COMPUTER ARCHITECTURE

PIPELINE ARCHITECTURE

☞ Chapter at a Glance

Review of Pipeline Architecture

Pipeline is an implementation technique whereby multiple instructions are overlapped in execution. Each step in the pipeline (called a *pipe stage*) completes a part of an instruction. Because all stages proceed at the same time, the length of a processor (clock) cycle is determined by the time required for the slowest pipe stage. Designer's goal: Balancing the length of each pipeline stage. If the stages are perfectly balanced, the time per instruction on the pipelined processor is,

$$\frac{\text{Time per instruction on un-pipelined machine}}{\text{Number of pipe stages}}$$

The Speedup from pipelining is equal to the number of pipe stages.

Ideal Pipeline Techniques

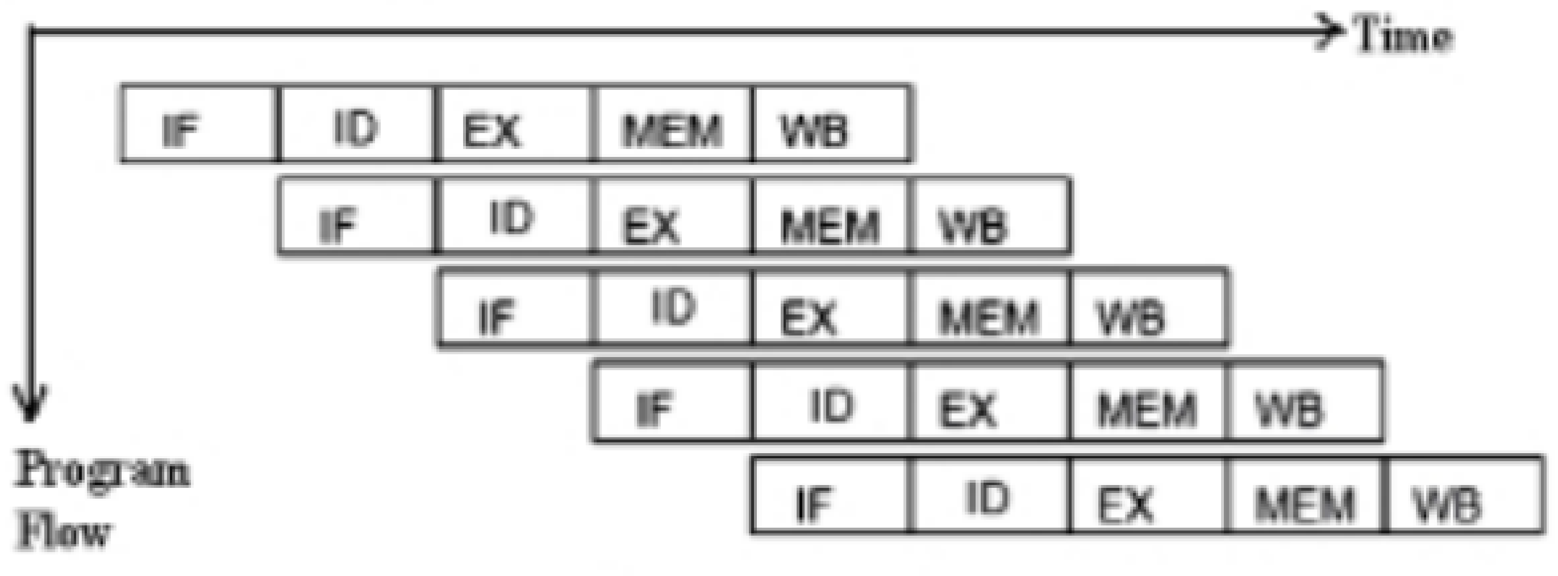


Fig: 1 Pipeline technique

- ***Pipeline throughput***
  The throughput of a pipeline is defined as the number of instructions that can be executed per time unit. All stages precede in synchronized fashion they all start at the same times (simplifies hardware design). The time required for moving one instruction down the pipeline is called a processor cycle (not to be confused with clock cycle). Because all pipe stages must be ready to proceed synchronously, the processor cycle is determined by the slowest stage.
- ***Performance of the pipeline***
  Pipelining achieves a reduction of the average execution time per instruction. This is in the sense that one can perform more instructions per clock cycle. This can be viewed in two ways:
  1. Decreasing the CPI. Typical way in which people view the performance increase
  2. Decreasing the cycle time (i.e., increasing the clock rate). The good news is that pipelining is typically invisible to the programmer.

Speed up, Efficiency and Throughput of Pipeline

A linear pipeline of k stages can process n tasks in k + (n–1) clock cycles, where k cycles are needed to complete the execution of the very first task and the remaining n–1 tasks require n–1 cycles. Thus the total time required is $T_k = [k + (n-1)]\tau$, where τ is the clock period.

Pipeline Hazards

In a pipeline, each instruction is supposed to start executing at a given clock cycle. Unfortunately there are cases in which an instruction cannot execute at its allotted clock cycle. These situations are called: pipeline hazards. Hazards further reduce the performance gain from the speedup.

- The hazard is a situation which prevents to fetch the next instructions in the instruction stream from executing during its designated clock cycle.
- Hazards reduce the performance from the ideal speedup gained by pipelining.
  - Data Hazards
  - Structural Hazards
  - Control Hazards
- Hazards can make it necessary to stall the pipeline.
  - When an instruction is stalled, all instructions issued later than the stalled instruction are also stalled.
  - No new instructions are fetched during the stall.

1. Pipeline stall

When hazards occur, the typical approach is to stall the pipeline. Delay an instruction in the pipeline to wait until another instruction completes execution, which is called stalling. When an instruction is stalled, all instructions issued after the stalled instruction are stalled. When an instruction is stalled, all instructions issued before the stalled instruction are allowed to continue. No new instruction is fetched during a stall.

2. Pipeline bubble

Pipeline Bubble (a technique also known as a pipeline break or pipeline stall) is a method for preventing data, structural, and branch hazards from occurring. As instructions are fetched, control logic determines whether a hazard will occur. If this is true, then the control logic inserts NOPs into the pipeline. Thus, before the next instruction (which would cause the hazard) is executed, the previous one will have had sufficient time to complete and prevent the hazard. If the number of NOPs is equal to the number of stages in the pipeline, the processor has been cleared of all instructions and can proceed free from hazards. This is called flushing the pipeline. All forms of stalling introduce a delay before the processor can resume execution.

Data Hazards

A hazard is created whenever there is dependence between instructions, and they are close enough that the overlap caused by pipelining, or other reordering of instructions, would change the order of access to the operand involved in the dependence. There are three types of data hazards can occur:

- ***Read After Write (RAW) hazards*:**
  RAW data hazard is the most common type. It appears when the next instruction tries to read from a source before the previous instruction writes to it. So, the next instruction gets the incorrect old value such as an operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.
- ***Write After Read (WAR) hazards:***
  WAR hazard appears when the next instruction writes to a destination before the previous instruction reads it. In this case, the previous instruction gets a new value incorrectly such as read an operand and writes soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.
- ***Write After Write (WAW) hazards:***
  WAW data hazard is situation when the next instruction tries to write to a destination before a previous instruction writes to it and it results in changes done in the wrong order. Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value. So the results of WAW hazards are: (i) Pipeline Latency and (ii) Instruction effects not completed before next operation begins.

Structural Hazards

A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time. A structural hazard might occur, for instance, if a program were to execute a branch instruction followed by a computation instruction.

Control Hazards

Controls hazards occur when the processor is told to branch – i.e., if a certain condition is true, and then jump from one part of the instruction stream to another – not necessarily to the next instruction sequentially. In such a case, the processor cannot tell in advance whether it should process the next instruction (when it may instead have to move to a distant instruction).

Multiple Choice Type Questions

1. The number of cycles required to complete *n* tasks in a *k* stage pipeline is [WBUT 2007, 2016, 2017]

a) $k + n - 1$ b) $nk + 1$ c) k d) none of these

Answer: (a)

2. A 4-ary 3-cube hypercube architecture has [WBUT 2007]

a) 3 dimensions with 4 nodes along each dimension
b) 4 dimensions what 3 nodes along each dimension
c) both (a) and (b) d) none of these

Answer: (a)

3. Which of these are examples of 2-dimensional topologies in static networks? **[WBUT 2007, 2010]**

a) Mesh **b) 3CCC networks** **c) Linear array** **d) None of these**

Answer: (a)

4. The seek time of a disk is 30 ms. It rotates at the rate of 30 rotations/second. The capacity of each track is 300 words. The access time is approximately **[WBUT 2007, 2008, 2010]**

a) 62 ms **b) 60 ms** **c) 47 ms** **d) none of these**

Answer: (d)

5. For two instructions *I* and *J* WAR hazard occur, if **[WBUT 2007, 2010, 2014]**

a) $R(I)\cap D(J)\neq\phi$ **b)** $R(I)\cap R(J)\neq\phi$

c) $D(I)\cap R(J)\neq\phi$ **d) none of these**

Answer: (c)

6. The performance of a pipelined processor suffers if **[WBUT 2008, 2009, 2011, 2013]**

a) the pipeline stages have different delays
b) Consecutive instructions are dependent on each other
c) the pipeline stages share hardware resources
d) all of these

Answer: (d)

7. A single bus structure is primarily found in **[WBUT 2008]**

a) Main frames **b) High performance machines**
c) Mini and Micro-computers **d) Supercomputers**

Answer: (c)

8. What will be the speed up for a four-stage linear pipeline, when the number of instruction n = 64? **[WBUT 2008, 2009]**

a) 4.5 **b) 7.1** **c) 6.5** **d) None of these**

Answer: (d)

9. Dynamic pipeline allows **[WBUT 2008, 2009, 2011, 2014, 2016, 2017]**

a) Multiples functions to evaluate **b) only streamline connection**
c) to perform fixed function **d) none of these**

Answer: (a)

10. The division of stages of a pipeline into sub-stages is the basis for **[WBUT 2009, 2014]**

a) pipelining **b) super-pipelining**
c) superscalar **d) VLIW processor**

Answer: (a)

11. A pipeline stage **[WBUT 2012, 2014, 2018, 2019]**

a) is sequential circuit b) is combinational circuit
c) consists of both sequential and combinational circuits
d) none of these

Answer: (c)

12. Utilization pattern of successive stages of a synchronous pipeline can be specified by **[WBUT 2012, 2015, 2017, 2018, 2019]**

a) Truth table b) Excitation table
c) Reservation table d) Periodic table

Answer: (c)

13. SPARC stands for **[WBUT 2012, 2018]**

a) Scalable Processor Architecture
b) Superscalar Processor A RISC Computer
c) Scalable Processor A RISC Computer
d) Scalable Pipeline Architecture

Answer: (a)

14. Portability is definitely an issue for which of the following architectures? **[WBUT 2012, 2018]**

a) VLIW processor b) Super Scalar processor
c) Super pipelined d) none of these

Answer: (b)

15. Which of the following is not the cause of possible data hazard? **[WBUT 2012, 2018, 2019]**

a) RAR b) RAW c) WAR d) WAW

Answer: (a)

16. What will be the speed up for a 4 segment linear pipeline when the number of instruction $n = 64$? **[WBUT 2013, 2014, 2019]**

a) 4.5 b) 3.82 c) 8.16 d) 2.95

Answer: (b)

17. Which type of data hazard is not possible? **[WBUT 2013]**

a) WAR b) RAW c) RAR d) WAW

Answer: (c)

18. MIPS means **[WBUT 2013]**

a) Multiple Instruction Per Second b) Millions of Instruction Per Second
c) Multi-Instruction Performed System d) none of these

Answer: (b)

19. The prefetching technique is a solution for **[WBUT 2014, 2016]**

a) data hazard **b) structural hazard**
c) control hazard **d) enhancing the speed of pipeline**

Answer: (c)

20. Suppose the time delay of the four stages of a pipeline are t1 = 60 ns, t2 = 50 ns, t3 = 90 ns, t4 = 80 ns respectively and the interface latch has a delay t1 = 10 ns, then the maximum clock frequency for the pipeline is **[WBUT 2016]**

a) 100 ns **b) 90 ns** **c) 190 ns** **d) 30 ns**

Answer: (b)

21. Pipelining uses **[WBUT 2017]**

a) data parallelism **b) temporal parallelism**
c) spatial parallelism **d) none of these**

Answer: (a)

22. An n-dimensional hypercube has **[WBUT 2017]**

a) n^n nodes **b) n^{-n} nodes**
c) 2^n nodes **d) none of these**

Answer: (c)

23. The throughput of a super scalar processor is **[WBUT 2019]**

a) less than 1 **b) 1**
c) more than 1 **d) not known**

Answer: (c)

Short Answer Type Questions

1. Define Speed-up. Deduce that the maximum speed-up in a k-stage pipeline processor is k. Is this maximum speed-up always achievable? Explain.
[WBUT 2006]

OR,

If there are no stalls (waits) then prove that the speedup is equal to the pipeline depth i.e., the number of pipeline stages. **[WBUT 2016]**

OR,

Show that the maximum speedup of a pipeline is equal to its stages. **[WBUT 2016]**

Answer:

Suppose we consider that
k = no. of stages in the pipeline
n= no. of processes to be execute
τ = time delay for each stage of the pipeline
and S = the Speed-up

then, $S = \dfrac{\text{Time require for non-pipeline process}}{\text{Time require for pipeline process}} = \dfrac{n.k.\tau}{(k+(n-1))\tau} = \dfrac{n.k}{k+(n-1)}$

Maximum Speed-up is, $S_k \rightarrow k$ when $n >> k$

The maximum Speed-up is never fully achievable because of data-dependencies between instructions, interrupts, program branches etc. So, many pipeline cycles may be wasted on a waiting state caused by out-of sequence instruction executions.

2. What are the different factors that can affect the performance of a pipelined system? Differentiate between WAR and RAW with a suitable example.
[WBUT 2007]

Answer:
Pipelining achieves a reduction of the average execution time per instruction. In the sense that pipeline can perform more instructions per clock cycle. This can be viewed in two ways:

- Decreasing the CPI. Typical way in which people view the performance increase
- Decreasing the cycle time (i.e., increasing the clock rate).

Pipelining increases the CPU instruction throughput. Pipelining does not decrease the execution time of an individual instruction. It increases the execution time due to overhead (clock skew and pipeline register delay) in the control of the pipeline.

Data hazards occur when data is modified. Ignoring potential data hazards can result in race conditions.
There are two situations a data hazard can occur in:

Read after Write (RAW):
An operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.

Write after Read (WAR):
Read an operand and write soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.
A RAW Data Hazard refers to a situation where we refer to a result that has not yet been calculated, for example:
i1. R2 <- R1 + R3
i2. R4 <- R2 + R3

The 1st instruction is calculating a value to be saved in register 2, and the second is going to use this value to compute a result for register 4. However, in a pipeline, when we fetch the operands for the 2nd operation, the results from the 1st will not yet have been saved, and hence we have a data dependency. We say that there is a data dependency with instruction 2, as it is dependent on the completion of instruction 1

A WAR Data Hazard represents a problem with concurrent execution, for example:
i_1. $r_1 <- r_2 + r_3$
i_2. $r_3 <- r_4 \times r_5$
If we are in a situation that there is a chance that i_2 may be completed before i_1 (i.e. with concurrent execution) we must ensure that we do not store the result of register 3 before i_1 has had a chance to fetch the operands.

3. What are the different parameters used in measuring CPU performance? Briefly discuss each. [WBUT 2008, 2015]

Answer:

To estimate the CPU performance, the measure that is generally most important is **execution time, T**. because we can write

Performance = 1 / Execution time

So, if the execution time increases the CPU performance decreases. There are three parameters to measure the performance of the CPU, i.e. speedup, efficiency and throughput.

When considering the impact of some performance improvement, the effect of the improvement is usually expressed in terms of the **speedup,** *S*, taken as the ratio of the execution time without the improvement (T_{wo}) to the execution time with the improvement (T_w):

$$S = T_{wo} / T_w$$

Speed-up as a direct percent can be represented as:

$$S = ((T_{wo} - T_w) / T_w) \times 100$$

Efficiency, E is the ratio of Speed-up to the number of processor used.

So, $\boldsymbol{E = S/p}$

Where S is the speed-up and p is the number of processor.

Throughput is the measure of number of computation over a unit time.

4. What is meant by pipeline stall? [WBUT 2008]

Answer:

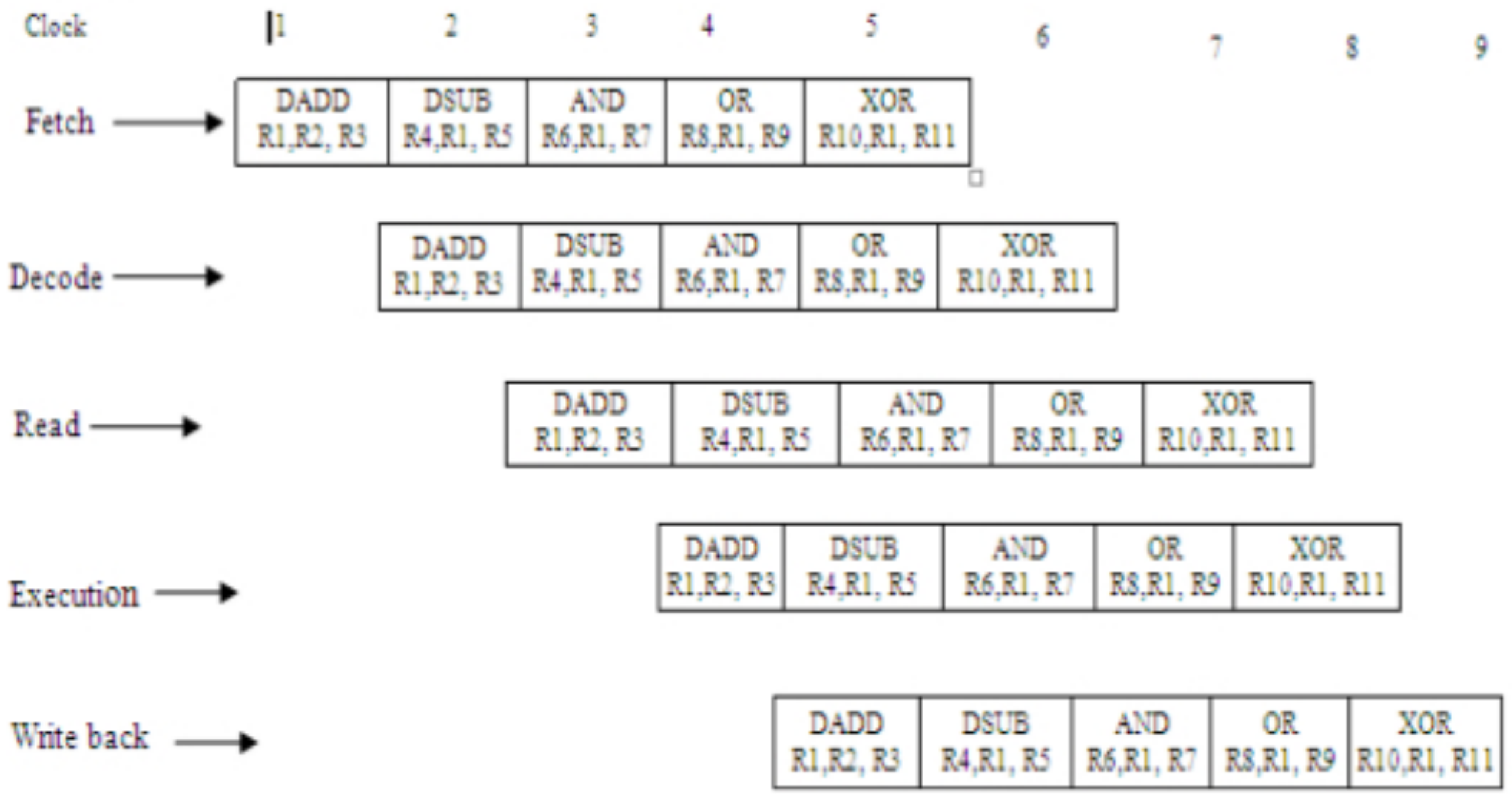
A pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle. Consider, for example, the case of an instruction fetch that incurs a cache miss. Assume also that a cache miss requires three extra time units. By this method we can prevent branch and structural hazards from occurring. As instructions are fetched, control logic determines whether or not a hazard could/will occur. If this is true, then the control logic inserts NOPs (No Operations) into the pipeline. Thus, before the next instruction (which will cause the hazard) is executed, the previous one will be sufficiently complete to prevent the hazard. If the number of NOPs is equal to the number of stages in the pipeline, the processor has been cleared of all instructions and can proceed free from hazards.

5. Consider the pipelined execution of these instructions: **[WBUT 2009]**

DADD	**R1, R2, R3**
DSUB	**R4, R1, R5**
AND	**R6, R1, R7**
OR	**R8, R1, R9**
XOR	**R10, R1, R11**

Explain how the above execution may generate a data hazard and describe a way to minimize the data hazard stalls using forwarding. Modify the above example to show a case where forwarding may not work.

Answer:



We have considered that the above pipeline technique is a five stage pipeline. The stages are fetch, decode, read, execution and write back. In the above figure, we also show that in which clock pulse what operation is performed.

Now, the data hazard will occur in 4th clock cycle, where both read and execution operations are performed on register R1 at the same time.

Result forwarding is a technique to minimize the stall in pipeline processing. The technique is that after execution of one instruction, if the result is transfer to the next instruction bypass the write back stage directly. i.e. without writing the result in the register pipeline transfer that result to the next instruction.

Result forwarding may not work in case of branch instruction of a pipeline execution. So, if there is a branch instruction in the above instruction set then result forwarding will not work.

6. Explain DMA working principle. **[WBUT 2009]**

Answer:

The main idea of direct memory access (DMA) is to transfer data between peripheral devices and main memory to bypass the role of the CPU. It allows peripheral devices to transfer data directly from and to memory without the intervention of the CPU. Having peripheral devices access memory directly would allow the CPU to do other work, which would lead to improved performance, especially in the cases of large transfers. The DMA controller controls one or more peripheral devices. It allows devices to transfer data to or from the system's memory without the help of the processor. Both the DMA and CPU use memory bus and only one or the other can use the memory at the same time. The DMA controller then sends a request to the CPU asking its permission to use the bus. The CPU returns an acknowledgment to the DMA controller granting it bus access. The DMA can now take control of the bus to independently conduct memory transfer. When the transfer is complete the DMA relinquishes its control of the bus to the CPU. Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the CPU asserts to indicate it has relinquished the bus. The figure below shows how the DMA controller shares the CPU's memory bus.

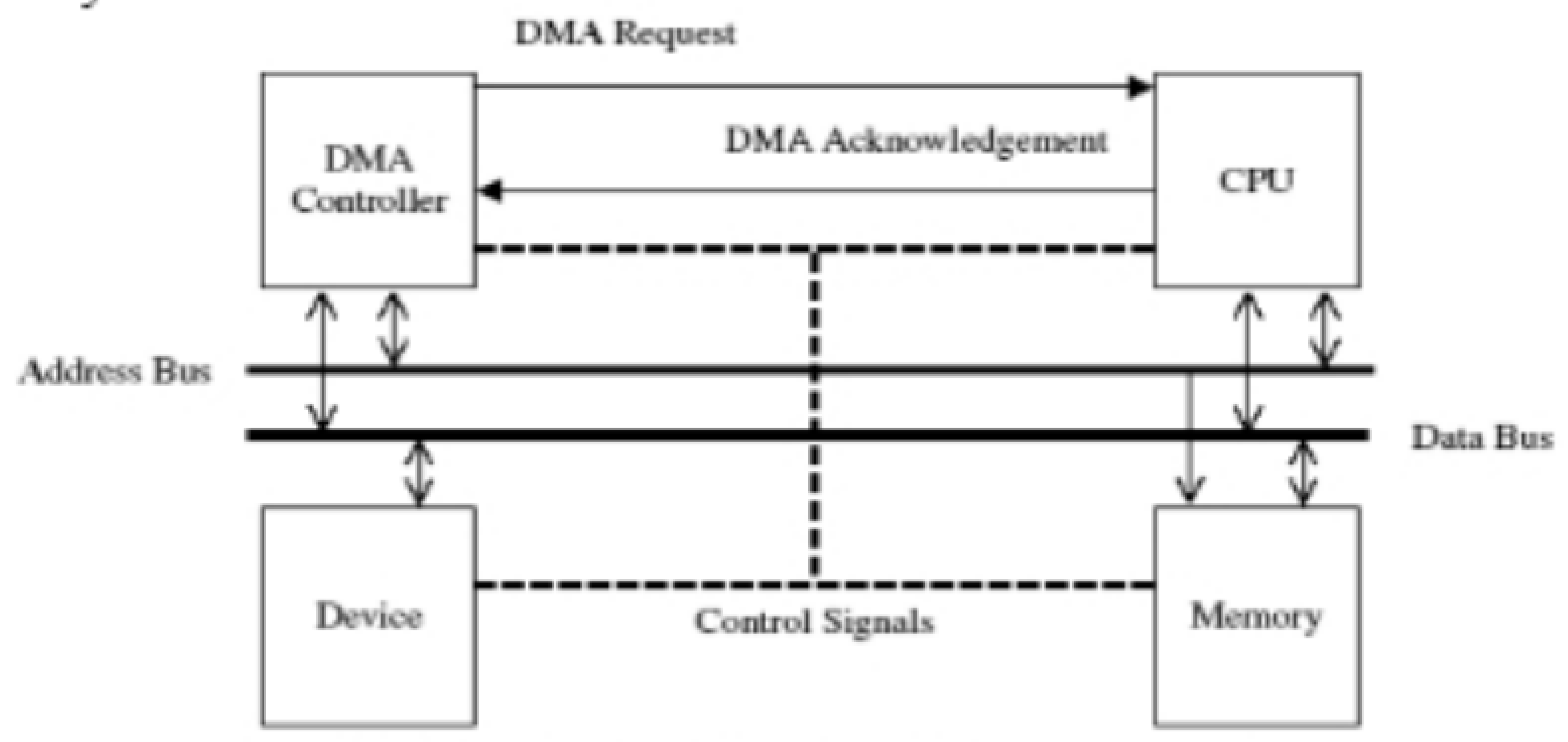


Fig: DMA controller shares the CPU's memory bus

A DMA controller has an address register, a word count register, and a control register. The address register contains an address that specifies the memory location of the data to be transferred. It is typically possible to have the DMA controller automatically increment the address register after each word transfer, so that the next transfer will be from the next memory location. The word count register holds the number of words to be transferred. The word count is decremented by one after each word transfer. The control register specifies the transfer mode. Direct memory access data transfer can be performed in burst mode or single cycle mode. In burst mode, the DMA controller keeps control of the bus until all the data has been transferred to (from) memory from (to) the peripheral device. This mode of transfer is needed for fast devices where data transfer cannot be stopped until the entire transfer is done. In single-cycle mode (cycle stealing), the DMA controller relinquishes the bus after each transfer of one data word. This minimizes the amount of time that the DMA controller keeps the CPU from controlling the bus, but it requires that the bus request/acknowledge sequence be performed for every single transfer. This overhead can result in a degradation of the performance. The single-cycle mode is preferred if the system cannot tolerate more than a few cycles of added interrupt latency or if the peripheral devices can buffer very large amounts of data, causing the DMA controller to tie up the bus for an excessive amount of time.

7. What are the different pipeline hazards and what are the remedies? [WBUT 2009]

OR,

Discuss data hazards briefly. [WBUT 2015]

OR,

What do you mean by hazards in pipeline? Describe the different types of hazards. [WBUT 2018]

OR,

How data hazards are detected and prevented? [WBUT 2019]

Answer:

In computer architecture, a hazard is a potential problem that can happen in a pipelined processor. There are typically three types of hazards: data hazards, branching hazards, and structural hazards.

Instructions in a pipelined processor are performed in several stages, so that at any given time several instructions are being executed, and instructions may not be completed in the desired order. A hazard occurs when two or more of these simultaneous (possibly out of order) instructions conflict.

i. Data hazards

Data hazards occur when data is modified. Ignoring potential data hazards can result in race conditions. There are three situations a data hazard can occur in:

- **Read after Write (RAW):** An operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.
- **Write after Read (WAR):** Read an operand and write soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.
- **Write after Write (WAW):** Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value.

The operands involved in data hazards can reside in memory or in a register.

ii. Structural hazards

A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time. A structural hazard might occur, for instance, if a program were to execute a branch instruction followed by a computation instruction. Because they are executed in parallel, and because branching is typically slow (requiring a comparison, program counter-related computation, and writing to registers), it is quite possible (depending on architecture) that the computation instruction and the branch instruction will both require the ALU at the same time.

iii. Branch hazards

Branching hazards (also known as control hazards) occur when the processor is told to branch - i.e., if a certain condition is true, then jump from one part of the instruction stream to another - not necessarily to the next instruction sequentially. In such a case, the processor cannot tell in advance whether it should process the next instruction (when it may instead have to move to a distant instruction). This can result in the processor doing unwanted actions.

8. Use 8-bit 2's complement integer to perform $-43+(-13)$. **[WBUT 2009]**

Answer:

$-43 = 11010101$

$-13 = 11110011$

$-43 + (-13) = 111001000$

So, discard carry and the result is 11001000. i.e. the 2's complement of 56.

9. What do you mean by pipeline processing? **[WBUT 2009]**

Answer:

Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit. The units are connected in a serial fashion and all of them operate simultaneously. The use of pipelining improves the performance compared to the traditional sequential execution of tasks. The figure below shows an illustration of the basic difference between executing four subtasks of a given instruction (in this case fetching F, decoding D, execution E, and writing the results W) using pipelining and sequential processing.

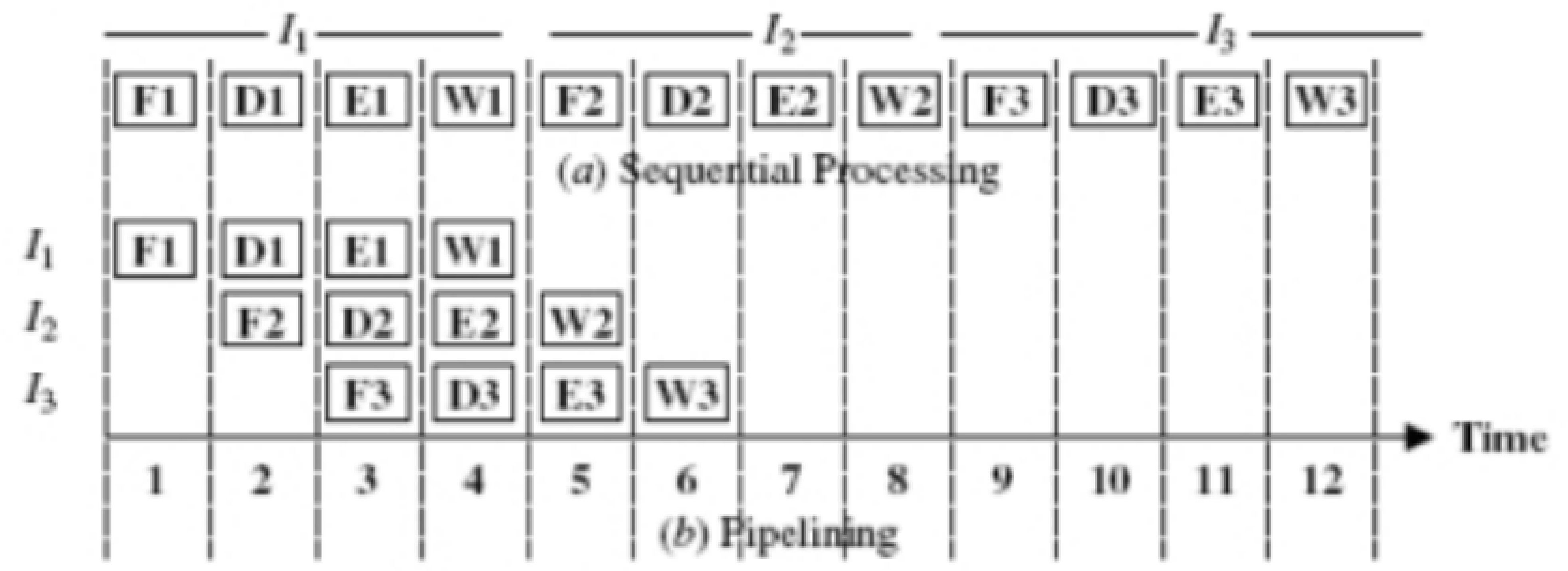


Fig: Pipeline vs sequential Processing

10. What are instruction pipeline and arithmetic pipeline? **[WBUT 2009]**

Answer:
An instruction pipeline is a technique used in the design of computers system to increase their instruction throughput. The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step. The principles used in instruction pipelining can be used in order to improve the performance of computers in performing arithmetic operations such as add, subtract, and multiply.
In a program there is several numbers of instructions. There are five steps to execute an instruction and the steps are:

i. Fetch
ii. Decode
iii. Operand fetch
iv. Execute
v. Write back

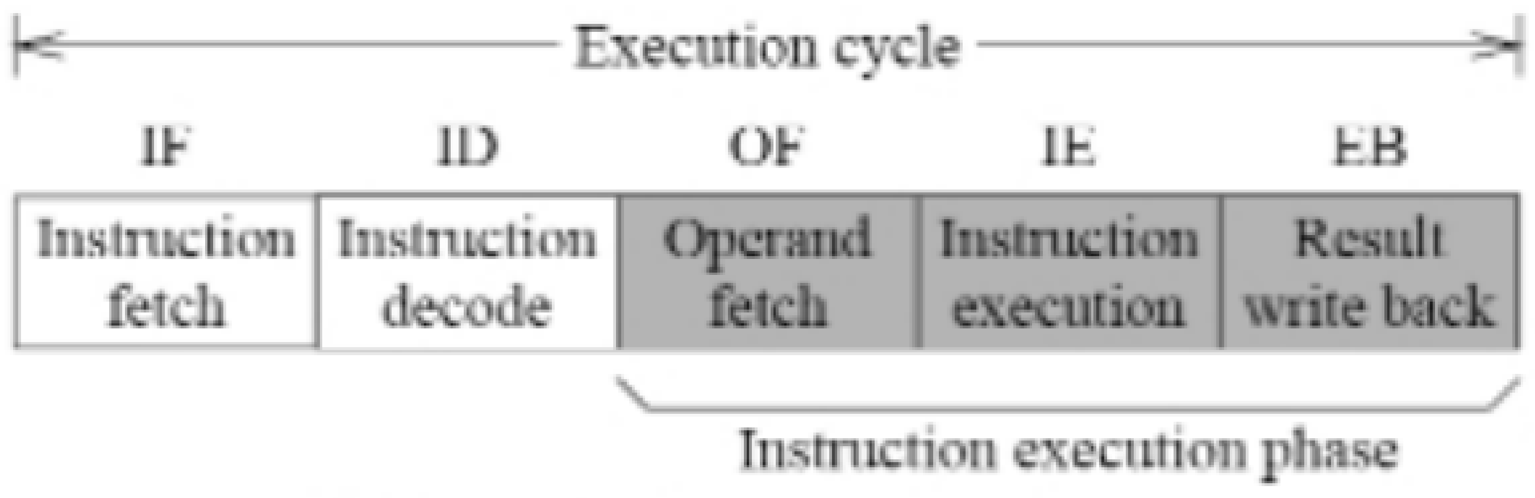
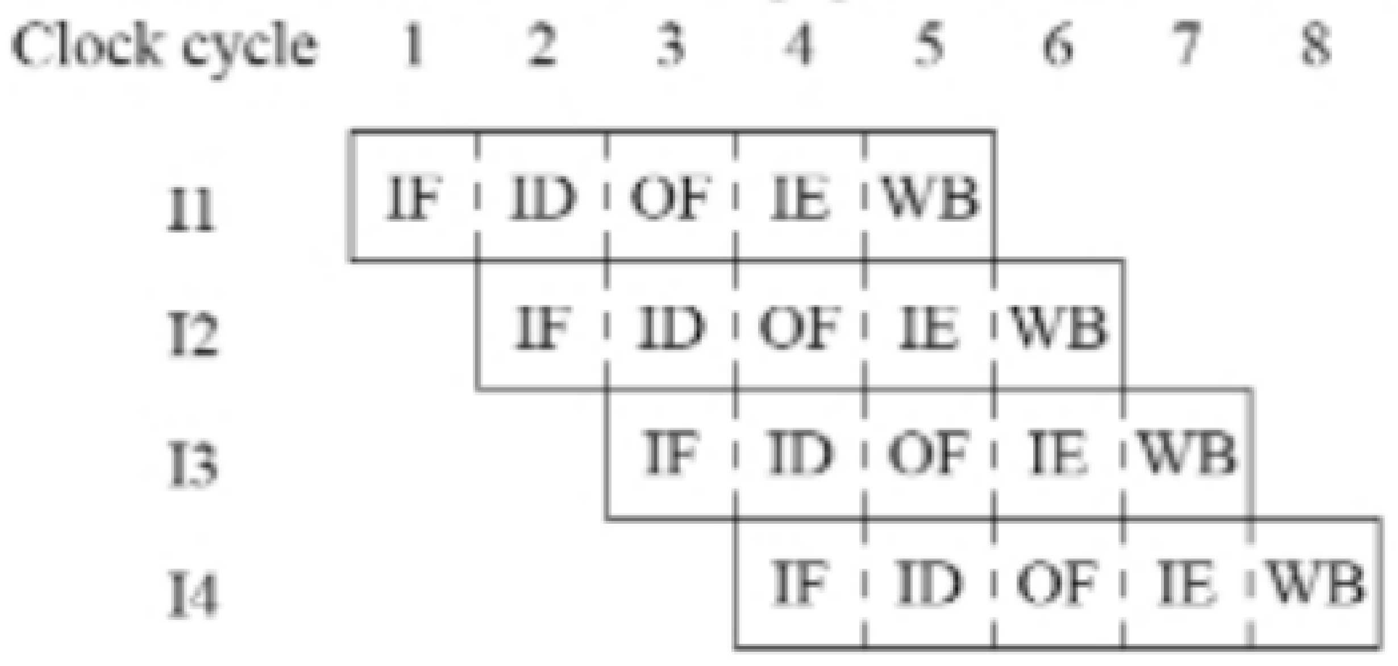


Fig: Instruction pipeline stages

The streams of instructions are executed in the pipeline in an overlapped manner.



