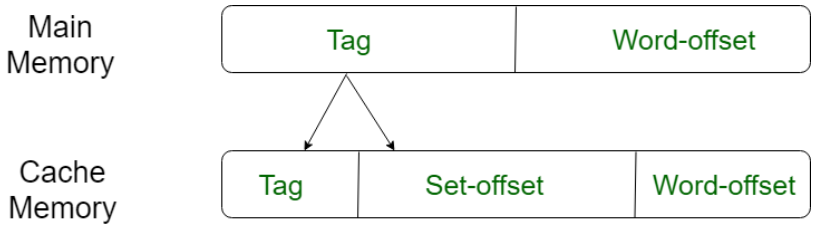
3. **Set-associative Mapping –**

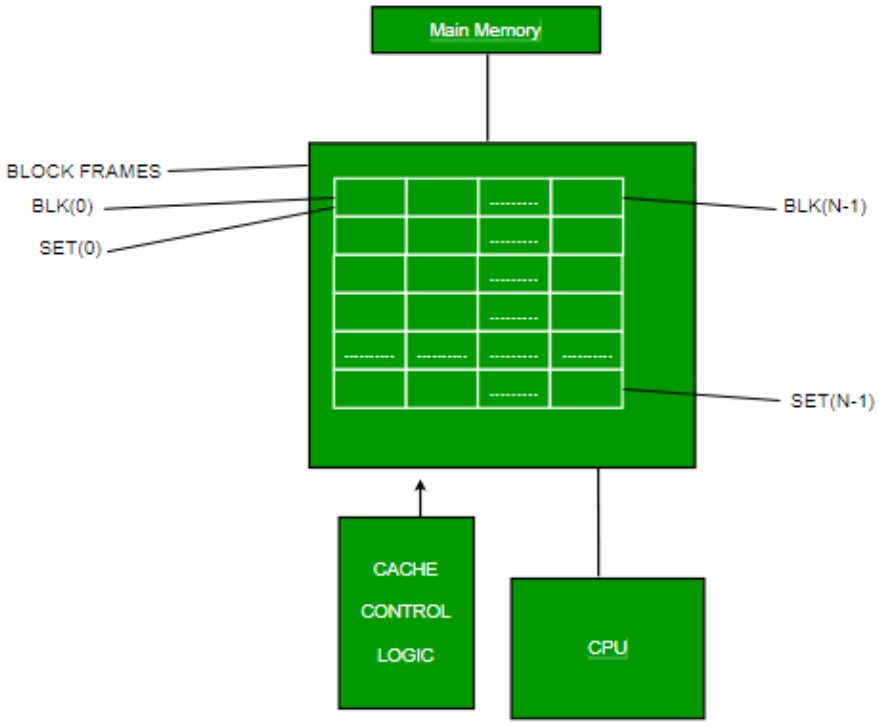
- This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

  In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

  m = v * k
  i= j mod v

where
i=cache set number
j=main memory block number
v=number of sets
m=number of lines in the cache number of sets
k=number of lines in each set





Set Associative Cache:

- **Set associative cache employs set associative cache mapping technique.**

The following steps explain the working of set associative cache-

After CPU generates a memory request,

- The set number field of the address is used to access the particular set of the cache.
- The tag field of the CPU address is then compared with the tags of all k lines within that set.
- If the CPU tag matches to the tag of any cache line, a cache hit occurs.
- If the CPU tag does not match to the tag of any cache line, a cache miss occurs.
- In case of a cache miss, the required word has to be brought from the main memory.
- If the cache is full, a replacement is made in accordance with the employed replacement policy.

EXAMPLES:

Problem-01:

Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find-

- Number of bits in tag
- Tag directory size

Solution-

Given-

Set size = 2

Cache memory size = 16 KB

Block size = Frame size = Line size = 256 bytes

Main memory size = 128 KB

- We consider that the memory is byte addressable.

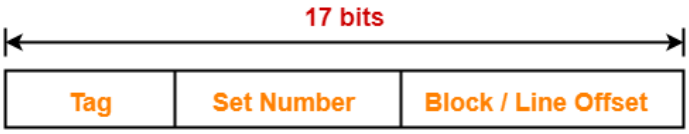
Number of Bits in Physical Address-

We have,

Size of main memory

= 128 KB = 2^{17} bytes

Thus, Number of bits in physical address = 17 bits



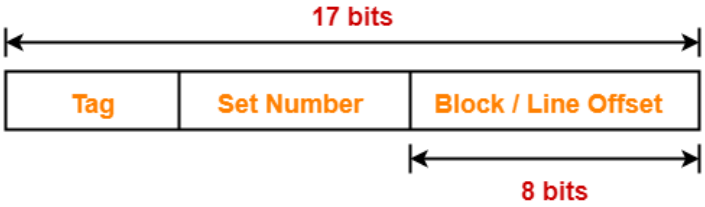
Number of Bits in Block Offset-

We have,

Block size

= 256 bytes = 2^{8} bytes

Thus, Number of bits in block offset = 8 bits



Number of Lines in Cache-

Total number of lines in cache

= Cache size / Line size

= 16 KB / 256 bytes = 2^{14} bytes / 2^{8} bytes = 64 lines

Thus, Number of lines in cache = 64 lines

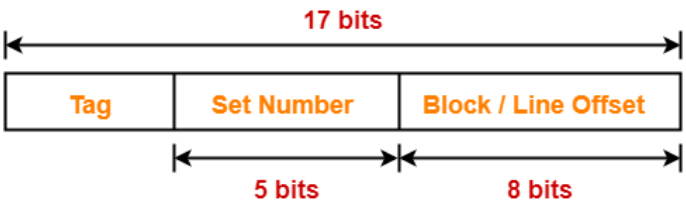
Number of Sets in Cache-

Total number of sets in cache

= Total number of lines in cache / Set size

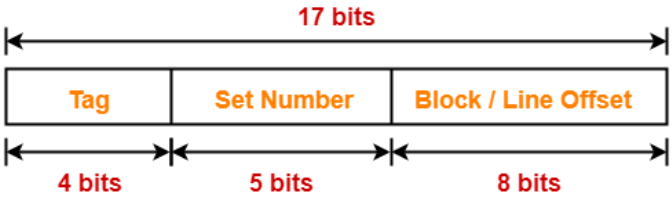
= 64 / 2 = 32 sets = 2^{5} sets

Thus, Number of bits in set number = 5 bits



Number of Bits in Tag-

Number of bits in tag
= Number of bits in physical address – (Number of bits in set number + Number of bits in block offset)
= 17 bits – (5 bits + 8 bits) = 17 bits – 13 bits = 4 bits
Thus, Number of bits in tag = 4 bits



Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 64 x 4 bits = 256 bits = 32 bytes
Thus, size of tag directory = 32 bytes

Problem-02:

Consider a 8-way set associative mapped cache of size 512 KB with block size 1 KB. There are 7 bits in the tag. Find-

- Size of main memory
- Tag directory size

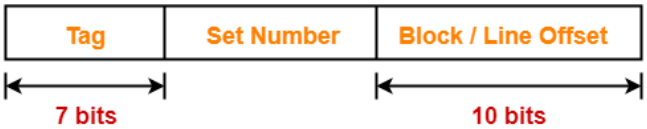
Solution-

Given-
Set size = 8
Cache memory size = 512 KB
Block size = Frame size = Line size = 1 KB
Number of bits in tag = 7 bits

- We consider that the memory is byte addressable.

Number of Bits in Block Offset-

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

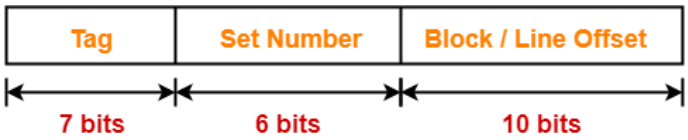


Number of Lines in Cache-

Total number of lines in cache
= Cache size / Line size
= 512 KB / 1 KB = 512 lines
Thus, Number of lines in cache = 512 lines

Number of Sets in Cache-

Total number of sets in cache
= Total number of lines in cache / Set size
= 512 / 8 = 64 sets = 2^6 sets
Thus, Number of bits in set number = 6 bits



Number of Bits in Physical Address-

Number of bits in physical address
= Number of bits in tag + Number of bits in set number + Number of bits in block offset
= 7 bits + 6 bits + 10 bits = 23 bits
Thus, Number of bits in physical address = 23 bits

Size of Main Memory-

We have,
Number of bits in physical address = 23 bits
Thus, Size of main memory
= 2^{23} bytes = 8 MB

Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 512 x 7 bits = 3584 bits = 448 bytes
Thus, size of tag directory = 448 bytes

Problem-03:

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

- Size of cache memory
- Tag directory size

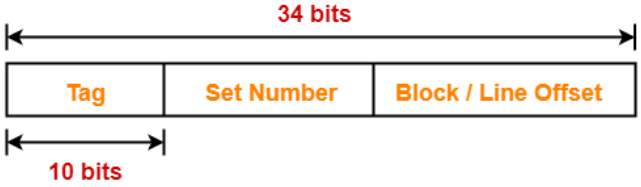
Solution-

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

- We consider that the memory is byte addressable.

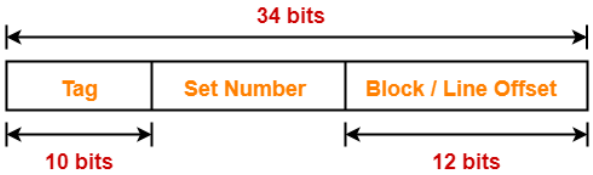
Number of Bits in Physical Address-

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



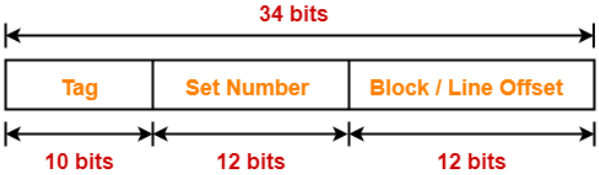
Number of Bits in Block Offset-

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



Number of Bits in Set Number-

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



Number of Sets in Cache-

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

Number of Lines in Cache-

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

Size of Cache Memory-

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

Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 2^{14} x 10 bits = 163840 bits = 20480 bytes = 20 KB
Thus, size of tag directory = 20 KB

Problem-04:

Consider a 8-way set associative mapped cache. The size of cache memory is 512 KB and there are 10 bits in the tag. Find the size of main memory.