Pipelining can be applied to arithmetic operations. As an example, we show a floating-point add pipeline in Figure below. The floating-point add unit has several stages:

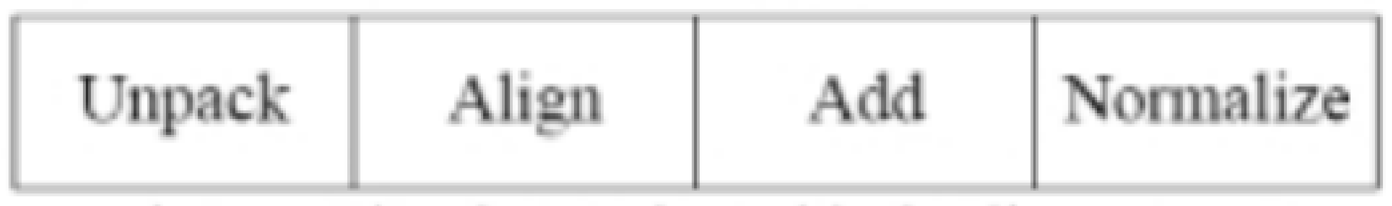


Figure: Floating point add pipeline stages

1. ***Unpack:*** The unpack stage partitions the floating-point numbers into the three field: the sign field, exponent field, and mantissa field. Any special cases such as not-a-number (NaN), zero, and infinities are detected during this stage.
2. ***Align:*** This stage aligns the binary points of the two mantissas by right-shifting the mantissa with the smaller exponent.
3. ***Add:*** This stage adds the two aligned mantissas.

4. ***Normalize:*** This stage packs the three fields of the result after normalization and rounding into the IEEE-754 floating-point format. Any output exceptions are detected during this stage.

11. Find 2's complement of $(1AB)_{16}$ represented in 16 bit format. [WBUT 2009]

Answer:

$(1AB)_{16}$ = $(0000\ 0001\ 1010\ 1011)_2$

2's complement of $(0000\ 0001\ 1010\ 1011)_2$ is

$(1111\ 1110\ 0101\ 0101)_2$

12. What are the different factors that can affect the performance in a pipelined system? Differentiate between WAR and RAW hazards. [WBUT 2010]

Answer:

Pipelining achieves a reduction of the average execution time per instruction. In the sense that pipeline can perform more instructions per clock cycle. This can be viewed in two ways:

- Decreasing the CPI. Typical way in which people view the performance increase
- Decreasing the cycle time (i.e., increasing the clock rate).

Pipelining increases the CPU instruction throughput. Pipelining does not decrease the execution time of an individual instruction. It increases the execution time due to overhead (clock skew and pipeline register delay) in the control of the pipeline.

WAR hazards	RAW hazards
1. **j** tries to write a destination before it is read by **i** , so **i** incorrectly gets the new value.	1. **j** tries to read a source before **i** writes it, so **j** incorrectly gets the old value.
2. WAR hazard is eliminated by register renaming of all the destination registers including those with a pending read or write for an earlier instruction.	2. RAW hazards are avoided by executing an instruction only when its operands are available.
3. WAR hazard if $D(i) \cap R(j) \neq \emptyset$	3. RAW hazard if $R(i) \cap D(j) \neq \emptyset$
4. For example: i1. $R4 \leftarrow R1 + R3$ i2. $R3 \leftarrow R1 + R2$ If we are in a situation that there is a chance that i2 may be completed before i1 (i.e. with concurrent execution) we must ensure that we do not store the result of register 3 before i1 has had a chance to fetch the operands.	4. For example: i1. $R2 \leftarrow R1 + R3$ i2. $R4 \leftarrow R2 + R3$ The first instruction is calculating a value to be saved in register 2, and the second is going to use this value to compute a result for register 4. However, in a pipeline, when we fetch the operands for the 2nd operation, the results from the first will not yet have been saved, and hence we have a data dependency.

13. "Instruction execution throughput increases in proportion with the number of pipeline stages." Is it true? Justify your statement. [WBUT 2012, 2015, 2017]

Answer:

Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit. The units are connected in a serial fashion and all of them operate simultaneously. The use of pipelining improves the performance compared to the traditional sequential execution of tasks. Consider the execution of m tasks (instructions) using n-stages (units) pipeline. We assume that the unit time $T = t$ units. Then the Throughput U(n) is, $m / (n+m-1) x t$

i.e. the no. of tasks executed per unit time. So, from the above equation, if we increase the number of stages in a pipeline it also increase the throughput of the pipeline.

14. For the code segment given below, explain how delayed branching can help:

I1	LOAD	R1, A
I2	Dec	R3, 1
I3	BrZero	R3, 15
I4	Add	R2, R4
I5	Sub	R5, R6
I6	Store	R5, B

[WBUT 2013]

Answer:

Instruction I2 performs "Dec R3,1" and I3 performs "BrZero R3,15". So, both I2 and I3 modify the register R3 at the same time. So, delayed branch actually first modify the value of R3 by "Dec R3,1" then update the value of R3 by "BrZero R3,15".

15. For the following code show how loop unrolling can help improve instruction level parallelism (ILP) performance:

Loop 1 : I1 :	Load R0, A (R1)	
	;	A is the starting address of array location
	;	R1 holds the initial; address of the element
I2 : Add R0, R2	;	R0 ← R0 + R2, R2 is a scalar
I3 : Store R0, A (R1)		
I4 : Add R1, –8	;	go to next word in Array of doubles
	;	whose address is 8 bytes earlier
I5 : BNE	R1, Loop1	

[WBUT 2013]

Answer:

Pipelining can overlap the execution of instructions when they are independent of one another. This potential overlap among instructions is called instruction-level parallelism (ILP) since the instructions can be evaluated in parallel. The simplest and most common way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of a loop. A loop is parallel unless there is a cycle in the dependencies, since the absence of a cycle means that the dependencies give a partial ordering on the statements. Statement I1 uses the value assigned in the previous iteration by statement I2, so there is a loop-carried dependency between I1 and I2. Despite this dependency, this loop can be made parallel because the dependency is not circular.

16. What is pipeline chaining? **[WBUT 2013]**

OR,

What do you mean by pipelined chaining? **[WBUT 2005, 2010]**

Answer:

Pipeline chaining is a linking process that occurs when results obtained from one pipeline unit are directly fed into the operand registers of another functional pipe. In other words, intermediate results do not have to be restored into memory and can be used even before the vector operation is completed. Chaining permits successive operations to be issued as soon as the first result becomes available as an operand. The desired functional pipes and operand registers must be properly reserved; otherwise, chaining operations have to be suspended until the demanded resources become available.

17. Compare between Control-Flow, Data-Flow and Demand-Driven mechanism. **[WBUT 2013]**

Answer:

The Control-Flow Architecture is a Von Neumann or control flow computing model. Here a program is a series of addressable instructions, each of which either specifies an operation along with memory locations of the operands or it specifies conditional transfer of control to some other instruction. The next instruction to be executed depends on what happened during the execution of the current instruction. The next instruction to be executed is pointed to and triggered by the PC. The instruction is executed even if some of its operands are not available yet.

But in Dataflow model, the execution is driven only by the availability of operand. There is no Program Counter and global updateable store. The two features of von Neumann model that become bottlenecks in exploiting parallelism are missing in Data flow Architecture.

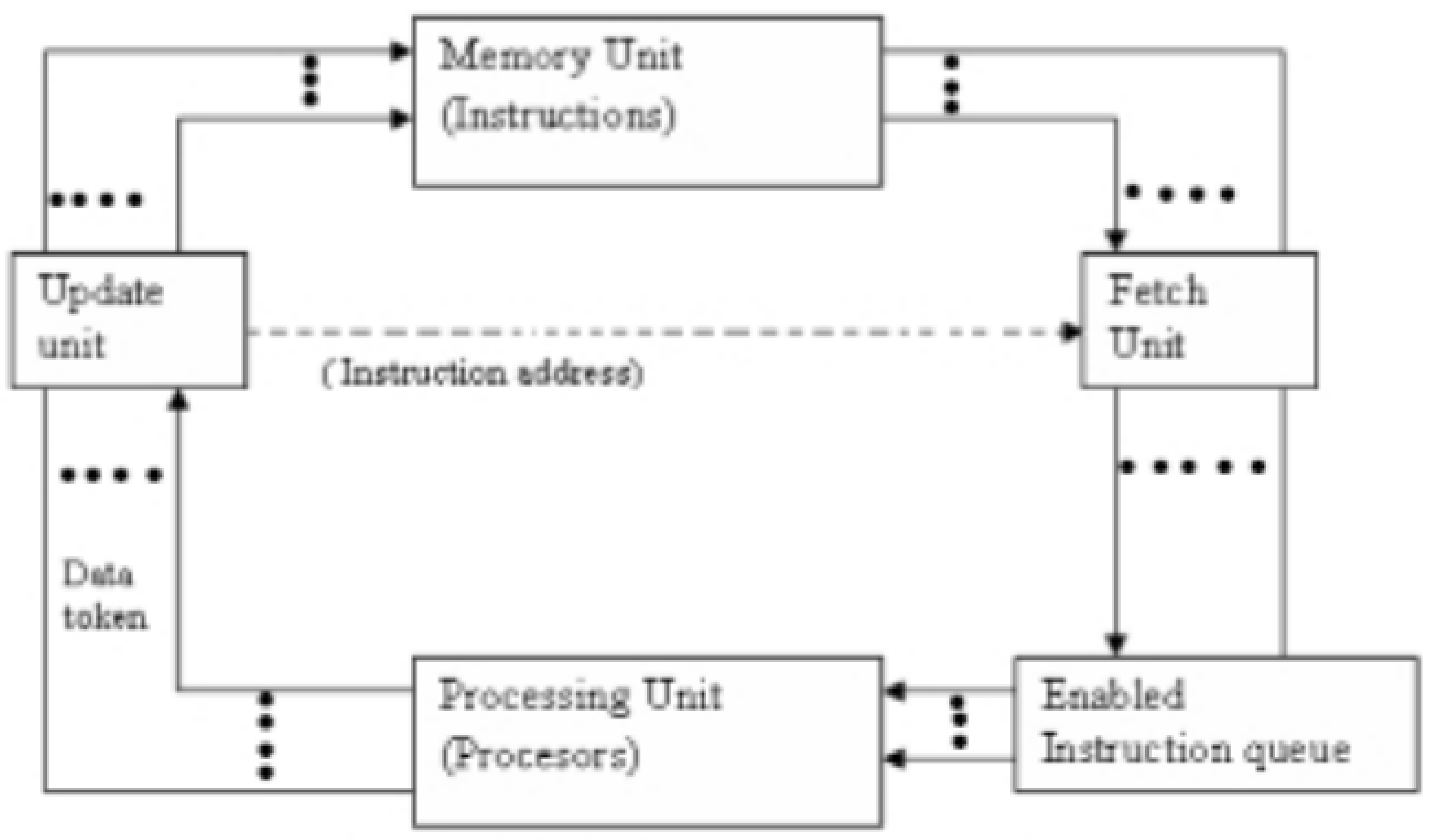


Fig: A static Data flow Computer

A demand driven access mechanism comprises logic apparatus at each program capable of seizing use of a shared communication channel for enabling access to a selected one program. The logic apparatus receives status signals from all programs so that if one program seeks access to the channel it will be enabled immediately upon an inactive status to the channel. If two or more programs simultaneously seek access to the channel, the logic apparatus establishes a priority order between them, thereby enabling access to only one program at a time. The priority ordering is based, in part, by the identity of the program last enabled, to thereby assure priority order allocation among the programs.

18. Draw pipeline execution diagram during the execution of the following instructions:
MUL R1, R2, R3
ADD R2, R3, R4
INC R4
SUB R6, R3, R7
Find out the delay in pipeline execution due to data dependency of the above instructions. [WBUT 2016]

Answer:

We have considered that the above pipeline technique is a five stage pipeline. The stages are fetch, decode, read, execution and write back. In the figure, we also show that in which clock pulse what operation is performed.

The operations of the instructions are,

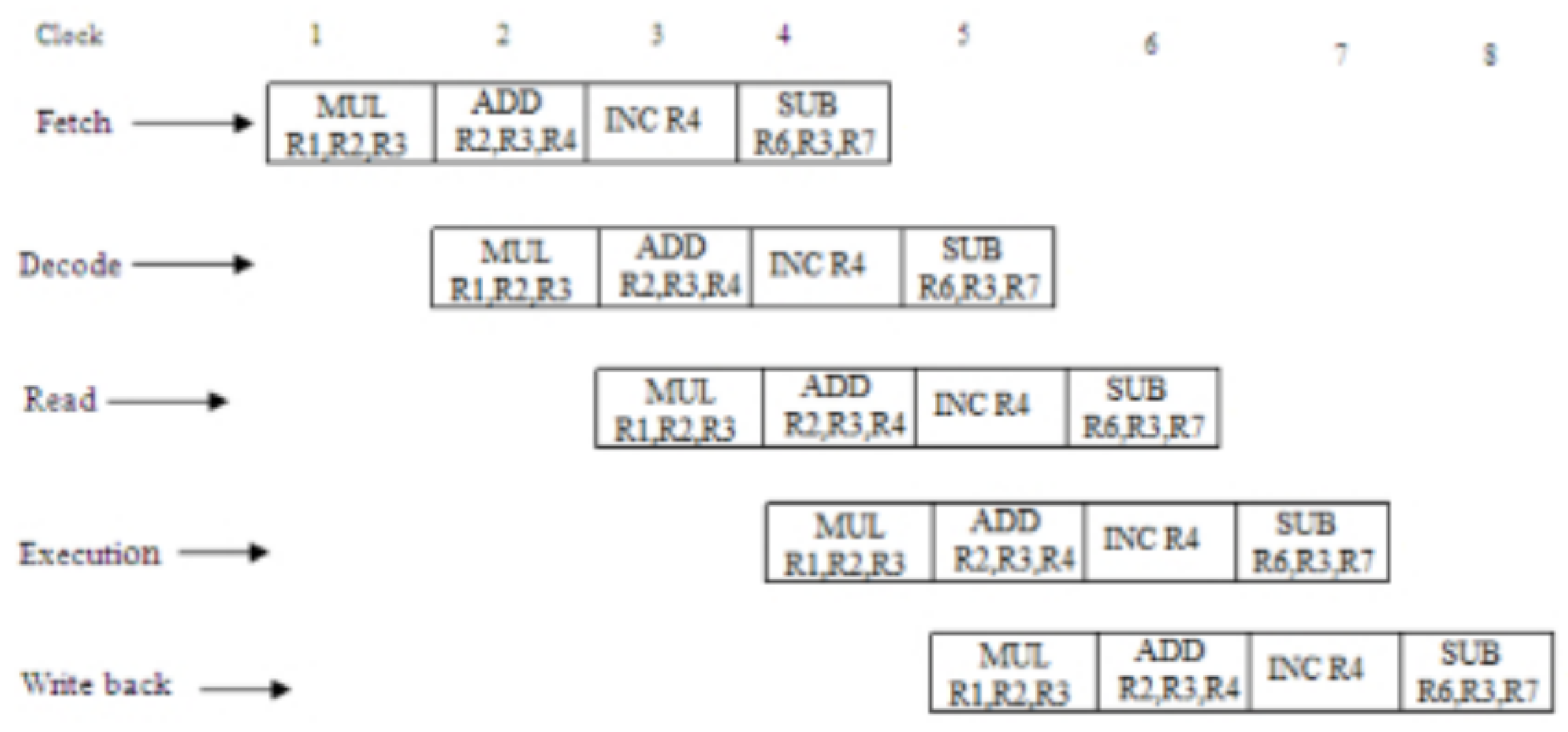
$R1 \leftarrow R2.R3$

$R2 \leftarrow R3+R4$

$R4 \leftarrow R4+1$

$R6 \leftarrow R3+R7$

So, from the above instructions, we can say that no data hazard will occur in the pipe line. The instructions are not dependent to each other. So, normal pipeline execution will occur.



19. How "Reservation Table" helps to study the performance of pipeline.
[WBUT 2016]

Answer:

There are two types of pipelines: static and dynamic. A static pipeline can perform only one function at a time, whereas a dynamic pipeline can perform more than one function at a time. A pipeline reservation table shows when stages of a pipeline are in use for a particular function. Each stage of the pipeline is represented by a row in the reservation table. Each row of the reservation table is in turn broken into columns, one per clock cycle. The number of columns indicates the total number of time units required for the pipeline to perform a particular function. To indicate that some stage S is in use at some time ty, an X is placed at the intersection of the row and column in the table corresponding to that stage and time. Figure 1 represents a reservation table for a static pipeline with three stages. When scheduling a static pipeline, only collisions between different input data for a particular function had to be avoided. With a dynamic pipeline, it is possible for different input data requiring different functions to be present in the pipeline at the same time. Therefore, collisions between these data must be considered as well. As with the static pipeline, however, dynamic pipeline scheduling begins with the compilation of a set of forbidden lists from function reservation tables. Next the collision vectors are obtained, and finally the sate diagram is drawn.

20. Consider the execution of a program of 15000 instructions by linear pipeline processor. The clock rate of pipeline is 25 MHz. Pipeline has five stages and one instruction is issued per clock cycle. Neglect pipelines due to branch instructions and out of sequence execution:

(i) Calculate the speedup program execution by pipeline as compared with that by non-pipelined processor.

(ii) What are the efficiency and throughput of the pipeline processor. [WBUT 2016]

Answer:

Information we get are:

n = 15,000 instructions or tasks

f = 25 MHz
k = 5 stages
Issued processor=1

(i) The Speedup (S_k) $= \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} = \frac{nk}{k+(n-1)} = \frac{(15,000)(5)}{5+(15,000-1)} = \frac{75,000}{15,004} = 4,999$

(ii) Efficiency (E_k) $= \frac{S_k}{k} = \frac{4,999}{5} = 0,999$

Throughput (T) $= \frac{nf}{k+(n-1)} = \frac{(15,000)(25)}{5+(15,000-1)} = \frac{375,000}{15,004} = 24,99\ MIPS$

21. Write down Amdahl's law of parallel processing. [WBUT 2017]

Answer:

Refer to Question No. 19 (d) of Long Answer Type Questions.

22. Differentiate between 3-address, 2-address, 1-address and 0-address instructions with suitable example. [WBUT 2019]

Answer:

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement X = (A + B) ∗ (C + D). We will use the symbols ADD, SUB, MUL, and DIV for the four arithmetic operations; MOV for the transfer-type operation; and LOAD and STORE for transfers to and from memory and AC register. We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

Three-Address Instructions: Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) ∗ (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

```
ADD R1, A, B  R1 ← M [A] + M [B]
ADD R2, C, D  R2 ← M [C] + M [D]
MUL X, R1, R2        M [X] ← R1 ∗ R2
```

It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.

Two-Address Instructions: Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) ∗ (C + D) is as follows:

```
MOV R1, A        R1 ← M [A]
ADD R1, B        R1 ← R1 + M [B]
MOV R2, C        R2 ← M [C]
ADD R2, D        R2 ← R2 + M [D]
```

MUL R1, R2	R1 ← R1∗R2
MOV X, R1	M [X] ← R1

The MOV instruction moves or transfers the operands to and from memory and processor registers.

One-Address Instructions: One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of tall operations. The program to evaluate X = (A + B) ∗ (C + D) is

LOAD A	AC ← M [A]
ADD B	AC ← A [C] + M [B]
STORE T	M [T] ← AC
LOAD C	AC ← M [C]
ADD D	AC ← AC + M [D]
MUL T	AC ← AC ∗ M [T]
STORE X	M [X] ← AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address Instructions: A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) ∗ (C + D) will be written for a stack organized computer. (TOS stands for top of stack)

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← (A + B)
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← (C + D)
MUL	TOS ← (C + D) ∗ (A + B)
POP X	M [X] ← TOS

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

23. What is instruction cycle? Compare and contrast hardwired vs. micro programmed control unit. [WBUT 2019]

Answer:

1st Part:

Instructions are processed under the direction of the control unit in a step-by-step manner. There are four fundamental steps in the instruction cycle:

1. **Fetch the instruction:** The next instruction is fetched from the memory address that is currently stored in the Program Counter (PC), and stored in the Instruction register (IR). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
2. **Decode the instruction:** The decoder interprets the instruction. During this cycle the instruction inside the IR (instruction register) gets decoded.
3. **Execute:** The Control Unit of CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU.
4. **Store result:** The result generated by the operation is stored in the main memory, or sent to an output device. Based on the condition of any feedback from the ALU, Program Counter may be updated to a different address from which the next instruction will be fetched.

2nd Part:

To execute an instruction, there are two types of control units Hardwired Control unit and Micro-programmed control unit.

1. Hardwired control units are generally faster than micro-programmed designs. In hardwired control, we saw how all the control signals required inside the CPU can be generated using a state counter and a PLA circuit. A micro-programmed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstructions and (2) generating control signals to execute each microinstruction.
2. Hardwired control unit generates the control signals needed for the processor using logic circuits. Micro-programmed control unit generates the control signals with the help of micro instructions stored in control memory.
3. Hardwired control unit is faster when compared to micro-programmed control unit as the required control signals are generated with the help of hardware.
4. It is difficult to modify as the control signals that need to be generated are hard wired. But in micro-program, it is easy to modify as the modification need to be done only at the instruction level.
5. More costlier as everything has to be realized in terms of logic gates and less costlier than hardwired control as only micro instructions are used for generating control signals.
6. The hardwired control cannot handle complex instructions as the circuit design for it becomes complex.
7. In hardwired control unit only limited numbers of instructions are used due to the hardware implementation. The control signals for many instructions can be generated in Micro-programmed control unit.

Long Answer Type Questions

1. What is a pipeline?
Consider the following reservation table:

	1	2	3	4
S1	X			X
S2		X		
S3			X	

Write down the forbidden latencies and initial collision vector. Draw the state diagram for scheduling the pipeline. Find out the sample and greedy cycle and MAL. If the pipeline clock rate is 25 MHz, then what is the throughput of the pipeline? What are the bounds on MAL? [WBUT 2007, 2011]

Answer:

1st Part:

Pipelining is a technique of splitting a sequential process into sub operations being execute of different segment that operates concurrently with all other segments. An instruction pipeline is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining assumes that with a single instruction (SIMD) concept successive instructions in a program sequence will overlap in execution.

A non-pipeline architecture is inefficient because some CPU components are idle while another module is active during the instruction cycle. Pipelining does not completely cancel out idle time in a CPU but making those modules work in parallel improves program execution significantly.

Processors with pipelining are organized inside into stages which can semi-independently work on separate jobs. Each stage is organized and linked into a 'chain' so each stage's output is inputted to another stage until the job is done. This organization of the processor allows overall processing time to be significantly reduced.

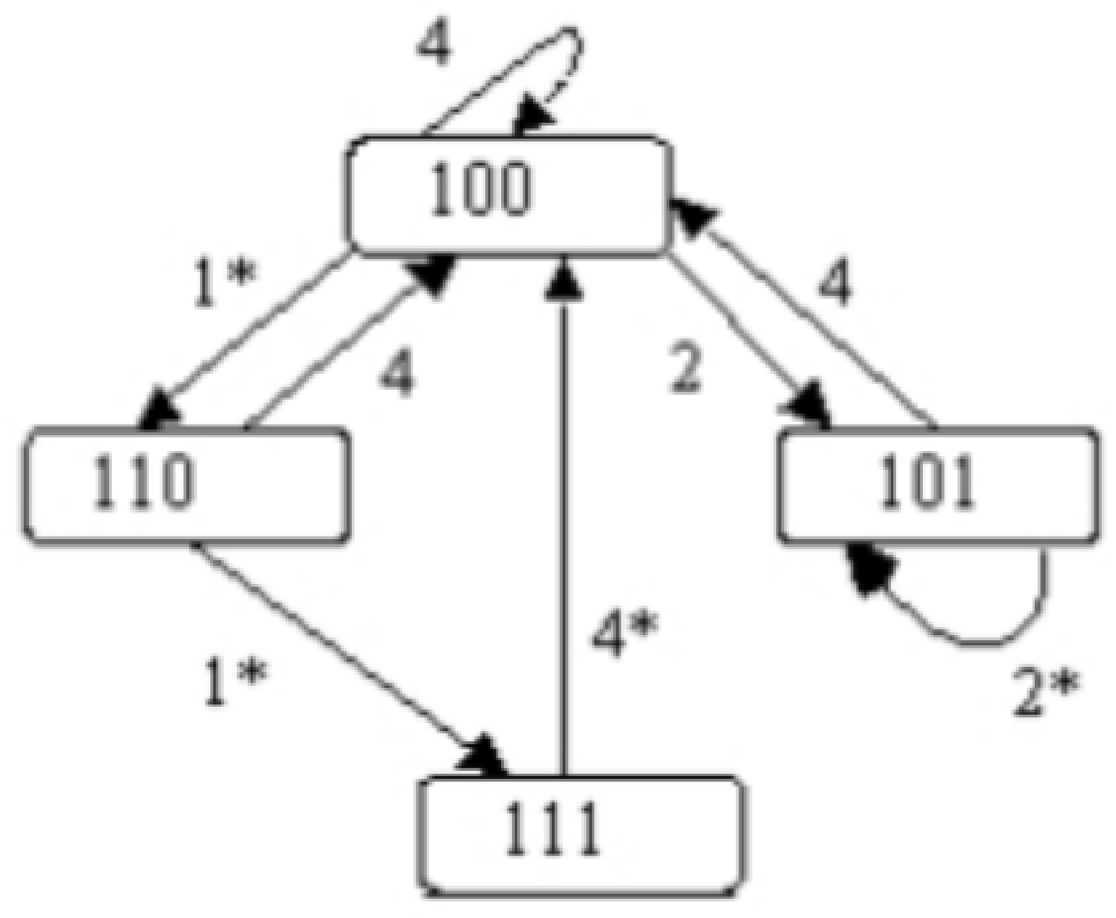
Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing. If 4 instructions that do not depend on the output of the first instruction are not available, the pipeline control logic must insert a stall or wasted clock cycle into the pipeline until the dependency is resolved. Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required. While pipelining can in theory increase performance over an unpipelined core by a factor of the number of stages (assuming the clock frequency also scales with the number of stages), in reality, most code does not allow for ideal execution.

2nd Part:

Forbidden latencies means the latencies that cause collision. Here the forbidden latency is 3.

The initial collision vector is 100.

The state diagram is given below



Simple cycle: (2), (4), (1,4), (1,1,4) and (2,4)

Greedy cycle: (2)

MAL = 2

Throughput = 25 MIPS

Upperbound = 2

Lowerbound = 2

2. a) What do you mean by "Data flow Computer"?

b) With simple diagram, explain Data flow architecture and compare it with control flow architecture.

c) Draw data flow graphs to represent the following computations:

i) X=A+B

ii) Y=X/B

iii) Z=A*X

iv) M=Z–Y

v) N=Z*X

vi) P=M/N **[WBUT 2008, 2014]**

Answer:

a) Data flow computer is a large, very powerful computer that has a number of processors all physically wired together with a large amount of memory and backing storage. Such computers are highly parallel in that they can carry out a large number of tasks at the same time. Data flow computers are used to execute processor intensive applications such as those associated with areas like molecular biology and simulation. Numerical calculations for the simulation of natural phenomena were conducted using a data-flow-type parallel processing computer. The computing time in the data-flow computer was approximately three-to-five times shorter than that of the usual medium-size computer of computing speed 3 MIPS (million instructions per second). Dynamic visualization of the computing process was realized using an image display directly connected to the memory of the data-flow computer.

b) ***Refer to Question No. 18 of Short Answer Type Questions.***

c)

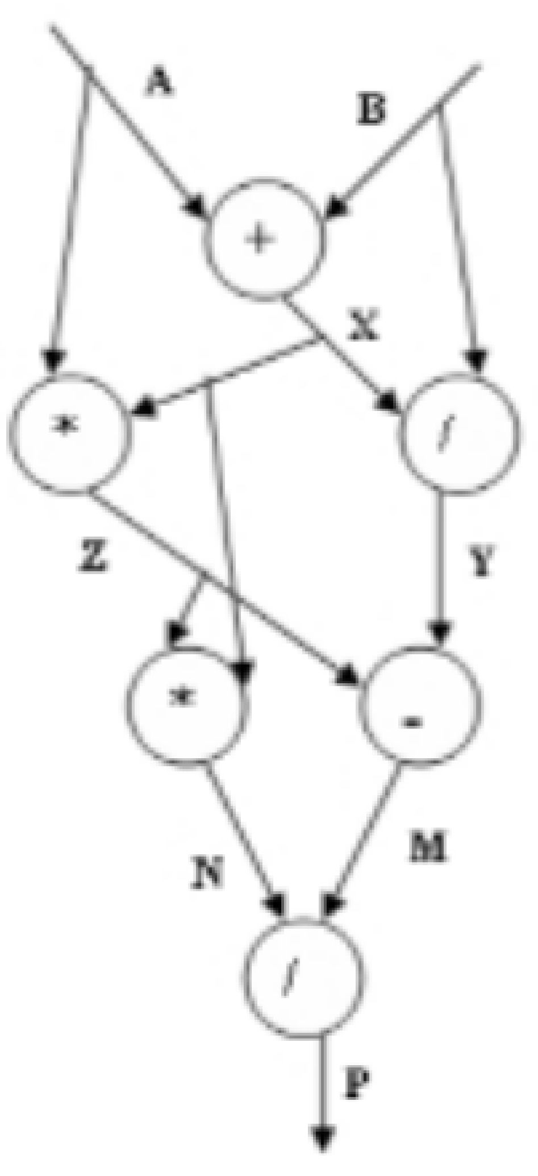


Fig: Data Flow graph for the above computation

3. What is floating point arithmetic operation? Explain all (addition, difference, multiplication, division) operations with example. [WBUT 2009]

Answer:

A floating-point (FP) number can be represented in the following form: $\pm m * b^e$
Where m is the mantissa and it represents the fraction part of the number and is normally represented as a signed binary fraction, e represents the exponent, and b represents the base (radix) of the exponent.

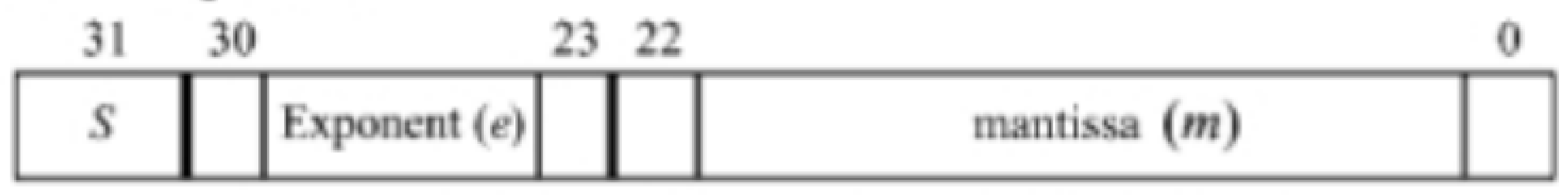


Fig: Representation of floating point number

Floating-Point Arithmetic Addition/Subtraction:

The difficulty in adding two FP numbers stems from the fact that they may have different exponents. Therefore, before adding two FP numbers, their exponents must be equalized, that is, the mantissa of the number that has smaller magnitude of exponent must be aligned.

Steps required add/subtract two Floating-Point numbers:

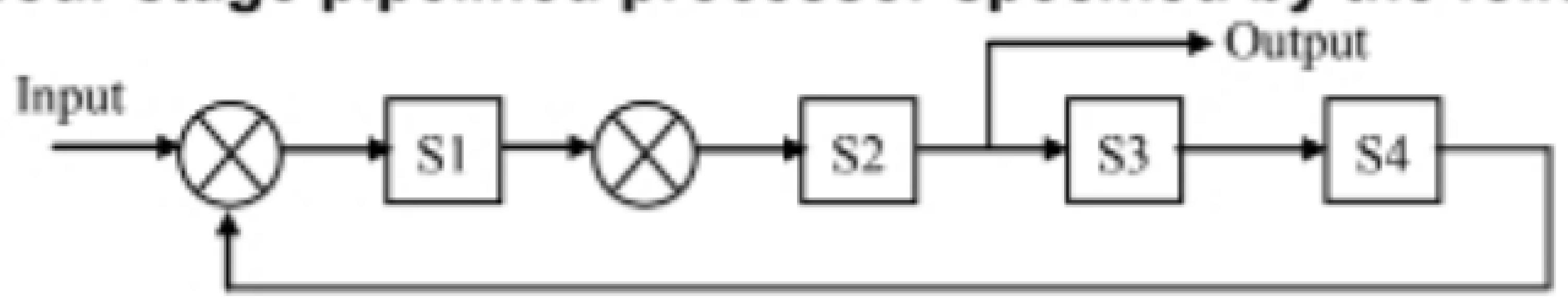
1. Compare the magnitude of the two exponents and make suitable alignment to the number with the smaller magnitude of exponent.

2. Perform the addition/subtraction.
3. Perform normalization by shifting the resulting mantissa and adjusting the resulting exponent.
Example: Consider adding the two FP numbers:
1. 1100 * 2^4 and 1.1000 * 2^2.
1. Alignment: 1.1000 * 2^2 has to be aligned to 0.0110 * 2^4
2. Addition: Add the two numbers to get 10.0010 * 2^4.
3. Normalization: The final normalized result is 1.0001 * 2^5 (assuming 4 bits are allowed after the radix point).

Floating-Point Arithmetic Multiplication
A general algorithm for multiplication of FP numbers consists of three basic steps. These are:
1. Compute the exponent of the product by adding the exponents together.
2. Multiply the two mantissas.
3. Normalize and round the final product.
Example Consider multiplying the two FP numbers
X = 1.000 * 2^{-2} and Y = -1.010 * 2^{-1}
1. Add exponents: -2 + (-1) = -3.
2. Multiply mantissas: 1.000 * - 1.010 = -1.010000.
The product is -1.0100 * 2^{-3}.

Floating-Point Arithmetic Division
A general algorithm for division of FP numbers consists of three basic steps:
1. Compute the exponent of the result by subtracting the exponents.
2. Divide the mantissa and determine the sign of the result.
3. Normalize and round the resulting value, if necessary.
Consider the division of the two FP numbers
X = 1.0000 * 2^{-2} and Y = -1.0100 * 2^{-1}.
1. Subtract exponents: -2 - (-1) = -1.
2. Divide the mantissas: (1.0000) / (-1.0100) = -0.1101.
3. The result is -0.1101* 2^{-1}.

4. Consider the four stage pipelined processor specified by the following diagram:



The pipeline has a total evaluation time of six clock cycles. All successor stages must be used after each clock cycle.
i) Specify the reservation table for above pipelined processor with six columns and four rows.

ii) What are the forbidden latencies and the initial collision vector? Draw the state transition diagram.
iii) Determine all simple cycles, greedy cycle and MAL.
iv) Determine the throughput of this pipelined processor. Given clock period as 20 ns. **[WBUT 2010]**

Answer:

i)

	1	2	3	4	5	6
S1	X				X	
S2		X				X
S3			X			
S4				X		

ii) The forbidden Latency is (0,4)
The pipeline collision vector (100010)

State 1 : 100010
Reaches State 2 (100110) after 1 cycle.
Reaches State 3 (101010) after 2 cycles.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 2 : 100110
Reaches State 5 (101110) after 1 cycle.
Reaches State 6 (111010) after 2 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 3 : 101010
Reaches State 7 (110110) after 1 cycle.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 4 : 110010
Reaches State 3 (101010) after 2 cycles.
Reaches State 4 (110010) after 3 cycles.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 5 : 101110
Reaches State 8 (111110) after 1 cycle.
Reaches State 1 (100010) after 5 cycles.
Reaches State 1 (100010) after 6 or more cycles.

State 6 : 111010
 Reaches State 4 (110010) after 3 cycles.
 Reaches State 1 (100010) after 5 cycles.
 Reaches State 1 (100010) after 6 or more cycles.

State 7 : 110110
 Reaches State 6 (111010) after 2 cycles.
 Reaches State 1 (100010) after 5 cycles.
 Reaches State 1 (100010) after 6 or more cycles.

State 8 : 111110
 Reaches State 1 (100010) after 5 cycles.
 Reaches State 1 (100010) after 6 or more cycles.

There are 8 states in the state diagram.

iii) Greedy cycle: (1115)
MAL = 2

iv) The throughput of this pipelined processor = (1/20)x (1/2) = 0.025 instructions per cycle.

5. a) What do you mean by MMX? Differentiate a data flow computer from a control flow computer.
b) List some potential problems with data flow computer implementation.
c) With simple diagram explain data flow architecture.
d) Draw data flow graphs to represent the following computations:
i) $P = X + Y$
ii) $Q = P \div Y$
iii) $R = X * P$
iv) $S = R - Q$
v) $T = R * P$
vi) $U = S \div T$ **[WBUT 2011, 2013, 2014]**

Answer:
a) MMX technology is an extension to the Intel Architecture (IA) designed to improve performance of multimedia and communication algorithms. The Pentium processor with MMX Technology is the first microprocessor to implement the new instruction set. All MMX chips have a larger internal L1 cache than their non-MMX counterparts. The Pentium processor with MMX implementation was the design of a new, dedicated, high-performance MMX pipeline, which was able to execute two MMX instructions with minimal logic changes in the existing units. Although adding a pipeline stage improves frequency, it decreases CPI performance, i.e., the longer the pipeline, the more work done

speculatively by the machine and therefore more work is being thrown away in the case of branch miss-prediction.

The control flow computers are either uniprocessor or parallel processors architecture. In uniprocessor system the instructions are executed sequentially and this is called control-driven mechanism. In parallel processors system control flow computers use shared memory. So, parallel executed instructions may cause side effects on other instructions and data due to shared memory. In control flow computer the sequence of execution of instructions is controlled by program counter register.

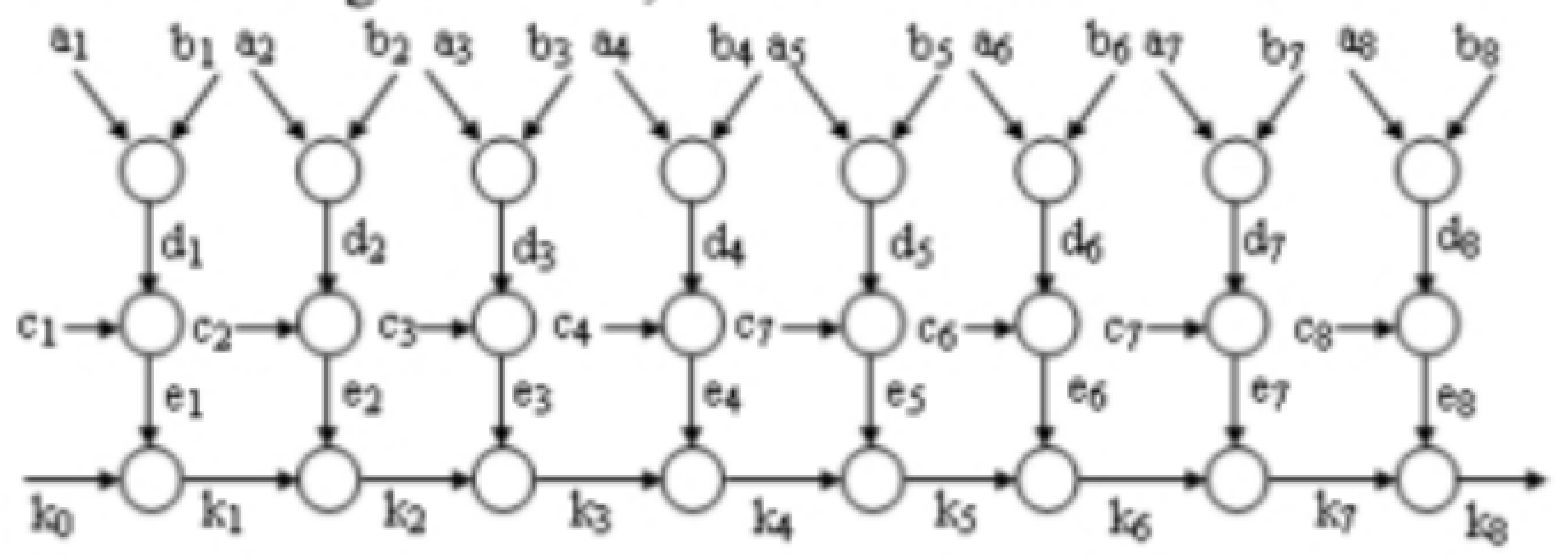
The data flow computers are based on a data driven mechanism. The fundamental difference is that instruction execution in a conventional computer is under program-flow control, whereas that in a data flow computer is driven by the data (operand) availability.

b) The data driven mechanism not requires any shared memory, program counter and control sequencer. It just checks data availability of needy instructions and to enable the chain reaction of asynchronous instruction execution.

c) Suppose, there are some instructions given below. Now we execute these instructions by a data flow machine.

```
Input:  a, b, c
        $k_0 = 0$
for i = 1 to 8 do
   begin
        $d_i = a_i + b_i$
        $e_i = d_i * c_i$
        $k_i = e_i + k_{i-1}$
   end
Output d, e, k.
```

In the above example, there are three instructions in the "for loop" and the "for loop" is executed 8 times. So, total 24 instructions will be executed. Suppose, each add, multiply and divide operation requires 1, 2 and 3 clock cycles to complete respectively. The data flow graph is shown in the figure below, for above instructions.


a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6 a7 b7 a8 b8
d1 d2 d3 d4 d5 d6 d7 d8
c1 c2 c3 c4 c7 c6 c7 c8
e1 e2 e3 e4 e5 e6 e7 e8
k0 k1 k2 k3 k4 k5 k6 k7 k8
Fig: data flow graph

The above instructions can be executed within 14 clock cycles as shown in the figure below.

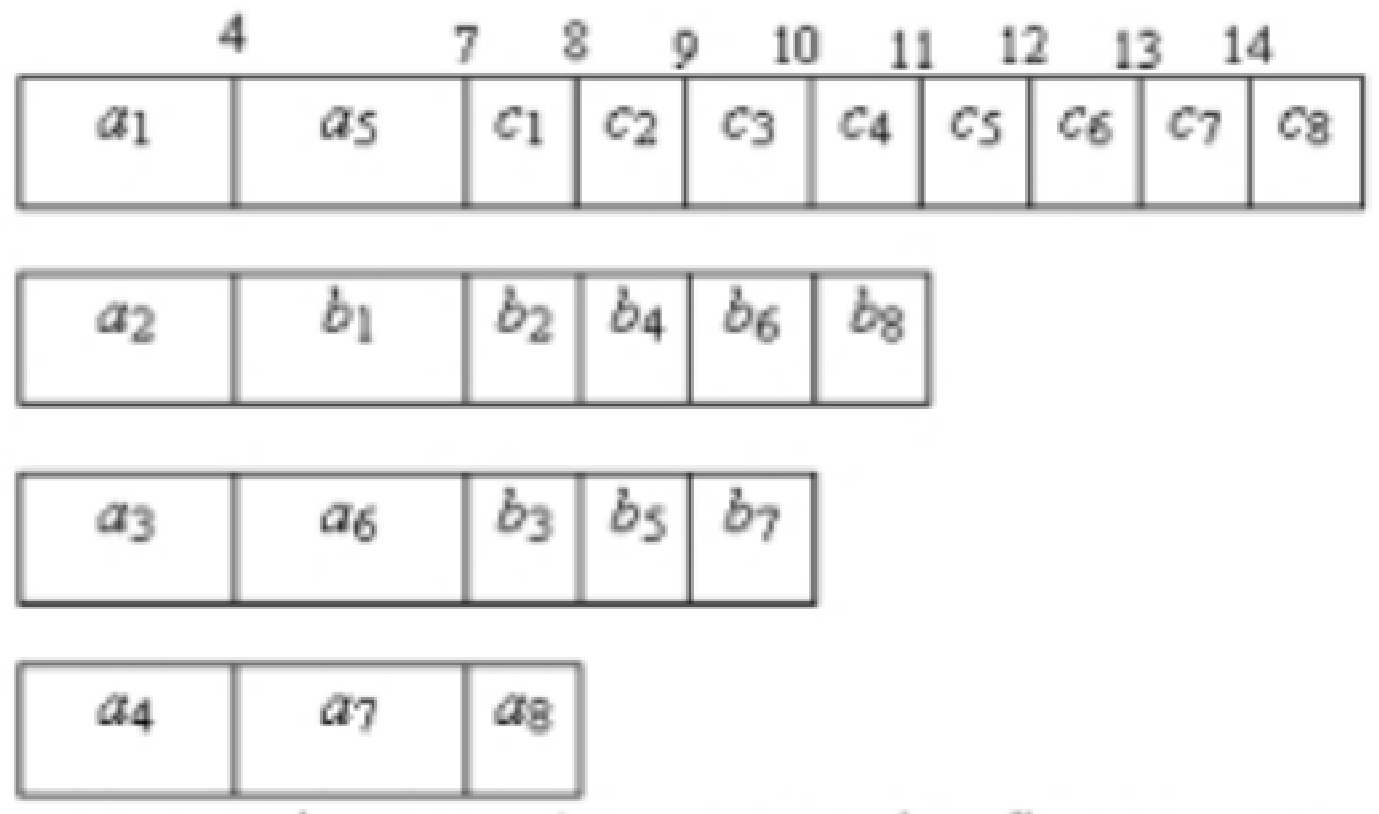
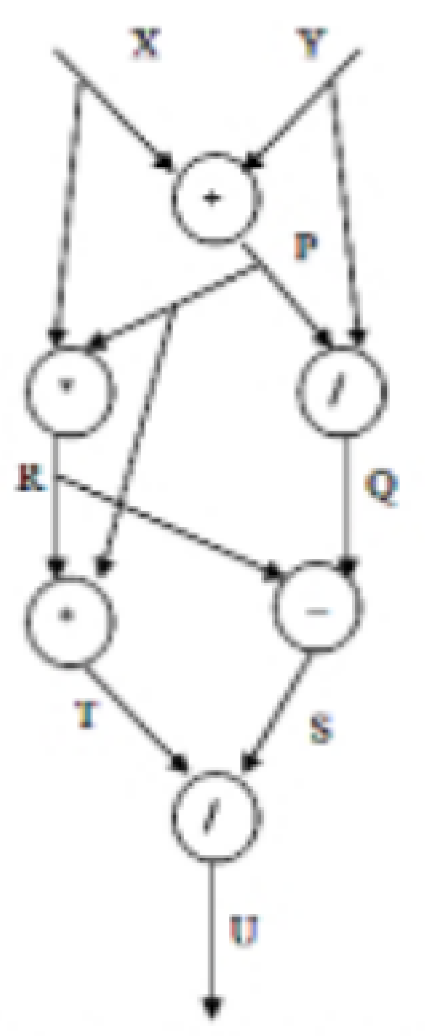

4
7
8
9
10
11
12
13
14
a_1 a_5 c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8
a_2 b_1 b_2 b_4 b_6 b_8
a_3 a_6 b_3 b_5 b_7
a_4 a_7 a_8

Fig: Data driven execution on a 4 processor dataflow computer in 14 cycles

The dataflow multiprocessor completes the execution in 14 cycles. If all the external inputs are available, instructions a_1, a_2, a_3 and a_4 are all ready for execution in the first three cycles produced then trigger the execution of a_5, b_1, a_6, and a_7 starting from cycle 4. The data-driven chain reactions are shown in figure above. The output c8 is the last one to produce, due to its dependence on all the previous c_i's.

The theoretical minimum time is 13 cycles along the critical path $a_1b_1c_1c_2...c_8$. The chain reaction control in dataflow is more difficult to implement and may result in longer overhead, as compared with the uniform operations performed by all - processors.

d)



6. a) What is the difference between Computer Organization and Computer Architecture? [WBUT 2012]

b) Why does the equation to calculate the CPU-time of a program often expressed in terms of average CPI of that processor? [WBUT 2012]

c) A 30% enhancement in speedup for a component of the processor has been proposed for a new architecture. If the enhancement is usable only for 50% for the time, what is fraction of the time must enhancement be used to achieve an overall speedup of 10? [WBUT 2012]

d) What are the different approaches taken by pipeline processor to handle branch instructions? Briefly illustrate any two approaches? [WBUT 2012]

OR,

What is branch hazard? Briefly discuss two methods to handle branch hazard. [WBUT 2014, 2017]

Answer:

a) Difference between Computer Organization and Computer Architecture:

- ***Computer organization***
  - Deals with all *physical components* of computer systems that interacts with each other to perform various functionalities
  - The lower level of computer organization is known as *microarchitecture* which is more detailed and concrete.
  - Examples of Organizational attributes includes Hardware details transparent to the programmer such as control signal and peripheral.

- ***Computer architecture***
  - Refers as a set of *attributes of a system* as seen by programmer
  - Examples of the Architectural attributes include the instruction set, the no of bits used to represent the data types, Input Output mechanism and technique for addressing memories

b) Performance analysis should help answering questions such as how fast can a program be executed using a given computer? In order to answer such a question, we need to determine the time taken by a computer to execute a given job. We define the clock cycle time as the time between two consecutive rising (trailing) edges of a periodic clock signal. Clock cycles allow counting unit computations, because the storage of computation results is synchronized with rising (trailing) clock edges. The time required to execute a job by a computer is often expressed in terms of clock cycles. We denote the number of CPU clock cycles for executing a job to be the cycle count (CC), the cycle time by CT, and the clock frequency by $f = 1/CT$. The time taken by the CPU to execute a job can be expressed as CPU time = $CC \times CT = CC/f$

It may be easier to count the number of instructions executed in a given program as compared to counting the number of CPU clock cycles needed for executing that program. Therefore, the average number of clock cycles per instruction (CPI) has been used as an alternate performance measure. The following equation shows how to compute the CPI.

CPI = CPU clock cycles for the program / Instruction count

CPU time = Instruction count x CPI x Clock cycle time

c) Say, the fraction of the time must enhancement is used to achieve an overall speedup = x

So, x. (30/50)=10 => x= 16.6

d) One of the major problems in designing an instruction pipeline is assuring a steady flow of instructions to initial stages of the pipeline. However, 15-20% of instructions in an assembly-level stream are (conditional) branches. Of these, 60-70% take the branch to a target address. Until the instruction is actually executed, it is impossible to determine whether the branch will be taken or not.

A number of techniques can be used to minimize the impact of the branch instruction (the branch penalty).

- ***Multiple streams:***
  - Replicate the initial portions of the pipeline and fetch both possible next instructions
  - Increases chance of memory contention
  - Must support multiple streams for each instruction in the pipeline

- ***Prefetch branch target***
  - When the branch instruction is decoded, begin to fetch the branch target instruction and place in a second prefetch buffer
  - If the branch is not taken, the sequential instructions are already in the pipe, so there is not loss of performance
  - If the branch is taken, the next instruction has been prefetched and results in minimal branch penalty (don't have to incur a memory read operation at the end of the branch to fetch the instruction)

- ***Loop buffer: Look ahead, look behind buffer***
  - Many conditional branches operations are used for loop control
  - Expand prefetch buffer so as to buffer the last few instructions executed in addition to the ones that are waiting to be executed
  - If buffer is big enough, entire loop can be held in it, this can reduce the branch penalty.

- ***Branch prediction***
  - Make a good guess as to which instruction will be executed next and start that one down the pipeline.
  - Static guesses: make the guess without considering the runtime history of the program : Branch never taken, Branch always taken, Predict based on the opcode
  - Dynamic guesses: track the history of conditional branches in the program.

- ***Delayed branch***
  - Minimize the branch penalty by finding valid instructions to execute in the pipeline while the branch address is being resolved.

- It is possible to improve performance by automatically rearranging instruction within a program, so that branch instruction occur later than actually desired
- Compiler is tasked with reordering the instruction sequence to find enough independent instructions to feed into the pipeline after the branch that the branch penalty is reduced to zero.

7. a) What are the major hurdles to achieve this ideal speed-up?
b) Discuss data hazard briefly.
c) Discuss briefly two approaches to handle branch hazards.
d) Consider a 4-stage pipeline that consists of Instruction Fetch (IF), Instruction Decode (ID), Execute (*Ex*) and Write Back (WB) stages. The times taken by these stages are 50 ns, 60 ns, 110 ns and 80 ns respectively. The pipeline registers are required after every pipeline stage, and each of these pipeline register consumes 10 ns delay. What is the speedup of the pipeline under ideal conditions compare to the corresponding non-pipelined implementation? [WBUT 2012]

Answer:

a) We define the speedup of a k -stage linear pipeline processor over an equivalent non-pipeline processor as
$S_k = n.k / k + (n-1)$

It should be noted that the maximum speedup is $S_k \rightarrow k$,for n >> k. In other words, the maximum speedup that a linear pipeline can provide us is k, where k is the number of stages in the pipe. The maximum speedup is never fully achievable because of data dependencies between instructions, interrupts, and other factors.

b) A data hazard is created whenever there is dependence between instructions, and they are close enough that the overlap caused by pipelining, or other reordering of instructions, would change the order of access to the operand involved in the dependence. Because of the dependence, we must preserve what is called *program order* that is the order that the instructions would execute in, if executed sequentially one at a time as determined by the original source program. There are three types of data hazards can occur:

- ***Read After Write (RAW) hazards*:**

RAW data hazard is the most common type. It appears when the next instruction tries to read from a source before the previous instruction writes to it. So, the next instruction gets the incorrect old value such as an operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.

- ***Write After Read (WAR) hazards:***

WAR hazard appears when the next instruction writes to a destination before the previous instruction reads it. In this case, the previous instruction gets a new value incorrectly such as read an operand and writes soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.

- ***Write After Write (WAW) hazards:***

WAW data hazard is situation when the next instruction tries to write to a destination before a previous instruction writes to it and it results in changes done in the wrong order. Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value. So the results of WAW hazards are:

  - Pipeline Latency
  - Instruction effects not completed before next operation begins

c) Branch hazards occur when the processor is told to branch – i.e., if a certain condition is true, and then jump from one part of the instruction stream to another – not necessarily to the next instruction sequentially. In such a case, the processor cannot tell in advance whether it should process the next instruction (when it may instead have to move to a distant instruction). To minimize the branch penalty, put in enough hardware so that we can test registers, calculate the branch target address, and update the PC during the second stage.

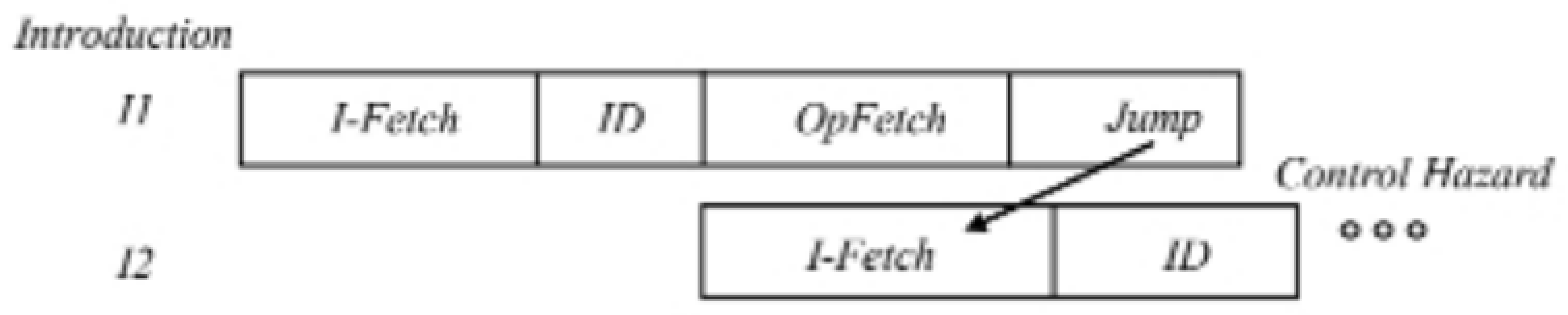


Fig: Situation of control hazards

There are two methods to prevent branch hazard as describe below:

- ***Branch Prediction:*** Branch predication is a strategy in computer architecture design for minimizing the costs, usually associated with conditional branches, particularly branches to short sections of code. It does this by allowing each instruction to conditionally either perform an operation or do nothing. Because computer programs respond to a user, there is no way around the fact that portions of a program need to be executed conditionally. Note that besides eliminating branches, less code is needed in total, provided the architecture provides predicated instructions. While this does not guarantee faster execution in general, it will if the do this and do that blocks of code are short enough. Typically, in order to claim a system has branch predication, most or all of the instructions must have this ability to execute conditionally based on a predicate.
- ***Delayed Branch:*** A branch delay instruction is an instruction immediately following a conditional branch instruction which is executed whether or not the branch is taken. The branch delay slot is a side-effect of pipelined architectures due to the branch hazard, i.e. the fact that the branch would not be resolved until the instruction has worked its way through the pipeline. A simple design would insert stalls into the

pipeline after a branch instruction until the new branch target address is computed and loaded into the program counter. Each cycle where a stall is inserted is considered one branch delay slot. The delayed branch always executes the next sequential instruction, with the branch taking place after that one instruction delay.

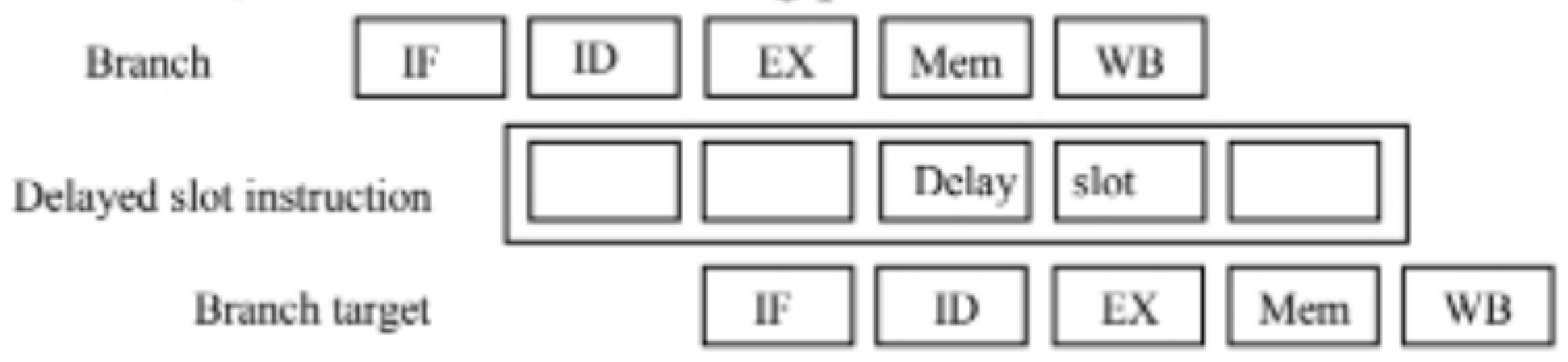


Fig: Situation of control hazards

d) Total time required for each instruction in the pipeline
= 50+10+60+10+110+10+80 = 330 ns
So, the speedup = (Time per instruction on non-pipeline machine/ No. of pipeline stages)
= 330/4 = 82.5

8. a) What do you mean by multiple issue processor? [WBUT 2012, 2018]
b) Briefly describe the VLIW processor architecture. [WBUT 2012]
c) What are the differences between superscalar processor and V.L.I.W processor?
d) Suppose your program consists of 2500 instructions. The proportion of different kinds of instructions in the program is as follow: [WBUT 2012]
Data transfer instruction 50%, arithmetic instruction 30% and branching related instructions 20%. The cycles consumed by these types of instructions are 2, 5 and 10 respectively. What will be the execution time for a 4 GHz processor to execute your program?

Answer:

a) Scoreboarding and Tomasulo are the two single-issue techniques that allow out-of-order execution. The multiple-issue processors are superscalar and VLIW (very long instruction word) processors. Most of today's general-purpose microprocessors are four- or six-issue superscalar often with an enhanced Tomasulo scheme. VLIW is the choice for most signal processors. The **issue logic** examines the waiting instructions in the instruction window and simultaneously assigns *(***issues***)* a number of instructions to the FUs up to a maximum issue bandwidth and Several instructions can be issued simultaneously (the issue bandwidth).

b) Recent high performance processors have depended on Instruction Level Parallelism (ILP) to achieve high execution speed. ILP processors achieve their high performance by causing multiple operations to execute in parallel, using a combination of compiler and hardware techniques. Very Long Instruction Word (VLIW) is one particular style of processor design that tries to achieve high levels of instruction level parallelism by executing long instruction words composed of multiple operations. The long instruction word called a MultiOp consists of multiple arithmetic, logic and control operations each of which would probably be an individual operation on a simple RISC processor. The VLIW processor concurrently executes the set of operations within a MultiOp thereby achieving instruction level parallelism. The remainder of this article discusses the technology, history, uses and the future of such processors. We now describe Defoe, an example processor used in this section to give the reader a feel for VLIW architecture and programming. Though it does not exist in reality, its features are derived from those of several existing VLIW processors. Figure 1 shows the architecture of the VLIW processor.

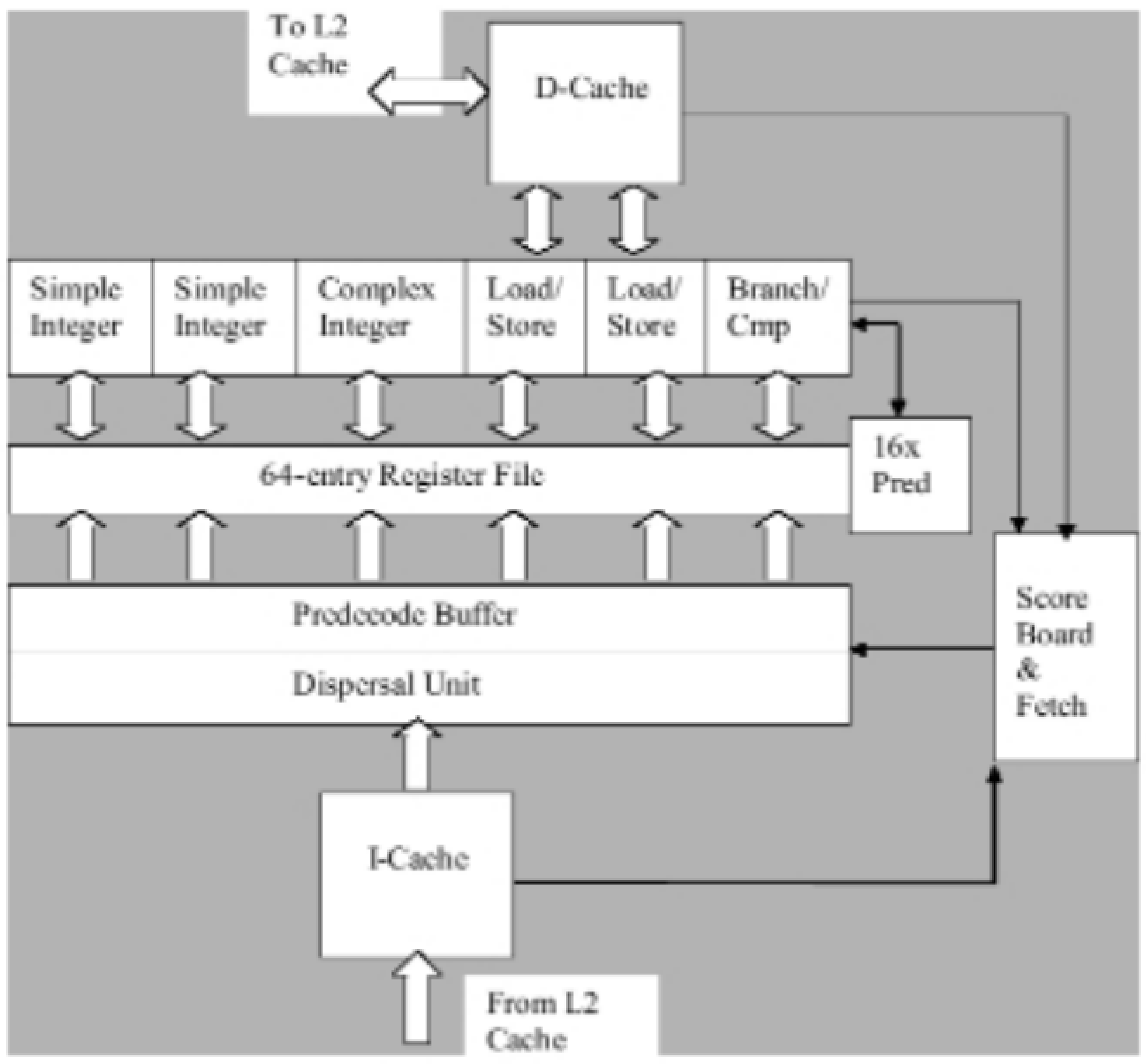


Fig: VLIW architecture

Functional unit:

- Two load/store units.
- Two simple ALUs that perform add, subtract, shift and logical operations on 64-bit numbers and packed 32, 16 and 8-bit numbers. In addition, these units also support multiplication of packed 16 and 8-bit numbers.
- One complex ALU that can perform multiply and divide on 64-bit integers and packed 32, 16 and 8-bit integers.
- One branch unit that performs branch, call and comparison operations.

c)

Superscalar Processor	VLIW Processor
• A superscalar architecture is one in which several instructions can be initiated simultaneously and executed independently. • Superscalar architectures include all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage.	• VLIW architectures rely on compile-time detection of parallelism. The compiler analysis the program and detects operations to be executed in parallel. such operations are packed into one "large" instruction. • After one instruction has been fetched all the corresponding operations are issued in parallel.
• Very complex - Much hardware is needed for run-time detection. - Power consumption can be very large. - The window of execution is limited	• No hardware is needed for run-time detection of parallelism. • The window of execution problem is solved: the compiler can potentially analyze the whole program in order to detect parallel operations.

d) Number of data transfer instructions = 2500 * 50% = 1250 and total no. of cycle consumed = 1250* 2 = 2500

Number of arithmetic instructions = 2500 * 30% = 750 and total no. of cycle consumed = 750* 5 = 3750

Number of branching related instructions = 2500 * 20% = 500 and total no. of cycle consumed = 500* 10 = 5000

Total clock cycle consumed for 2500 instructions = 2500 + 3750+ 5000=11250

Frequency of the processor (f) = 4 GHz
So, clock period ($T = 1/f$) = .25 ns
The execution time for a 4 GHz processor to execute this program= .25 * 11250 = 2.81μs

9. What do you understand by instruction pipelining and arithmetic pipelining? Why pipeline scheduling is necessary and how it is done? [WBUT 2013]

Answer:

1st Part:

Refer to Question No. 10 of Short Answer Type Questions.

2nd Part:

Pipeline instruction scheduling may be done either before or after register allocation or both before and after it. The advantage of doing it before register allocation is that this results in maximum parallelism. The disadvantage of doing it before register allocation is that this can result in the register allocator needing to use a number of registers exceeding those available. This will cause spill/fill code to be introduced which will reduce the performance of the section of code in question.

If the architecture being scheduled has instruction sequences that have potentially illegal combinations (due to a lack of instruction interlocks) the instructions must be scheduled after register allocation. This second scheduling pass will also improve the placement of the spill/fill code.

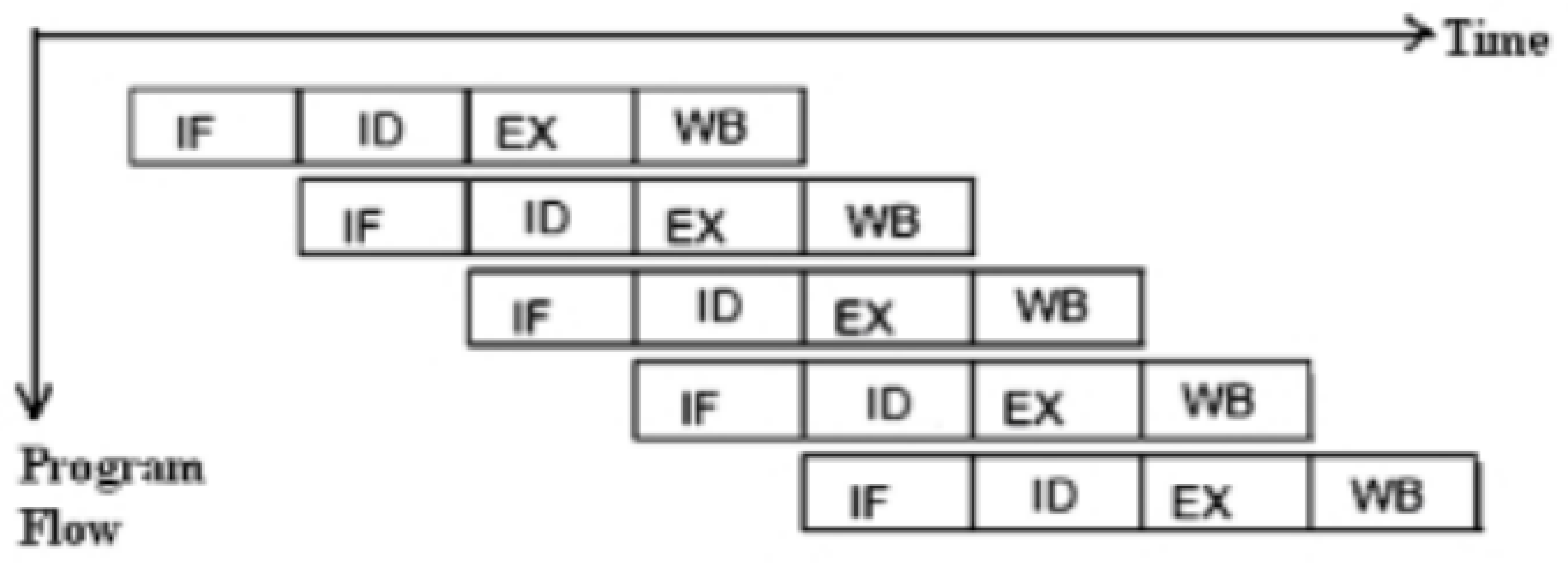
If scheduling is only done after register allocation then there will be false dependencies introduced by the register allocation that will limit the amount of instruction motion possible by the scheduler.

10. Define pipelining technique. Assume a 4 stage pipeline: [WBUT 2013]
Fetch: Read the instruction from the memory
Decode: Decode the instruction

Execute: Execute the instruction
Write: Store the result in destination location
Draw the space-time diagram for pipelining.

Answer:

Pipeline is an implementation technique whereby multiple instructions are overlapped in execution. Each step in the pipeline (called a pipe stage) completes a part of an instruction.



Instruction Fetch Cycle (IF): In this cycle CPU sends the address which is held by program counter (PC) to the memory. Then fetch the current instruction from memory and update the content of PC for the next instruction.

Instruction Decode/Register Fetch Cycle (ID): In this cycle CPU decodes the instruction and reads the registers from the register file and performs the equality test on the registers for a possible branch. Then sign-extend the offset field of the instruction in case it is needed and computes the possible branch target address by adding the sign-extended offset to the incremented PC.

Execution/Effective Address Calculation (EX): In the execution cycle the ALU operates on the operands prepared in the prior cycle. If it is memory reference instructions, the ALU adds the base register and the offset to form the effective address or if it is Register-

Register instruction then the ALU performs the operation specified by the ALU opcode on the values from the register file. If it is Register-Immediate instruction then the ALU performs the operation specified by the opcode on the first value from the register file and the sign-extended immediate.

Write-Back cycle (WB): Register-Register ALU instruction or Load instruction: Write the result into the register file.

11. a) What is arithmetic and instruction pipeline?
b) Consider the following reservation table

	1	2	3	4
S1	×		×	
S2		×		
S3			×	
S4		×		×

List the set of forbidden latencies and collision vector. Draw the state transition diagram. List all simple cycle from state diagram. Identify the greedy cycles among simple cycles. Find out minimum average latency (MAL). Find out maximum throughput of this pipeline if the clock rate is 25 MHz.

OR,

Consider the four-stage pipelined processor specified by the following reservation table: [WBUT 2018]

	1	2	3	4
S1	X		X	
S2		X		
S3			X	
S4		X		X

(i) List the set of forbidden latencies and collision vector.
(ii) Draw the state transition diagram.
(iii) List all simple cycles from state diagram.
(iv) Identify the simple cycles among greedy cycles.
(v) Find out minimum average latency
c) What are bounds on MAL? [WBUT 2014]

Answer:

a) ***Refer to Question No. 10 of Short Answer Type Questions.***

b)

Forbidden Latencies are: 0, 2
Pipeline collision Vector is: (1010)
Greedy Cycle is: (1, 3)*

State Diagram is
State 1: 1010
Reaches State 2 (1110) after 1 cycle.
Reaches State 1 (1010) after 3 cycles.
Reaches State 1 (1010) after 4 or more cycles.
State 2: 1110
Reaches State 1 (1010) after 3 cycles.
Reaches State 1 (1010) after 4 or more cycles.

There are 2 states in the state diagram:
State 1 represents 1010
State 2 represents 1110

Minimal Average Latency (MAL) is : 2
Pipeline clock period be $\tau = 40$ ns.
So, throughput = MAL /(no. of stages in the pipeline* time period)
$= 2/ (4 * 40 \times 10^{-9}) = 1.25$ MIPS
Lower bound of MAL = 2 (maximum number of check marks in any row of the reservation table)
Upper bound of MAL= 2+1=3 (no. of 1's in initial collision vector +1)

c) ***Refer to Question No. 4 of Long Answer Type Questions.***

12. What do you mean by multiple issue processors?
Briefly describe the VLIW processor architecture. What are the limitations of VLIW? [WBUT 2014]

Answer:

Refer to Question No. 8 (a) & (b) of Long Answer Type Questions.

13. a) What is meant by pipeline hazard? Briefly discuss different pipeline hazards.
b) What do you mean by job collision in pipeline processor? Show how collisions occur in the following static pipeline.

	0	1	2	3	4
S_0	X				X
S_1		X		X	
S_2			X		

c) Consider the execution of a program of 20,000 instructions by a linear pipeline processor with a clock rate 40 MHz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-order executions are ignored. Calculate the speed-

up of the pipeline over its equivalent non-pipeline processor, the efficiency and throughput. [WBUT 2015]

Answer:

a) ***Refer to Question No. 7 of Short Answer Type Questions.***

b) The number of cycles between initiations is called the latency. The first step is to identify the forbidden latencies revealed by the diagram. A latency is forbidden if it will lead to a collision. The table shows that stage S_0 is required during both the first and last cycles. We cannot initiate a new task after only one cycle, or there would be an immediate collision. There is a systematic way to identify all forbidden latencies from a given reservation table. For each row containing more than one X, we write down the distance between every pair of X's. Each such distance represents a forbidden latency. In our example, row S_0 contains two X's which can form one distinct pair. The latency blocked by these pair is 4. Row S_1 yields a forbidden latency in 2. There are no forbidden latencies for row S_3, since it contains only one X.

The set of forbidden latencies is then summarized in a bit string called a collision vector. The collision vector contains one bit of each possible latency. The bits are numbered from left to right (but some authors use the opposite order). The collision vector for our example table is 11010.

c) Speedup, $S_k = nk\tau / [k + (n-1)]\tau = (20000 \times 5 \times 40) / (5 + (20000-1)) \times 40 = 5$

14. Consider the following pipeline reservation table.

	1	2	3	4	5	6	7	8
S1	X					X		X
S2		X		X				
S3			X		X		X	

a) What are the forbidden latencies?

b) Draw the state transition diagram.

c) List all the simple cycles and greedy cycles.

d) Determine the optimal constant latency cycle and the minimal average latency.

e) Let the pipeline clock period be $\tau = 20ns$. Determine the throughput of the pipeline. [WBUT 2016]

Answer:

a) The Forbidden Latencies for your data 0, 2, 4, 5, 7

b) The State Diagram in a Text Manner for your data

State 1 : 10101101

Reaches State 2 (11111111) after 1 cycle.

Reaches State 3 (11101101) after 3 cycles.

Reaches State 3 (11101101) after 6 cycles.

Reaches State 1 (10101101) after 8 or more cycles.

State 2 : 11111111
Reaches State 1 (10101101) after 8 or more cycles.
State 3 : 11101101
Reaches State 3 (11101101) after 3 cycles.
Reaches State 3 (11101101) after 6 cycles.
Reaches State 1 (10101101) after 8 or more cycles.
There are 3 states in the state diagram:
State 1 represents 10101101
State 2 represents 11111111
State 3 represents 11101101

c) The Greedy Cycle is (1, 8)*

d) The Minimum Average Latency = 4.5

e) Throughput,
W= (η / T) * (1/ MAL) = $(1/ 20 \times 10^{-9})$ * (1/4.5) = .222

15. With the use of Amdahl's law, conclude, among the given options which possible improvement is the best one.

Instruction type	Frequency	CPI
ALU	40%	1
Branch	20%	4
Load	30%	2
Store	10%	3

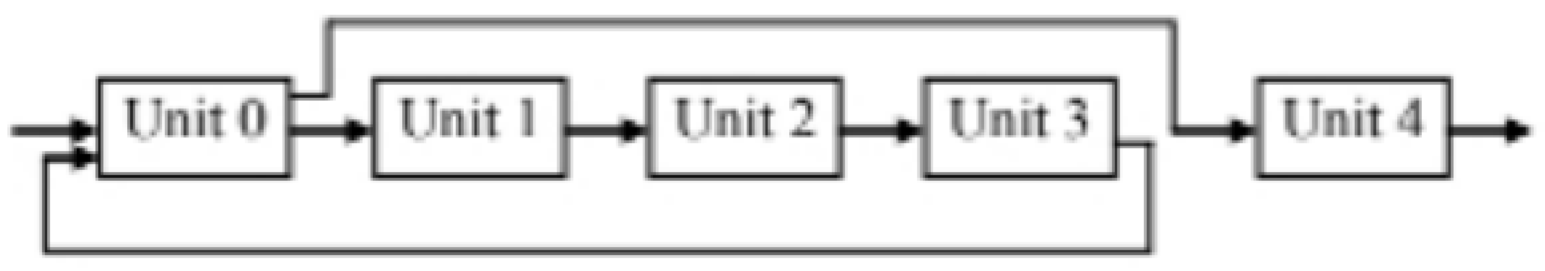
Possible improvements:
1. Branch CPI can be decreased from 4 to 3.
2. Increase clock frequency from 2 to 2.3 GHz.
3. Store CPI can be decreased from 3 to 2. **[WBUT 2016]**

Answer:
To improve the performance, we have to calculate CPI for all cases.
In the given problem,
CPI= 0.4*1+0.2*4+0.3*2+0.1*3= 2.1
Now, clock is 2GHz = 2000MHz
Thus MIPS ((millions of instructions per second) = 2.1*2000= 4200
Now in case 1, branch CPI can be decreased from 4 to 3.
CPI= 0.4*1+0.2*3+0.3*2+0.1*3= 1.9

If the clock is 2GHz, then
MIPS = 1.9*2000=3800
Now in case 2, Increase clock frequency from 2 to 2.3 GHz.
When clock is 2.3 GHz,

MIPS((millions of instructions per second) = 2.1*2300=4830
Now in case 2, Store CPI can be decreased from 3 to 2.
CPI= 0.4*1+0.2*4+0.3*2+0.1*2= 2
If the clock is 2GHz, then
MIPS= 2*2000= 4000
So, among the given options case 2 is the best one for improvement.