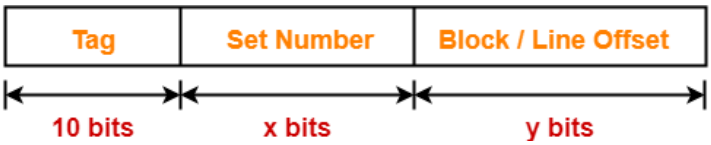
Solution-

Given-
Set size = 8
Cache memory size = 512 KB
Number of bits in tag = 10 bits

- We consider that the memory is byte addressable.

Let-
Number of bits in set number field = x bits

Number of bits in block offset field = y bits



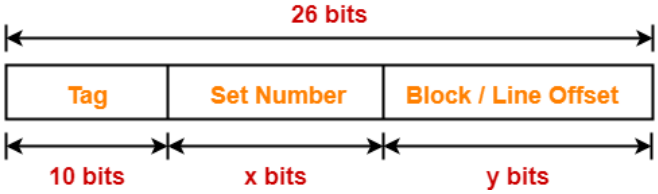
Sum of Number Of Bits Of Set Number Field And Block Offset Field-

We have,
Cache memory size = Number of sets in cache x Number of lines in one set x Line size
Now, substituting the values, we get-
512 KB = 2^x x 8 x 2^y bytes

- ⇨ 2^{19} bytes = 2^{3+x+y} bytes
- ⇨ 19 = 3 +x + y
- ⇨ x + y = 19 – 3
- ⇨ x + y = 16

Number of Bits in Physical Address-

Number of bits in physical address
= Number of bits in tag + Number of bits in set number + Number of bits in block offset
= 10 bits + x bits + y bits
= 10 bits + (x + y) bits
= 10 bits + 16 bits = 26 bits
Thus, Number of bits in physical address = 26 bits



Size of Main Memory-

We have,
Number of bits in physical address = 26 bits
Thus, Size of main memory
= 2^{26} bytes = 64 MB
Thus, size of main memory = 64 MB

Problem-05:

Consider a 4-way set associative mapped cache. The size of main memory is 64 MB and there are 10 bits in the tag. Find the size of cache memory.

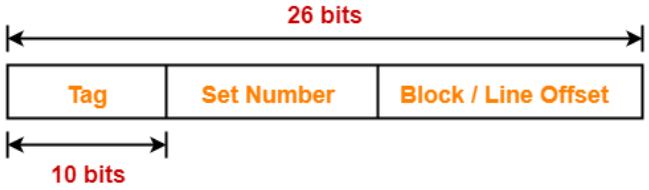
Solution-

Given-
Set size = 4
Main memory size = 64 MB
Number of bits in tag = 10 bits

- We consider that the memory is byte addressable.

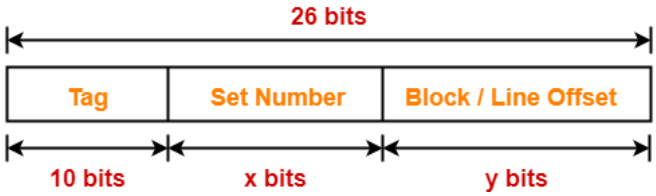
Number of Bits in Physical Address-

We have,
Size of main memory
= 64 MB
= 2^{26} bytes
Thus, Number of bits in physical address = 26 bits



Sum Of Number Of Bits Of Set Number Field And Block Offset Field-

Let-
Number of bits in set number field = x bits
Number of bits in block offset field = y bits



Then, Number of bits in physical address
= Number of bits in tag + Number of bits in set number + Number of bits in block offset
So, we have-
26 bits = 10 bits + x bits + y bits

- ⇨ 26 = 10 + (x + y)
- ⇨ x + y = 26 – 10
- ⇨ x + y = 16

Thus, Sum of number of bits of set number field and block offset field = 16 bits

Size of Cache Memory-

Cache memory size
= Number of sets in cache x Number of lines in one set x Line size
= 2^x x 4 x 2^y bytes
= 2^{2+x+y} bytes
= 2^{2+16} bytes
= 2^{18} bytes
= 256 KB
Thus, size of cache memory = 256 KB

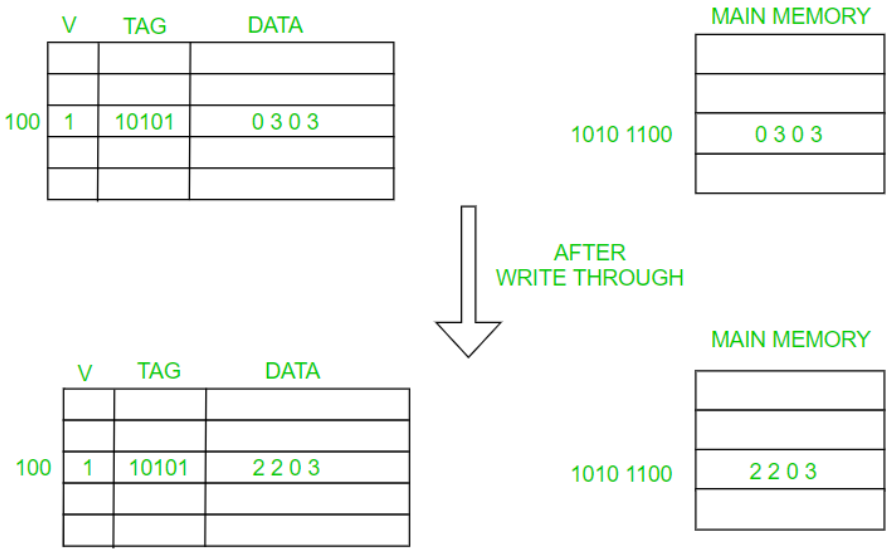
Write Through and Write Back in Cache

- Cache is a technique of storing a copy of data temporarily in rapidly accessible storage memory. Cache stores most recently used words in small memory to increase the speed in which a data is accessed. It acts like a buffer between RAM and CPU and thus increases the speed in which data is available to the processor.
- Whenever a Processor wants to write a word, it checks to see if the address it wants to write the data to, is present in the cache or not. If address is present in the cache i.e., **Write Hit**.
- We can update the value in the cache and avoid a expensive main memory access. But this results in **Inconsistent Data** Problem. As both cache and main memory have different data, it will cause problem in two or more devices sharing the main memory (as in a multiprocessor system). This is where **Write Through** and **Write Back Protocol** comes into picture.

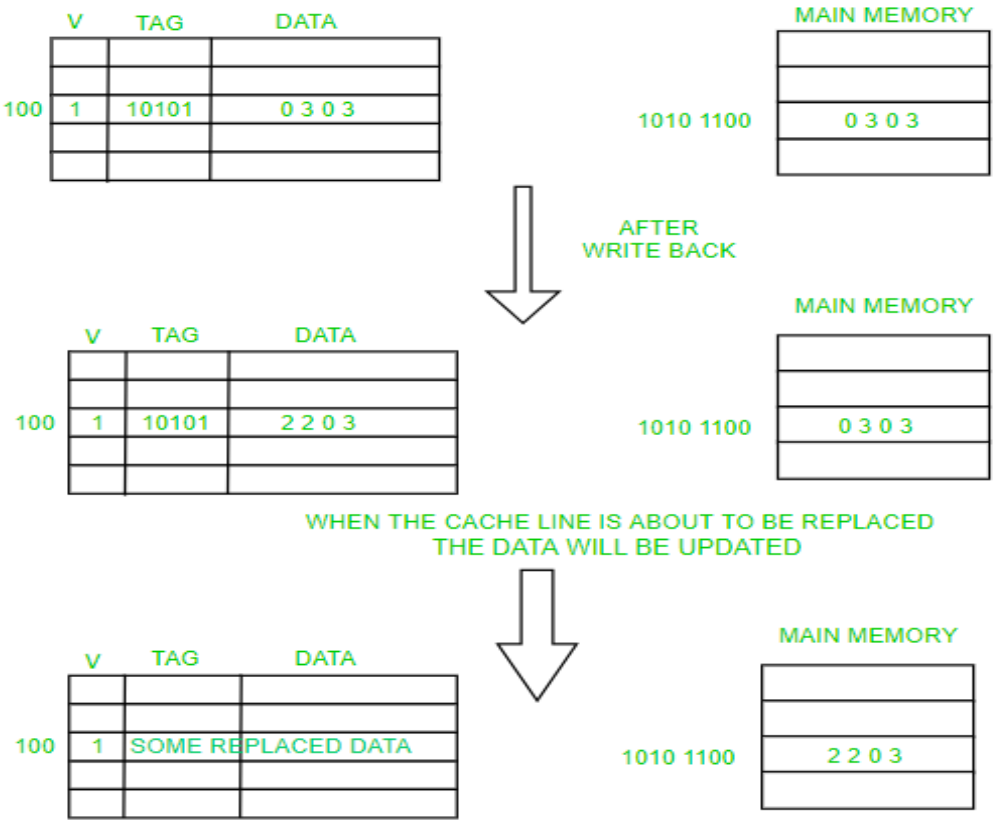
Write Through:

- In write through, data is **simultaneously updated to cache and memory**. This process is simpler and more reliable. This is used when there are no frequent writes to the cache(Number of write operation is less).

- It helps in data recovery (In case of power outage or system failure). A data write will experience latency (delay) as we have to write to two locations (both Memory and Cache). It Solves the inconsistency problem. But it questions the advantage of having a cache in write operation (As the whole point of using a cache was to avoid multiple accessing to the main memory).