16. a) Consider the following block diagram of a circuit. Form the Reservation table.



b) An instruction requires four stages to execute:
Stage 1 (instruction fetch) requires 30 ns:
Stage 2 (instruction decode) = 9 ns, stage 3 (instruction execute) = 20 ns and stage 4 (store results) = 10 ns. An instruction must proceed through the stages in sequence. What is the minimum asynchronous time for any single instruction to complete?
c) We want to set this up as a pipelined operation. How many stages should we have and at what rate should we clock the pipeline? [WBUT 2016]

Answer:

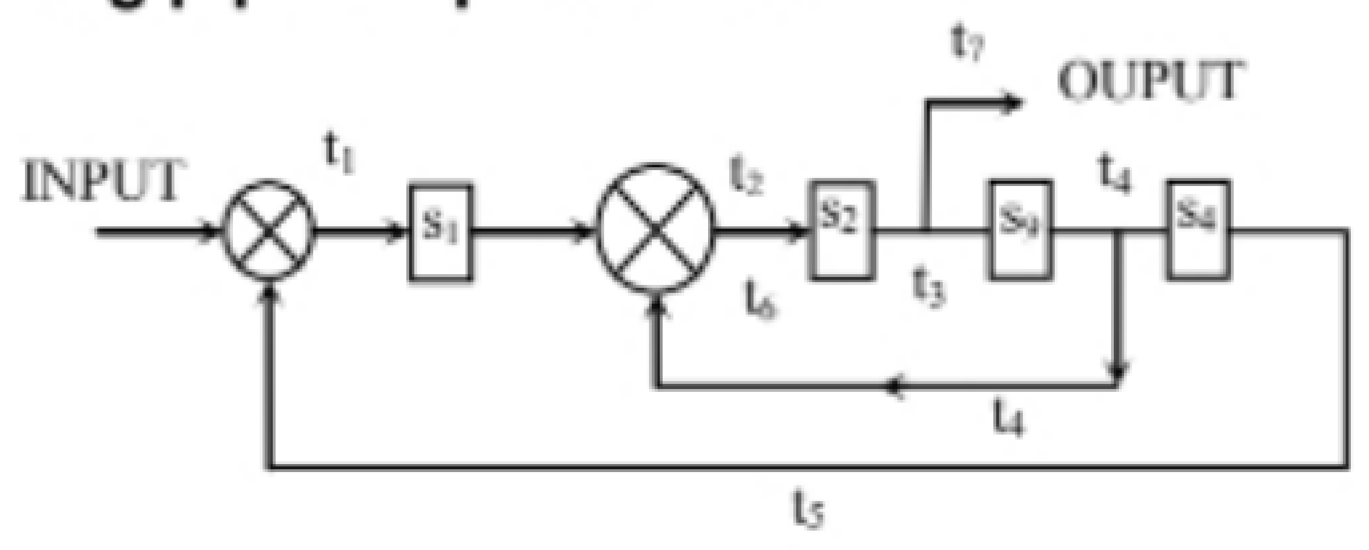
a)

	Time					
	t_0	t_1	t_2	t_3	t_4	t_5
Unit 0	X				X	
Unit 1		X				
Unit 2			X			
Unit 3				X		
Unit 4		X				X

b) The minimum asynchronous time for any single instruction to complete = (30 + 9+20+10) ns = 69ns

c) To set up a pipeline operation, we need to construct minimum five stages in the pipeline. The stages are instruction fetch, instruction decode, operant read, instruction execution and store results. We should calculate that which stage requires the highest time for execution and this amount of time we have to set for all the stages of the pipeline.

17. Consider the following pipeline processor:



Where, S = number of stages & T = clock cycles
S_I is number of stages and T_I is clock cycle
a) Specify the reservation table for this pipeline with six columns and four rows.
b) List the set of forbidden latencies between task initiations.
c) Draw the state diagram which shows all possible latency cycles.
d) List all greedy cycles from the state diagram.
e) What is the value of minimal average latency?
f) What is the maximal throughput of this pipeline? **[WBUT 2017]**

Answer:

a) Reservation table

Time --->							
		1	2	3	4	5	6
Stages	S1	X				X	
	S2		X				X
	S3			X			
	S4				X		

b) Forbidden latencies: 0, 4

c) Draw the state diagram which shows all possible latency cycles.
The Pipeline Collision Vector: (1 0 0 0 1 0)

State diagram
State 1: 100010
  Reaches State 2 (100110) after 1 cycle.
  Reaches State 3 (101010) after 2 cycles.
  Reaches State 4 (110010) after 3 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 2 : 100110
  Reaches State 5 (101110) after 1 cycle.
  Reaches State 6 (111010) after 2 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 3 : 101010
  Reaches State 7 (110110) after 1 cycle.
  Reaches State 4 (110010) after 3 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 4 : 110010
  Reaches State 3 (101010) after 2 cycles.
  Reaches State 4 (110010) after 3 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 5 : 101110
  Reaches State 8 (111110) after 1 cycle.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 6 : 111010
  Reaches State 4 (110010) after 3 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 7 : 110110
  Reaches State 6 (111010) after 2 cycles.
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

State 8 : 111110
  Reaches State 1 (100010) after 5 cycles.
  Reaches State 1 (100010) after 6 or more cycles.

There are 8 states in the state diagram:
State 1 represents 100010
State 2 represents 100110
State 3 represents 101010
State 4 represents 110010
State 5 represents 101110
State 6 represents 111010
State 7 represents 110110
State 8 represents 111110

d) List all greedy cycles from the state diagram.
The Greedy Cycle: (1, 1, 1, 5)*

e) The Minimum Average Latency: 2

f) The Throughput: 0.5 instruction per cycle

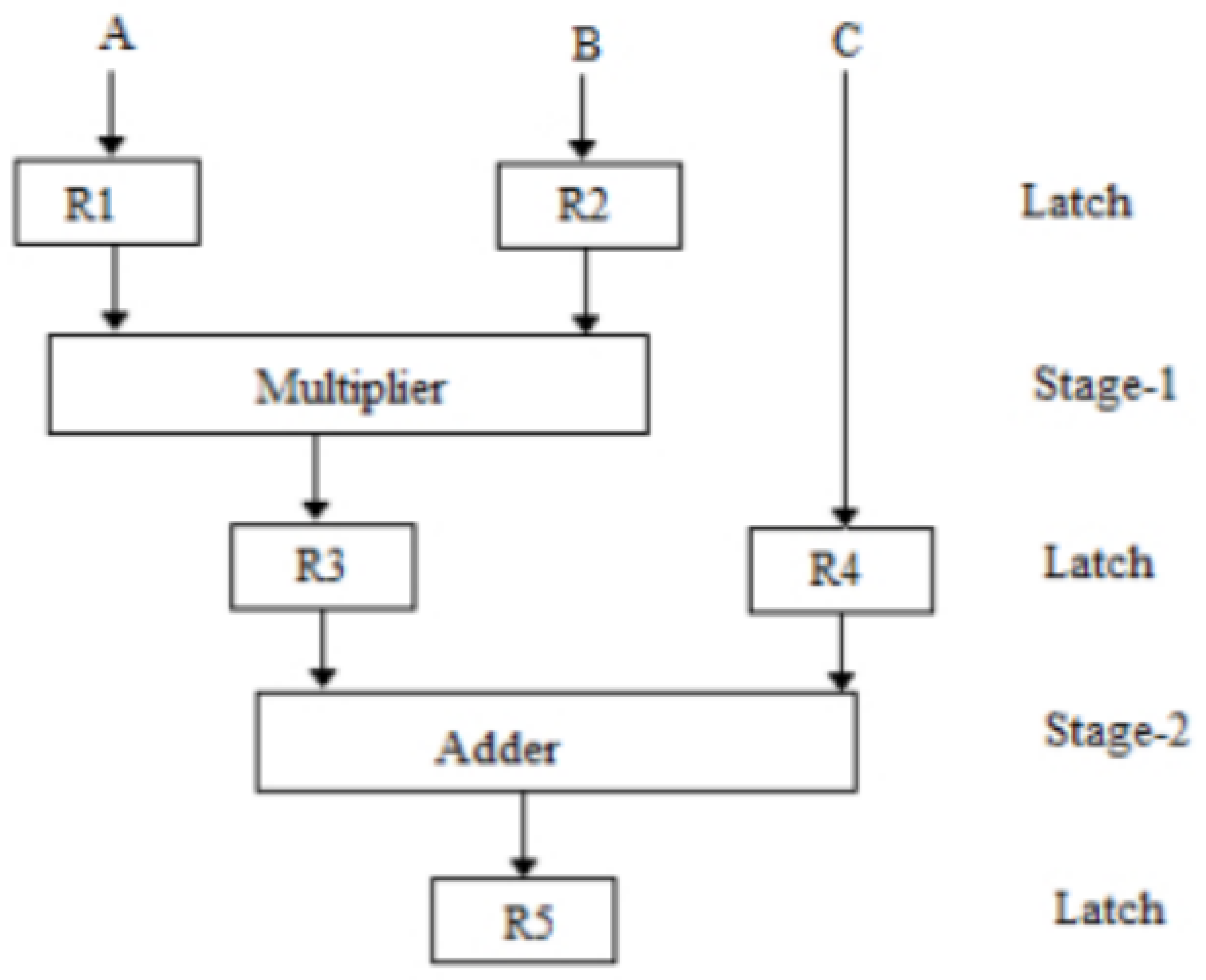
18. With the help of a neat diagram show the structure of a typical arithmetic pipeline performing (A*B+C). **[WBUT 2017]**

Answer:

The sub operations to be performed for arithmetic expression A*B+C in each stage of the pipeline are as follows:

Sub-operation 1: R1← A, R2←B	Input A and B
Sub-operation 2: R3← R1*R2, R4←C	Multiply and input C
Sub-operation 3: R5← R3 + R4	Add C to product

Five registers are loaded with new data in every clock period. The corresponding pipeline is shown below.



19. Consider the following pipeline reservation table

	1	2	3	4	5	6	7	8
S1	X					X		X
S2		X		X				
S3			X		X		X	

a) **Find forbidden and permissible latency set**
b) **Draw the state transition diagram**
c) **Find all simple cycles**
d) **Find the minimum average latency**

e) What are the performance measuring parameters of a pipelining system? Explain briefly. **[WBUT 2019]**

Answer:

a) to d) ***Refer to Question No.14 of Long Answer Type Questions.***

e) The ability to overlap stages of a sequential process for different input tasks (data or operations) results in an overall theoretical completion time of

$$T_{pipe} = m.P+(n-1).P$$

Here n is the number of input tasks, m is the number of stages in the pipeline, and P is the clock. The three basic performance measures for the pipeline are as follows:

Speed up:

K-stage pipeline processes n tasks in $k + (n-1)$ clock cycles: k cycles for the first task and $(n-1)$ cycles for the remaining $(n-1)$ tasks

Total time to process n tasks $T_k = [k + (n-1)]$

For the non-pipelined processor $T_1 = n.k$

Speed up factor is

\Speedup=*(Time taken by non-pipelined implementation) /(Time taken by pipelined implementation)*

Speedup= $nky/(K+(n-1))y=nk/(k+(n-1))$

if $n \to \infty$, speedup = $nk/n=k$

As the k stage reaches n and the value of n is approximated to infinity. The speedup factor is equivalent to k

Throughput:

Throughput is the outputs produced per clock cycle and that throughput will be equal to 1, in case of ideal situation that means, when the pipeline is producing one output per clock cycle.

$U(n) = m*f / n+(m-1)$

Efficiency: The efficiency of n stages in a pipeline is defined as ratio of the actual speedup to the maximum speed.

$E(n) = m / n+m-1$

20. Simplify the following program segment using internal forwarding and register tagging techniques:

R0 $\leftarrow$ (M1)
R0 $\rightarrow$ (R0)+(M2)
R0 $\rightarrow$ (R0)*(M2)
M4 $\rightarrow$ (R0) **[WBUT 2019]**

Answer:

- Internal Forwarding: A "short-circuit" technique to replace unnecessary memory accesses by register-register transfers in a sequence of fetch-arithmetic-store operations

- Register Tagging: Use of tagged registers , buffers and reservation stations, for exploiting concurrent activities among multiple arithmetic units
- Store-Fetch Forwarding –
  (M ← R1, R2 ← M) replaced by (M ← R1, R2 ← R1)
- Fetch-Fetch Forwarding –
  (R1 ← M, R2 ← M) replaced by (R1 ← M, R2 ← R1)
- Store-Store Overwriting –
  (M ← R1, M ← R2) replaced by (M ← R2)]

Simplify the given program segment using internal forwarding and register tagging techniques:

R0 ← (M1)
R0 → (R0)+(M2)
R0 → (R0)*(M2)
M4 → (R0)
	R0 ← M1, R1 ← M2, R3← M2
	R0← R0 + R1
	R0← R0 + R3
	M4 ← R0

R0 ⟵ (MI)	R0 ⟵M1,R1 ⟵M2,R3 ⟵M2
R0 ⟶ (R0)+(M2)	R0 ⟵R0 + R1
R0 ⟶ (R0)*(M2)	R0 ⟵R0 + R3
M4 ⟶ (R0)	M4 ⟵R0

21. Write short notes of the following:

a) Pipeline hazards	**[WBUT 2005, 2011, 2014]**
b) Reservation Table	**[WBUT 2008]**
c) Branch handling in instruction pipeline	**[WBUT 2011]**
d) Amdahl's law and its significance	**[WBUT 2012, 2014]**
e) The VLIW processor architecture	**[WBUT 2018]**
f) Data flow computer	**[WBUT 2018]**

Answer:

a) With pipelining, each instruction is supposed to start executing at a given clock cycle. Unfortunately there are cases in which an instruction cannot execute at its allotted clock cycle. These situations are called: pipeline hazards. Hazards further reduce the performance gain from the speedup.

So,

- The hazard is a situation which prevents to fetch the next instructions in the instruction stream from executing during its designated clock cycle.
- Hazards reduce the performance from the ideal speedup gained by pipelining.
  - Data Hazards
  - Structural Hazards
  - Control Hazards
- Hazards can make it necessary to stall the pipeline.

- When an instruction is stalled, all instructions issued later than the stalled instruction are also stalled.
- No new instructions are fetched during the stall.

Data Hazard
Data hazards may be classified as one of three types, depending on the order of read and write accesses in the instructions. By convention, the hazards are named by the ordering in the program that must be preserved by the pipeline.
There are three types of data hazards can occur:

- ***Read After Write (RAW) hazards*:**

RAW data hazard is the most common type. It appears when the next instruction tries to read from a source before the previous instruction writes to it. So, the next instruction gets the incorrect old value such as an operand is modified and read soon after. Because the first instruction may not have finished writing to the operand, the second instruction may use incorrect data.

- ***Write After Read (WAR) hazards:***

WAR hazard appears when the next instruction writes to a destination before the previous instruction reads it. In this case, the previous instruction gets a new value incorrectly such as read an operand and writes soon after to that same operand. Because the write may have finished before the read, the read instruction may incorrectly get the new written value.

- ***Write After Write (WAW) hazards:***

WAW data hazard is situation when the next instruction tries to write to a destination before a previous instruction writes to it and it results in changes done in the wrong order. Two instructions that write to the same operand are performed. The first one issued may finish second, and therefore leave the operand with an incorrect data value. So the results of WAW hazards are:

- Pipeline Latency
- Instruction effects not completed before next operation begins

Structural Hazards
A structural hazard occurs when a part of the processor's hardware is needed by two or more instructions at the same time. A structural hazard might occur, for instance, if a program were to execute a branch instruction followed by a computation instruction. Because they are executed in parallel, and because branching is typically slow (requiring a comparison, program counter-related computation, and writing to registers), it is quite possible that the computation instruction and the branch instruction will both require the ALU at the same time. Hardware cannot support the combination of instructions that we want to execute in the same clock cycle.

Control hazards
Controls hazards occur when the processor is told to branch – i.e., if a certain condition is true, and then jump from one part of the instruction stream to another – not necessarily to the next instruction sequentially. In such a case, the processor cannot tell in advance whether it should process the next instruction (when it may instead have to move to a distant instruction). To minimize the branch penalty, put in enough hardware so that we can test registers, calculate the branch target address, and update the PC during the second stage.

b) Scheduling and control are important factors in the design of nonlinear and dynamic pipelines. Any time the hardware is reconfigured, or any time the same stage is used more than once in a computation, a structural hazard may exist, meaning there is the possibility of a *collision* in the pipeline. A collision is an attempt to use the same stage for two or more operations at the same time. If two or more sets of inputs arrive at the same stage simultaneously, at the very least the pipeline will compute erroneous results for at least one set of inputs. Depending on the details of physical construction, the outputs of different stages could even be short-circuited to a common input, possibly causing damage to the circuitry. Thus collisions are to be avoided at all costs when controlling a pipeline.

How can we determine when collisions might occur in a pipeline? One graphical tool we can use to analyze pipeline operation is called a *reservation table.* A reservation table is just a chart with rows representing pipeline stages and columns representing time steps (clock cycles). Marks are placed in cells of the table to indicate which stages of the pipeline are in use at which time steps while a given computation is being performed. The simplest reservation table is one for a static, linear pipeline. Table below is an example of a reservation table for a five-stage static linear pipeline.

Notice that all the marks in this simple reservation table lie in a diagonal line. This is because each stage is used once and only once, in numerical order, in performing each computation. Even if we permuted the order of the stages, there would be still be only one mark in each row because no stage would be used more than once per operation. As long as this is the case, we will be able to initiate a new operation in the pipeline on each clock cycle. Suppose instead that we had a nonlinear pipeline. Some of the stages are used more than once per computation. This pipeline would have a reservation table as depicted in Table. Notice that in this case rows 2 and 3 of the table contain more than one mark, depicting the repeated use of the corresponding stages.

	t_0	t_1	t_2	t_3	t_4
Stage 1	X				
Stage 2		X			
Stage 3			X		
Stage 4				X	
Stage 5					X

Fig: Reservation table for static linear pipeline

	t_0	t_1	t_2	t_3	t_4
Stage 1	X				
Stage 2		X		X	
Stage 3			X		X

Fig: Reservation table for non linear pipeline

Latches are used between pipeline stages to get Minimum Average Latency (MAL). An optimization technique based on introducing a method to search the most proper location of no compute delay latches between nonlinear pipeline stages is given. The idea is to find a new collision vector which is adaptable with pipeline topology and modifies reservation table, yielding MAL at minimum execution time. This approach not only reduces execution time of hardware, but also minimizes favorite collision vector search time.

c) In pipeline process the branch instructions are those that tell the processor to make a decision about what the next instruction to be executed should be based on the results of another instruction. Branch instructions can be troublesome in a pipeline if a branch is conditional on the results of an instruction which has not yet finished its path through the pipeline.
For example:

```
Loop : add   $r3,  $r2,  $r1
       sub   $r6,  $r5,  $r4
       equal $r3, $r6, Loop
```

The example above instructs the processor to add r1 and r2 and put the result in r3, then subtract r4 from r5, storing the difference in r6. In the third instruction, beq stands for branch if equal. If the contents of r3 and r6 are equal, the processor should execute the instruction labeled "Loop." Otherwise, it should continue to the next instruction. In this example, the processor cannot make a decision about which branch to take because neither the value of r3 or r6 have been written into the registers yet.

The processor could stall, but a more sophisticated method of dealing with branch instructions is branch prediction. The processor makes a guess about which path to take - if the guess is wrong, anything written into the registers must be cleared, and the pipeline must be started again with the correct instruction. Some methods of branch prediction depend on stereotypical behavior. Branches pointing backward are taken about 90% of the time since backward-pointing branches are often found at the bottom of loops. On the other hand, branches pointing forward are only taken approximately 50% of the time. Thus, it would be logical for processors to always follow the branch when it points backward, but not when it points forward. Other methods of branch prediction are less static: processors that use dynamic prediction keep a history for each branch and use it to predict future branches. These processors are correct in their predictions 90% of the time.

d) Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the problem size remains the same when parallelized. For example, if for a given problem size a parallelized implementation of an algorithm can run 12% of the algorithm's operations arbitrarily quickly (while the remaining 88% of the operations are not parallelizable), Amdahl's law states that the maximum speedup of the parallelized version is $1/(1-0.12) = 1.136$ times as fast as the non-parallelized implementation.

More technically, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S. (For example, if 30% of the computation may be the subject of a speed up, P will be 0.3; if the improvement makes the portion affected twice as fast, S will be 2.) Amdahl's law states that the overall speedup of applying the improvement will be:

$$\frac{1}{(1-P)+\frac{P}{S}}$$

To see how this formula was derived, assume that the running time of the old computation was 1, for some unit of time. The running time of the new computation will be the length of time the unimproved fraction takes (which is $1 - P$), plus the length of time the improved fraction takes. The length of time for the improved part of the computation is the length of the improved part's former running time divided by the speedup, making the length of time of the improved part P/S. The final speedup is computed by dividing the old running time by the new running time, which is what the above formula does.

e) The VLIW processor architecture:
Refer to Questions No .8 (b) of Long Answer Type Questions.

f) Data flow computer:
Refer to Questions No. 2 (a) of Long Answer Type Questions.

MEMORY

☞ Chapter at a Glance

Memory Hierarchy

The hierarchical arrangement of storage in current computer architectures is called the memory hierarchy. It is designed to take advantage of memory locality in computer programs. Each level of the hierarchy has the properties of higher speed, smaller size, and lower latency than lower levels.

Inclusion, Coherence, and Locality Properties

1. ***Inclusion Property***

   According to the inclusion property, the memory contains that present in the upper level of memory hierarchy must present also lower level of memory. So, we can state the inclusion property as $M_1 \subset M_2 \subset M_3 \subset ... \subset M_n$

   At the time of the processing, required portion of memory M_n are copied into M_{n-1}. Similarly, subsets of M_{n-1} are copied into M_{n-2}, and so on. So, if a word is found in memory M_i , then copies of the same word can be also found in all levels as M_{i+1}, M_{i+2}, ..., M_n. But, a word stored in M_{i+1} may not be found in M_i.

2. ***Cache Coherence Property***

   Memory coherence property can manage the memories of a multiprocessor system so that no data is lost or overwritten before the data is transferred from a cache to the target memory. Memory caching is effective because most programs access the same data or instructions over and over. When multiple processors with separate caches share a common memory, it is necessary to keep the caches in a state of coherence by ensuring that any shared operand that is changed in any cache is changed throughout the entire system. This is done by a directory-based system.

3. ***Locality of Reference***

   The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly. Locality occurs often because of the way in which computer programs are created. Generally, related data is stored in nearby locations in storage. Locality often occurs because code contains loops that tend to reference arrays or other data structures by indices. Increasing and exploiting locality of reference are common techniques for optimization. This can happen on several levels of the memory hierarchy. Paging obviously benefits from spatial locality. A cache is a simple example of exploiting temporal locality, because it is a specially designed faster but smaller memory area, generally used to keep recently referenced data and data near recently referenced data, which can lead to potential performance increases.

Reducing Cache Misses: There are three methods to reduce the miss penalty of memory access.

1. ***Increase block size:*** If we increase the block size then it reduces the miss penalty. Large block can store large amount of data. So, it will reduce the compulsory miss. But increase the Capacity miss.
2. ***Sub-block Placement to Reduce Miss Penalty:*** Generally we don't load full block of memory contain on a miss penalty and maintain valid invalid bits per sub-block.
3. ***Early Restart and Critical Word First:*** In this strategy, to reduce miss penalty, don't wait for full block to be loaded before restarting CPU. Early restart the memory access. As soon as the requested word of the block arrives, send it to the CPU.

Cache Memory

Cache Memory is a small quantum amount of fast memory which sits between standard main memory and CPU as shown in the figure below. It holds copies of recently accessed instructions and data**.** It may be located on CPU chip or module. When CPU requests contents of memory location, then it first checks cache for this data.

$$\begin{tikzcd}\boxed{\text{CPU}} \arrow[r, shift left] & \boxed{\text{Cache}} \arrow[l, shift left] \arrow[r, shift left] & \boxed{\text{Main Memory}} \arrow[l, shift left]\end{tikzcd}$$

Fig: Position of cache memory

Different Mapping function of Cache Memory: A mapping function is the method used to locate a memory address within a cache. It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache. There are three kinds of mapping functions such as: (i) Direct Mapping, (ii) Associative Mapping, (iii) Set Associative Mapping.

Virtual Memory

A virtual memory system attempts to optimize the use of the main memory with the hard disk. In effect, virtual memory is a technique for using the secondary storage to extend the apparent limited size of the physical memory beyond its actual physical size. It is usually the case that the available physical memory space will not be enough to host all the parts of a given active program. Those parts of the program that are currently active are brought to the main memory while those parts that are not active will be stored on the magnetic disk. If the segment of the program containing the word requested by the processor is not in the main memory at the time of the request, then such segment will have to be brought from the disk to the main memory. The most relevant principle is that of keeping active segments in the high-speed main memory and moving inactive segments back to the hard disk.

Implementation of Virtual Memory: Virtual memory may be implemented using – (i) Paging and (ii) Segmentation.

Multiple Choice Type Questions

1. A computer with cache access time of 100ns, a main memory access time of 1000 ns, and a hit ratio of 0.9 produces an average access time of **[WBUT 2007, 2010, 2014]**

a) 250 ns **b) 200 ns** **c) 190 ns** **d) none of these**

Answer: (c)

2. Consider the high speed 40 ns memory cache with a successful hit ratio of 80%. The regular memory has an access time of 100ns. What is the effective access time for CPU to access memory? **[WBUT 2007, 2008, 2009, 2011]**

a) 52 ns **b) 60 ns** **c) 70 ns** **d) 80 ns**

Answer: (a)

3. Assuming a Main memory of size 32 k × 12, Cache memory of size 512 × 12 and block size of 1 word, the addressing relationships using direct mapping would be **[WBUT 2007]**

a) tag field – 6 bits, index field – 9 bits **b) tag field – 9 bits, index field –6 bits**
c) tag field – 7 bits, index field – 8 bits **d) none of these**

Answer: (a)

4. Associative memory is a **[WBUT 2008, 2009]**

a) pointer addressable memory **b) very cheap memory**
c) content addressable memory **d) slow memory**

Answer: (c)

5. The principle of locality justifies the use of **[WBUT 2008, 2009]**

a) Interrupts **b) Polling** **c) DMA** **d) Cache memory**

Answer: (d)

6. How many address bits are required for a 512×4 memory? **[WBUT 2008, 2009]**

a) 512 **b) 4** **c) 9** **d) $A_0 - A_6$**

Answer: (c)

7. Assume a system where main memory is of size 16K × 12 and cache memory is of size 1K × 12. For a direct mapping system which statement is correct? **[WBUT 2011]**

a) Tag field is 9 bits and index field is 6 bits
b) Tag field is 4 bits and index field is 10 bits
c) Tag field is 7 bits and index field is 8 bits
d) none of these

Answer: (b)

8. In absence of TLB, to access a physical memory location in a paged-memory system how many memory accesses are required? **[WBUT 2012, 2018, 2019]**

a) 1 **b) 2** **c) 3** **d) 4**

Answer: (b)

9. A direct mapped cache memory with n blocks is nothing but which of the following set associative cache memory organizations? [WBUT 2012, 2015, 2018]
a) 0-way set associative b) 1-way set associative
c) 2-way set associative d) n-way set associative
Answer: (d)

10. In which type of memory mapping there will be conflict miss? [WBUT 2013]
a) direct mapping b) set associative mapping
c) associative mapping d) both (a) & (b)
Answer: (d)

11. Virtual address space can be divided into some fixed size fields of [WBUT 2013, 2019]
a) segments b) blocks c) pages d) none of these
Answer: (c)

12. Which is not the property of a memory module? [WBUT 2013, 2019]
a) inclusion b) consistency c) capability d) locality
Answer: (c)

13. Effective access time $\left(T_{eff}\right)$ of memory is given by [WBUT 2014]
a) $T_{eff} = \dfrac{1}{\sum_{i=1}^{n} f_i t_i}$ b) $T_{eff} = \sum_{i=1}^{n} f_i t_i$
c) $T_{eff} = \sum_{i=1}^{n} \dfrac{f_i}{t_i}$ d) none of these
Answer: (b)

14. The compiler optimization technique is used to reduce [WBUT 2015, 2017, 2018]
a) cache miss penalty b) cache miss rate
c) cache hit time d) none of these
Answer: (b)

15. The cache coherence is a potential problem especially [WBUT 2015, 2018]
a) in asynchronous parallel algorithm execution in multiprocessor
b) in synchronous parallel algorithm execution in multiprocessor
c) in asynchronous parallel algorithm execution in data flow m/c
d) in synchronous parallel algorithm execution in data flow m/c
Answer: (c)

16. A computer with cache access time of 100 ns, a main memory access time of 1000 ns and a hit ratio of 0.9 produces an average access time of [WBUT 2016]
a) 250 ns b) 200 ns c) 190 ns d) 80 ns

Answer: (c)

17. In which type of memory mapping conflict miss occurs? [WBUT 2019]

a) set associative	**b) direct**
c) associative	**d) only (a) & (b)**

Answer: (b)

18. The fastest data access is provided using [WBUT 2019]

a) caches	**b) DRAM's**	**c) SRAM's**	**d) registers**

Answer: (d)

Short Answer Type Questions

1. Consider the performance of a main memory organization, when a cache miss has occurred as [WBUT 2007]

i) 4 clock cycles to send the address
ii) 24 clock cycles for the access time per word
iii) 4 clock cycles to send a word of data.

Estimate:

a) The miss penalty for a cache block of 4 words.
b) The miss penalty for a 4 way interleaved main memory with a cache block of 4 words.

Answer:

To find a single word from main memory, the processor wastes 4+24=28 cycles.
Since, size of the main memory block=size of the cache memory block

To find 4 words in main memory processor access it one time and it takes the whole block to cache.
So, miss penalty = $4\times 28+4=116$

For a 4-way inter leaved memory, the 4 words be present in 4 different banks.
So, the first 4 cycles, all the addresses are activated and the time required to read = 24 cycles.

Let, it requires 4 clock cycle to send a word of data.
So, the total miss penalty = $4+(24+4\times 4)=44$

2. What do you mean by m-way memory interleaving? In the system with pipeline processing, is the memory interleaving useful? If yes, explain why? [WBUT 2008]

Answer:

Interleaving is a technique used to improve the memory performance. Memory interleaving increases bandwidth by allowing simultaneous access of more than one chunk of memory. This improves the performance of the processor because it can transfer more information to / from memory in the same amount of time. It also helps to alleviate the processor-memory bottleneck that is a major limiting factor in overall performance.

Interleaving works by dividing the system memory into multiple blocks. If there are m numbers of blocks then this is called the m-way memory interleaving. In general *two-way* or *four-way* interleaving technique is used. Each block of memory is accessed using different sets of control lines, which are merged together on the memory bus. When a read or write is begun to one block, a read or write to other blocks can be overlapped with the first one.
In an interleaved system, a main memory of size 2^l is divided into *m* modules, where *m* is a positive integer (usually, $m=2^n$ *for* some integer *n* such that $0<n<l$, *l* being the number of bits in a main memory address). Each main memory address is mapped to a module, and to an address within that module. Such a mapping is called a *hashing scheme*. Clearly, the mapping must be one-to-one.

3. A computer has cache access time of 100nanosecs, a main memory access time of 1000 nanosecs and a hit ratio of 0.9.
i) Find the average access time of the memory system.
ii) Suppose that in the computer, there is no cache memory, and then find the average access time, when the main memory access time is 1000 nanosecs. Compare the two access times. [WBUT 2008]

Answer:
i) Suppose the average access time of memory system is
$t_{av} = 100 \times .9 + 1000 \times .1 = 190$ ns

ii) If there is no cache memory, then the average access time is equal to the main memory access time, i.e., 1000ns.
So, with cache memory, the average access time = 190ns and without cache memory, the average access time = 1000ns

4. Consider a computer where the clock per instruction (CPI) is 1.0 when all memory accesses hit (no memory stalls) in the cache. Assume each clock cycle is 2 ns. The only data accesses are loads and stores, and these total 50% of the instructions. Assume the following formula for calculating execution time:
CPU execution time = (CPU clock cycles + Memory stall cycles) × Clock cycle time.
For a program consisting of 100 instructions:

- **Calculate the CPU execution time assuming there are no misses.**
- **Calculate the CPU execution time considering the miss penalty is 25 clock cycles and the miss rate is 2%.**

Discuss the difference between write through and write back cache policies.
[WBUT 2009]

Answer:
Case 1:
If there is no miss then CPU execution time for 100 instructions
$= (1+0) * 2 * 100$ ns $= 200$ ns

Case 2:
The CPI of cache memory = Misses per instruction × Miss penalty

Here miss rate = 2 % and miss penalty = 25 clock cycle
So, The CPI of cache memory = 0.02 × 25 = 0.5
Now, CPU execution time for 100 instructions
= (CPU clock cycles + Memory stall cycles) × Clock cycle time x no. of instructions
= (0.5 + 0) * 2 * 100 = 100

Cache write-through policy and write-back policies:
There are two cache write policies: write-through policy and write-back policy. For write operations, if the word is in the cache, then there may be two copies of the word, one in the cache, and one in main memory. If both are updated simultaneously, this is referred to as write-through policy. i.e., both cache memory and main memory are updated at the same time.

In the write-back policy, the update is performed only in the cache memory. When the cache block is replaced then it updates the main memory contain.

5. Assume the performance of 1-word wide primary memory organization is
- **4 clock cycles to send the address**
- **56 clock cycles for the access time per word**
- **4 clock cycles to send a word of data**

Given a cache block of 4 words, and that a word is 8 bytes, calculate the miss penalty and the effective memory bandwidth.
Re-compute the miss penalty and the memory bandwidth assuming we have
- **Main memory width of 2 words**
- **Main memory width of 4 words**
- **Interleaved main memory with 4 bands with each bank 1-word wide.**

[WBUT 2009]

Answer:
In the first case, total (4+56+4)= 64 clock cycle is required to access 1 word data from memory and 1 word data = 8 bytes.
Memory bandwidth = (8 / 64) bytes per clock cycle
= 0.125 bytes/clock cycle

In the second case, main memory width is 2 words. So, total (4+56+4) = 64 clock cycle is required to access 2 word data from memory and 1 word data = 8 bytes.
Memory bandwidth = (8*2 / 64) bytes per clock cycle
= 0.25 bytes/clock cycle

In the third case, main memory width is 4 words. So, total (4+56+4) = 64 clock cycle is required to access 4 word data from memory and 1 word data = 8 bytes.
Memory bandwidth = (8*4 / 64) bytes per clock cycle = 0.5 bytes/clock
In the fourth case, interleaved main memory has 4 banks and each bank is 1 word wide. So, total (4+56+4) = 64 clock cycle is required to access (1*4) = 4 word data from memory and 1 word data = 8 bytes.
Memory bandwidth = (8*4 / 64) bytes per clock cycle = 0.5 bytes/clock

6. What are logical address and physical address? If segment no. is 8, page no. is 04, word no. is 40. Segment no. 8 hold 30 and page no. 30 hold 019, what will be corresponding physical address? For answer figure is essential. [WBUT 2009]

Answer:

In a system, there are two types of addresses: logical address and physical address. Logical address is generated by CPU and this is the address at which a memory cell or storage element appears to reside from the perspective of an executing application program. The physical address is the address of main memory word which is permanent and cannot change. A logical address may be different from the physical address due to the operation of an address translator or mapping function.

Each segment is divided into a number of equal sized pages. The basic unit of transfer of data between the main memory and the disk is the page, that is, at any given time; the main memory may consist of pages from various segments. In this case, the virtual address is divided into a segment number, a page number, and displacement within the page. Address translation is the same as explained above except that the physical segment base address obtained from the segment table is now added to the virtual page number in order to obtain the appropriate entry in the page table. The output of the page table is the page physical address, which when concatenated with the word field of the virtual address results in the physical address.

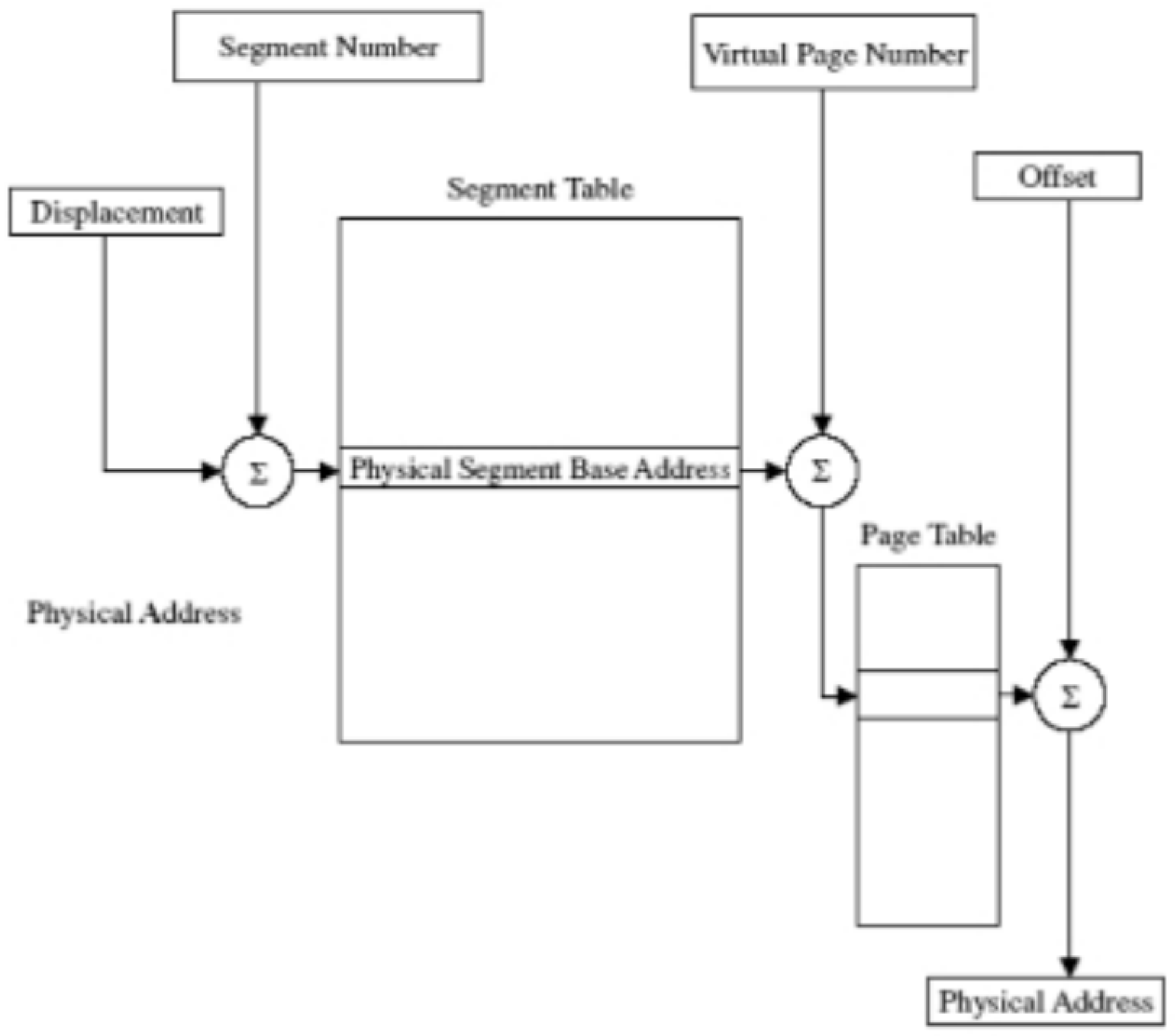


Fig: paged segment address translation

[The above figure shows the paged segment address translation. But there are incomplete data in that problem.]

7. What is the limitation of direct mapping method? Explain with example how it can be improved in set-associative mapping. [WBUT 2009]

Answer:

Direct mapping technique is the simplest technique of cache mapping. It places an incoming main memory block into a specific fixed cache block location. The placement is done based on a fixed relation between the incoming block number, i, the cache block number, j, and the number of cache blocks, N:

$$j = i \bmod N$$

The main disadvantage of Direct mapping technique is the inefficient use of the cache. This is because according to this technique, a number of main memory blocks may compete for a given cache block even if there exist other empty cache blocks. The expected poor utilization of the cache by the direct mapping technique is mainly due to the restriction on the placement of the incoming main memory blocks in the cache i.e., the many-to-one property. This disadvantage should lead to achieving a low cache hit ratio.

The set-associative mapping combines aspects of both direct mapping and associative mapping technique. The set-associative mapping scheme combines the simplicity of direct mapping with the flexibility of associative mapping. In the set-associative mapping technique, the cache is divided into a number of sets. Each set consists of a number of blocks. A given main memory block maps to a specific cache set based on the equation

$$S = i \bmod S$$

Where S is the number of sets in the cache, i is the main memory block number, and s is the specific cache set to which block i maps. So, in this technique, the blocks of main memory are connected to the different sets of the cache memory by direct mapping technique. But in a set there are number of cache blocks and a specific block of main memory may transfer to any block of a specific set of the cache memory by fully associative cache mapping technique. So, the cache replacement is reduced and hit ratio is increased.

8. Describe different techniques to reduce Miss Rate. [WBUT 2010]

OR,

What are the three different types of cache misses? Explain the technique to reduce the cache miss. [WBUT 2018]

OR,

Describe different techniques to reduce cache miss rate. [WBUT 2019]

Answer:

One of the techniques to reduce the cache miss rate is Compiler-controlled prefetch. This may reduce the cache miss rate. While this approach yields better prefetch "hit" rates than hardware prefetch, it does so at the expense of executing more instructions. Thus, the compiler tends to concentrate on prefetching data that are likely to be cache misses anyway. Loops are key targets since they operate over large data spaces and their data accesses can be inferred from the loop index in advance.

Another method of reduce cache miss rate is Compiler optimizations. This method does NOT require any hardware modifications. Yet it can be the most efficient way to eliminate cache misses. The improvement results from better code and data organizations. For example, code can be rearranged to avoid conflicts in a direct-mapped cache, and accesses to arrays can be reordered to operate on blocks of data rather than processing rows of the array.

9. Assume that main memory size is of 32kB×12. Cache memory size is of 512×12 and block size is of 1 word. Describe the following: [WBUT 2010]
a) Direct mapping technique
b) Associative mapping technique.

Answer:

The size of the main memory is 32kb × 12 = $2^{15} \times 12$ bytes
i.e. there are 15 bit address line and 12 bit data bus in the system
The size of the cache memory is 512 × 12 = $2^9 \times 12$ bytes
i.e. there are 9 bit address bus and 12 bit data bus in the system.
The block size is 1 word = 2^4 words in a block.

a) So, in direct mapping technique,
The CPU address = 15 bit
The no. of words in a block = 2^4
The no. of blocks = $2^9/2^4 = 2^5$
No. of bits in Tag field = 15 – 9 = 6 bits

CPU address

Tag Field	Block Field	Word Field
6 bits	5 bits	4 bits

b) In Associative mapping technique
No. of words in a block = 2^4
No. of bits in the Tag field = 15 – 4 = 11bits

CPU address

Tag Field	Word Field
11 bits	4 bits

10. What is the cache coherence problem? Suggest one method to solve this problem. [WBUT 2011]

OR,

What do you mean by cache coherence problem? Describe one method to remove this problem and indicate its limitations. [WBUT 2013, 2016, 2017]

Answer:

A protocol for managing the caches of a multiprocessor system so that no data is lost or overwritten before the data is transferred from a cache to the target memory. When two or more computer processors work together on a single program, known as multiprocessing, each processor may have its own memory cache that is separate from the larger RAM that the individual processors will access. Memory caching is effective because most programs access the same data or instructions over and over. When multiple processors with separate caches share a common memory, it is necessary to keep the caches in a state of coherence by ensuring that any shared operand that is changed in any cache is changed throughout the entire system.

This is done in either of two ways: through a directory-based or a snooping system. In a directory-based system, the data being shared is placed in a common directory that maintains the coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed the directory either updates or invalidates the other caches with that entry. In a snooping system, all caches monitor or snoop the bus to determine if they have a copy of the block of data that is requested on the bus.

11. How does principle of locality help in memory hierarchy design? [WBUT 2012]

OR,

Explain the concept of locality of reference and state its importance to memory systems. [WBUT 2018]

Answer:

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly. Locality occurs often because of the way in which computer programs are created. Generally, related data is stored in nearby locations in storage.

- ***Different Types of Locality:***
  - **Temporal Locality (Locality in Time):** The concept that a memory location that is referenced by a program at one point in time will be referenced again sometime in the near future. If a memory element is referenced, it will tend to be referenced again soon (e.g., loops, reuse). It refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next.
  - **Spatial Locality (Locality in Space):** If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight line code, array access). Spatial locality refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time. The concept that likelihood of referencing a memory location by a program is higher if a memory location near it was just referenced.
  - **Sequential locality:** In typical programs, the execution of instructions follows a sequential order (or the program order) unless branch instructions create out- of-order executions.

12. How is a block chosen for replacement in set-associative cache to resolve a cache miss? **[WBUT 2012]**

Answer:

The cache is divided into a number of sets containing an equal number of blocks. Each block in main memory maps into one set in cache memory similar to that of direct mapping. Within the set, the cache acts as associative mapping where a block can occupy any line within that set. Replacement algorithms may be used within the set. However, an incoming block maps to any block in the assigned cache set. Therefore, the address issued by the processor is divided into three distinct fields. These are the Tag, Set, and Word fields. The Set field is used to uniquely identify the specific cache set that ideally should hold the targeted block. The Tag field uniquely identifies the targeted block within the determined set. The Word field identifies the element (word) within the block that is requested by the processor.

$\longleftarrow$ Main Memory Address $\longrightarrow$

Tag Field	Set Field	Word Field

Memory address field of set-associative mapped cache organization

Having shown the division of the main memory address, we can now proceed to explain the protocol used by the MMU(Memory management unit) to satisfy a request made by the processor for accessing a given element. The steps of the protocol are:

1. Use the Set field to determine the specified set.
2. Use the Tag field to find a match with any of the blocks in the determined set. A match in the tag memory indicates that the specified set determined in step 1 is currently holding the targeted block, that is, a cache hit.
3. Among the words contained in hit cache block, the requested word is selected using a selector with the help of the Word field.

If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory, deposited in the specified set first, and the targeted element (word) is made available to the processor. The cache Tag memory and the cache block memory have to be updated accordingly.

13. A certain program generates the following sequence of word addresses:
4, 5, 12, 8, 10, 28, 6, 10
A page has four words; the number of page frames in main memory is 3. How many page faults are generated if optimum page replacement policy is used? **[WBUT 2013]**

Answer:

Sequence of word address:
4, 5, 12, 8, 10, 28, 6, 10

4	4	4	8	8	8	6	
	5	5	5	10	10	10	
		12	12	12	28	28	

Total page fault is 7 for optimum page replacement policy.

14. A system has 48 bit virtual address, 36 bit physical address and 128 MB main memory address. If the system has 4096 bytes pages, how many virtual and physical pages can have address support? How many page frames of main memory are there? [WBUT 2013]

Answer:

128 MB = 2^{27} Bytes
Physical Address space = 2^{36} Bytes
Page size = 4096 Bytes = 2^{12} Bytes
No. of Physical Pages = $2^{36} / 2^{12} = 2^{24}$
No. of Virtual Pages = $2^{48} / 2^{12} = 2^{36}$
No. of Page Frames in main memory = $2^{27} / 2^{12} = 2^{15}$

15. What is the objective of OPT page replacement algorithm policy of virtual memory? Using LRU, show the page-fault rate for the reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
[WBUT 2013]

Answer:

The page to be replaced is the one that will not be used for the longest period of time. This algorithm requires future knowledge of the reference string which is not usually available.

[Size of the page frame is not given]

We assume that it is 4.

Reference string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7		3		3						3				7		
	0	0	0		0		0						0				0		
		1	1		1		4						1				1		
			2		2		2						2				2		

Total page fault 8.

16. An address space is specified by 28 bits and corresponding memory space of 26 bits. If a page consists of 4K words
i) How many pages and blocks are there in the system?
ii) The associative memory page-table contains the following entries.

Page	Block
0	0
1	1
5	2
6	3

Make a list of all virtual addresses (in decimal and in binary), that will cause a page fault. [WBUT 2013]

Answer:

No. of pages = 228 / 212 = 216 [page size = 4 K = 212]
No. of blocks = 226 / 212 = 214

The list of virtual address that caused page fault are 2, 3, 4 and 7.

17. What is the drawback of direct mapped cache? How is it resolved in set associative cache? [WBUT 2014, 2017]

Answer:

Refer to Question No. 7 of Short Answer Type Questions.

Long Answer Type Questions

1. a) How does the Cache memory effect the throughput of a computer system? [WBUT 2007]

b) Distinguish between Write back and Write through Cache. [WBUT 2007]

OR,

Briefly explain the two write policies: write through and write back with advantages and disadvantages. [WBUT 2013]

c) What effect does memory bandwidth have on the effective memory access time? [WBUT 2007]

d) What is Cache coherence? How can this problem be overcome? [WBUT 2007]

OR,

Briefly describe cache coherence problem with an example. Suggest one software protocol for this. [WBUT 2015]

Answer:

a) In general, throughput is as amount of data transferred from sender to receiver in a specified period of time. Throughput strongly depends on the latency. However, in many cases it can provide better performance than expected by simply dividing cache line size by latency, because many cache lines can be transferred in parallel. A cache memory system immediately supplies data codes to a central processing unit, and new data codes are written from the central processing unit into the cache memory system so as to enhance the hit ratio. The new data codes are transferred to a main memory system while there are no predicted bus requests or communication between the central processing unit, the main memory system, and the cache memory system, so that data throughput is improved without negatively affecting the hit ratio. The performance of cache-based multiprocessors for general-purpose computing and for multitasking is analyzed with simple throughput models. A private cache is associated with each processor, and multiple buses connect the processors to the shared, interleaved memory. Simple models based on dynamic instruction mix statistics are introduced to evaluate upper bounds on the throughput when independent tasks are run on each processor. With these models, one can obtain a first estimate of the MIPS (millions of instructions per second) rate of a multiprocessor.

b) Write-through technique, data is updated both on the cache and in the main memory. If there is a write buffer for main memory and it is empty, information is written into cache and write buffer. CPU continues working while the write buffer writes the word to memory. If the write buffer is full, the cache and the CPU must wait until the buffer is empty.

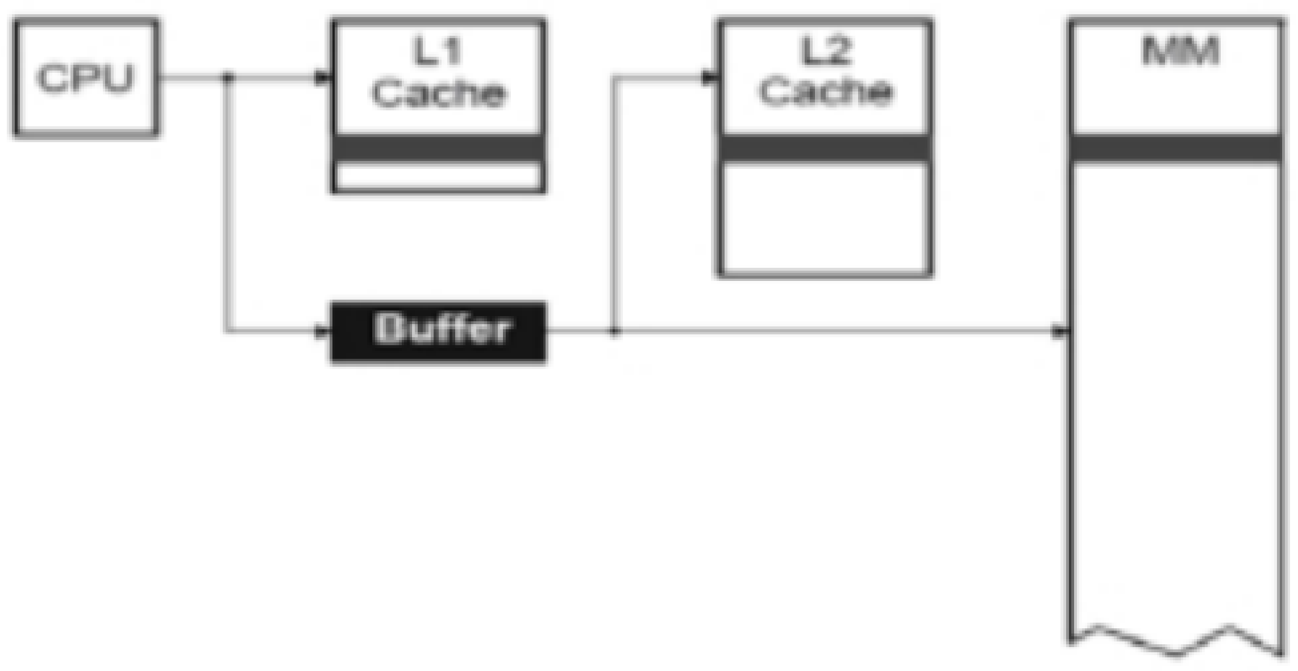


Fig: Write through policy (All the memories have the same copy)

In write-back policy data is written to the cache, and updated in the main memory only when the cache line is replaced. Information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced. This requires an additional information (either hardware or software), called dirty bits. A dirty bit is attached to each tag of the cache. Whenever the information in cache is different from the one in main memory, then write back to main memory.

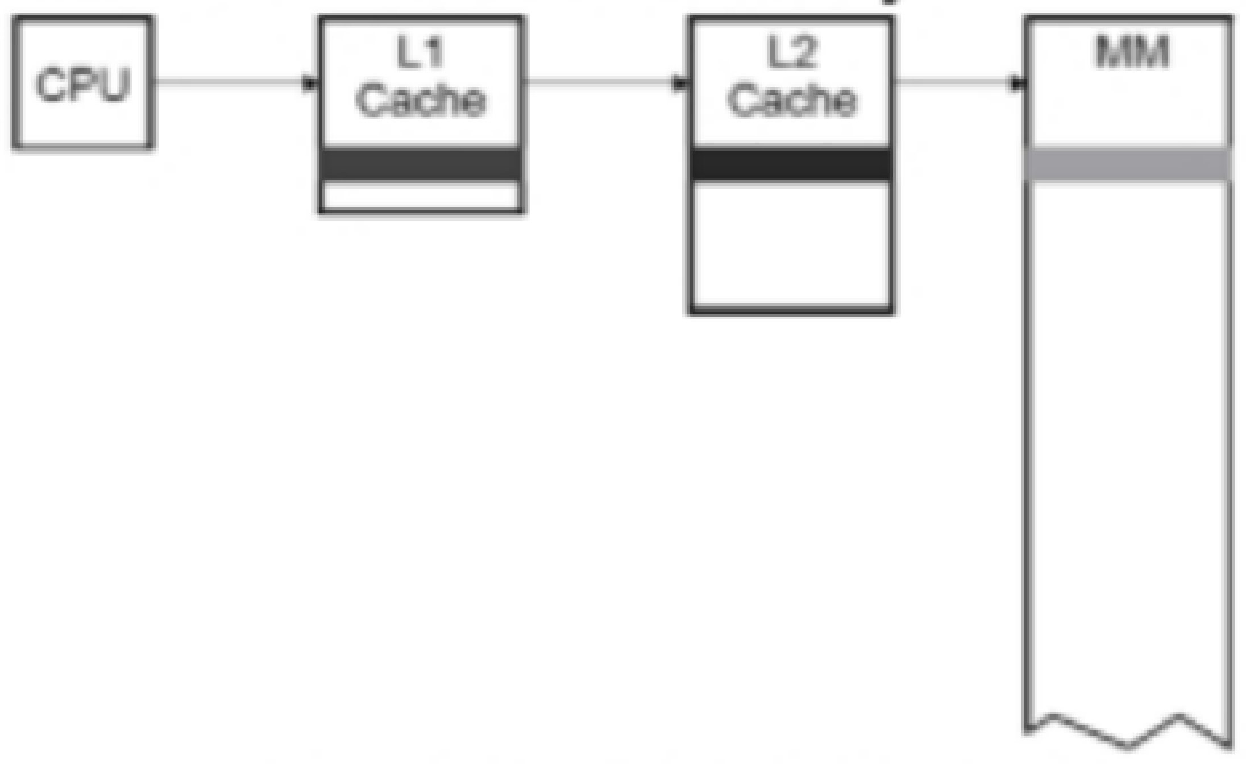


Fig: Write back Policy (All the memories have the same copy)

c) The bandwidth provides a measure of the number of bits per second that can be accessed. The bandwidth b_i refers to the rate at which information is transferred from i-th level to its adjacent levels.

Access time refers to how quickly the memory can respond to a read or write request. The access time t_i refers that the time between request to i-th level of memory and word arrives from that level of memory to the processor.

So, according the above definition it is obvious that if we increase bandwidth the access time will be less. Because increasing bandwidth bit transfer rate will increase.

d) Memory coherence property can manage the memories of a multiprocessor system so that no data is lost or overwritten before the data is transferred from a cache to the target memory. Memory caching is effective because most programs access the same data or instructions over and over. When multiple processors with separate caches share a common memory, it is necessary to keep the caches in a state of coherence by ensuring that any shared operand that is changed in any cache is changed throughout the entire system. This is done by a directory-based system. In a directory-based system, the data being shared is placed in a common directory that maintains the coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed the directory either updates or invalidates the other caches with that entry.

2. a) What is memory Management Unit (MMU)?
b) What are the advantages of using cache memory organization? Define hit ratio. Compare and contrast associative mapping and direct mapping. [WBUT 2008]

Answer:

a) Memory Management Unit (MMU) is the hardware component that manages virtual memory systems. Among the functions of such devices are the translation of virtual addresses to physical addresses, memory protection, cache control, bus arbitration, and, in simpler computer architectures, bank switching. Typically, the MMU is part of the CPU, though in some designs it is a separate chip. The MMU includes a small amount of memory that holds a table matching virtual addresses to physical addresses. This table is called the Translation Look-aside Buffer (TLB). All requests for data are sent to the MMU, which determines whether the data is in RAM or needs to be fetched from the mass storage device. If the data is not in memory, the MMU issues a page fault interrupt.

b) The advantage using a cache of the memory hierarchy is to keep the information expected to be used more frequently by the CPU in the cache. The end result is that at any given time some active portion of the main memory is duplicated in the cache. Therefore, when the processor makes a request for a memory reference, the request is first search in the cache. If the request corresponds to an element that is currently residing in the cache, we call that a cache hit. On the other hand, if the request corresponds to an element that is not currently in the cache, we call that a cache misses.

A cache hit ratio, h_c, is defined as the probability of finding the requested element in the cache. A cache miss ratio $(1 - h_c)$ is defined as the probability of not finding the requested element in the cache. A cache hit ratio, h_c, is defined as the probability of finding the requested element in the cache. A cache miss ratio $(1-h_c)$ is defined as the probability of not finding the requested element in the cache.

$$\textbf{Hit ratio } (\mathbf{h_i}) = \frac{\text{No. of hit}}{\text{No. of hit} + \text{No. of miss}} = \frac{\text{No. of hit}}{\text{Total CPU access}}$$

Compare and contrast associative mapping and direct mapping:
i. In direct mapping technique, each block of main memory is transferred to a fixed block of cache memory by the procedure $j = i \bmod m$. But in associative mapping technique, the blocks of main memory may transfer to any block of cache memory.
ii. Conflict Miss may occur in Direct mapping technique but Capacity Miss may occur in Associative mapping technique.

3. a) What is cache memory? Define global miss & local miss with a suitable example. [WBUT 2010, 2019]
b) Describe different techniques to reduce Miss Penalty. [WBUT 2010, 2019]

OR,

What is cache memory? How does it increase the performance of a computer? What is miss penalty? [WBUT 2018]

Answer:

a) Cache memory is extremely fast memory that is built into a computer's central processing unit (CPU), or located next to it on a separate chip. The CPU uses cache memory to store instructions that are repeatedly required to run programs, improving overall system speed. Most modern desktop and server CPUs have at least three independent caches: an **instruction cache** to speed up executable instruction fetch, a **data cache** to speed up data fetch and store, and a translation look-aside buffer used to speed up virtual-to-physical address translation for both executable instructions and data. When the processor needs to read or write a location in main memory, it first checks whether that memory location is in the cache. This is accomplished by comparing the address of the memory location to all tags in the cache that might contain that address. If the processor finds that the memory location is in the cache, we say that a *cache hit* has occurred; otherwise, we speak of a *cache miss*. In the case of a cache hit, the processor immediately reads or writes the data in the cache line.

Local miss: Local miss means the misses with respect to the memory accesses to this cache memory or we can say that the misses in this cache divided by the total number of memory accesses to this cache memory.

Global miss: Global miss means the misses with respect to the memory access by CPU. i.e. the misses in this cache divided by the total number of memory accesses generated by the CPU.

b) Many techniques to reduce miss penalty affect the CPU. This technique ignores the CPU, concentrating on the interface between the cache and main memory. Adding another level of cache between the original cache and memory simplifies the decision. The first-level cache can be small enough to match the clock cycle time of the fast CPU. Yet the second-level cache can be large enough to capture many accesses that would go to main memory, thereby lessening the effective miss penalty.

Multilevel caches require extra hardware to reduce miss penalty, but not this second technique. It is based on the observation that the CPU normally needs just one word of the block at a time. This strategy is impatience: Don't wait for the full block to be loaded before sending the requested word and restarting the CPU. Here are two specific strategies:

- Critical word first—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Critical-word-first fetch is also called wrapped fetch and requested word first.
- Early restart—Fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution.

Generally these techniques only benefit designs with large cache blocks, since the benefit is low unless blocks are large. The problem is that given spatial locality, there is more than random chance that the next miss is to the remainder of the block.
One another approach to lower miss penalty is to remember what was discarded in case it is needed again. Since the discarded data has already been fetched, it can be used again at small cost. Such "recycling" requires a small, fully associative cache between a cache and its refill path. This *victim cache* contains only blocks that are discarded from a cache because of a miss and are checked on a miss to see if they have the desired data before going to the next lower-level memory. If it is found there, the victim block and cache block are swapped.

4. a) An 8 KB 4-way set associative write back cache is organized as multiple blocks, each of 32-byte size. Assume that the processor generates 36 bits addresses. Calculate the total size of memory required by cache controller to store the tags for cache?
b) What are the approaches to improve miss penalty?
c) A CPU generates 32-bit virtual addresses. The page size is 4 kB. The processor has a TLB which can hold a total is 4 kB. The processor has a TLB which can hold a total of 256 page table entries. The TLB is 8-way set associative. Calculate the TLB tag size. [WBUT 2012]

Answer:
a) Total size of the cache memory = 8 KB = 2^{18} byte
4-way set associative means each set there are 4 blocks i.e. 2^2
Size of the each block is 32 byte i.e. 2^5.
So, the address field contains 5 bit word, 2 bit set and 11 bit tag field.

b) Many techniques to reduce miss penalty affect the CPU. This technique ignores the CPU, concentrating on the interface between the cache and main memory. Adding another level of cache between the original cache and memory simplifies the decision. The first-level cache can be small enough to match the clock cycle time of the fast CPU. Yet the second-level cache can be large enough to capture many accesses that would go to main memory, thereby lessening the effective miss penalty.

Multilevel caches require extra hardware to reduce miss penalty, but not this second technique. It is based on the observation that the CPU normally needs just one word of the block at a time. This strategy is impatience: Don't wait for the full block to be loaded before sending the requested word and restarting the CPU. Here are two specific strategies:
Critical word first—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Critical-word-first fetch is also called wrapped fetch and requested word first.
Early restart—Fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution.

Generally these techniques only benefit designs with large cache blocks, since the benefit is low unless blocks are large. The problem is that given spatial locality, there is more than random chance that the next miss is to the remainder of the block.

One another approach to lower miss penalty is to remember what was discarded in case it is needed again. Since the discarded data has already been fetched, it can be used again at small cost. Such "recycling" requires a small, fully associative cache between a cache and its refill path. This victim cache contains only blocks that are discarded from a cache because of a miss and are checked on a miss to see if they have the desired data before going to the next lower-level memory. If it is found there, the victim block and cache block are swapped.

c) 32 bit virtual address. The page size is 4 KB= 2^{12}
The TLB can hold 256 page table entries i.e. 2^8
The TLB is 8-way set associative i.e. 2^3
So, the size of the tag field = 32 – (12+8+3) = 9

5. What are the major differences between segmentation and Paging? Why is the page size is usually a power of 2? [WBUT 2013]

Answer:

Paging – Computer memory is divided into small partitions that are all the same size and referred to as, page frames. Pages are fixed units of computer memory allocated by the computer for storing and processing information. Paging is a virtual memory scheme which is transparent to the program at the application level. When a process is loaded it gets divided into pages which are the same size as those previous frames. The process pages are then loaded into the frames.

Segmentation – Computer memory is allocated in various sizes (segments) depending on the need for address space by the process. In segmentation, the address space is typically divided into a preset number of segments like data segment (read/write), code segment (read-only), stack(read/write) etc. These segments may be individually protected or shared between processes. Commonly you will see what are called "Segmentation Faults" in programs, this is because the data that about to be read or written is outside the permitted address space of that process.

Recall that paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

6. A computer has 1 KB 4-way set associative cache and 1 MB main memory. If the block size is 64 B, then in which cache set are the words $(ABCDE)_{16}$ and $(EDCBA)_{14}$ mapped? [WBUT 2014]

Answer:
Cache size= 1KB= 2^{10}
Block size= 64 Bytes = 2^6
So, no. of blocks in cache memory= $2^{10} / 2^6 = 2^4$
This is 4-way set associative mapping.
So, no. of sets in cache memory= $2^4 / 2^2 = 4$
The size of main memory = 1 MB= 2^{20}
No. of blocks in main memory = $2^{20} / 2^6 = 2^{14}$.
So, the word $(ABCDE)_{16}$ will be in the block no. $(10995)_{10}$.
[Take the binary value of $(ABCDE)_{16}$ = 10101011110011011110
Consider the first 14 bits as the block no.]
Now, this block will be in the set no. = 10995 mod 4 = 3
Similarly, the word $(EDCBA)_{16}$ will be in the block no. $(15218)_{10}$ and this will be in the set no. = 15218 mod 4 = 2

7. a) What is meant by the cache miss penalty? Briefly discuss "early restart" technique to reduce miss penalty.
b) Let us consider a memory system consisting of main memory and cache memory. In case of a cache miss, assume the performance of the basic memory organization as:

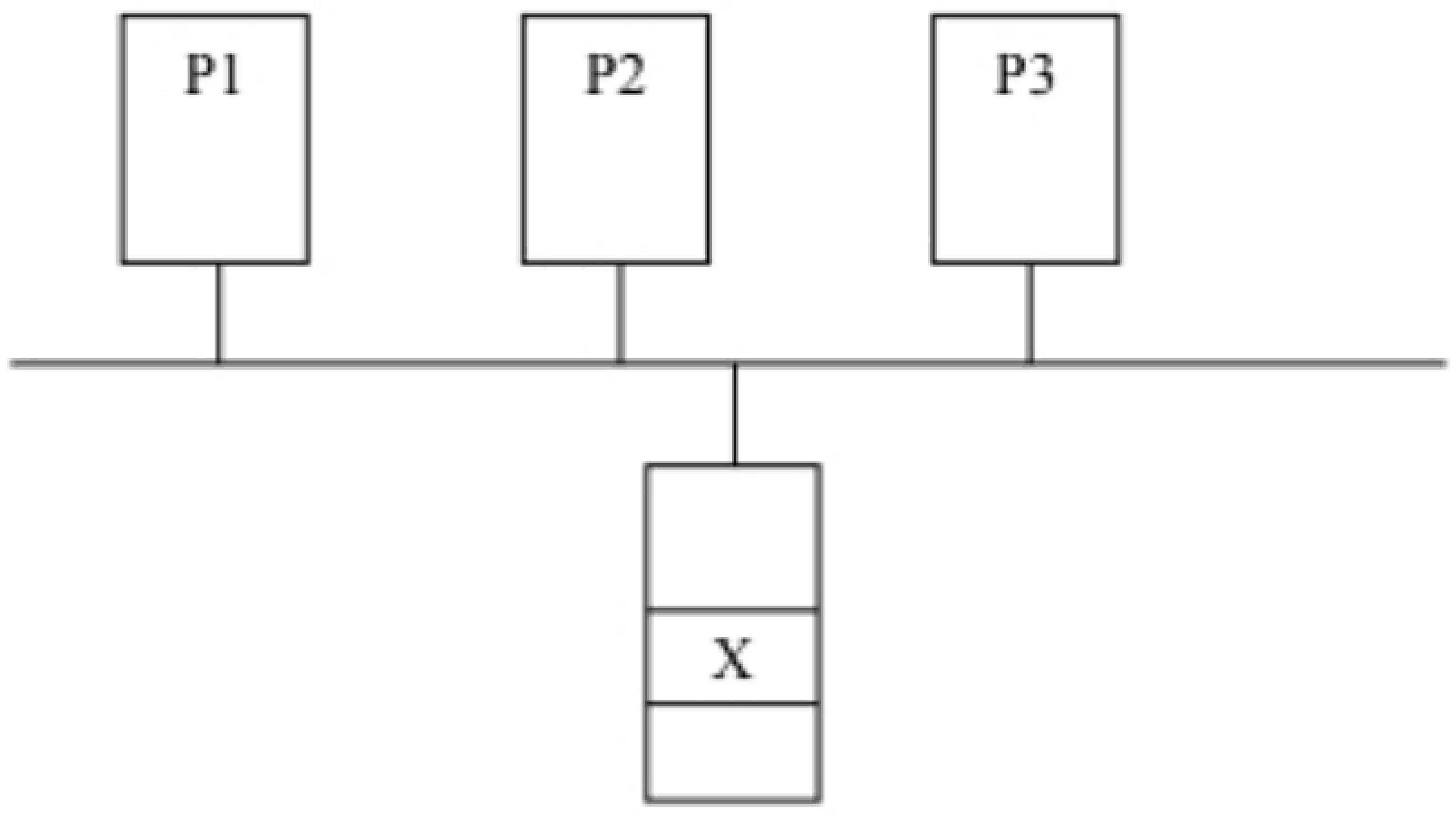
- **4 clock cycles to send the address**
- **24 clock cycles for the access time per word**
- **4 clock cycles to send a word of data.**

i) What will be the miss penalty, given a cache block of four words?
ii) What will be the memory bandwidth? [WBUT 2015]

Answer:
a) ***Refer to Question No. 4 of Long Answer Type Questions.***

b) ***Refer to Question No. 1 of Short Answer Type Questions.***

8. What is the difference between broadcast and invalidate protocols? Explain MESI protocol.



The value of X (shared memory) is 50. P1 and P3 want to read X and store in their cache memories. At t1 time P1 wants to write on X for three times. After that P3 wants to read for two times. After that first P3 writes on X and then P2 wants to read.
Explain the above mentioned scenario using Write through update, Write back update, Write through invalidate. Write back invalidate protocols. [WBUT 2016]

Answer:

1st part:

Difference between broadcast and invalidate protocols:

The performance differences between write broadcast and write invalidate protocols arise from three characteristics:

1. Multiple writes to the same word with no intervening reads require multiple write broadcasts in an update protocol, but only one initial invalidation is in a write invalidate protocol.
2. With multiword cache blocks, each word written in a cache block requires a write broadcast in an update protocol, although only the first write to any word in the block needs to generate an invalidate in an invalidation protocol. An invalidation protocol works on cache blocks, while an update protocol must work on individual words (or bytes, when bytes are written). It is possible to try to merge writes in a write broadcast scheme.
3. The delay between writing a word in one processor and reading the written value in another processor is usually less in a write update scheme, since the written data are immediately updated in the reader's cache.

2nd part:

MESI protocol: The MESI protocol is an Invalidate based cache coherence protocol and is one of the most common protocol which support write-back caches. By using write back caches, we save a lot on bandwidth which is generally wasted on a write through cache. There is always a dirty state present in write back caches which indicates that the data in the cache is different from that in main memory. This protocol reduces the number of Main memory transactions with respect to the MSI protocol. This marks a significant improvement in the performance.

3rd part:

• Write-update: When a local cache block is updated, the new data block is broadcast to all caches containing a copy of the block for updating them. i.e. Every write is observable and every write goes on the bus. i.e. only one write can take place at a time in any processor. So, P1 performs 3 write operations first. Then P3 performs one write operation.

• Write-invalidate: Invalidate all remote copies of cache when a local cache block is updated. The simplest snoopy protocol is the write invalidate protocol based on write through caches. The caches and memory are interconnected by a bus. At the start of the simulation, each processor sends its first request to its cache and, when it receives a response, issues the next request in the succeeding clock period. Any packet sent to the bus is forwarded to all the caches and the memory. The memory only responds to packets sent to it; a packet sent from the memory contains the same source address as the packet it received.

The state of a cache block copy of local processor can take one of the two states:
Valid State:

- All processors can read safely.
- Local processor can also write

Invalid State: (not in cache)

- Block being invalidated.
- Block being replaced
- When a remote processor writes to its cache copy, all other cache copies become invalidated.

9. What do you mean by memory fragmentation? What is the advantage of using Paging? Explain Virtual memory concept with an example where logical address space is 8 kb, physical address space is 4 kb, page size is 1 kb. Explain page fault with FIFO and LRU Algorithm. [WBUT 2016]

Answer:

1st part:

Memory fragmentation occurs when a system contains memory that is technically free but that the computer can't utilize. The memory allocator, which assigns needed memory to various tasks, divides and allocates memory blocks as they are required by programs. When data is deleted, more memory blocks are freed up in the system and added back to the pool of available memory. When the allocator's actions or the restoration of previously occupied memory segments leads to blocks or even bytes of memory that are too small or too isolated to be used by the memory pool, fragmentation has occurred. It means that the memory is divided into parts of fixed size and when some processes try to occupy the memory space, they sometimes are not able to occupy the whole memory leading to some holes in the memory. This is memory fragmentation. It is of 2 types:

1. External fragmentation
2. Internal Fragmentation

2nd part:

Advantage of using Paging:

- Allocating memory is easy and cheap
- Eliminates external fragmentation
- Data (page frames) can be scattered all over Physical Memory
- Pages are mapped appropriately anyway
- Allows demand paging
- More efficient swapping
- No need for considerations about fragmentation
- Just swap out page least likely to be used

3rd part:

The virtual memory concept with logical address space is 8 kb, physical address space is 4 kb, page size is 1 kb is given below:

The size of the page and page frame are same. System loads pages into frames and translates addresses. Now we can represent,

- logical address: va = (p,w)
- physical address: pa = (f,w), where

|p| determines number of pages in VM

|f| determines number of frames in PM

|w| determines page/frame size

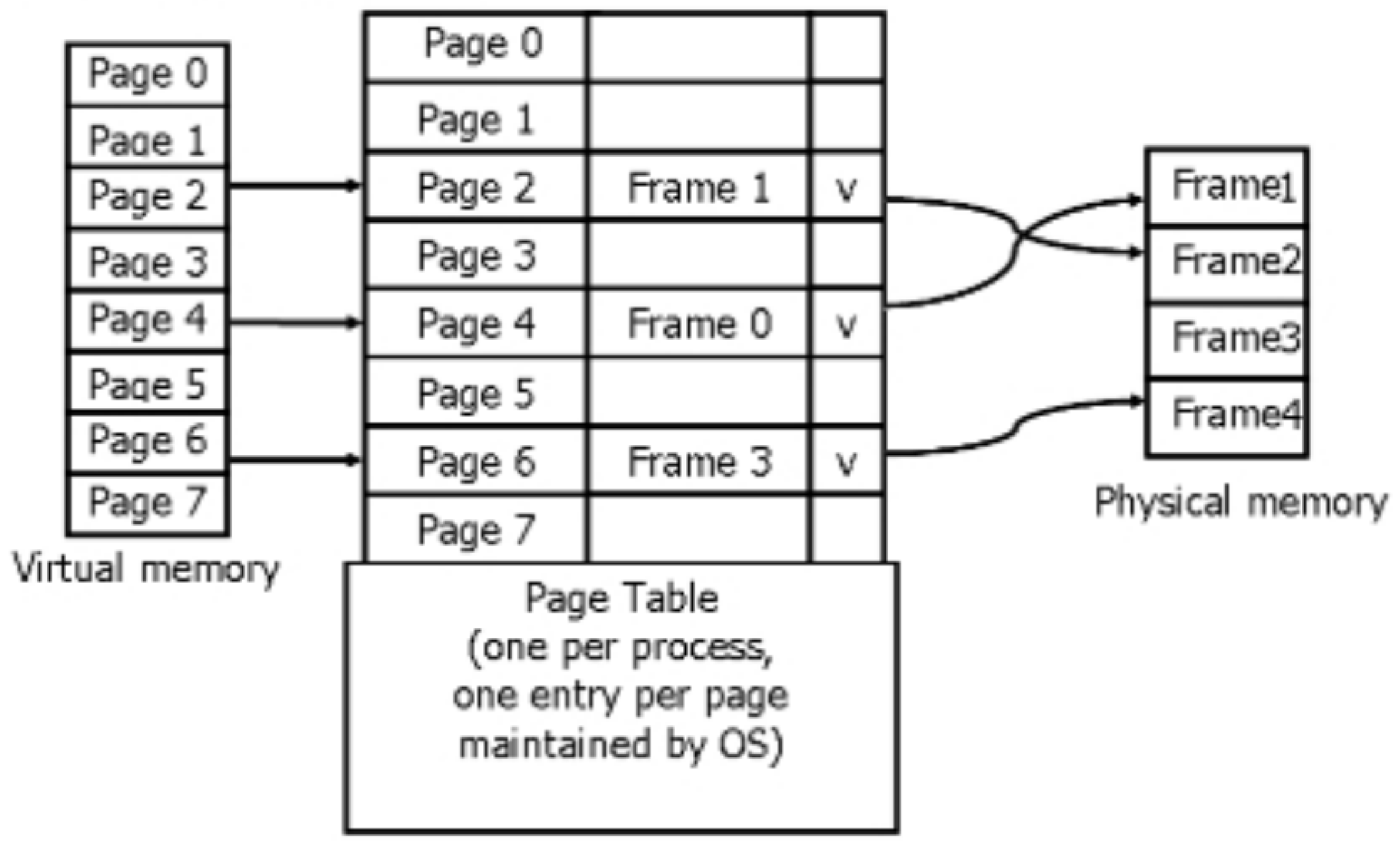


Fig: Mapping between virtual address and physical address

In the above figure, the size of the logical address space is 8 kb, physical address space is 4 kb, page size is 1 kb. So, there are 8 pages in virtual memory and 4 frames in physical memory.