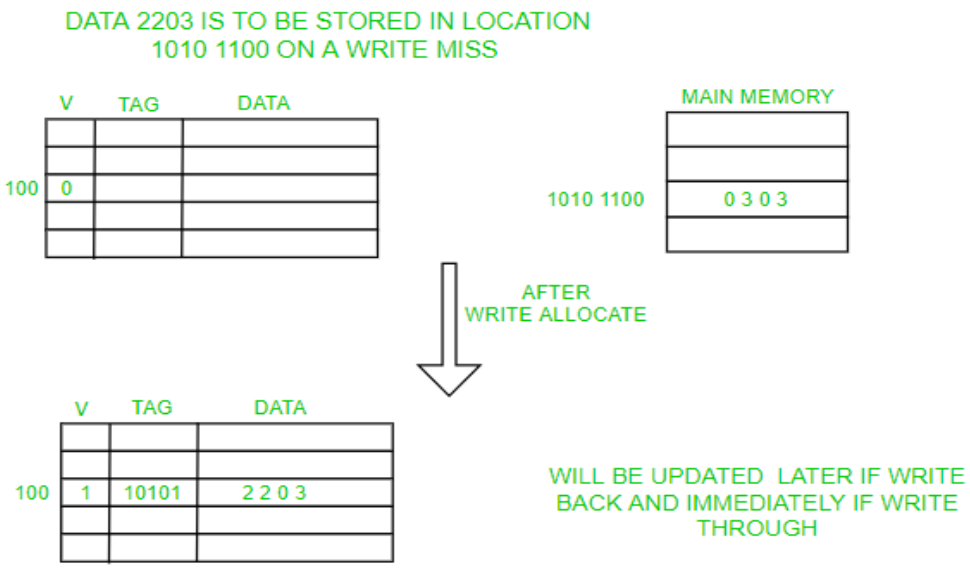
Write Back:



- The data is updated only in the cache and updated into the memory in later time. Data is updated in the memory only when the cache line is ready to replaced (cache line replacement is done using Belady's Anomaly, Least Recently Used Algorithm, FIFO, LIFO and others depending on the application). Write Back is also known as **Write Deferred**.

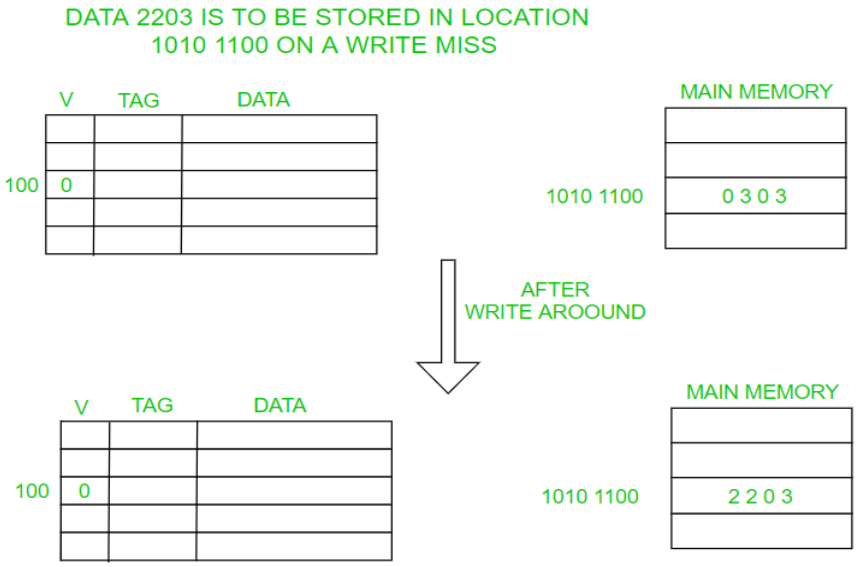
- **Dirty Bit** : Each Block in the cache needs a bit to indicate if the data present in the cache was modified(Dirty) or not modified(Clean).If it is clean there is no need to write it into the memory. It designed to reduce write operation to a memory. If **Cache fails** or if **System fails or power outage** the modified data will be lost. Because its nearly impossible to restore data from cache if lost.
- If write occurs to a location that is not present in the Cache (Write Miss), we use two options, **Write Allocation** and **Write Around**.

Write Allocation:



- In Write Allocation data is loaded from the memory into cache and then updated. Write allocation works with both Write back and Write through. But it is generally used with Write Back because it is unnecessary to bring data from the memory to cache and then updating the data in both cache and main memory. Thus, Write Through is often used with No write Allocate.

Write Around:



Here data is Directly written/updated to main memory without disturbing cache. It is better to use this when the data is not immediately used again.

Differentiate between Write Through and Write Back Methods

During a read operation, when the CPU determines a word in the cache, the main memory is not included in the transfer. Thus, there are two ways that the system can proceed when the operation is a write.

1. Write Through Method: The simplest method is to update the main memory with every memory write operation, when the cache memory is update in parallel when it contains the word at the specified address. This can be known as the write through method.

2. Write Back Method:

During write operation, only the cache location is updated in the write back method. Then, the location is marked by a flag so that it is later copied to the main memory when the word is removed from the cache. For the write back method, the reason is that during the time a word remains in the cache, it can be updated multiple times. Thus, as long as the word remains in the cache, it does not matter if the copy in the main cache. This is only when the word is displaced from the cache which needs an exact copy that is rewritten into main memory.

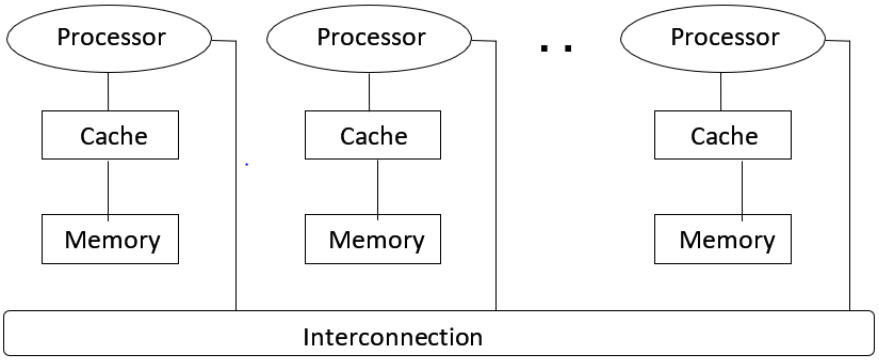
Differentiate Between Write Through and Write Back Methods:

WRITE THROUGH METHOD	WRITE BACK METHOD
In this method main memory is updated with every memory write operation as well as cache memory is updated in parallel if it contains the word at the specified address.	In this method only cache location is updated during write operation.
Main memory always contains same data as cache.	Main memory and cache memory may have different data.
Number of memory write operation in a typical program is more.	Number of memory write operation in a typical program is less
When I/O device communicated through DMA would receive most recent data.	When I/O device communicated through DMA would not receive most recent data.
It is a process of writing cache and main memory simultaneously.	It is a process of writing cache and data is removed from cache, first copied to main memory.

Cache Coherence and Synchronization

Cache coherence problem

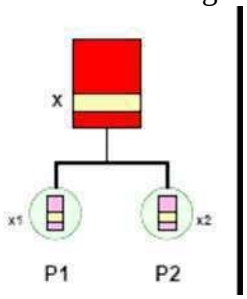
- An important problem that must be addressed in many parallel systems - any system that allows multiple processors to access (potentially) multiple copies of data is cache coherence. The existence of multiple cached copies of data creates the possibility of inconsistency between a cached copy and the shared memory or between cached copies themselves.



There are three common sources of cache inconsistency:

i. Inconsistency in data sharing:

- In a memory hierarchy for a multiprocessor system data inconsistency may occur between adjacent levels or within the same level. The cache inconsistency problem occurs only when multiple private caches are used. Thus, it is, the possible that a wrong data being accessed by one processor because another processor has changed it, and not all changes have yet been propagated. Suppose we have two processors, A and B, each of which is dealing with memory word X, and each of which has a cache. If processor A changes X, then the value seen by processor B in its own cache will be wrong, even if processor A also changes the value of X in main memory.



- In above example initially, x1 = x2 = X = 5. P1 writes X: =10 using write-through. P2 now reads X and uses its local copy x2 but finds that X is still 5.
- **Thus, P2 does not know that P1 modified X.**
- Thus, the cache inconsistency problem occurs when multiple private cache are used and especially the problem arose by writing the shared variables.

ii. Process migration (even if jobs are independent):

- This problem occurs when a process containing shared variable X migrates from process 1 to process2 using the write back cache on the right. Thus, another important aspect of coherence is serialization of writes - that is, if two processors try to write 'simultaneously', then:
  - (i) the writes happen sequentially (and it doesn't really matter who gets to write first - provided we have sensible arbitration); and
  - (ii) all processors see the writes as occurring in the same order. That is, if processors A and B both write to X, with A writing first, then any other processors (C, D, E) all see the same thing.

iii. DMA I/O:

- This inconsistency problem occurs during the I/O operation that bypass the cache. This problem is present even in a uniprocessor and can be removed by OS cache flushes).
- In practice, these issues are managed by a memory bus, which by its very nature ensures write serialization, and also allows us to broadcast invalidation signals (we essentially just put the memory address to be invalidated on the bus). We can add an extra valid bit to cache tags to mark then invalid. Typically, we would use a write-back cache because it has much lower memory bandwidth requirements. Each processor must keep track of which cache blocks are dirty - that is, that it has written to - again by adding a bit to the cache tag. If it sees a memory access for a word in a cache block it has marked as dirty, it intervenes and provides the (updated) value. There are numerous other issues to address when considering cache coherence.
- One approach to maintaining coherence is to recognize that not every location needs to be shared (and in fact most don't), and simply reserve some space for non-cacheable data such as semaphores, called a coherency domain.
- Using a fixed area of memory, however, is very restrictive. Restrictions can be reduced by allowing the MMU to tag segments or pages as non-cacheable. However, that requires the OS, compiler, and programmer to be involved in specifying data that is to be coherently shared. For example, it would be necessary to distinguish between the sharing of semaphores and simple data so that the data can be cached once a processor owns its semaphore, but the semaphore itself should never be cached.

In order to remove this data inconsistency there are a number of approaches based on hardware and software techniques few are given below:

- No caches are used which is not a feasible solution
- Make shared-data non-cacheable this is the simplest software solution but produce low performance if a lot of data is shared
- software flush at strategic times: e.g., after critical sections, this is relatively simple technique but has low performance if synchronization is not frequent
- hardware cache coherence this can be achieved by making memory and caches coherent (consistent) with each other, in other words if the memory and other processors see writes then without intervention of the software
- absolute coherence i.e. all copies of each block have same data at all times
- It is not necessary what is required is appearance of absolute coherence that is done by making temporary incoherence is OK (e.g., write-back cache)
- In general, a cache coherence protocols consist of the set of possible states in local caches, the state in shared memory and the state transitions caused by the messages transported through the interconnection network to keep memory coherent. There are basically two kinds of protocols depends on how writes are handled.

Other protocols used for cache coherence and synchronization:

i. Snooping Cache Protocol (for bus-based machines)

- With a bus interconnection, cache coherence is usually maintained by adopting a "snoopy protocol", where each cache controller "snoops" on the transactions of the other caches and guarantees the validity of the cached data. In a (single-) multi-stage network, however, the unavailability of a system "bus" where transactions are broadcast makes snoopy protocols not useful. Directory based schemes are used in this case. In case of snooping protocol processors perform some form of snooping - that is, keeping track of other processor's memory writes. ALL caches/memories see and react to ALL bus events. The protocol relies on global visibility of requests (ordered broadcast). This allows the processor to make state transitions for its cache-blocks.

ii. Write Invalidate protocol

- The states of a cache block copy changes with respect to read, write and replacement operations in the cache. The most common variant of snooping is a write invalidate protocol.
- Consider a scenario, when processor A writes to X, it broadcasts the fact and all other processors with a copy of X in their cache mark it invalid. When another processor (B, say) tries to access X again then there will be a cache miss and either

i. in the case of a write-through cache the value of X will have been updated (actually, it might not because not enough time may have elapsed for the memory write to complete - but that's another issue); or

ii. in the case of a write-back cache processor A must spot the read request, and substitute the correct value for X.

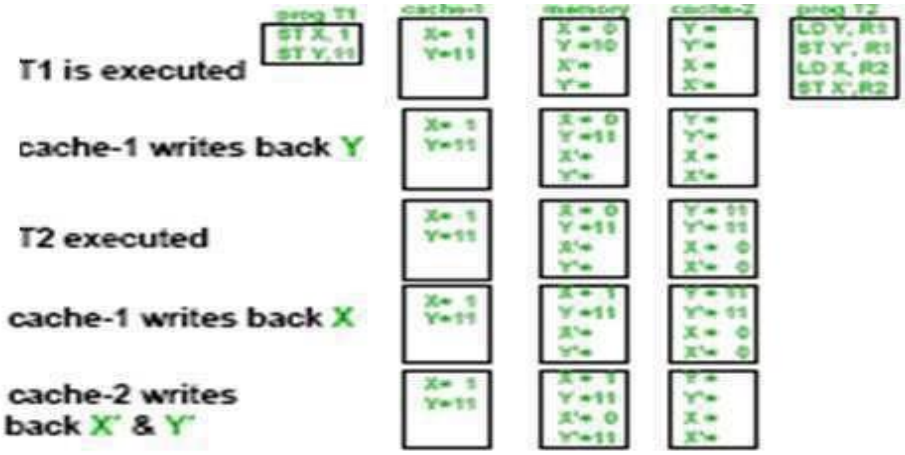


Figure: Write back with cache

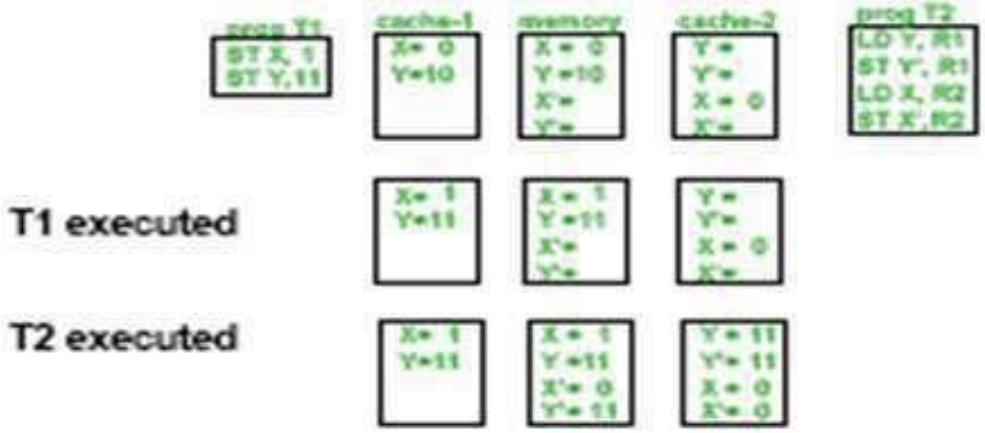


Figure: Write through with cache

- An alternative (but less-common) approach is write broadcast. This is intuitively a little more obvious - when a cached value is changed, the processor that changed it broadcasts the new value to all other processors. They then update their own cached values. The trouble with this scheme is that it uses up more memory bandwidth. A way to cut this is to observe that many memory words are not shared - that is, they will only appear in one cache. If we keep track of which words are shared and which are not, we can reduce the amount of broadcasting necessary. There are two main reasons why more memory bandwidth is used: in an invalidation scheme, only the first change to a word requires an invalidation signal to be broadcast, whereas in a write broadcast scheme all changes must be signaled; and in an invalidation scheme only the first change to any word in a cache block must be signaled, whereas in a write broadcast scheme every word that is written must be signaled. On the other hand, in a write broadcast scheme we do not end up with a cache miss when trying to access a changed word, because the cached copy will have been updated to the correct value.

Processor Activity	Bus Activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of Memory Location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Write Broadcast for X	1	1	1
CPU B reads X		1	1	1

Figure: write-back with broadcast if different processors operate on different data items, these can be cached.

1. Once these items are tagged dirty, all subsequent operations can be performed locally on the cache without generating external traffic.
2. If a data item is read by number of processors, it transitions to the shared state in the cache and all subsequent read operations become local.

In both cases, the coherence protocol does not add any overhead.

iii. Directory-based Protocols:

- When a multistage network is used to build a large multiprocessor system, the snoopy cache protocols must be modified. Since broadcasting is very expensive in a multistage network, consistency commands are sent only to caches that keep a copy of the block. This leads to Directory Based protocols.
- A directory is maintained that keeps track of the sharing set of each memory block. Thus, each bank of main memory can keep a directory of all caches that have copied a particular line (block). When a processor writes to a location in the block, individual messages are sent to any other caches that have copies. Thus, the Directory-based protocols selectively send invalidation/update requests to only those caches having copies—the sharing set leading the network traffic limited only to essential updates. Proposed schemes differ in the latency with which memory operations are performed and the implementation cost of maintaining the directory.
- The memory must keep a bit-vector for each line that has one bit per processor, plus a bit to indicate ownership (in which case there is only one bit set in the processor vector).

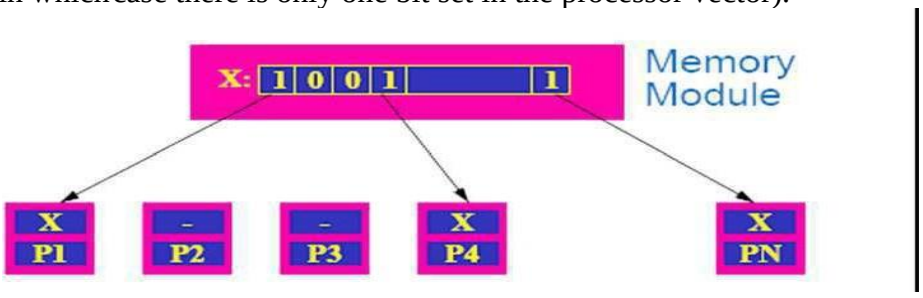


Figure: Directory based protocol

- These bitmap entries are sometimes referred to as the presence bits. Only processors that hold a particular block (or are reading it) participate in the state transitions due to coherence operations. Note that there may be other state transitions triggered by processor read, write, or flush (retiring a line from cache) but these transitions can be handled locally with the operation reflected in the presence bits and state in the directory. If different processors operate on distinct data blocks, these blocks become dirty in the respective caches and all operations after the first one can be performed locally.
- If multiple processors read (but do not update) a single data block, the data block gets replicated in the caches in the shared state and subsequent reads can happen without triggering any coherence overheads.
- Various directory-based protocols differ mainly in how the directory maintains information and what information is stored. Generally speaking, the directory may be central or distributed. Contention and long search times are two drawbacks in using a central directory scheme. In a distributed-directory scheme, the information about memory blocks is distributed. Each processor in the system can easily "find out" where to go for "directory information" for a particular memory block.
- Directory-based protocols fall under one of three categories:
    i. Full-map directories,
    ii. limited directories, and
    iii. chained directories.

i. This full-map protocol is extremely expensive in terms of memory as it stores enough data associated with each block in global memory so that every cache in the system can simultaneously store a copy of any block of data. It thus defeats the purpose of leaving a bus-based architecture.

ii. A limited-map protocol stores a small number of processor ID tags with each line in main memory. The assumption here is that only a few processors share data at one time. If there is a need for more processors to share the data than there are slots provided in the directory, then broadcast is used instead.

iii. Chained directories have the main memory store a pointer to a linked list that is itself stored in the caches. Thus, an access that invalidates other copies goes to memory and then traces a chain of pointers from cache to cache, invalidating along the chain. The actual write operation stalls until the chain has been traversed. Obviously, this is a slow process.