4th part:

FIFO:

FIFO Page Replacement

- A simple and obvious page replacement strategy is ***FIFO***, i.e. first-in-first-out.
- As new pages are brought in, they are added to the tail of a queue, and the page at the head of the queue is the next victim. In the following example, 20 page requests result in 15 page faults:

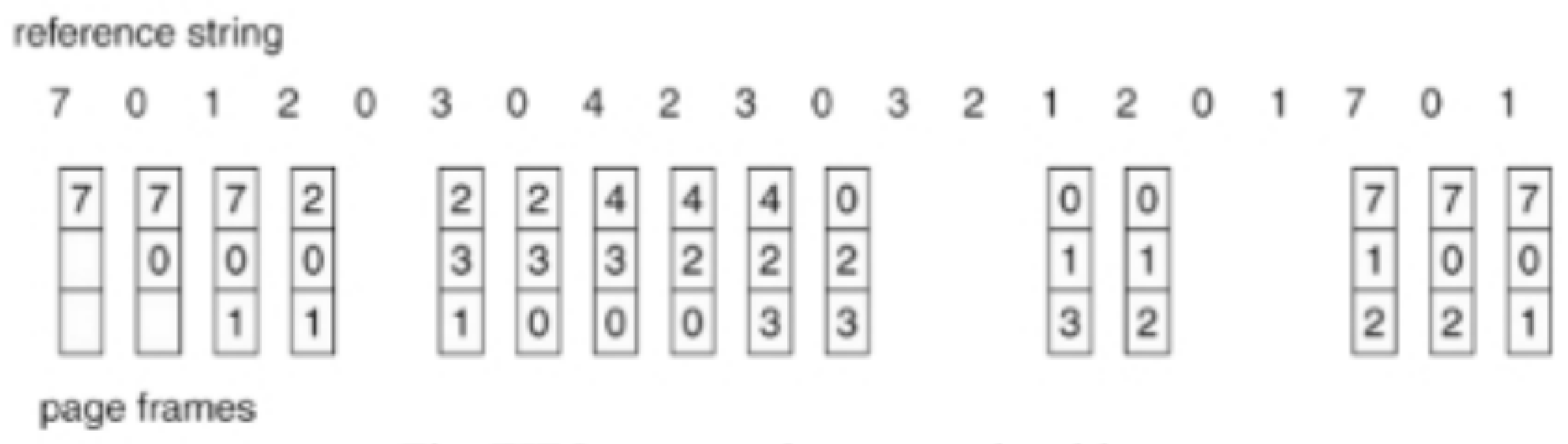


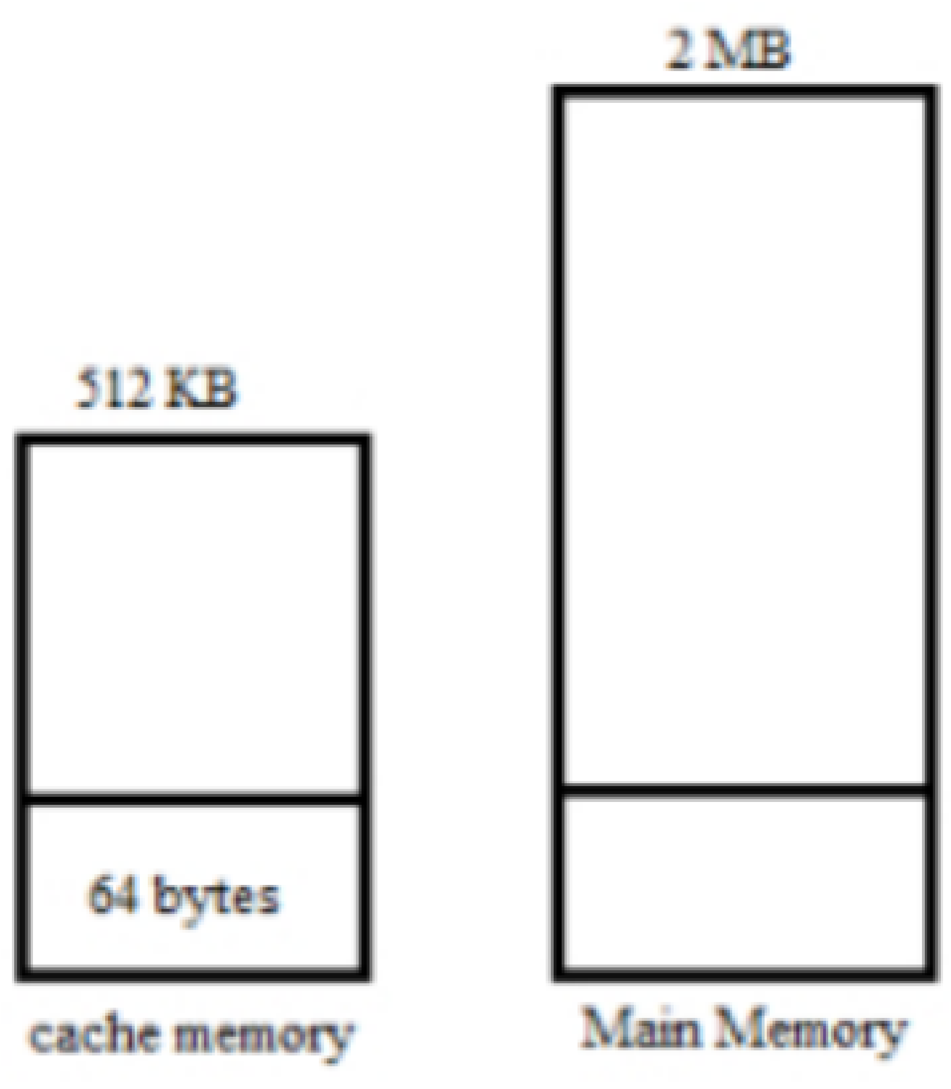
Fig: FIFO page-replacement algorithm

LRU: ***Refer to Question No. 15 of Short Answer Type Questions.***

10. a) A computer has 512kB cache memory and 2MB main memory. If the block size is 64 bytes then find subfield for
i) associative memory
ii) direct mapping
iii) set-associative mapping
b) How does cache memory increase the speed of processing? Explain.
[WBUT 2017]

Answer:

a)

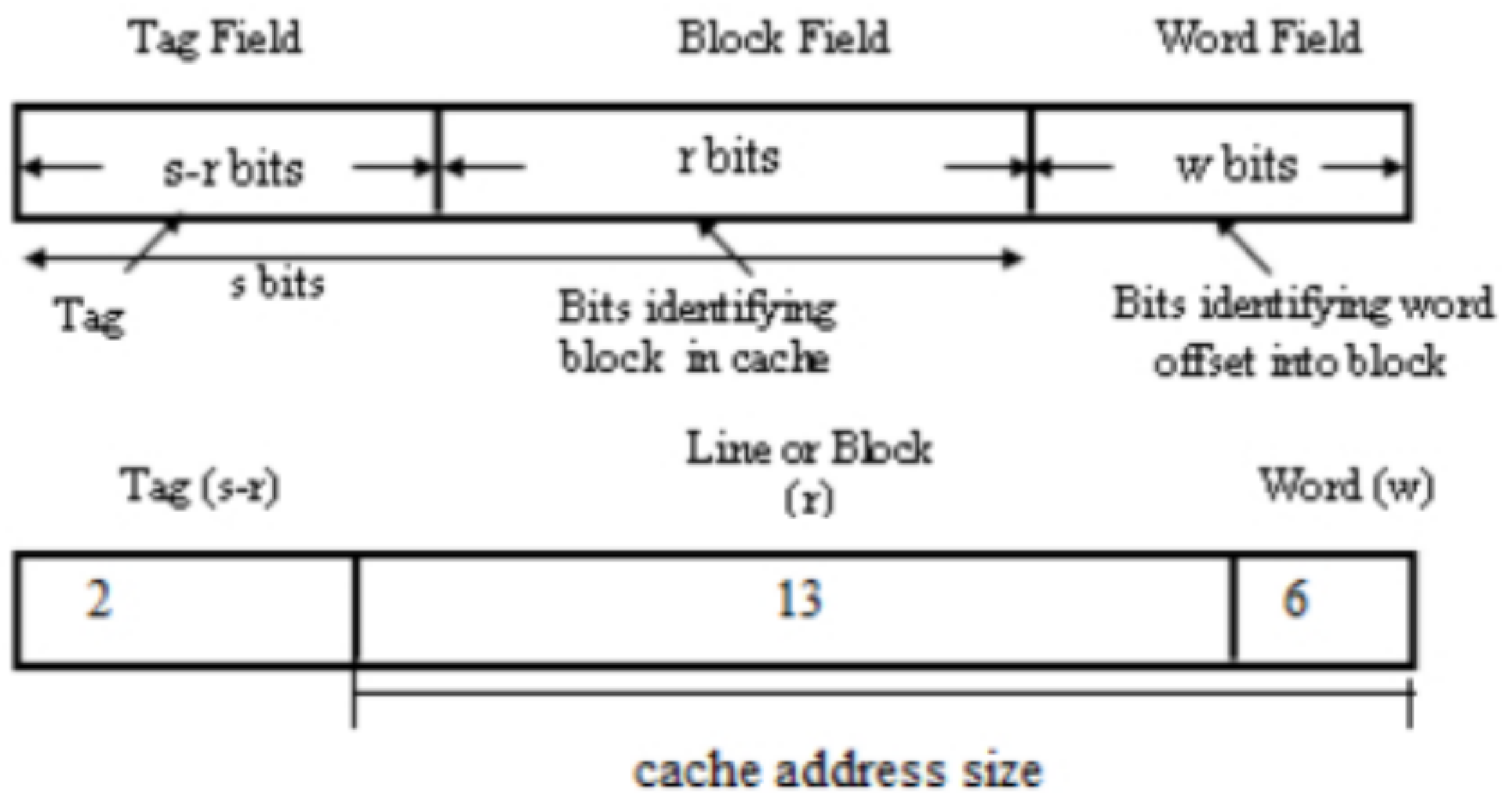


i)

Tag field	Word field
T bits	**W bits**

Tag field	Word field
15 bits	6 bits

ii)



iii) set-associative mapping should be m-way set associative. The value of 'm' may be 4 or 8 or 16 etc. Here no value of m is given. So, the question is wrong.

b) Reduce the cache miss rate, a cache memory can increase the speed of the processing. Three techniques to reduce the miss rate:

- ***Larger cache block size:*** Larger block size will reduce compulsory misses. Assuming that the cache size is constant, a larger block size also reduces the number of blocks. Increases conflict misses. If a program was one long basic block, doubling the block size would result in half as many instruction cache misses. Every new instruction cache block executed would be a compulsory miss (because it was never visited before), but the program would span half as many blocks.

- ***Larger caches:*** Large caches reduce capacity misses. It might increase the hit time. Larger caches take more advantage of temporal locality since cache blocks are not replaced as quickly. This is particularly important for scientific computing data caches that may work on large matrices. Larger caches are slower caches. The miss rate may be reduced at the expense of longer hit time and the size of the cache is proportionally increased with cost. So, this is the limitations of large cache.

- ***Compiler optimization:*** This is an alternative to hardware prefetching. Some CPUs include prefetching instructions. These instructions request that data be moved into either a register or cache. These special instructions can either be faulting or non-faulting. Non-faulting instructions do nothing (no-operation) if the memory access would cause an exception. Of course, prefetching does not help if it interferes with normal CPU memory access or operation. Thus, the cache must be non-blocking. This allows the CPU to overlap execution with the prefetching of data. While this approach yields better prefetch "hit" rates than hardware prefetch, it does so at the expense of executing more instructions. Thus, the compiler tends to concentrate on prefetching data that are likely to be cache misses anyway. Loops are key targets since they operate over large data spaces and their data accesses can be inferred from the loop index in advance. This method does NOT require any hardware modifications. Yet it can be the most efficient way to eliminate cache misses. The improvement results from better code and data organizations. For example, code can be rearranged to avoid conflicts in a direct-mapped cache, and accesses to arrays can be reordered to operate on blocks of data rather than processing rows of the array.

11. A hierarchical cache main memory sub-system has the following specifications:
Cache access time: 50 ns
Main memory access time: 500 ns
80% of memory request for read
Hit ratio: 0.9 for read access and write through scheme is used.
i) Calculate the average access time of the memory system considering only memory read cycle
ii) Calculate the average access time of memory system both read and write cycle.
[WBUT 2017]

Answer:
i) The average access time of the memory system considering only memory read cycle = (50*.9 + 500*.1)*.8 = 76nsec

ii) The average access time of memory system both read and write cycle= (50*.9 + 500*.1) = 95nsec

12. What is cache mapping? What is set associative mapping in cache memory?
[WBUT 2018]

Answer:
Cache mapping is the method by which the contents of main memory are brought into the cache and referenced by the CPU. The mapping method used directly affects the performance of the entire computer system.
Set associative mapping —Set associative cache mapping combines the best of direct and associative cache mapping techniques. As with a direct mapped cache, blocks of main memory data will still map into as specific set, but they can now be in any N-cache block frames within each set.

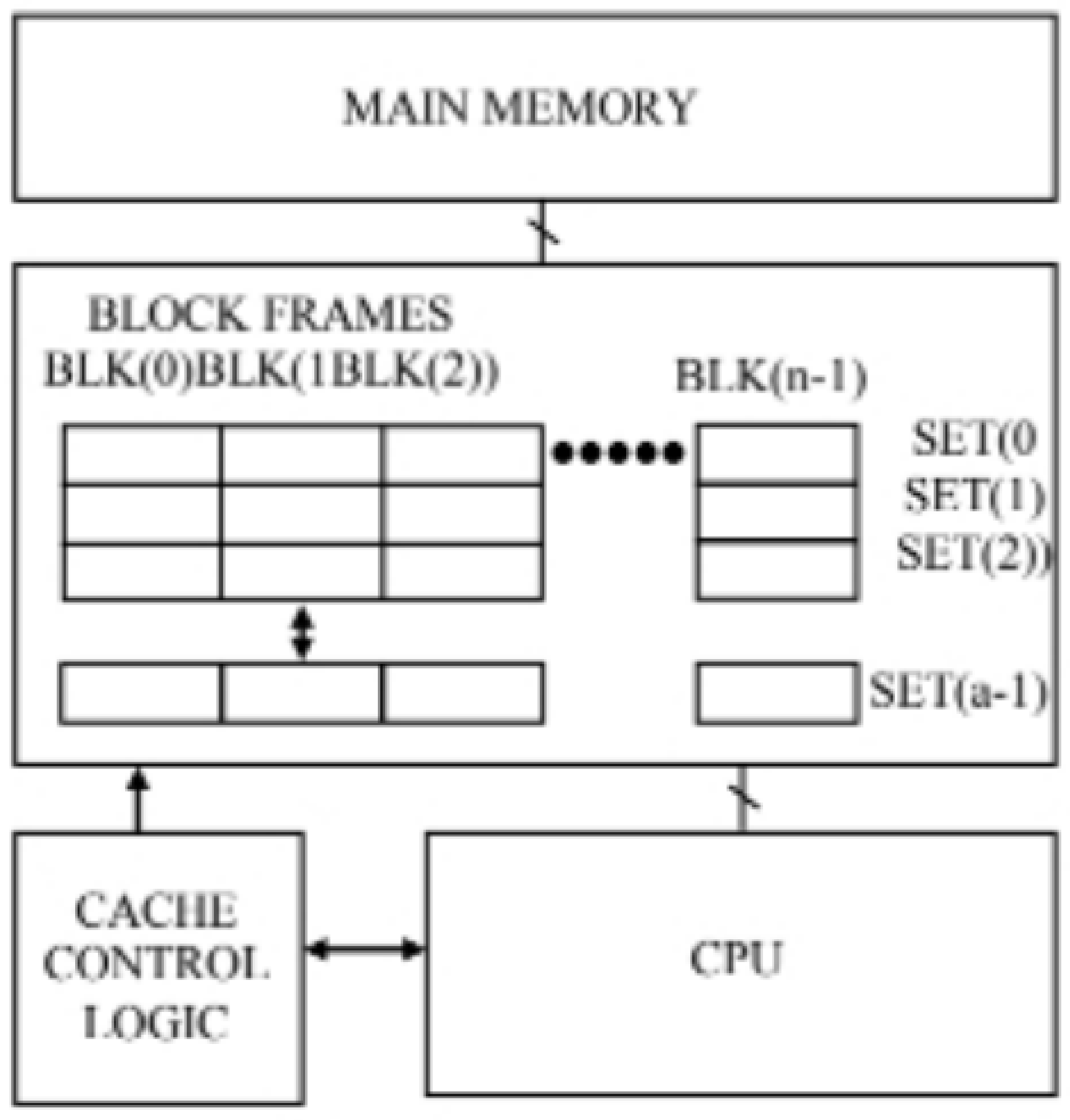


Fig: Example of set association mapping used in cache memory

13. Write short notes on the following:

a) Write through and write back caches	**[WBUT 2011]**
b) Cache coherence problem and its solutions	**[WBUT 2012, 2014, 2018]**
c) Virtual memory	**[WBUT 2019]**
d) TLB	**[WBUT 2019]**
e) Paging	**[WBUT 2019]**

Answer:

a) Write through and write back caches:

Write-through technique, data is updated both on the cache and in the main memory. If there is a write buffer for main memory and it is empty, information is written into cache and write buffer. CPU continues working while the write buffer writes the word to memory. If the write buffer is full, the cache and the CPU must wait until the buffer is empty.

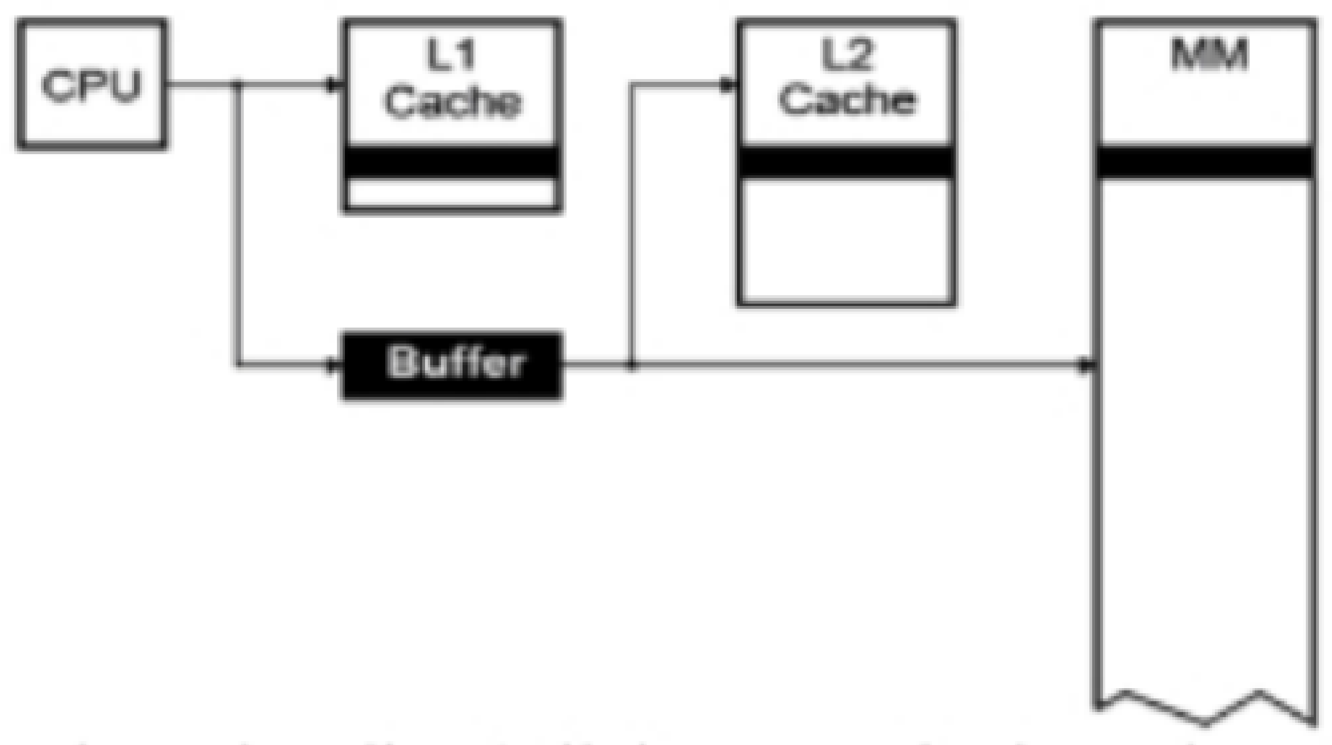


Fig: Write through policy (All the memories have the same copy)

In write-back policy data is written to the cache, and updated in the main memory only when the cache line is replaced. Information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced. This requires an additional information (either hardware or software), called dirty bits. A dirty bit is attached to each tag of the cache. Whenever the information in cache is different from the one in main memory, then write back to main memory.

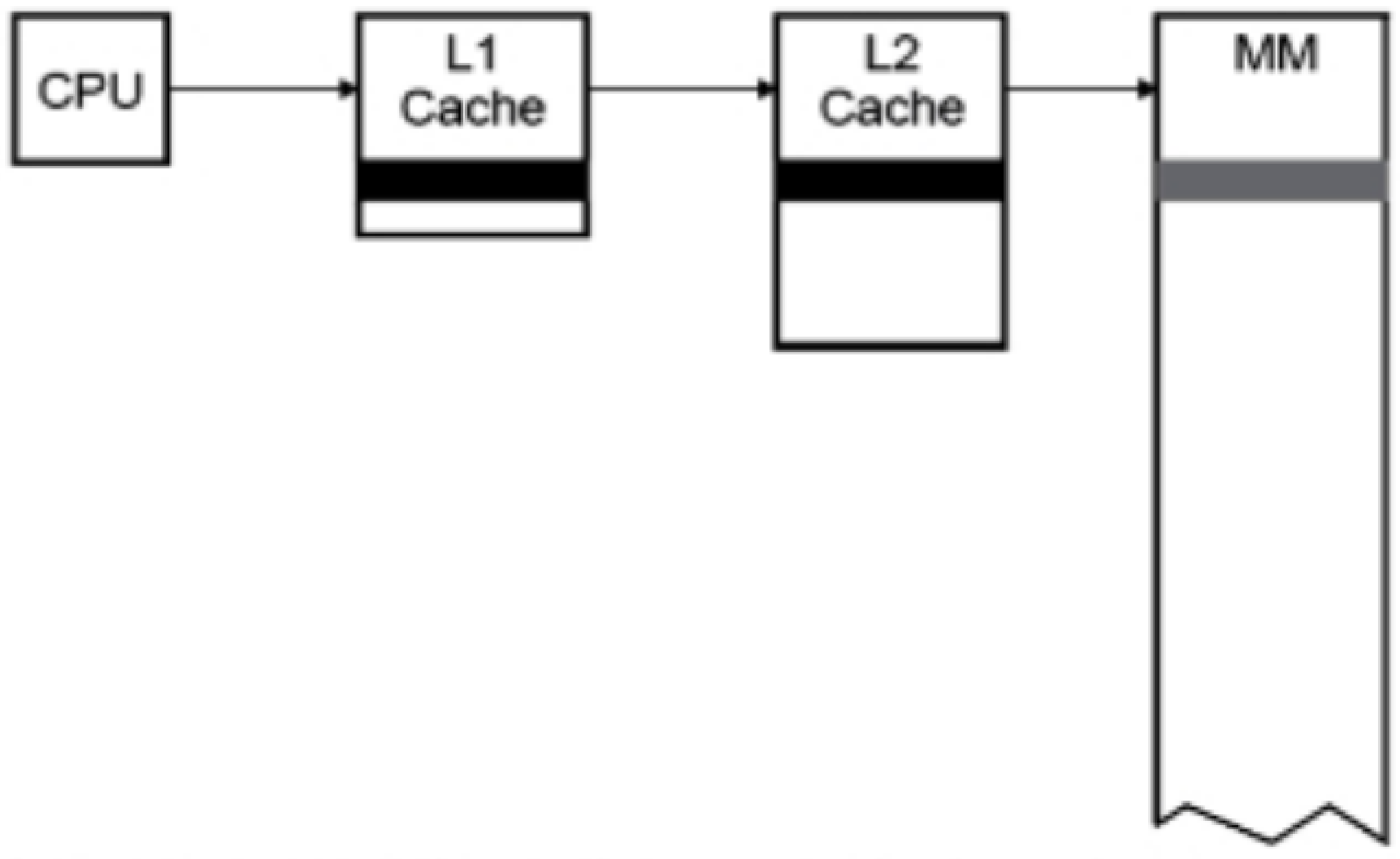
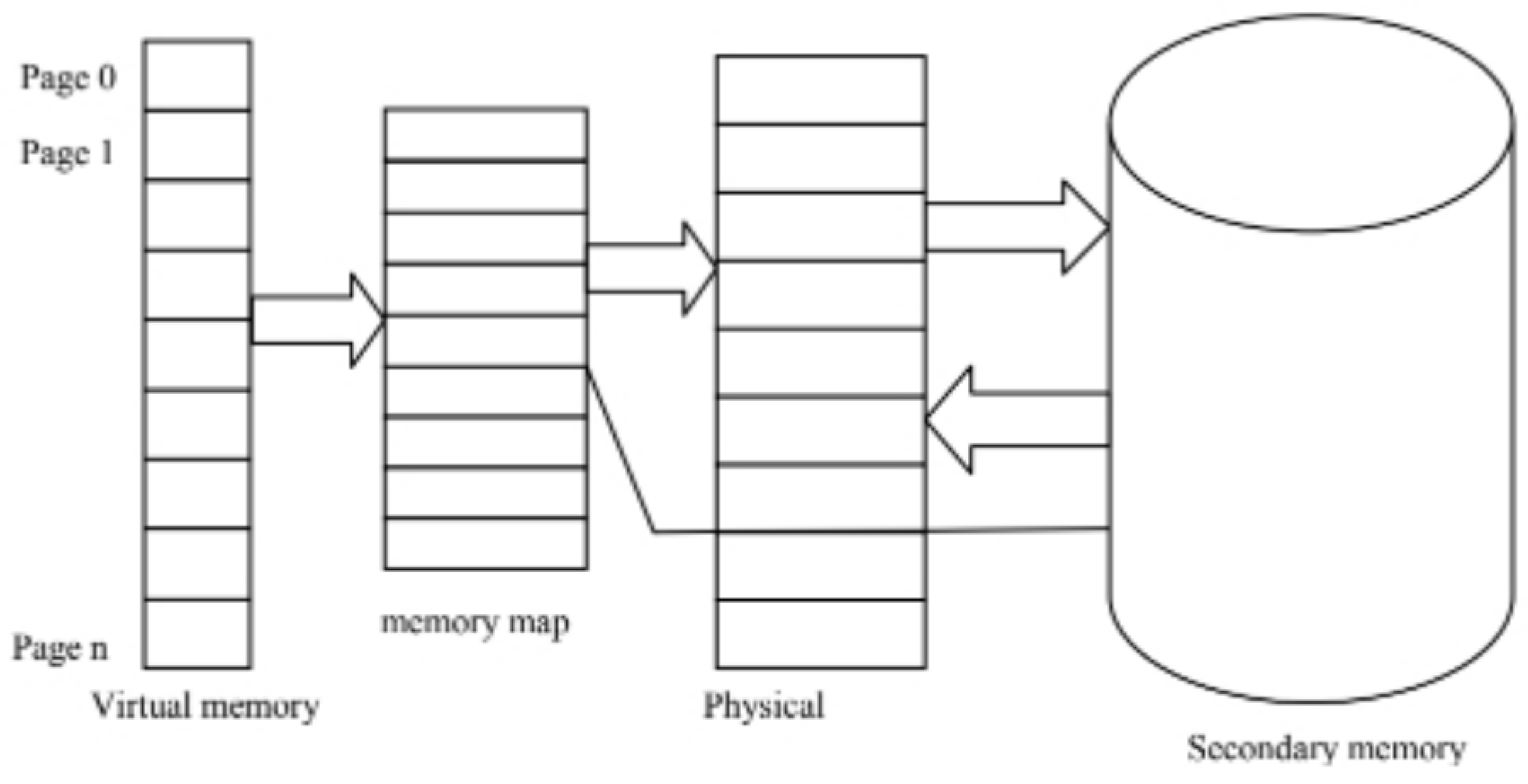


Fig: Write back Policy (All the memories have the same copy)

b) Cache coherence problem and its solutions:
Refer to Question No. 10 of Short Answer Type Questions.

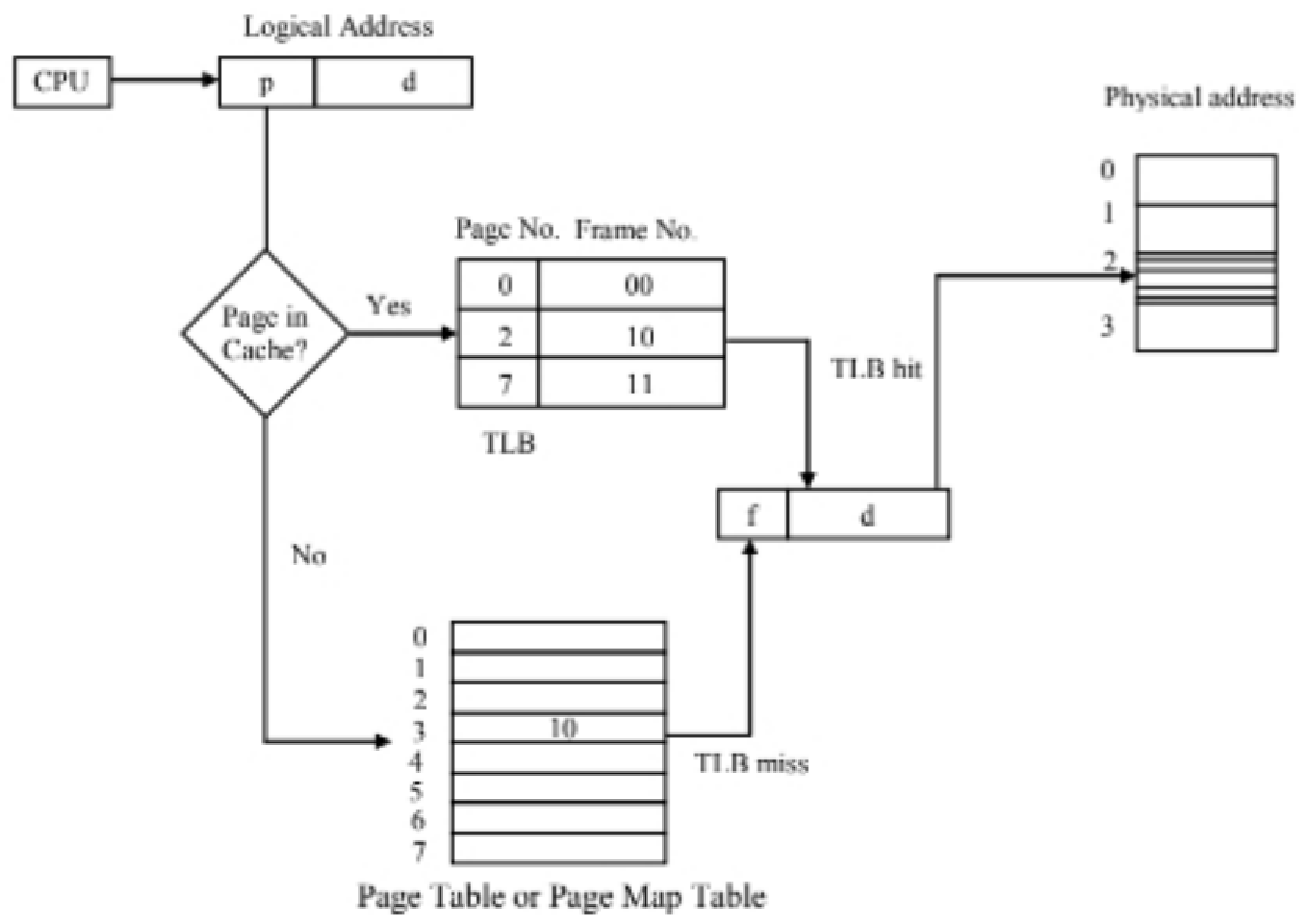
c) Virtual memory:
Virtual memory is a valuable concept in computer architecture that allows you to run large, sophisticated programs on a computer even if it has a relatively small amount of RAM. A computer with virtual memory artfully juggles the conflicting demands of multiple programs within a fixed amount of physical memory. A PC that's low on memory can run the same programs as one with abundant RAM, although more slowly.

Physical vs Virtual Addresses: A computer accesses the contents of its RAM through a system of addresses, which are essentially numbers that locate each byte. Because the amount of memory varies from PC to PC, determining which software will work on a given computer becomes complicated. Virtual memory solves this problem by treating each computer as if it has a large amount of RAM and each program as if it uses the PC exclusively. The OS translates virtual addresses into physical ones, dynamically fitting programs into RAM as it becomes available. Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available. Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.



d) TLB:

In Operating System (Memory Management Technique: Paging, for each process page table will be created, which will contain Page Table Entry (PTE). This PTE will contain information like frame number (The address of main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit etc). This page table entry (PTE) will tell where in the main memory the actual page is residing. Now, the system has to place the page table such that the overall access time (or reference time) will be less. To overcome this problem a high-speed cache is set up for page table entries called a TLB. TLB is nothing but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



Steps in TLB hit:

1. CPU generates virtual address.
2. It is checked in TLB (present).
3. Corresponding frame number is retrieved, which now tells where in the main memory page lies.

Steps in Page miss:

1. CPU generates virtual address.
2. It is checked in TLB (not present).
3. Now the page number is matched to page table residing in main memory.
4. Corresponding frame number is retrieved, which now tells where in the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e., either FIFO, LRU or MFU etc).

e) Paging:

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames. One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes. Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

Drawbacks of Paging:

1. Size of Page table can be very big and therefore it wastes main memory.
2. CPU will take more time to read a single word from the main memory.

How to decrease the page table size:

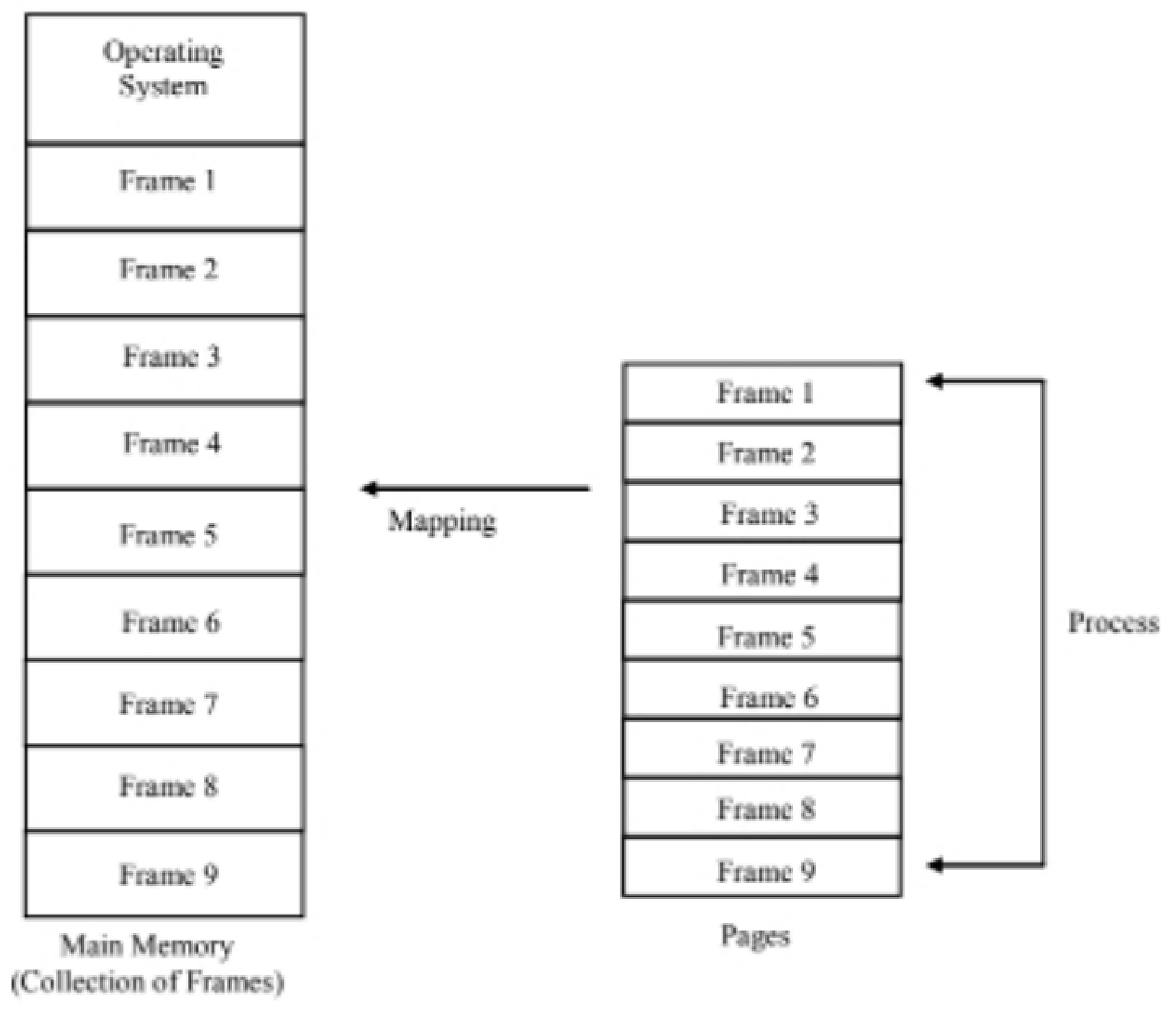
1. The page table size can be decreased by increasing the page size but it will cause internal fragmentation and there will also be page wastage.
2. Other way is to use multilevel paging but that increases the effective access time therefore this is not a practical approach.

How to decrease the effective access time:

1. CPU can use a register having the page table stored inside it so that the access time to access page table can become quite less but the register are not cheaper and they are very small in compare to the page table size therefore, this is also not a practical approach.
2. To overcome these many drawbacks in paging, we have to look for a memory that is cheaper than the register and faster than the main memory so that the time taken by the CPU to access page table again and again can be reduced and it can only focus to access the actual word.

Locality of reference:

In operating systems, the concept of locality of reference states that, instead of loading the entire process in the main memory, OS can load only those number of pages in the main memory that are frequently accessed by the CPU and along with that, the OS can also load only those page table entries which are corresponding to those many pages.



VECTOR PROCESSOR

☞ Chapter at a Glance

Vector Processor

Vector processors are specialized, heavily pipelined processors that perform efficient operations on entire vectors and matrices at once. This class of processor is suited for applications that can benefit from a high degree of parallelism. Register-register vector processors require all operations to use registers as source and destination operands. Memory-memory vector processors allow operands from memory to be routed directly to the arithmetic unit. A vector is a fixed-length, one-dimensional array of values, or an ordered series of scalar quantities. Various arithmetic operations are defined over vectors, including addition, subtraction, and multiplication.

Vector Instructions: There are three different types of vector instructions. They are:

1. ***Vector-vector instruction:*** From different vector registers one or more than one vector operands enter in a functional pipeline unit and result is send to another vector register. This type of vector operation is called vector-vector instruction as shown in the given figure below, where V_a , V_b, V_c are different vector registers and it can define by the following two mapping functions f_1 and f_2.

   $$f_1 : V_a \rightarrow V_b \text{ and } f_2 : V_b \times V_c \rightarrow V_a$$

2. ***Vector-scalar instruction:*** In vector scalar instructions the input operands of the functional unit enter from scalar register and vector register both and produce a vector output as shown in the figure below, where V_a , V_b are different vector registers and S_a is a scalar register. It can also define by the following mapping function f_1.

   $$f_1 : S \times V_a \rightarrow V_b$$

3. ***Vector-Memory instruction:*** vector –memory instruction can be defined by vector load or vector store operations between vector register and memory. it can also defined by the following two mapping functions f_1 and f_2.

   $$f_1 : M \rightarrow V \quad [\text{ vector load }] \text{ and } f_2 : V \rightarrow M \quad [\text{ vector store}]$$

Pipeline Chaining

Pipeline chaining is a linking process that occurs when results obtained from one pipeline unit are directly fed into the operand registers of another functional pipe. In other words, intermediate results do not have to be restored into memory and can be used even before the vector operation is completed. Chaining permits successive operations to be issued as soon as the first result becomes available as an operand. The desired functional pipes and operand registers must be properly reserved; otherwise, chaining operations have to be suspended until the demanded resources become available.

Superscalar, Superpipeline and Superscalar Superpipelined architecture

- Superscalar and super-pipelined processors utilize parallelism to achieve peak performance that can be several times higher than that of conventional scalar processors. In order for this potential to be translated into the speedup of real programs, the compiler must be able to schedule instructions so that the parallel hardware is effectively utilized.
- Superscalar processing has multiple functional units are kept busy by multiple instructions. Current technology has enabled, and at the same time created the need to issue instructions in parallel. As execution pipelines have approached the limits of speed, parallel execution has been required to improve performance.

VLIW Architecture

Very Long Instruction Word (VLIW) refers to a CPU architecture designed to take advantage of instruction level parallelism. A processor that executes every instruction one after the other i.e. a non-pipelined scalar architecture may use processor resources inefficiently, potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously, or even executing multiple instructions entirely simultaneously as in superscalar architectures. The VLIW approach executes operation in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of execution of operations is handled by the compiler.

Multiple Choice Type Questions

1. Which of the following types of instructions are useful in handling sparse vectors or sparse matrices often encountered in practical vector processing application? [WBUT 2007]
a) Vector-Scalar instruction b) Masking instruction
c) Vector-memory instructions d) None of these
Answer: (b)

2. The vector stride value is required [WBUT 2009, 2011, 2018]
a) to deal with the length of vectors b) to find the parallelism in vectors
c) to access the elements in multi-dimensional vectors
d) to execute vector instruction
Answer: (a)

3. Basic difference between Vector and Array processors is [WBUT 2010, 2014]
a) pipelining b) interconnection network c) register d) none of these
Answer: (a)

4. Stride in Vector processor is used to [WBUT 2010, 2014]
a) differentiate different data types b) registers
c) differentiate different data d) none of these
Answer: (c)

5. Array process is present in **[WBUT 2013, 2017]**
a) MIMD **b) MISD** **c) SISD** **d) SIMD**
Answer: (d)

6. The vector stride value is required **[WBUT 2015]**
a) to deal with the length of vectors
b) to find the parallelism in vectors
c) to access the elements in multi-dimensional vectors
d) none of these
Answer: (c)

7. The task of a vectorizing compiler is **[WBUT 2015]**
a) to find the length of vectors
b) to convert sequential scalar instructions into vector instructions
c) to process multi-dimensional vectors
d) to execute vector instructions
Answer: (d)

8. Array processors perform computations to exploit **[WBUT 2015]**
a) temporal parallelism **b) spatial parallelism**
c) sequential behavior of programs **d) modularity of programs**
Answer: (b)

9. In which type of processor array processing is possible? **[WBUT 2019]**
a) SIMD **b) MIMD** **c) MISD** **d) SISD**
Answer: (a)

Short Answer Type Questions

1. How do you speed up memory access in case of vector processing?
[WBUT 2005, 2007]

Answer:
Let r be the vector speed ratio and f be the vectorization ratio. For example, if the time it takes to add a vector of 64 integers using the scalar unit is 10 times the time it takes to do it using the vector unit, then r = 10. Moreover, if the total number of operations in a program is 100 and only 10 of these are scalar (after vectorization), then $f = 90$ (i.e. 90% of the work is done by the vector unit). It follows that the achievable speedup is:
(Time without the vector unit) / Time with the vector unit
In general, the speedup is:
$r / [(1-f)r + f]$

So even if the performance of the vector unit is extremely high (r = oo) we get a speedup less than 1/(1-f), which suggests that the ratio f is crucial to performance since it poses a limit on the attainable speedup. This ratio depends on the efficiency of the compilation. Vector instructions that access memory have a known access pattern. If the vector's elements are all adjacent, then fetching the vector from a set of heavily interleaved memory banks works very well. The high latency of initiating a main memory access versus accessing a cache is amortized, because a single access is initiated for the entire vector rather than to a single word. Thus, the cost of the latency to main memory is seen only once for the entire vector, rather than once for each word of the vector. In this way we can speed up memory access in case of vector processing.

2. Discuss vector instruction format. [WBUT 2006]

OR,

Define the various types of vector instructions. [WBUT 2010, 2014]

OR,

Describe different types of vector instructions. [WBUT 2018]

Answer:

There are three different types of vector instructions basically basis of their mathematical mapping as given below.

Vector-vector instruction: From different vector registers one or more than one vector operands enter in a functional pipeline unit and result is send to another vector register. This type of vector operation is called vector-vector instruction as shown in the given figure below, where Va, Vb, Vc are different vector registers and it can define by the following two mapping functions f1 and f2.

$$f1 : Va \rightarrow Vb \text{ and } f2 : Vb \times Vc \rightarrow Va$$

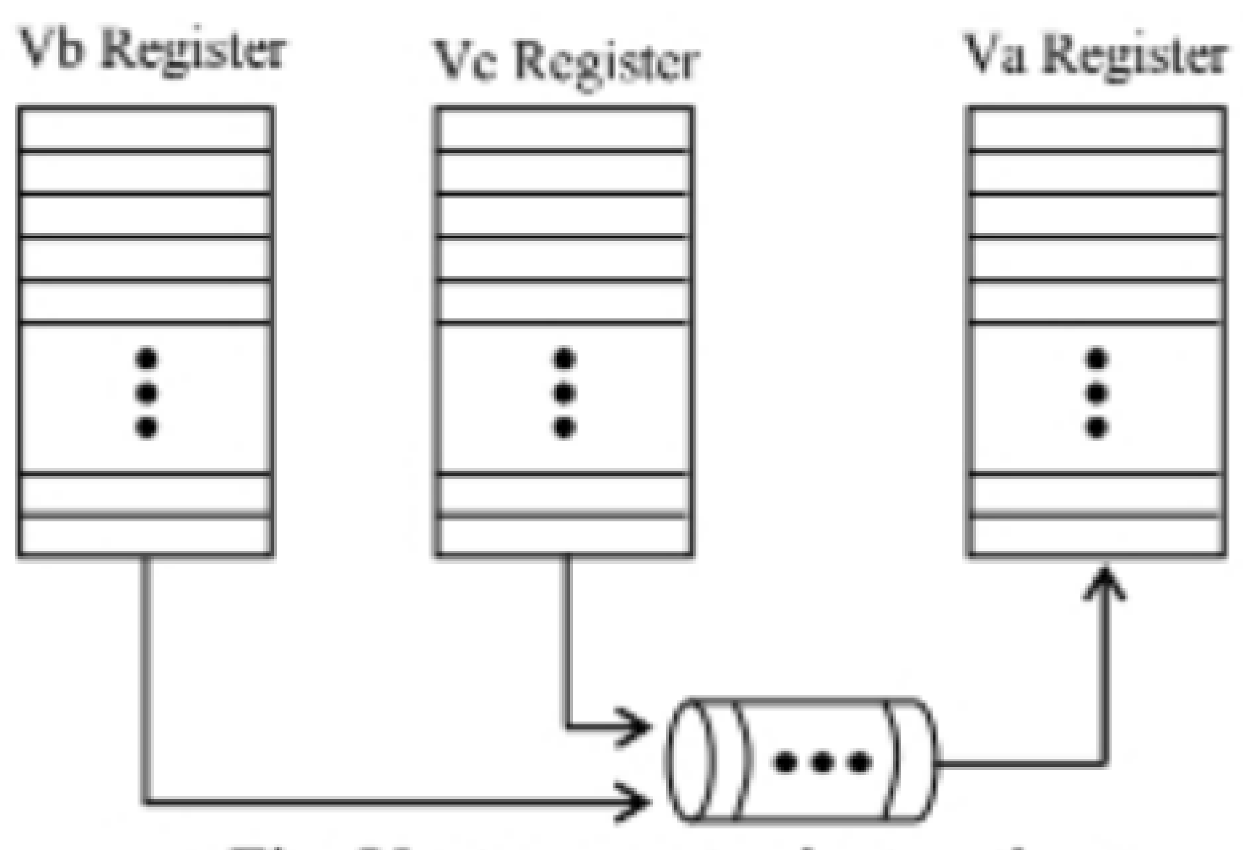


Fig: Vector –vector instruction

Vector-scalar instruction: In vector scalar instructions the input operands of the functional unit enter from scalar register and vector register both and produce a vector output as shown in the figure below, where Va , Vb are different vector registers and Sa is a scalar register. It can also define by the following mapping function f1.

$$f1 : S \times Va \rightarrow Vb$$

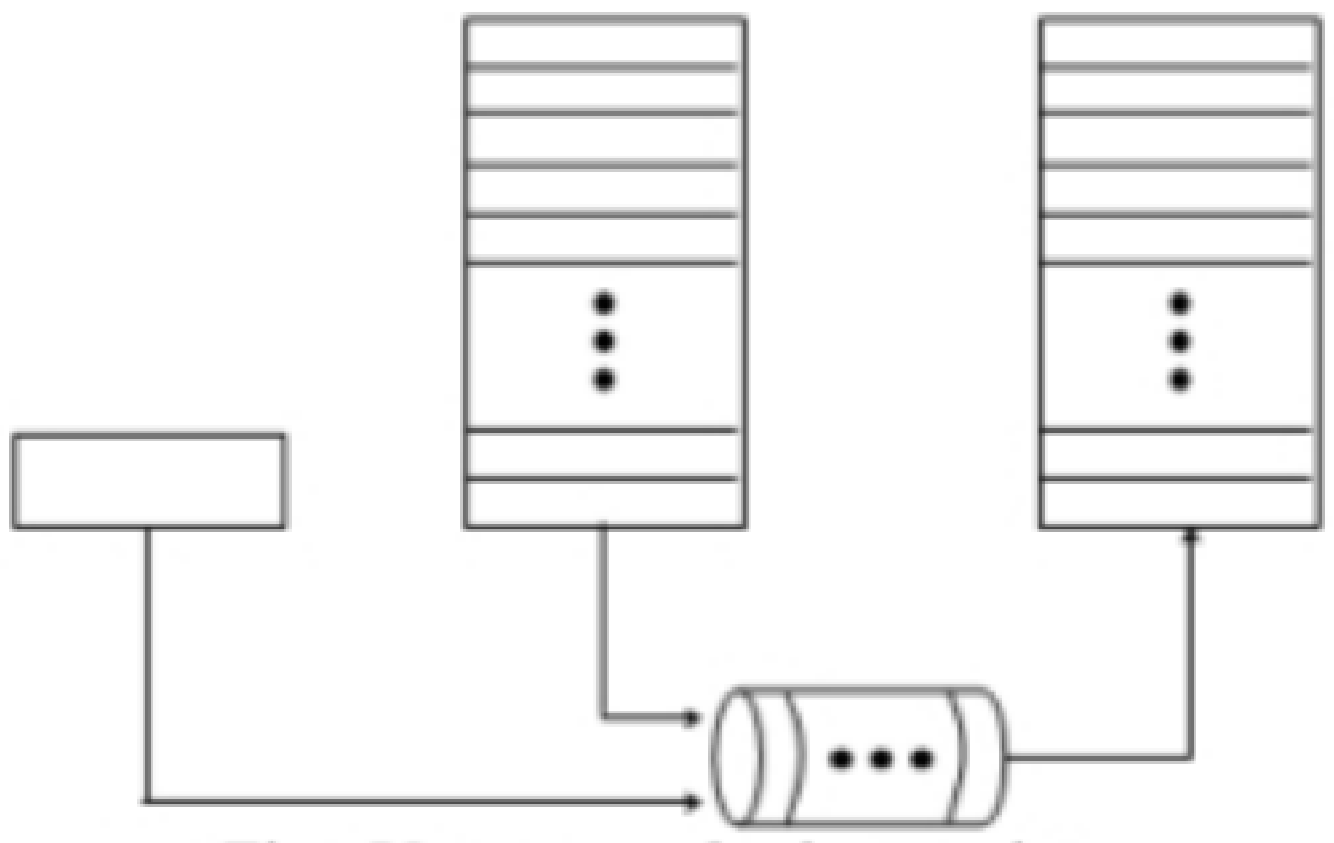


Fig: Vector –scalar instruction

Vector-Memory instruction: vector –memory instruction can be defined by vector load or vector store operations between vector register and memory. it can also defined by the following two mapping functions f1 and f2.

f1 : M → V [vector load] and f2 : V → M [vector store]

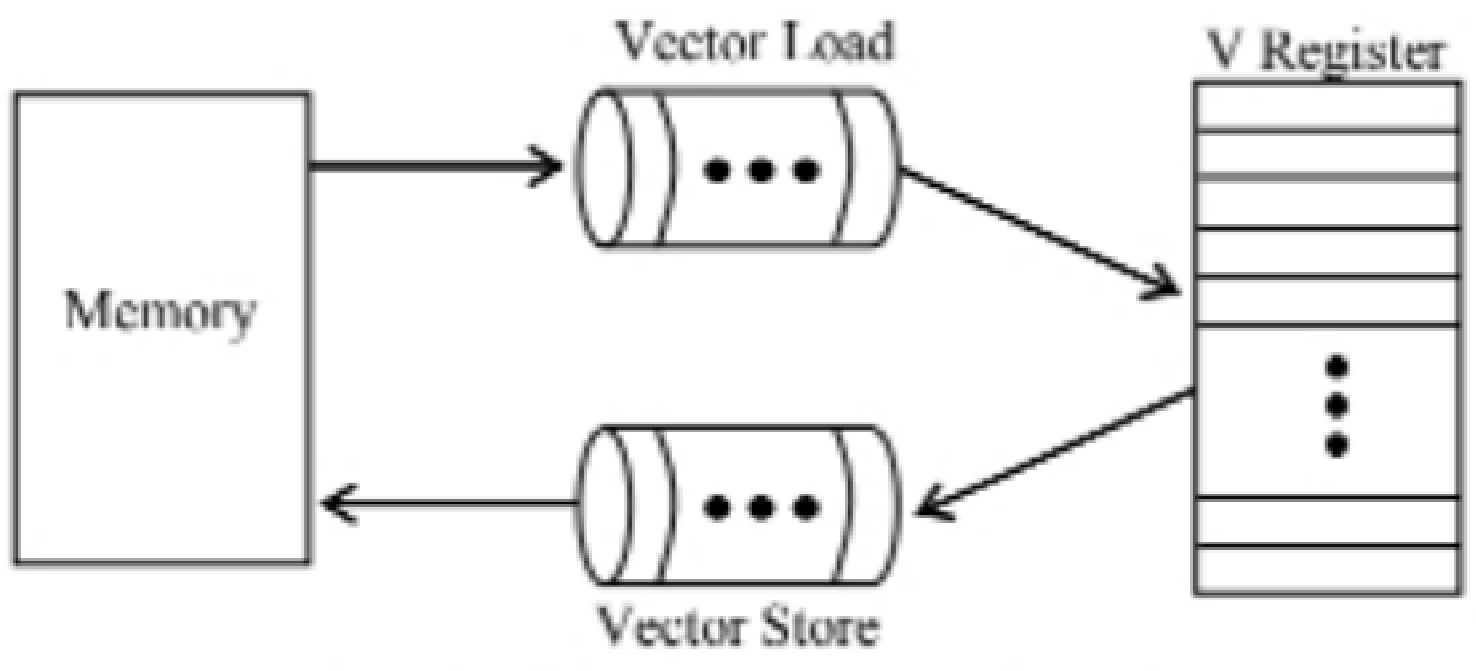


Fig: Vector memory instructions

3. Compare superscalar, superpipeline and superscalar superpipelined architecture. [WBUT 2007]

Answer:

Superscalar and super-pipelined processors utilize parallelism to achieve peak performance that can be several times higher than that of conventional scalar processors. Superscalar machines can issue several instructions per cycle. A system was developed and used to measure instruction-level parallelism for a series of benchmarks. The average degree of super pipelining metric is introduced. Our simulations suggest that this metric is already high for many machines. These machines already exploit all of the instruction-level parallelism available in many non-numeric applications, even without parallel instruction issue or higher degrees of pipelining.

Superscalar

Superscalar processing has multiple functional units are kept busy by multiple instructions. As execution pipelines have approached the limits of speed, parallel execution has been required to improve performance. Super-pipelined machines can issue only one instruction per cycle, but they have cycle times shorter than the latency of any functional unit. In some cases superscalar machines still employ a single fetch-decode-dispatch pipe that drives all of the units. For example, the Ultra SPARC splits execution after the third stage of a unified pipeline. However, it is becoming more common to have multiple fetch-decode-dispatch pipes feeding the functional units. Superscalar operation is limited by the number of independent operations that can be extracted from an instruction stream.

Super-pipeline
Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

- Further divide each of the pipeline stages into n substages.
- Provide n pipelines that are overlapped.

The first approach requires faster logic and the ability to subdivide the stages into segments with uniform latency. The second approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time.

Super-pipelining is limited by the speed of logic, and the frequency of unpredictable branches. Stage time cannot productively grow shorter than the inter stage latch time, and so this is a limit for the number of stages. The MIPS R4000 is sometimes called a super-pipelined machine. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics.

Superscalar-Super-pipeline
We may also combine superscalar operation with super-pipelining and the result is potentially the product of the speedup factors. However, it is even more difficult to interlock between parallel pipes that are divided into many stages. Also, the memory subsystem must be able to sustain a level of instruction throughput corresponding to the total throughput of the multiple pipelines -- stretching the processor/memory performance gap even more. Of course, with so many pipes and so many stages, branch penalties become huge, and branch prediction becomes a serious bottleneck.

But the real problem may be in finding the parallelism required to keep all of the pipes and stages busy between branches. Consider that a machine with 12 pipelines of 20 stages must always have access to a window of 240 instructions that are scheduled so as to avoid all hazards, and that the average of 40 branches that would be present in a block of that size are all correctly predicted sufficiently in advance to avoid stalling in the prefetch unit.

4. Compare superscalar, super-pipeline and VLIW techniques.
[WBUT 2008, 2011, 2014, 2016]

Answer:
Superscalar
A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units on the processor. A superscalar CPU architecture implements a form of parallelism called

instruction-level parallelism within a single processor. Superscalar processing has multiple functional units are kept busy by multiple instructions. In some cases superscalar machines still employ a single fetch-decode-dispatch pipe that drives all of the units.
Superscalar operation is limited by the number of independent operations that can be extracted from an instruction stream. It has been shown in early studies on simpler processor models, that this is limited, mostly by branches, to a small number. The superscalar technique is traditionally associated with several identifying characteristics.

These characteristics are,

- Instructions are issued from a sequential instruction stream
- CPU hardware dynamically checks for data dependencies between instructions at run time (versus software checking at compile time)
- Accepts multiple instructions per clock cycle

Super-pipeline
Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

- Further divide each of the pipeline stages into n substages.
- Provide n pipelines that are overlapped.

The first approach requires faster logic and the ability to subdivide the stages into segments with uniform latency. The second approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time.
Super-pipelining is limited by the speed of logic, and the frequency of unpredictable branches. Stage time cannot productively grow shorter than the inter stage latch time, and so this is a limit for the number of stages. The MIPS R4000 is sometimes called a super-pipelined machine. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics. On more irregular applications, there is little performance advantage.

VLIW
Superscalar and VLIW architectures both exhibit instruction- level parallelism, but differ in their approach. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate. Each functional unit is not a separate CPU core but an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier. Superscalar CPU design emphasizes improving the instruction dispatcher accuracy, and allowing it to keep the multiple functional units in use at all times.

Very Long Instruction Word (VLIW) refers to a CPU architecture designed to take advantage of instruction level parallelism. A processor that executes every instruction one after the other i.e. a non-pipelined scalar architecture may use processor resources inefficiently, potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously, or even executing multiple instructions entirely simultaneously as in superscalar architectures.

The VLIW approach executes operation in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of execution of operations is handled by the compiler.

5. Discuss about strip mining and vector stride in vector processors.
[WBUT 2008, 2012]

Answer:

Vector lengths do not often correspond to the length of the vector registers. For shorter vectors, we can use a vector length register applied to each vector operation. If a vector to be processed has a length greater than that of the vector registers, then **strip-mining** is used, whereby the original vector is divided into equal size segments i.e. equal to the size of the vector registers and these segments are processed in sequence. The process of strip-mining is usually performed by the compiler but in some architecture it could be done by the hardware. The strip-mined loop consists of a sequence of convoys.
The vector elements are ordered to have a fixed addressing increment between successive elements called as stride or skew distance. i.e. It is the distance separating elements in memory that will be adjacent in a vector register. The value of the stride could be different for different variable. When a vector is loaded into a vector register then the stride is 1, meaning that all the elements of vector are adjacent. Non-unit strides can cause major problems for the memory system, which is based on unit stride (i.e. all the elements are one after another in different interleaved memory banks). Caches deal with unit stride, and behave badly for non-unit stride. To account for non-unit stride, most systems have a stride register that the memory system can use for loading elements of a vector register. However, the memory interleaving may not support rapid loading. The vector strides technique is used when the elements of vectors are not adjacent.

6. What is vector processor? Give the block diagram to indicate the architecture of a typical Vector Processor with multiple function pipes.
[WBUT 2008, 2010-short note]

Answer:

Vector processors are specialized, heavily pipelined processors that perform efficient operations on entire vectors and matrices at once. This class of processor is suited for applications that can benefit from a high degree of parallelism. Register-register vector processors require all operations to use registers as source and destination operands. Memory-memory vector processors allow operands from memory to be routed directly to the arithmetic unit. A vector is a fixed-length, one-dimensional array of values, or an ordered series of scalar quantities. Various arithmetic operations are defined over vectors, including addition, subtraction, and multiplication.

A vector processor includes a set of vector registers for storing data to be used in the execution of instructions and a vector functional unit coupled to the vector registers for executing instructions. The functional unit executes instructions using operation codes provided to it which operation codes include a field referencing a special register. The special register contains information about the length and starting point for each vector instruction. A series of new instructions to enable rapid handling of image pixel data are provided.

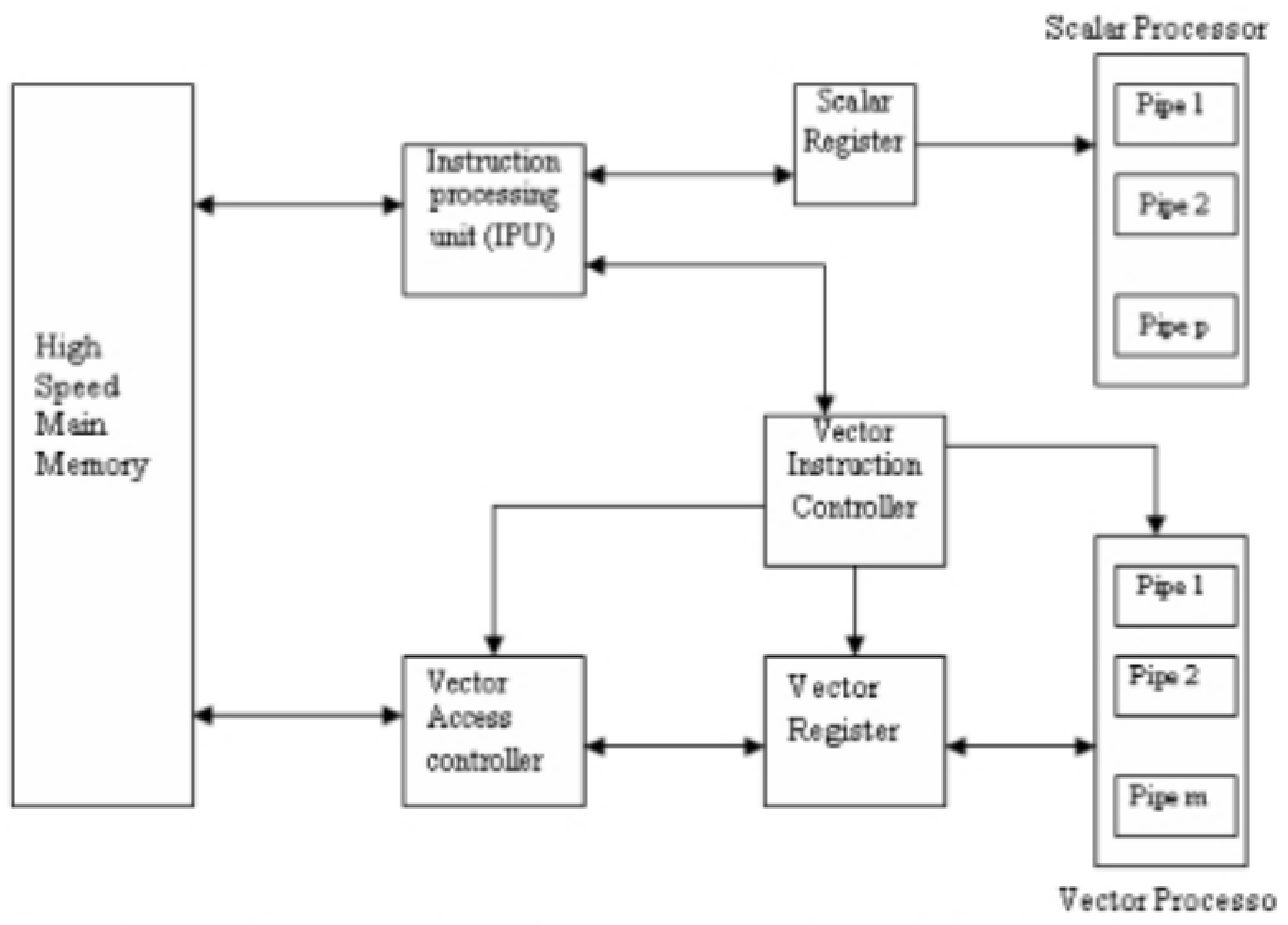


Fig: The architecture of a vector processor with multiple function pipes

7. Explain the concept of strip mining used in vector processors. Why do vector processors use memory banks? [WBUT 2009]

Answer:

When a vector has a length greater than that of the vector registers then segmentation of the long vector into fixed length segments is necessary. This technique is called strip-mining. One vector segment is processed at a time. As an example, the vector segment length is 64 elements in Cray computers. Until the entire vector elements in each segment are processed, the vector register cannot be assigned to another vector operation. Strip-mining is restricted by the number of available vector, registers and vector chaining.

To allow faster access to vector elements stored in memory, the memory of a vector processor is often divided into memory banks. Interleaved memory banks associate successive memory addresses with successive banks cyclically. One memory access (load or store) of a data value in a memory bank takes several clock cycles to complete. Each memory bank allows only one data value to be read or stored in a single memory access. But more than one memory bank may be accessed at the same time. When the elements of a vector stored in an interleaved memory are read into a vector register, the reads are staggered across the memory banks so that one vector element is read from a bank per clock cycle. If one memory access takes n clock cycles, then n elements of a vector may be fetched at a cost of one memory access. This is n times faster than the same number of memory accesses to a single bank.

8. What is Vector array processor? Explain with example. [WBUT 2009]

Answer:
A vector processor is a processor that can operate on entire vectors with one instruction, i.e. the operands of some instructions specify complete vectors. For example, consider the following add instruction:

$$C = A + B$$

In both scalar and vector machines this means "add the contents of A to the contents of B and put the sum in C." In a scalar machine the operands are numbers, but in vector processors the operands are vectors and the instruction directs the machine to compute the pair-wise sum of each pair of vector elements. A processor register, usually called the vector length register, tells the processor how many individual additions to perform when it adds the vectors. A key division of vector processors arises from the way the instructions access their operands. In the *memory to memory* organization the operands are fetched from memory and routed directly to the functional unit. Results are streamed back out to memory as the operation proceeds. In the *register to register* organization operands are first loaded into a set of *vector registers*, each of which can hold a segment of a register, for example 64 elements. The vector operation then proceeds by fetching the operands from the vector registers and returning the results to a vector register.

9. Discuss different types of vector instruction. [WBUT 2011]

Answer:
There are three different types of vector instructions basically basis of their mathematical mapping as given below.

1. ***Vector-vector instruction:*** From different vector registers one or more than one vector operands enter in a functional pipeline unit and result is send to another vector register. This type of vector operation is called vector-vector instruction as shown in the given figure below, where V_a , V_b, V_c are different vector registers and it can define by the following two mapping functions f_1 and f_2.
   $f_1 : V_a \rightarrow V_b$ and $f_2 : V_b \text{ x } V_c \rightarrow V_a$
2. ***Vector-scalar instruction:*** In vector scalar instructions the input operands of the functional unit enter from scalar register and vector register both and produce a vector output as shown in the figure below, where V_a , V_b are different vector registers and S_a is a scalar register. It can also define by the following mapping function f_1.
   $f_1 : S \text{ x } V_a \rightarrow V_b$
3. ***Vector-Memory instruction:*** vector –memory instruction can be defined by vector load or vector store operations between vector register and memory. it can also defined by the following two mapping functions f_1 and f_2.
   $f_1 : M \rightarrow V$ [vector load] and $f_2 : V \rightarrow M$ [vector store]

10. What is vector chaining? How can it speedup the processing? Explain with suitable example. [WBUT 2018]

Answer:
In computing, chaining is a technique used in computer architecture in which scalar and vector registers generate interim results which can be used immediately, without additional memory references which reduce computational speed. i.e. in chaining data forwarding from one vector functional unit to another unit without waiting for completing the previous instruction.

Without chaining, the system must wait for last element of result to be written before starting dependent instruction as shown in fig. 1(a). It takes three unit of time for 'Load', 'Mul' and 'Add' instructions. But with chaining technique, the system can start dependent instruction as soon as first result appears as shown in fig. 1(b). The pipelined vector processors have an optimization called chaining: When a vector load operation executes, whose result is used by a vector multiply, and that result by a vector add, the machine would not wait with the vector multiply until the vector load is finished, but pass the first value of the vector right from the load unit to the multiply unit, and the first value of that result directly to the add unit.

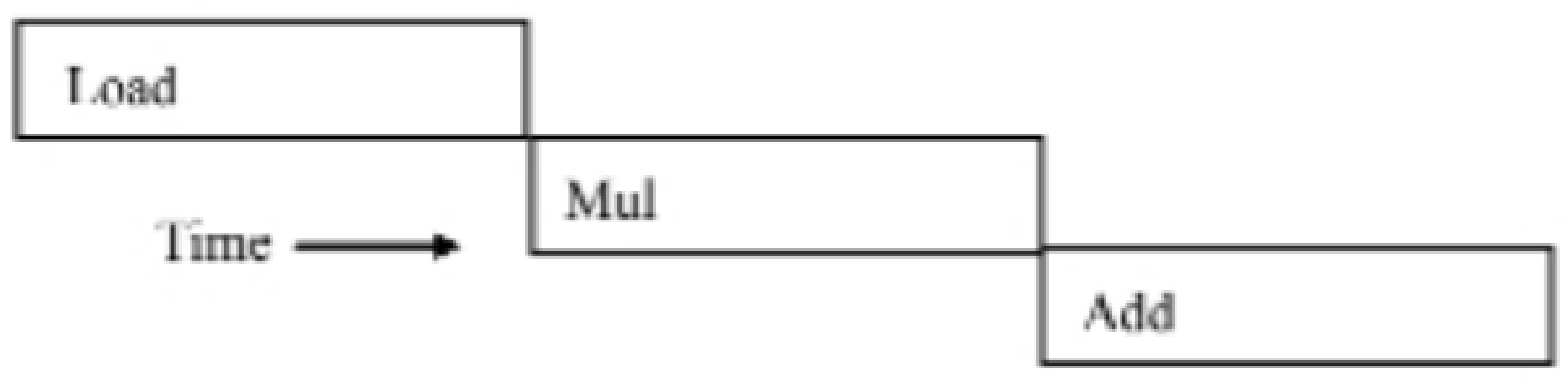


Fig. 1(a)

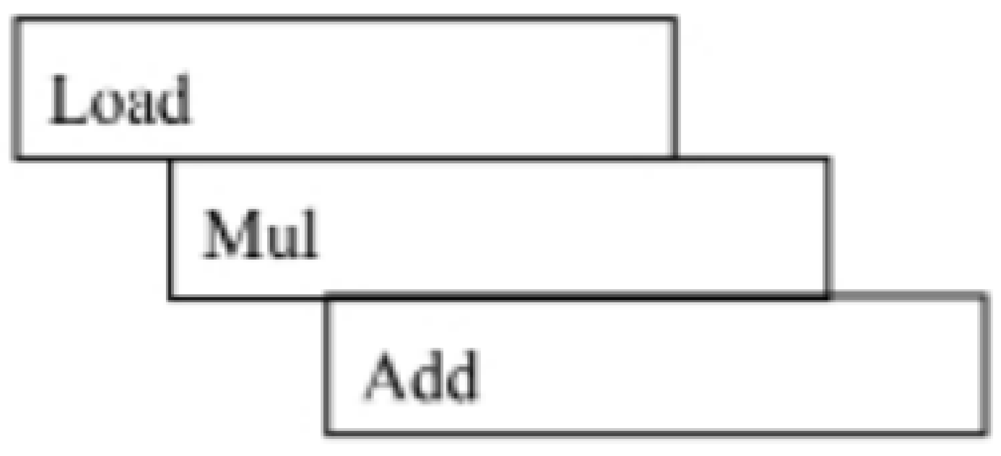


Fig. 1(b)

Long Answer Type Questions

1. What are the different types of vector operations? Give different fields in a vector instruction. [WBUT 2005, 2013]

Answer:
There are two primary types of vector operations:
- Vector-register operations
- Memory-memory vector operations

In vector-register operations, all vector operations—except load and store—are among the vector registers. All major vector computers use vector-register architecture, including the Cray Research processors (Cray-1, Cray-2). In memory-memory vector operations, all vector operations are memory to memory. The first vector computers were of this type, as were CDC's vector computers.

The vector instructions of the following types:

- Vector-vector instructions:
  - f1: Vi --> Vj (e.g. MOVE Va, Vb)
  - f2: Vj x Vk --> Vi (e.g. ADD Va, Vb, Vc)
- Vector-scalar instructions:
  - f3: s x Vi --> Vj (e.g. ADD R1, Va, Vb)
- Vector-memory instructions:
  - f4: M --> V (e.g. Vector Load)
  - f5: V --> M (e.g. Vector Store)
- Vector reduction instructions:
  - f6: V --> s (e.g. ADD V, s)
  - f7: Vi x Vj --> s (e.g. DOT Va, Vb, s)
- Gather and Scatter instructions:
  - f8: M x Va --> Vb (e.g. gather)
  - f9: Va x Vb --> M (e.g. scatter)
- Masking instructions:
  - fa: Va x Vm --> Vb (e.g. MMOVE V1, V2, V3)

Gather and scatter are used to process sparse matrices/vectors. The gather operation uses a base address and a set of indices to access from memory "few" of the elements of a large vector into one of the vector registers. The scatter operation does the opposite. The masking operations allow conditional execution of an instruction based on a "masking" register.