- Duplicate directories can be expensive to implement, and there is a problem with keeping them consistent when processor and bus accesses are asynchronous. For a write-through cache, consistency is not a problem because the cache must go out to the bus anyway, precluding any other master from colliding with its access. But in a write-back cache, care must be taken to stall processor cache writes that change the directory while other masters have access to the main memory.
- On the other hand, if the system includes a secondary cache that is inclusive of the primary cache, a copy of the directory already exists. Thus, the snooping logic can use the secondary cache directory to compare with the main memory access, without stalling the processor in the main cache. If a match is found, then the comparison must be passed up to the primary cache, but the number of such stalls is greatly reduced due to the filtering action of the secondary cache comparison.
- A variation on this approach that is used with write-back caches is called dirty inclusion, and simply requires that when a primary cache line first becomes dirty, the secondary line is similarly marked. This save writing through the data, and writing status bits on every write cycle, but still enables the secondary cache to be used by the snooping logic to monitor the main memory accesses. This is especially important for a read- miss, which must be passed to the primary cache to be satisfied.

- The previous schemes have all relied heavily on broadcast operations, which are easy to implement on a bus. However, buses are limited in their capacity and thus other structures are required to support sharing for more than a few processors. These structures may support broadcast, but even so, broadcast- based protocols are limited.
- The problem is that broadcast is an inherently limited means of communication. It implies a resource that all processors have access to, which means that either they contend to transmit, or they saturate on reception, or they have a factor of N hardware for dealing with the N potential broadcasts.

NOTES:

- Snoopy cache protocols are not appropriate for large-scale systems because of the bandwidth consumed by the broadcast operations
- In a multistage network, cache coherence is supported by using cache directories to store information on where copies of cache reside.
- A cache coherence protocol that does not use broadcast must store the locations of all cached copies of each block of shared data. This list of cached locations whether centralized or distributed is called a cache directory. A directory entry for each block of data contains number of pointers to specify the locations of copies of the block.

Distributed directory schemes

- In scalable architectures, memory is physically distributed across processors. The corresponding presence bits of the blocks are also distributed. Each processor is responsible for maintaining the coherence of its own memory blocks. Since each memory block has an owner its directory location is implicitly known to all processors. When a processor attempts to read a block for the first time, it requests the owner for the block. The owner suitably directs this request based on presence and state information locally available. When a processor writes into a memory block, it propagates an invalidate to the owner, which in turn forwards the invalidate to all processors that have a cached copy of the block. Note that the communication overhead associated with state update messages is not reduced.
- Distributed directories permit O(p) simultaneous coherence operations, provided the underlying network can sustain the associated state update messages. From this point of view, distributed directories are inherently more scalable than snoopy systems or centralized directory systems. The latency and bandwidth of the network become fundamental performance bottlenecks for such systems.

Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

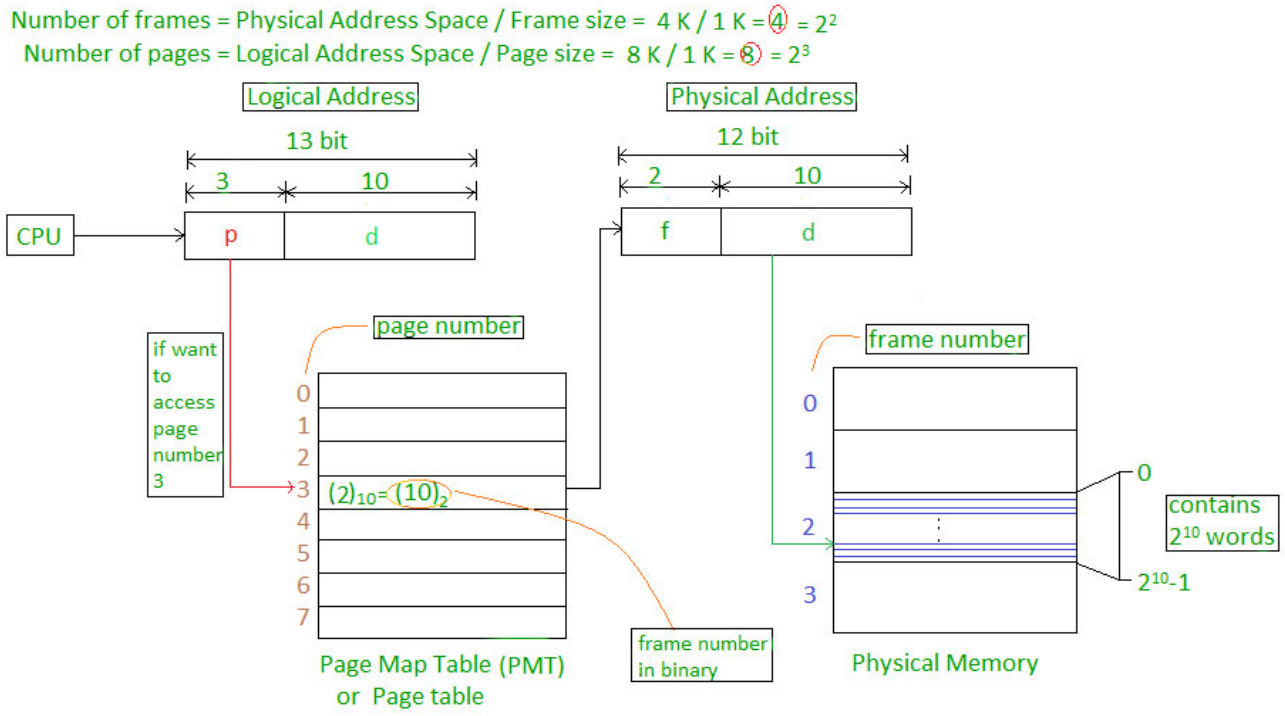
- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 * 2^{20}$ words, then Physical Address = $\log_2 2^{24}$ = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device, and this mapping is known as paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



Address generated by CPU is divided into

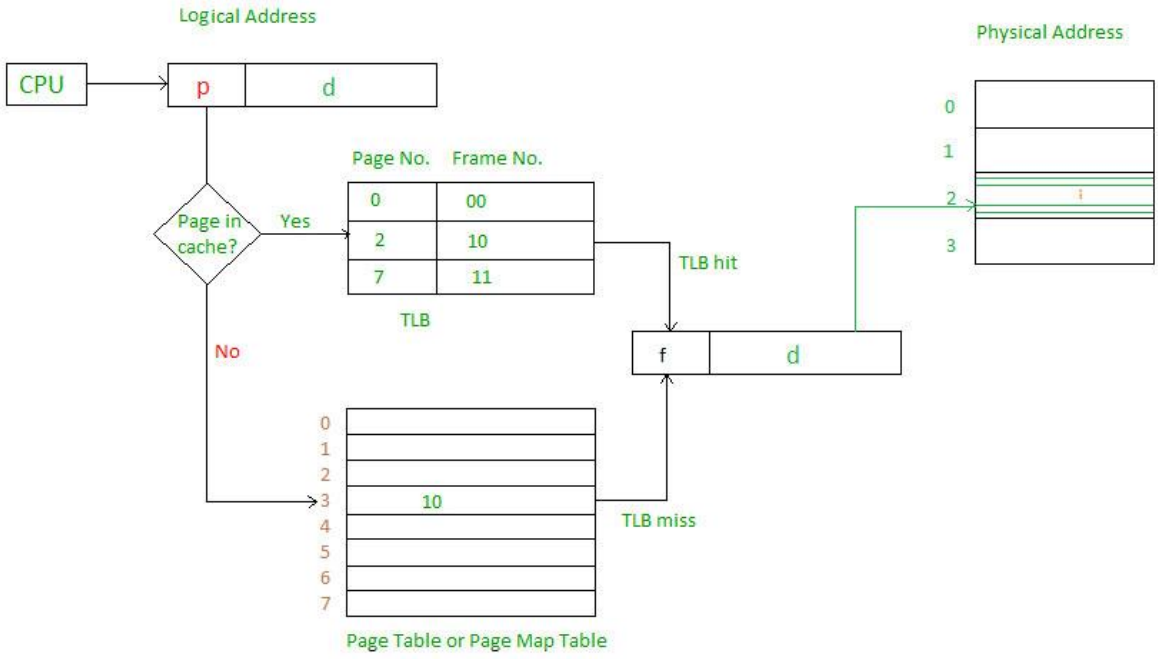
- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB (translation Look-aside buffer), a special, small, fast look up hardware cache.

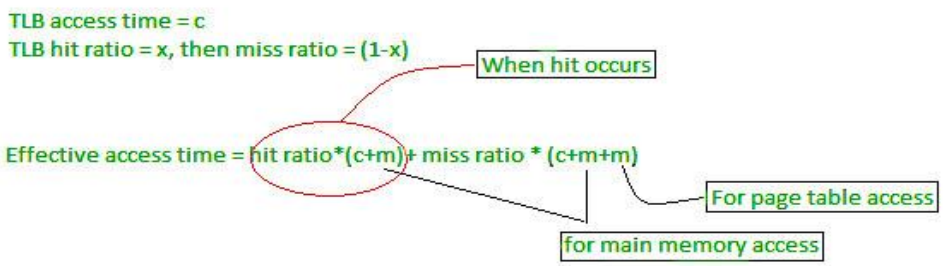
- The TLB is associative, high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then corresponding value is returned.



Main memory access time = m

If page table are kept in main memory,

Effective access time = m (for page table) + m (for particular page in page table)



virtual memory

- Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory. Virtual memory is a very common technique used in the operating systems (OS) of computers.
- Virtual memory uses hardware and software to allow a computer to compensate for physical memory shortages, by temporarily transferring data from random access memory (RAM) to disk storage. In essence, virtual memory allows a computer to treat secondary memory as though it were the main memory.
- Today, most PCs come with up to around 4 GB of RAM. However, sometimes this isn't enough to run all the programs a user might want to use at once. This is where virtual memory comes in.
- Virtual memory can be used to swap data that has not been used recently and move it over to a storage device like a hard drive or solid-state drive (SDD). This will free up more space on the RAM.
- Virtual memory is important for improving system performance, multitasking, using large programs and flexibility. However, users shouldn't rely on virtual memory too much, because using virtual data is considerably slower than the use of RAM. If the OS has to swap data between virtual memory and RAM too often, it can make the computer feel very slow, this is called **thrashing.**
- Virtual memory was developed at a time when physical memory, also referenced as RAM, was expensive. Computers have a finite amount of RAM, so memory can run out, especially when multiple programs run at the same time. A system using virtual memory uses a section of the hard drive to emulate RAM. With virtual memory, a system can load larger programs or multiple programs running at the same time, allowing each one to operate as if it has infinite memory and without having to purchase more RAM.

How virtual memory works?

- Virtual memory uses both computer hardware and software to work. When an application is in use, data from that program is stored in a physical address using RAM. More specifically, virtual memory will map that address to RAM using a memory management unit (MMU). The OS will make and manage memory mappings by using page tables and other data structures. The MMU, which acts as an address translation hardware, will automatically translate the addresses.
- If at any point later the RAM space is needed for something more urgent, the data can be swapped out of RAM and into virtual memory. The computer's memory manager is in charge of keeping track of the shifts between physical and virtual memory. If that data is needed again, a context switch can be used to resume execution again.
- While copying virtual memory into physical memory, the OS divides memory into page-files or swap-files with a fixed number of addresses. Each page is stored on a disk, and when the page is needed, the OS copies it from the disk to main memory and translates the virtual addresses into real addresses.
- However, the process of swapping virtual memory to physical is rather slow. This means that using virtual memory generally causes a noticeable reduction in performance. Because of swapping, computers with more RAM are seen to have better performance.

Types of virtual memory:

- A computer's MMU handles memory operations, including managing virtual memory. In most computers, the MMU hardware is integrated into the CPU. There are two ways in which virtual memory is handled: paged and segmented.
- Paging divides memory into sections or paging files, usually approximately 4 KB in size. When a computer uses up its RAM, pages not in use are transferred to the section of the hard drive designated for virtual memory using a swap file. A swap file is a space set aside on the hard drive as the virtual memory extensions of the computer's RAM. When the swap file is needed, it's sent back to RAM using a process called page swapping. This system ensures that the computer's OS and applications don't run out of real memory.

i. paging without virtual memory.

- The paging process includes the use of page tables, which translate the virtual addresses that the OS and applications use into the physical addresses that the MMU uses. Entries in the page table indicate whether the page is in real memory. If the OS or a program doesn't find what it needs in RAM, then the MMU responds to the missing memory reference with a page fault exception to get the OS to move the page back to memory when it's needed. Once the page is in RAM, its virtual address appears in the page table.
- Segmentation is also used to manage virtual memory. This approach divides virtual memory into segments of different lengths. Segments not in use in memory can be moved to virtual memory space on the hard drive. Segmented information or processes are tracked in a segment table, which shows if a segment is present in memory, whether it's been modified and what its physical address is. In addition, file systems in segmentation are only made up of a list of segments mapped into a process's potential address space.

ii. virtual memory with segmentation.

- Segmentation and paging differ as a memory model in terms of how memory is divided; however, it can also be combined. Some virtual memory systems combine segmentation and paging. In this case, memory gets divided into frames or pages. The segments take up multiple pages, and the virtual address includes both the segment number and the page number.

How to manage virtual memory?

- Operating systems have default settings that determine the amount of hard drive space to allocate for virtual memory. That setting will work for most applications and processes, but there may be times when it's necessary to manually reset the amount of hard drive space allocated to virtual memory, such as with applications that depend on fast response times or when the computer has multiple HDDs.
- When manually resetting virtual memory, the minimum and maximum amount of hard drive space to be used for virtual memory must be specified. Allocating too little HDD space for virtual memory can result in a computer running out of RAM. If a system continually needs more virtual memory space, it may be wise to consider adding RAM. Common operating systems may generally recommend users not increasing virtual memory beyond 1.5 times the amount of RAM.
- Managing virtual memory may be a different experience on different types of operating systems, however. And IT professionals should understand the basics when it comes to managing physical memory, virtual memory and virtual addresses. For example, here are some tips on managing virtual memory on the Windows 10 operating system.

Benefits of using virtual memory

Benefits of virtual memory include:

- its ability to handle twice as many addresses as main memory;
- frees applications from managing shared memoryand saves users from having to add memory modules when RAM space runs out;
- increased security because of memory isolation;
- multiple larger applications can be run simultaneously;
- allocating memory is relatively cheap;
- doesn't need external fragmentation;
- effective CPU use;
- data can be moved automatically; and
- pages in the original process can be shared during a fork system call.

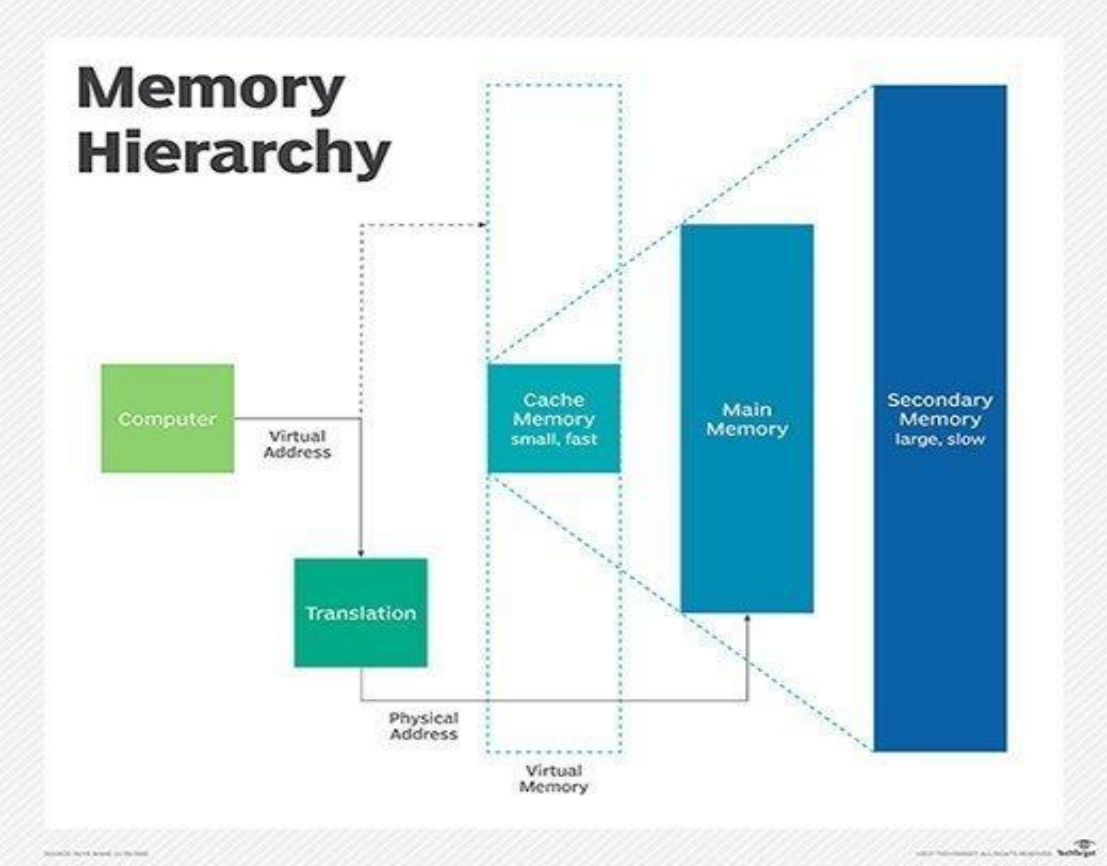


Figure: The memory hierarchy of a computer considering primary and secondary storage.

- In addition, in a virtualized computing environment, administrators can use virtual memory management techniques to allocate additional memory to a virtual machine (VM) that has run out of resources. Such virtualization management tactics can improve VM performance and management flexibility.

Limitations

- The use of virtual memory has its tradeoffs, particularly with speed. It's generally better to have as much physical memory as possible, so programs work directly from RAM or physical memory.
- The use of virtual memory slows a computer because data must be mapped between virtual and physical memory, which requires extra hardware support for address translations.

- The size of virtual storage is limited by the amount of secondary storage, as well as the addressing scheme with the computer system.
- Thrashing can happen if the amount of RAM is too small, which will make the computer perform slower.
- It may take time to switch between applications using virtual memory.

Virtual memory vs. physical memory

- When talking about the differences between virtual and physical memory, the biggest distinction is normally seen to be in speed. RAM is considerably faster than virtual memory. RAM, however, tends to be more expensive than virtual memory.
- When a computer requires storage, RAM is the first used. Virtual memory is used when the RAM is filled, because it's slower.
- Users can actively add RAM to a computer by buying and installing more RAM chips if they are experiencing slowdowns due to memory swaps happening too often. The amount of RAM depends on what's installed on a computer. Virtual memory, on the other hand, is limited by the size of the computer's hard drive. Virtual memory settings can often be controlled through the operating system.

History

- Before virtual memory was developed, computers had RAM and secondary memory. Early computers used magnetic core memory for main memory and magnetic drums for their secondary memory. Computer memory was expensive and usually in short supply back in the 1940s and 1950s. As computer programs grew in size and complexity, developers had to worry that their programs would use up all of a computer's main memory and run out of memory.
- In those early days, programmers used a process called overlaying to run programs that were larger than available memory. Parts of a program that weren't continually in use were set up as overlays that, when needed, would overwrite the existing overlay in memory. It required extensive programming to make overlaying work, and that was a key impetus for the development of automated virtual memory.
- German physicist Fritz-Rudolf Güntsch has been credited with developing the concept of virtual memory in 1956, though this point is contested. Güntsch did, however, end up describing a form of cache memory.
- The first apparent real instance of a virtual memory system came from the University of Manchester, in their attempt to develop a one-level storage system for the Atlas computer. The system used paging to map virtual addresses to a programmer on to the primary memory. Atlas was developed in 1959 and later commissioned in 1962.
- In 1961, the first commercial computer with virtual memory was released by the Burroughs Corporation. This version of virtual memory used segmentation as opposed to paging.
- In 1969, IBM researchers demonstrated that virtual memory overlay systems worked better than the earlier manual systems. Up until this point, there was still a debate over this. Mainframes and minicomputers in the 1970s generally used virtual memory. Virtual memory technology was not included in early personal computers because developers thought running out of memory would not be a problem in those machines. That assumption proved incorrect. Intel introduced virtual memory in the protected mode of the 80286 processor in 1982, and paging support when the 80386 came out in 1985.

Operating System - Virtual Memory

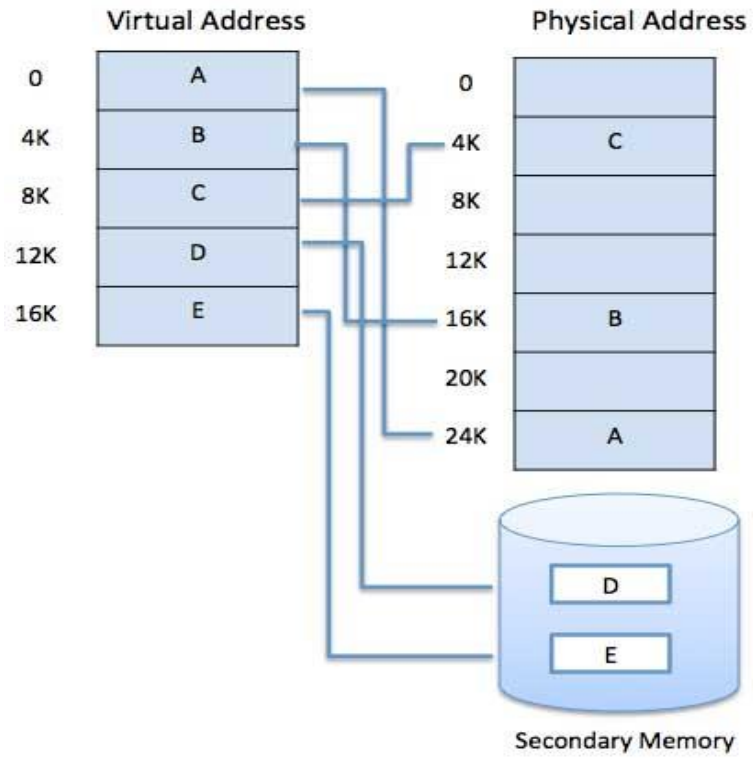
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

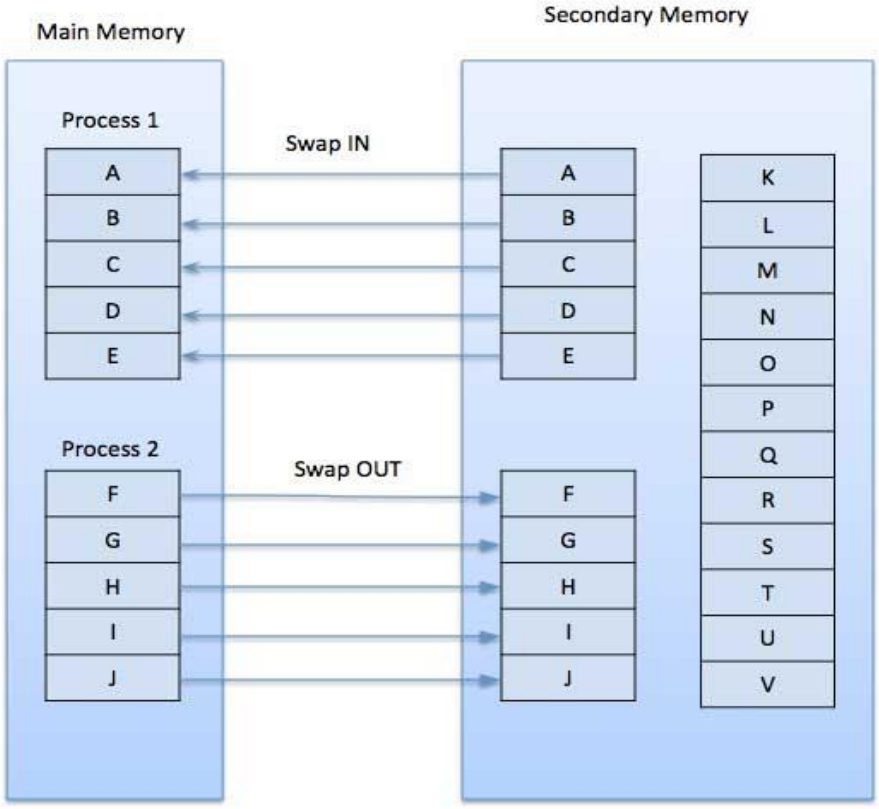
Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement-

- Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.
- Page replacement is required when-
- All the frames of main memory are already occupied.
- Thus, a page has to be replaced to create a room for the required page.

Page Replacement Algorithms-

- Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.
- Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.
- When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.
- A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults.

Various page replacement algorithms are-

- FIFO Page Replacement Algorithm
- LRU Page Replacement Algorithm
- Optimal Page Replacement Algorithm

NOTE: A good page replacement algorithm is one that minimizes the number of page faults.

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

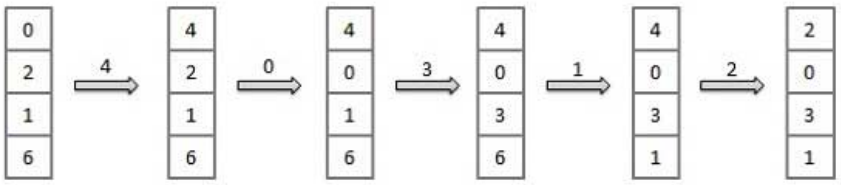
- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

i. First In First Out (FIFO) algorithm

- As the name suggests, this algorithm works on the principle of "**First in First out**".
- It replaces the oldest page that has been present in the main memory for the longest time.
- It is implemented by keeping track of all the pages in a queue.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



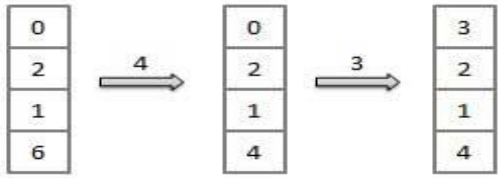
Fault Rate = 9 / 12 = 0.75

ii. Optimal Page replacement algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- This algorithm replaces the page that will not be referred by the CPU in future for the longest time.
- It is practically impossible to implement this algorithm.
- This is because the pages that will not be used in future for the longest time cannot be predicted.
- However, it is the best-known algorithm and gives the least number of page faults.
- Hence, it is used as a performance measure criterion for other algorithms.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



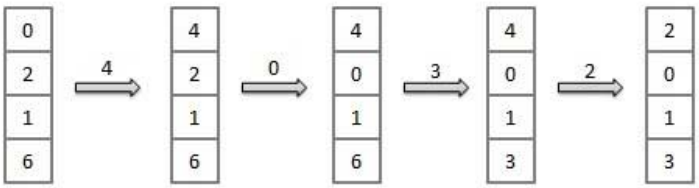
Fault Rate = 6 / 12 = 0.50

iii. Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

<u>Numerical on Optimal, LRU and FIFO</u>

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

- Optimal Page Replacement Algorithm
- FIFO Page Replacement Algorithm
- LRU Page Replacement Algorithm

Optimal Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

Page Fault in OS-

- A page fault occurs when a page referenced by the CPU is not found in the main memory.
- The required page has to be brought from the secondary memory into the main memory.
- A page has to be replaced if all the frames of main memory are already occupied.

Problem-01:

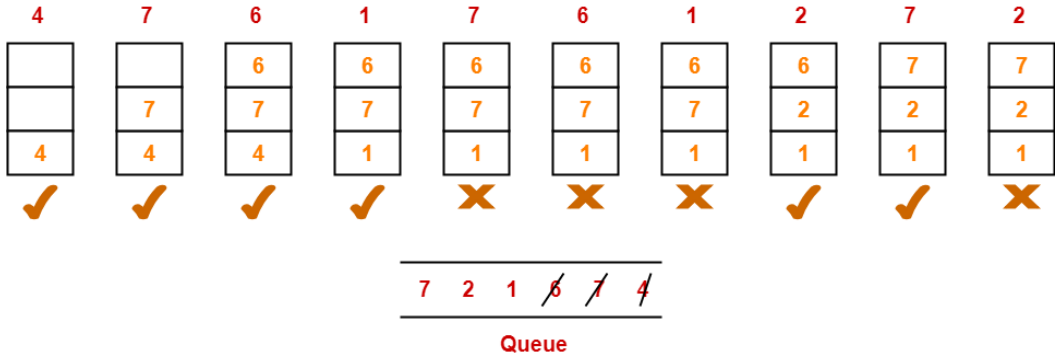
A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



From here,

Total number of page faults occurred = 6

Calculating Hit ratio-

Total number of page hits
= Total number of references – Total number of page misses or page faults
= 10 – 6 = 4
Thus, Hit ratio
= Total number of page hits / Total number of references
= 4 / 10 = 0.4 or 40%

Calculating Miss ratio-

Total number of page misses or page faults = 6
Thus, Miss ratio
= Total number of page misses / Total number of references
= 6 / 10 = 0.6 or 60%

Alternatively,
Miss ratio = 1 – Hit ratio
= 1 – 0.4 = 0.6 or 60%

Problem-02:

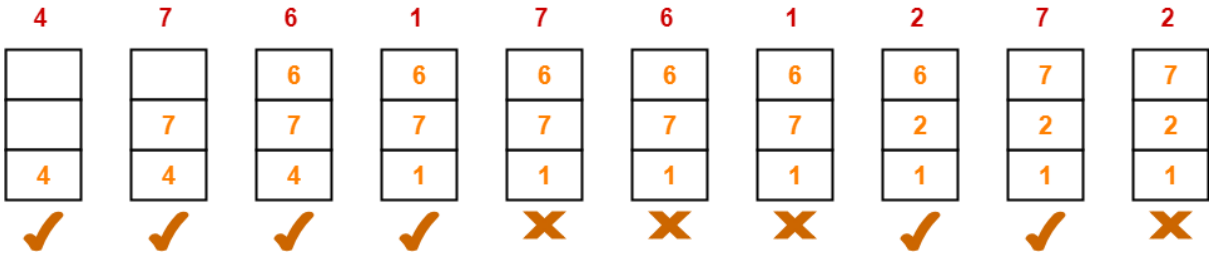
A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



From here,
Total number of page faults occurred = 6

Calculating Miss ratio-

Hit ratio = Total number of page hits / Total number of references
= (Total number of references-number of page fault) / Total number of references
= 4 / 10 = 0.4 or 40%

Calculating Miss ratio-

Total number of page misses or page faults = 6
Thus, Miss ratio
= Total number of page misses / Total number of references
= 6 / 10 = 0.6 or 60%

Problem-03:

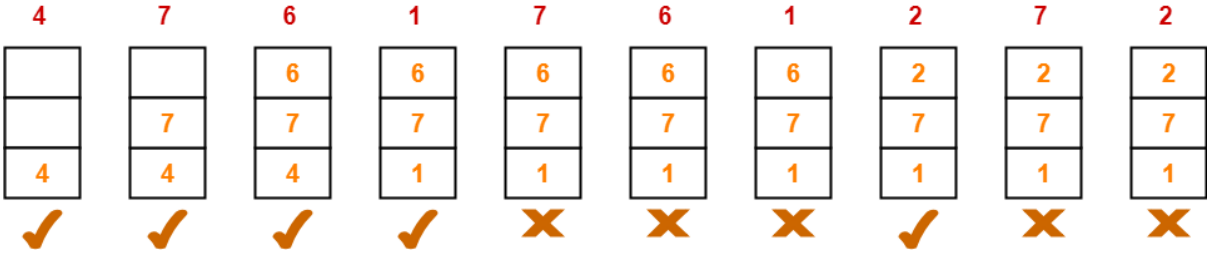
A system uses 3 page frames for storing process pages in main memory. It uses the Optimal page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below-

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



From here,
Total number of page faults occurred = 5

In the similar manner as above-
Hit ratio = 0.5 or 50%
Miss ratio = 0.5 or 50%

Computer Architecture: Input/Output Organisation

Input/Output Subsystem

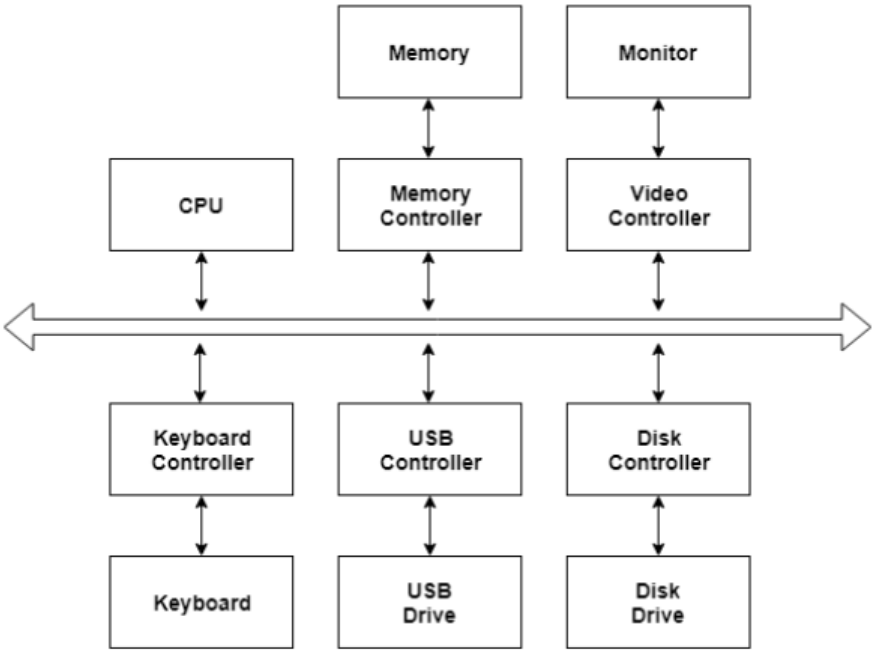
- I/O devices are very important in the computer systems. They provide users the means of interacting with the system. So, there is a separate I/O system devoted to handling the I/O devices.

The different Components of the I/O systems are −

I/O Hardware

- There are many I/O devices handled by the operating system such as mouse, keyboard, disk drive etc. There are different device drivers that can be connected to the operating system to handle a specific device. The device controller is an interface between the device and the device driver.

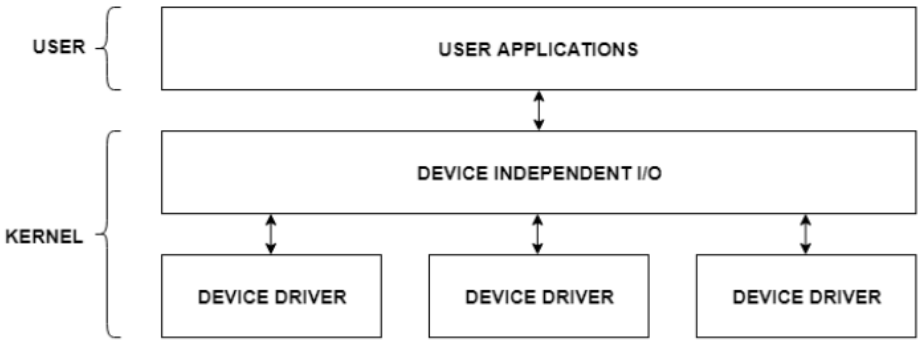
A diagram to represent this is −



I/O Application Interface

- The user applications can access all the I/O devices using the device drivers, which are device specific codes. The application layer sees a common interface for all the devices.

This is illustrated using the below image −



Most of the devices are either block I/O and character I/O devices. Block devices are accessed one block at a time whereas character devices are accessed one character at a time.

I/O Software

- The I/O software contains the user level libraries and the kernel modules. The libraries provide the interface to the user program to perform input and output. The kernel modules provide the device drivers that interact with the device controllers.
- The I/O software should be device independent so that the programs can be used for any I/O device without specifying it in advance. For example - A program that reads a file should be able the read the file on a hard disk, floppy disk, CD-ROM etc. without having to change the program each time.

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

- **Input peripherals**: Allows user input, from the outside world to the computer.
  Example: Keyboard, Mouse etc.
- **Output peripherals**: Allows information output, from the computer to the outside world.
  Example: Printer, Monitor etc.
- **Input-Output peripherals**: Allows both input (from outside world to computer) as well as, output (from computer to the outside world).
  Example: Touch screen etc.

Interfaces

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Interface
2. I/O Interface

Input-Output Interface

- Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called **input-output interface units** because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer

- Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:
  - Programmed I/O
  - Interrupt Initiated I/O
  - Direct Memory Access

i. Programmed I/O

- Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.
- Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

ii. Interrupt Initiated I/O

- In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.
- This problem can be overcome by using **interrupt initiated I/O**. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

iii. Direct Memory Access

- Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as **DMA**.
- In this, the interface transfer data to and from the memory through memory bus. A DMA controller manages to transfer data between peripherals and memory unit.
- Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc. It is also used for intra chip data transfer in multicore processors. In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.

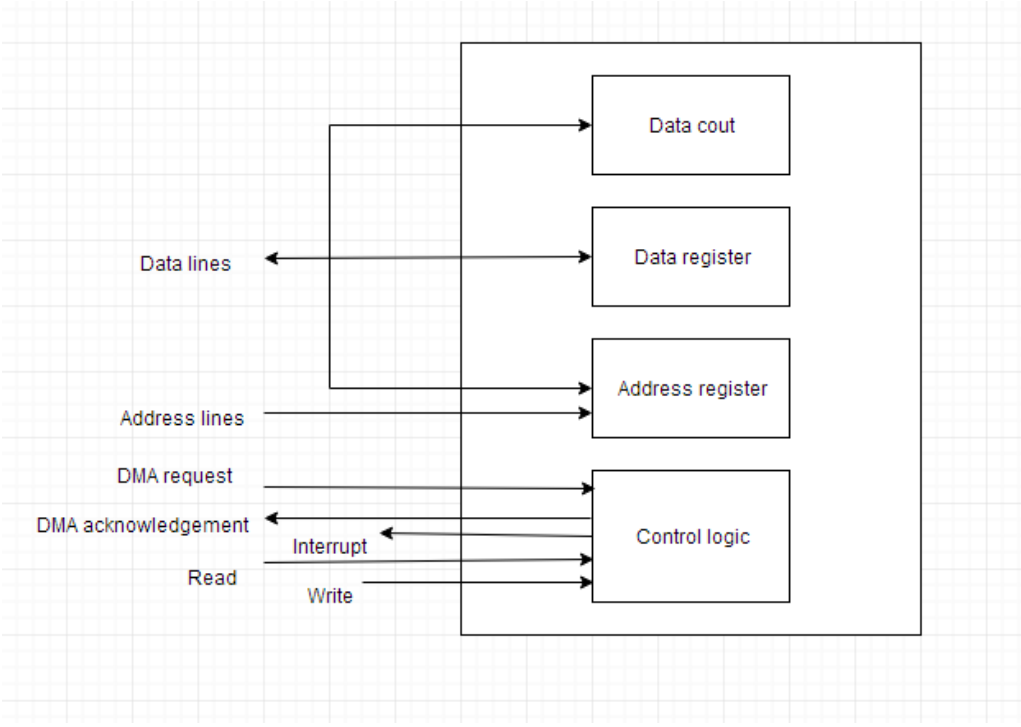


Figure: Block diagram of DMA