2. a) What are strip mining and vector stride, in respect to vector processors?
b) Both vector processors and array processors are specialized to operate on vectors. What are the main differences between them? [WBUT 2005, 2010]

OR,

Differentiate between vector processor and array processor with example.
[WBUT 2018]

Answer:

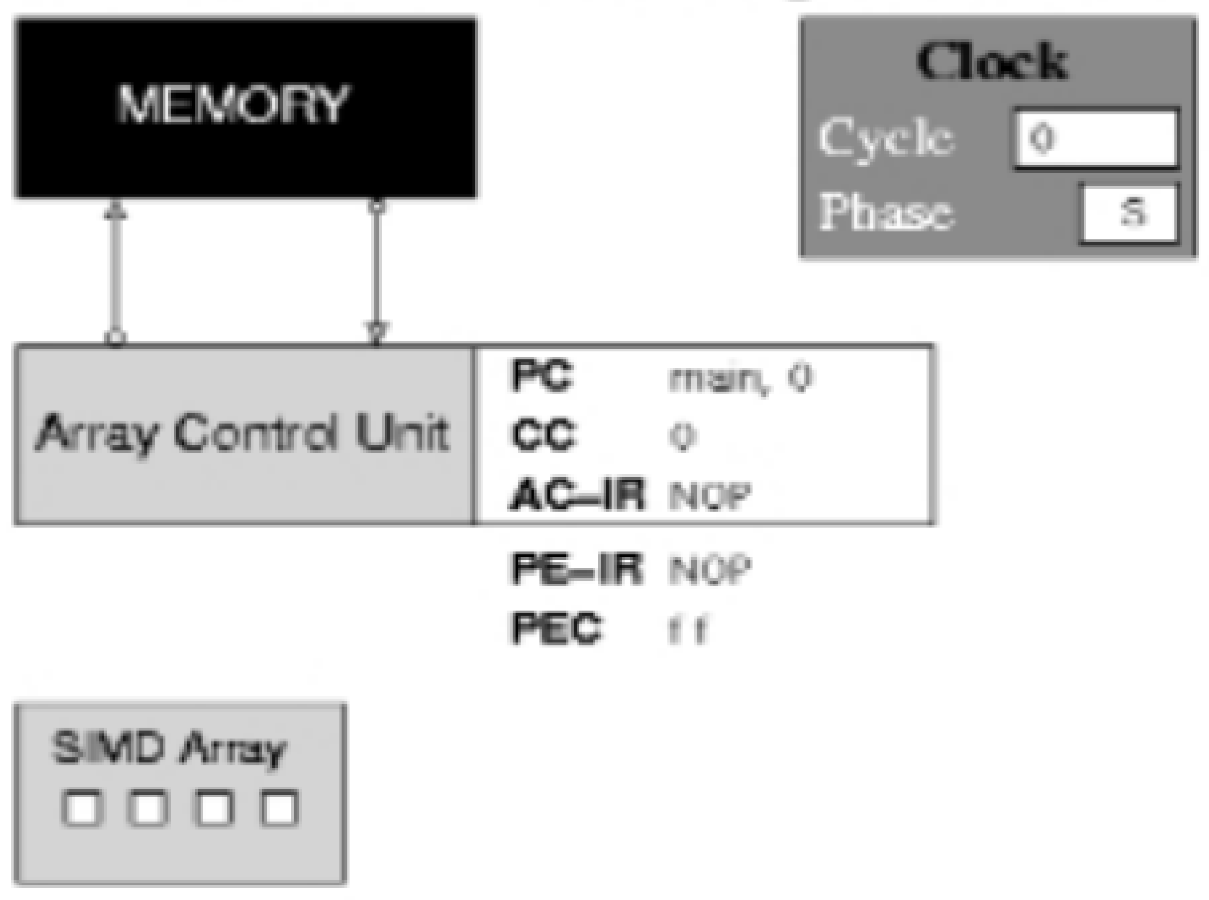
a) Now, we discuss about the different address position in memory of adjacent elements in a vector and these addresses may not be sequential. For vector processors without caches, we need a technique to fetch elements of a vector that are not adjacent in memory. A vector instruction is said to be ***stride i,*** if the distance between its two successive data references is ***i*** words or ***i*** double words apart. This distance separating elements that are to be gathered into a single register is called the **stride**. Once a vector is loaded into a vector register it acts as if it had logically adjacent elements.

The vector stride, like the vector starting address, can be put in a general-purpose register. Then instruction can be used to fetch the vector into a vector register. In some vector processors the loads and stores always have a stride value stored in a register, so that only a single load and a single store instruction are required. Complications in the memory system can occur from supporting strides greater than one. When multiple accesses contend for a bank, a memory bank conflict occurs and one access must be stalled. A bank conflict, and hence a stall, will occur if

$$\frac{\text{Number of banks}}{\text{Least common multiple (Stride, Number of banks)}} < \text{Bank busy time}$$

When a vector has a length greater than the vector registers then segmentation of the long vector into fixed length segments is necessary. This technique is called strip-mining. One vector segment is processed at a time. As an example, the vector segment length is 64 elements in Cray computers. Until the entire vector elements in each segment are processed, the vector register cannot be assigned to another vector operation. Strip-mining is restricted by the number of available vector, registers and vector chaining.

b) The SIMD-1 Array Processor consists of a Memory, an Array Control Unit (ACU) and the one-dimensional SIMD array of simple processing elements (PEs). The figures show a 4-processor array. The figures shows the initial image seen when the model is loaded.



The ACU is a simple load/store, register-register arithmetic processor. It has 16 general purpose registers, a Program Counter (PC), a Condition code Register (CC) and an Instruction Register (AC-IR). The Program Counter has two fields: label and offset. The label field is initially set to "main" and the offset to zero. The ACU also uses two other registers, the Processing Element Instruction Register (PE-IR) and the Processing Element Control register (PEC) which are global registers used to communicate with the SIMD Array. The Processing Elements operate in lock step, *i.e.* each active PE (determined by the state of its PEC bit) obeys the same instruction at the same time. Whenever a PE ACC is updated by a PE instruction, the PE sends the new ACC value to each of its neighbors.
When first loaded, the model contains a program which reverses the order of the values held in memory locations 0 and 2 of the Processing Elements (initially in locations 0 and 2 of each of their memories) and leaves the results in location 1 and 3 of each of their memories.

A vector processor is also a CPU design that is able to run mathematical operations on multiple data elements simultaneously. This is in contrast to a scalar processor which handles one element at a time. A computer with built-in instructions that perform multiple calculations on vectors (one-dimensional arrays) simultaneously. It is used to solve the same or similar problems as an array processor; however, a vector processor passes a vector to a functional unit, whereas an array processor passes each element of a vector to a different arithmetic unit.

Vector processors are based on a single-instruction, multiple data architecture that is distinctly different than SIMD extension to scalar/superscalar processors. Each vector data path has some data independence from the others allowing data path dependent operations. This allows easier control for wider machines. Single chip vector processors can still be low power and easy to program even with eight parallel vector units. For many communications algorithms, characterized by high data parallelism, vector single-instruction machines end up being the ideal balance of instruction/programming simplicity and compactness, while still supporting complex processing requirements and high performance.

3. a) How do vector processors improve the speed of instruction execution over scalar processors? Illustrate with an example.
b) What is vectorizing compiler? Why do we need it in a vector processor?
[WBUT 2015]

Answer:
a) Many performance optimization schemes are used in vector processors. Memory banks are used to reduce load/store latency. Strip mining is used to generate code so that vector operation is possible for vector operands whose size is less than or greater than the size of vector registers. Vector chaining - the equivalent of forwarding in vector processors - is used in case of data dependency among vector instructions. Special scatter and gather instructions are provided to efficiently operate on sparse matrices.

b) An intelligent compiler must be develop to detect the concurrency among vector instructions which can be realized with pipelining or with the chaining of pipelines. A vectorizing compiler would regenerate parallelism lost in the use of sequential languages. It is desirable to use high level programming languages with rich parallel constructs on vector processors. The following four stages have been recognized in the development of parallelism in advanced programming. The parameter in parentheses indicates the degree of parallelism explorable at each stage:

- Parallel algorithm (A)
- High-level language (L)
- Efficient object code(O)
- Target machine code(M)

The degree of parallelism refers to the number of independent operations that can be performed simultaneously. In the ideal situation with well developed parallel user languages, we should expect $A \geq L \geq O \geq M$, as in the figure below.

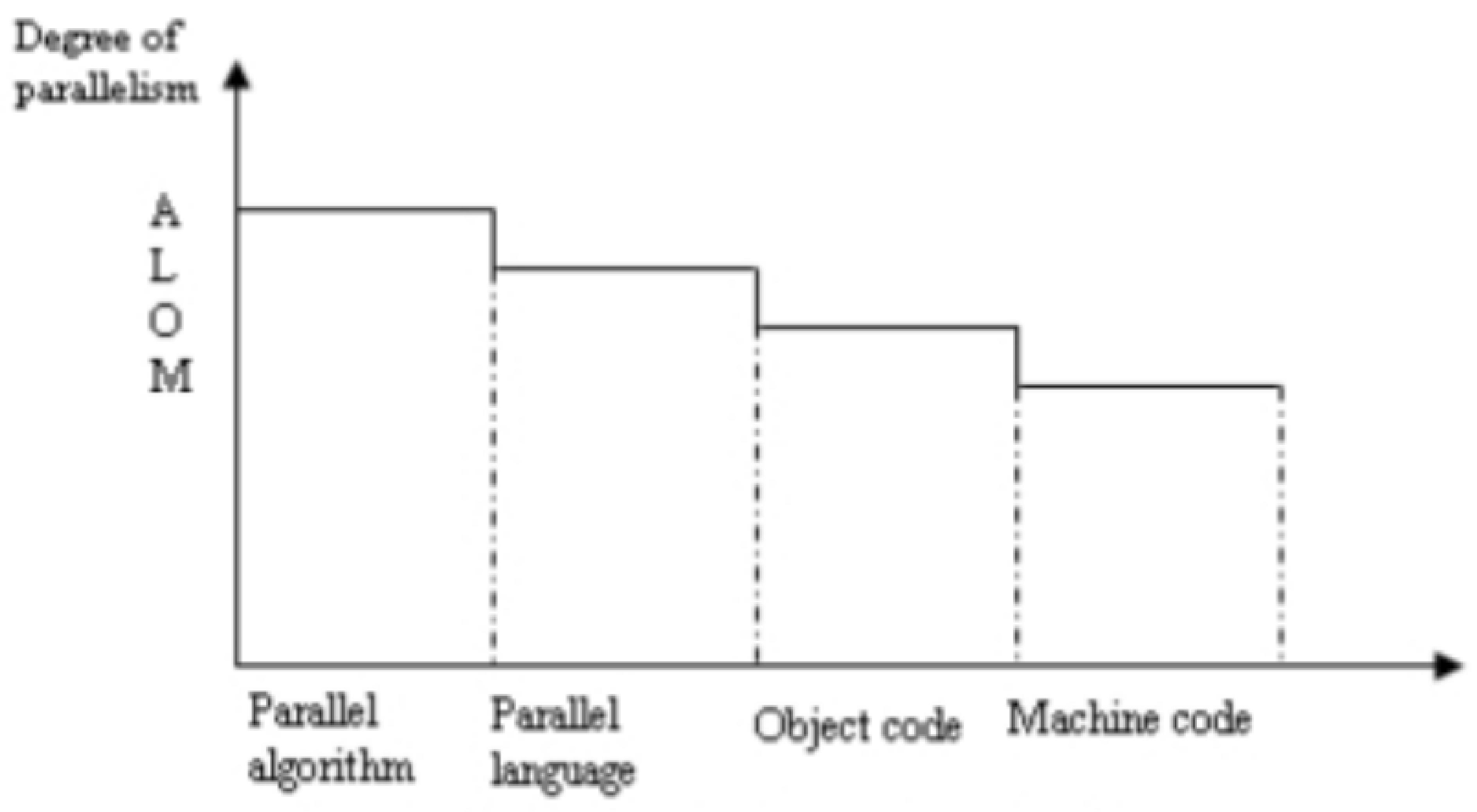


Fig: The ideal case of using parallel algorithm

At present any parallelism in an algorithm is lost when it is expressed in a sequential high-level language. In order to promote parallel processing in machine hardware, an intelligent compiler is needed to regenerate the parallelism through vectorization as shown in figure below.

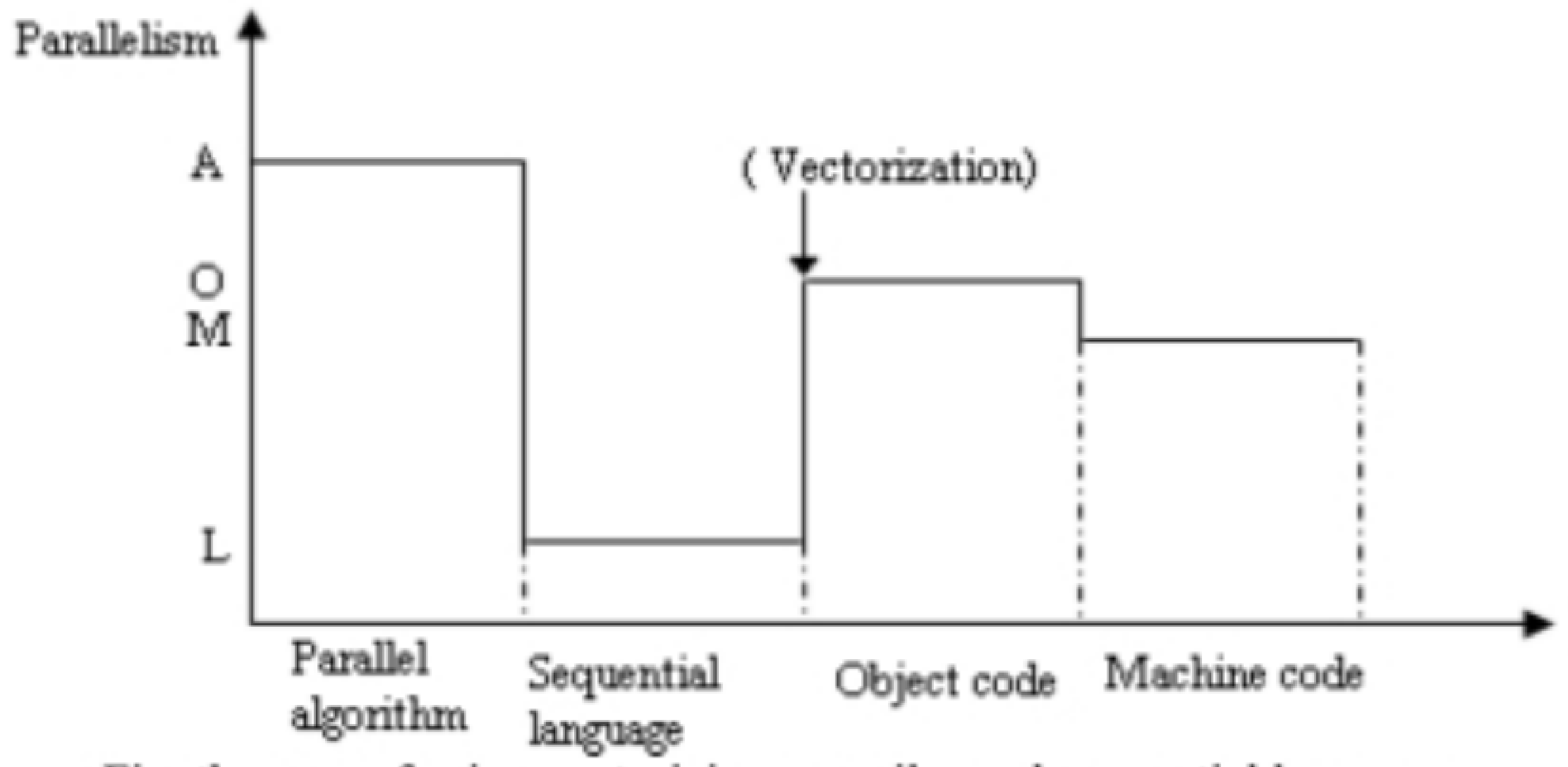


Fig: the case of using vectorizing compiler and sequential language

The process to replace a block of sequential code by vector instructions is called vectorization and the system software which does this regeneration of parallelism is called a vectorizing compiler.

4. Differentiate between Vectored and Non-vectored interrupts. [WBUT 2017]

Answer:

A vectored interrupt is where the CPU actually knows the address of the Interrupt Service Routine in advance. All it needs is that the interrupting device sends its unique vector via a data bus and through its I/O interface to the CPU. The CPU takes this vector, checks an interrupt table in memory, and then carries out the correct ISR for that device. So the vectored interrupt allows the CPU to be able to know what ISR to carry out in software (memory). A non-vectored interrupt is where the interrupting device never sends an interrupt vector. An interrupt is received by the CPU, and it jumps the program counter to a fixed address in hardware. So, the difference between vectored and non-vectored interrupt is that in vectored interrupt the new address is generated by the processor automatically. For instance, if 8085 microprocessor is interrupted through RST 5.5 pin the processor will multiply 5.5 by 8 and automatically convert it to Hex address. While in a non-vectored interrupt it is necessary for the user to provide the address of subroutine using the CALL instruction.

5. Explain low-order interleaved memory and its advantages. [WBUT 2018]

Answer:

The idea of interleaving memory is that memory is divided into banks.
Each bank is to be considered as having the same addressable unit as the main memory.
In low–order interleaving, consecutive addresses in the memory will be found in different memory banks.

Consider a 64–word memory that is 4–way interleaved. This means that there are four memory banks, each holding 16 words.
If this memory is also low–order interleaved, we have the following allocation of words to banks.

Bank 0: Words 00, 04, 08, 12, 16, 20, 24, …, 60
Bank 1: Words 01, 05, 09, 13, 17, 21, 25, …, 61
Bank 2: Words 02, 06, 10, 14, 18, 22, 26, …, 62
Bank 3 Words 03, 07, 11, 15, 19, 23, 27, …, 63
Again, we have not yet specified the size of the memory word.
In traditional (flat) layouts, memory banks can be allocated a continuous block of memory addresses, which is very simple for the memory controller and gives equal performance in completely random access scenarios, when compared to performance levels achieved through interleaving.

6. Write short notes on the following:

a) Scalar and vector processors	**[WBUT 2006, 2007, 2018]**
b) Memory to memory vector architecture	**[WBUT 2010]**
c) Vectorizing compilers	**[WBUT 2010]**
d) Vector registers architectures	**[WBUT 2011]**
e) Vector Stride	**[WBUT 2012]**
f) Array processor & Vector processor	**[WBUT 2014]**

Answer:

a) Scalar and vector processors:

A vector processor is a CPU design that is able to run mathematical operations on multiple data elements simultaneously. This is in contrast to a scalar processor which handles one element at a time. A computer with built-in instructions that perform multiple calculations on vectors (one-dimensional arrays) simultaneously. It is used to solve the same or similar problems as an array processor; however, a vector processor passes a vector to a functional unit, whereas an array processor passes each element of a vector to a different arithmetic unit.

Vector processors are based on a single-instruction, multiple data architecture that is distinctly different than SIMD extension to scalar/superscalar processors. Each vector data path has some data independence from the others allowing data path dependent operations. This allows easier control for wider machines. Single chip vector processors can still be low power and easy to program even with eight parallel vector units. For many communications algorithms, characterized by high data parallelism, vector single-instruction machines end up being the ideal balance of instruction/programming simplicity and compactness, while still supporting complex processing requirements and high performance.

A vector processor for executing vector instructions comprises a plurality of vector registers and a plurality of pipeline arithmetic logic units. The vector registers are constructed with a circuit which operates in a speed equal to 2n times as fast as the processing speed of the pipeline arithmetic logic units. Either the read or the write operation from or to the vector registers are carried out in the time obtained by a processing cycle of each of the pipeline arithmetic logic units multiplied by n/2.

The simplest processors are scalar processors. Each instruction executed by a scalar processor typically manipulates one or two data items at a time. RISC processors are in this category. A scalar processor that includes a plurality of scalar arithmetic logic units and a special function unit. Each scalar unit performs, in a different time interval, the same operation on a different data item, where each different time interval is one of a plurality of successive, adjacent time intervals. Each unit provides an output data item in the time interval in which the unit performs the operation and provides a processed data item in the last of the successive, adjacent time intervals. The special function unit provides a special function computation for the output data item of a selected one of the scalar units, in the time interval in which the selected scalar unit performs the operation, so as to avoid a conflict in use among the scalar units. A vector processing unit includes an input data buffer, the scalar processor, and an output orthogonal converter.

b) Memory to memory vector architecture:
To maintain an initiation rate of one word fetched or stored per clock, the memory system must be capable of producing or accepting this data. This is usually done by creating multiple memory banks. There are significant numbers of banks is useful for dealing with vector loads or stores that access rows or columns of data.
In the register to register machines the vectors have a relatively short length, 64 in the case of the Cray family, but the startup time is far less than on the memory to memory machines. Thus these machines are much more efficient for operations involving short vectors, but for long vector operations the vector registers must loaded with each segment before the operation can continue.

Vector-Memory instruction: vector –memory instruction can be defined by vector load or vector store operations between vector register and memory. it can also defined by the following two mapping functions f_1 and f_2 .

$f_1 : M \rightarrow V$ [vector load] and $f_2 : V \rightarrow M$ [vector store]

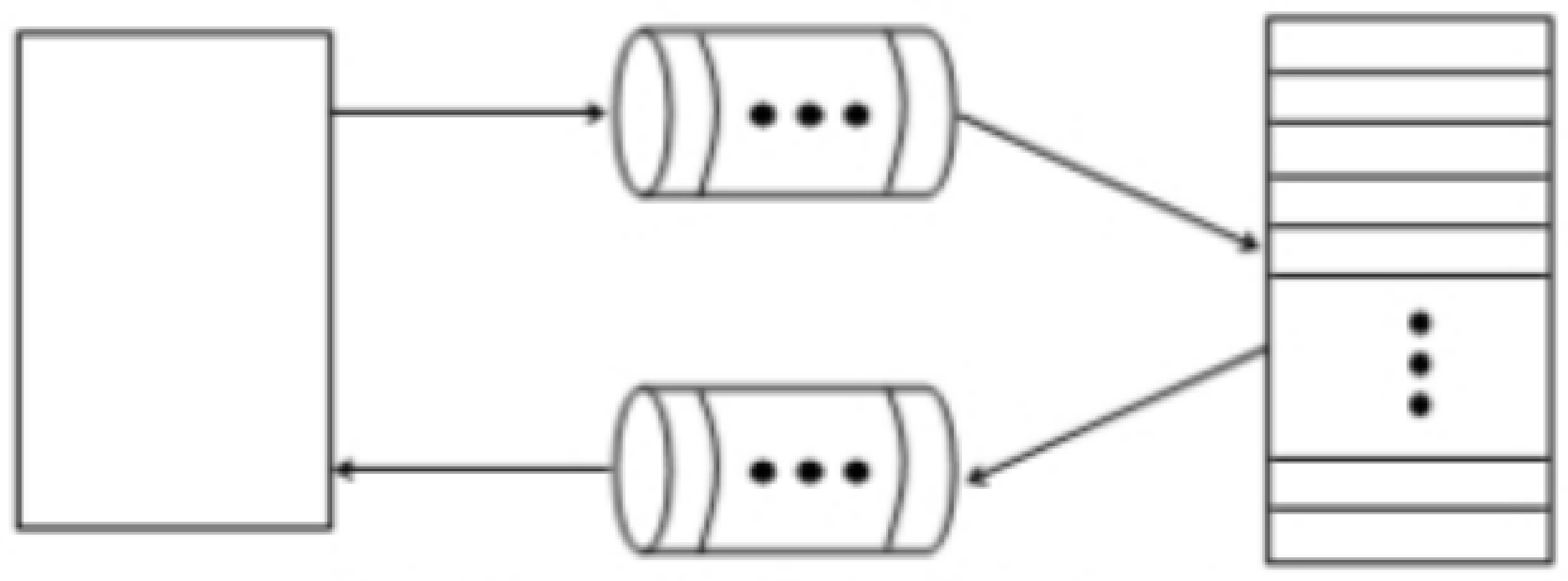


Fig: Vector memory instructions

Register to register machines now dominate the vector computer market, with a number of offerings from Cray Research Inc., including the Y-MP and the C-90.

The basic processor architecture of the Cray supercomputers has changed little since the Cray-1 was introduced in 1976. There are 8 vector registers, named V0 through V7, which each hold 64 64-bit words. There are also 8 scalar registers, which hold single 64-bit words, and 8 address registers (for pointers) that have 20-bit words. Instead of a cache, these machines have a set of backup registers for the scalar and address registers; transfer to and from the backup registers is done under program control, rather than by lower level hardware using dynamic memory referencing patterns.

The original Cray-1 had 12 pipelined data processing units; newer Cray systems have 14. There are separate pipelines for addition, multiplication, computing reciprocals (to divide x by y, a Cray computes x.(1/y)), and logical operations. The cycle time of the data processing pipelines is carefully matched to the memory cycle times. The memory system delivers one value per clock cycle through the use of 4-way interleaved memory.

c) Vectorizing compilers: ***Refer to Question No. 4(b) of Long Answer Type Questions.***

d) Vector registers architectures:
Each vector register is a fixed-length bank holding a single vector. Each vector register must have at least two read ports and one write port. This will allow a high degree of overlap among vector operations to different vector registers. The read and write ports, which total at least 16 read ports and 8 write ports, are connected to the functional unit inputs or outputs by a pair of crossbars.

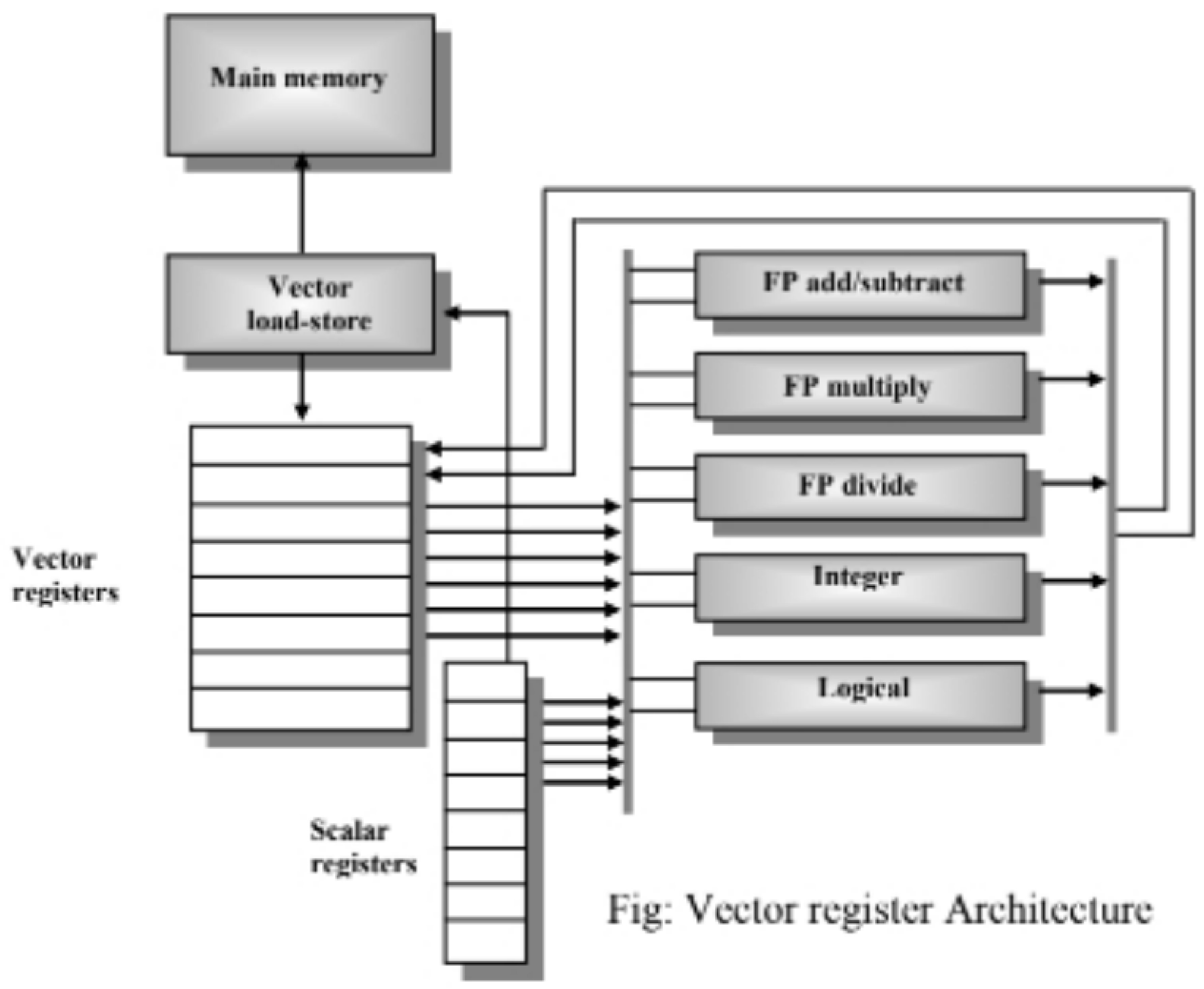


Fig: Vector register Architecture

In the above figure, there are eight 64-element vector registers, and all the functional units are vector functional units.

Vector functional units: In a vector functional unit each unit is fully pipelined and can start a new operation on every clock cycle. A control unit is needed to detect hazards. In the above figure, there are five functional units.

Vector load-store unit: This is a vector memory unit that loads or stores a vector to or from memory. Here, vector loads and stores are fully pipelined, so that words can be moved between the vector registers and memory with a bandwidth of one word per clock cycle, after an initial latency.

Set of scalar registers: Scalar registers can also provide data as input to the vector functional units, as well as compute addresses to pass to the vector load-store unit.

Vector Execution Time: The execution time of a sequence of vector operations primarily depends on three factors:

- The length of the operand vectors,
- Structural hazards among the operations and
- Data dependences.

We can compute the time for a single vector instruction depending on the vector length and the initiation rate that is the rate at which a vector unit consumes new operands and produces new results. All modern supercomputers have vector functional units with multiple parallel pipelines that can produce two or more results per clock cycle.

e) Vector Stride: ***Refer to Question No. 5 of Short Answer Type Questions.***

f) Array processor & Vector processor:

Vector and array processing are essentially the same because, with slight and rare differences, a vector processor and an array processor are the same type of processor. A processor, or central processing unit (CPU), is a computer chip that handles most of the information and functions processed through a computer. A vector processor employs multiple vector pipelines. An array processor uses a number of processing elements operating in parallel. An array processor is a SIMD type processor and requires a host processor (control processor). An array processor is a synchronous parallel processor containing multiple ALUs. Each ALU contains local memory. The ALU together with the local memory is called a processing element (PE). The PEs are synchronized to perform same operation simultaneously. The host processor is a scalar processor. The instructions are fetched and decoded by the control processor. The vector instructions are sent to PEs for distributed execution over different elements of the vector operand. These vector elements are contained in the local memories. The PEs are passive devices without instruction decoding capabilities. Vector and array processing technology is not usually used in home or office computers. This technology is most often seen in high-traffic servers. Servers are racks of storage drives designed to house and allow access to information from several different users at different computers located on a computer network. Scalar processing technology operates on different principles than vector and array processing technology and is the most common type of processing hardware used in the average computer. A superscalar processor is a processor that operates like a scalar processor, but it has many different units within the CPU which each handle and process data simultaneously. The higher-performance superscalar processor type is also equipped with programming that makes it efficiently assign data processing to the available scalar units within the CPU. Most modern home computer processors are superscalar.

FLYNN'S TAXONOMY OF COMPUTER ARCHITECTURE

☞ Chapter at a Glance

Flynn's classification

The most popular taxonomy of computer architecture was defined by Flynn in 1966. Flynn's classification scheme is based on the notion of a stream of information. Two types of information flow into a processor: instructions and data. The instruction stream is defined as the sequence of instructions performed by the processing unit. The data stream is defined as the data traffic exchanged between the memory and the processing unit. According to Flynn's classification, either of the instruction or data streams can be single or multiple. Computer architecture can be classified accordingly into the following four distinct categories:

- Single-instruction single-data streams (SISD)
- Single-instruction multiple-data streams (SIMD)
- Multiple-instruction single-data streams (MISD) Multiple-instruction multiple-data streams (MIMD)

Different Types of SIMD Models

Synchronous array of parallel processors is called an array processor, which consists of multiple processing elements (PEs) under the supervision of one control unit (CU). An array processor can handle single instruction and multiple data (SIMD) streams. In this sense, array processors are also known as SIMD computers. SIMD machines are especially designed to perform vector computations over matrices or arrays of data. There are two different types of SIMD models:

- Distributed memory model
- Shared memory model

Shared Memory Models

In shared memory SIMD model instructions are stored by the host to control memory and data are directly stored to the shared memory through data bus. Array control unit is attached to the control memory. This array control unit separates scalar instructions and vector instructions. Scalar instructions are transferred to the scalar processor for executions. Vector instructions are transferred to different PEs by broadcast bus.

Multiple Choice Type Questions

1. Advantage of MMX technology lies in [WBUT 2010]
a) Multimedia application b) VGA
c) CGA d) none of these

Answer: (a)

2. Array Processor is present in [WBUT 2010]
a) SIMD b) MISD c) MIMD d) none of these

Answer: (a)

3. Which one of the following has no practical usage? [WBUT 2010, 2014, 2016]
a) SISD b) SIMD c) MISD d) MIMD

Answer: (c)

4. The expression for Amdahl's law is [WBUT 2011, 2016, 2017]
a) $S(n)=1/f$ where $n \to \infty$ b) $S(n)=f$ where $n \to \infty$
c) $S(n)=1/T$ where $n \to \infty$ d) None of these

Answer: (c)

5. Which MIMD systems are best according to scalability with respect to the number of processors? [WBUT 2011]
a) Distributed memory computers b) ccNUMA systems
c) nccNUMA systems d) Symmetric multiprocessors

Answer: (a)

6. Superscalar processors have CPI of [WBUT 2011, 2017]
a) less than 1 b) greater than 1 c) more than 2 d) greater than 3

Answer: (a)

7. The main memory of a computer has 2 cm blocks while the cache has 2 c blocks. If the cache uses the set associative mapping scheme with 2 blocks per set; then block k of the main memory maps to the set [WBUT 2011, 2016]
a) (*k* mod *m*) of the cache b) (*k* mod *c*) of the cache
c) (*k* mod 2*c*) of the cache d) (*k* mod 2*m*) of the cache

Answer: (a)

8. As the bus in a multiprocessor is a shared resource, so there must be some mechanism to resolve the conflict. The ……………. algorithm form the below mentioned is not a conflict resolution technique. [WBUT 2016]
a) state priority algorithm b) FIFO algorithm
c) LRU algorithm d) Daisy Chaining algorithm

Answer: (a)

9. The MMX technology uses [WBUT 2018]
a) Pipelining technique b) Vectorizing technique

c) SIMD technique **d) MIMD technique**

Answer: (c)

Short Answer Type Questions

1. Discuss Flynn's classification of parallel computers.
[WBUT 2006, 2007, 2009, 2010, 2017]

OR,

Describe Flynn's classification of computer architecture. [WBUT 2012, 2019]

OR,

Explain in brief with neat diagrams the Flynn's classifications of computers.
[WBUT 2013, 2018]

OR,

Explain Flynn's classification. [WBUT 2016]

Answer:

The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

Single Instruction, Single Data stream (SISD)

A sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are the traditional uni-processor machines like a PC or old mainframes.

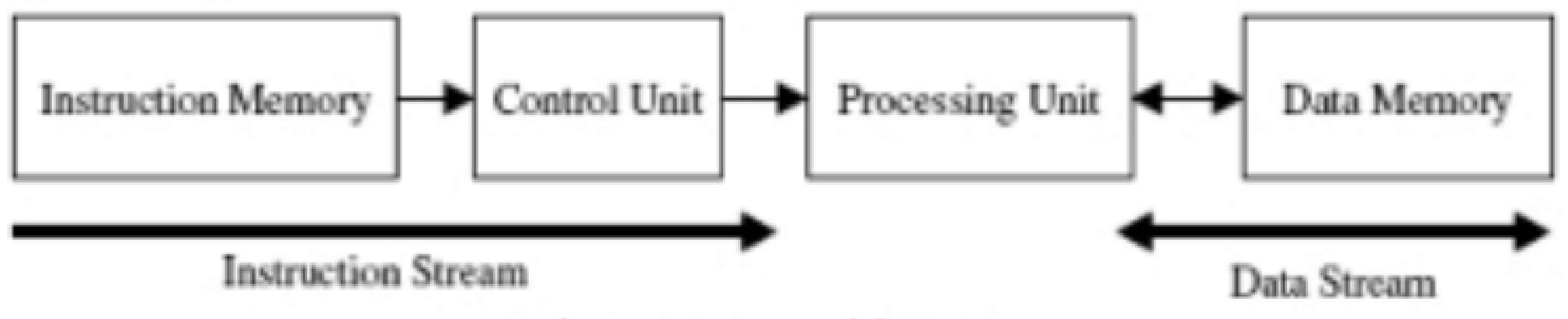


Fig: SISD Architecture

Single Instruction, Multiple Data streams (SIMD)

A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or CPU.

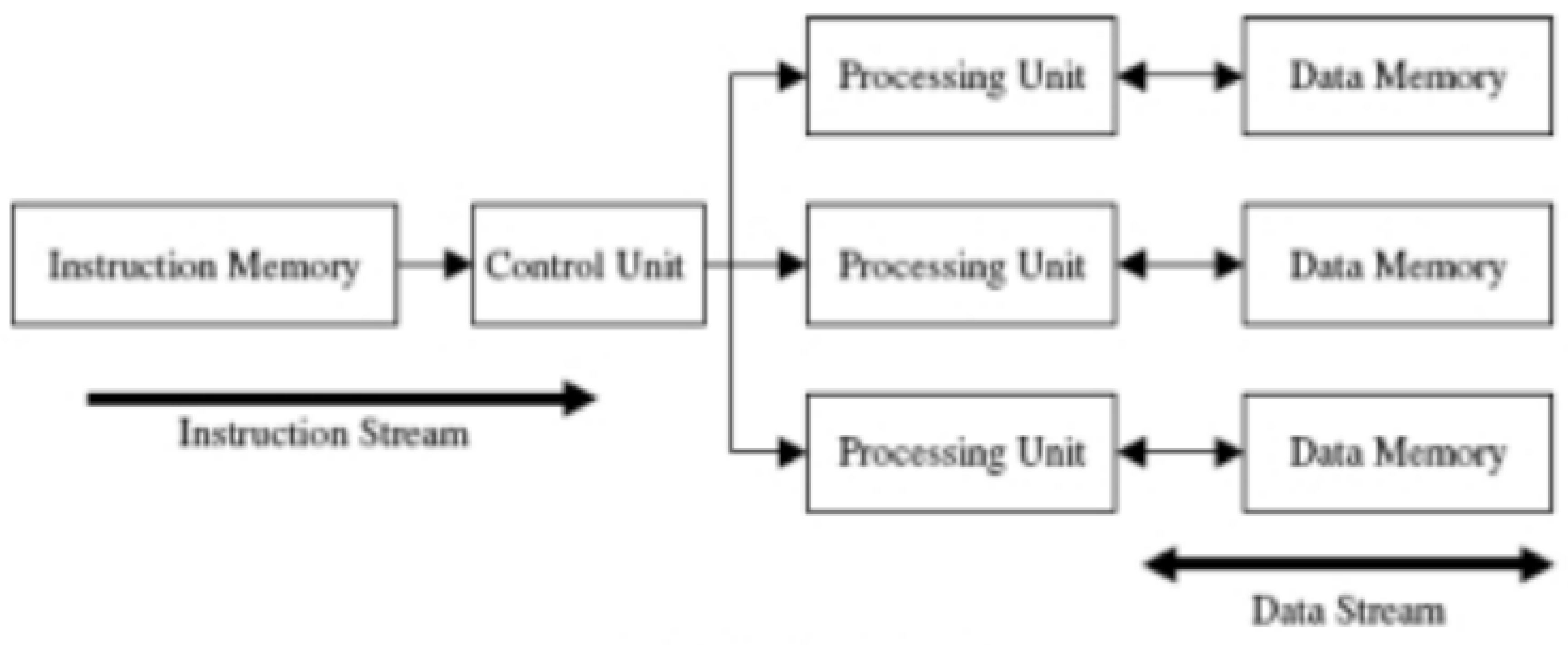


Fig: SIMD Architecture

Multiple Instructions, Single Data stream (MISD)
Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

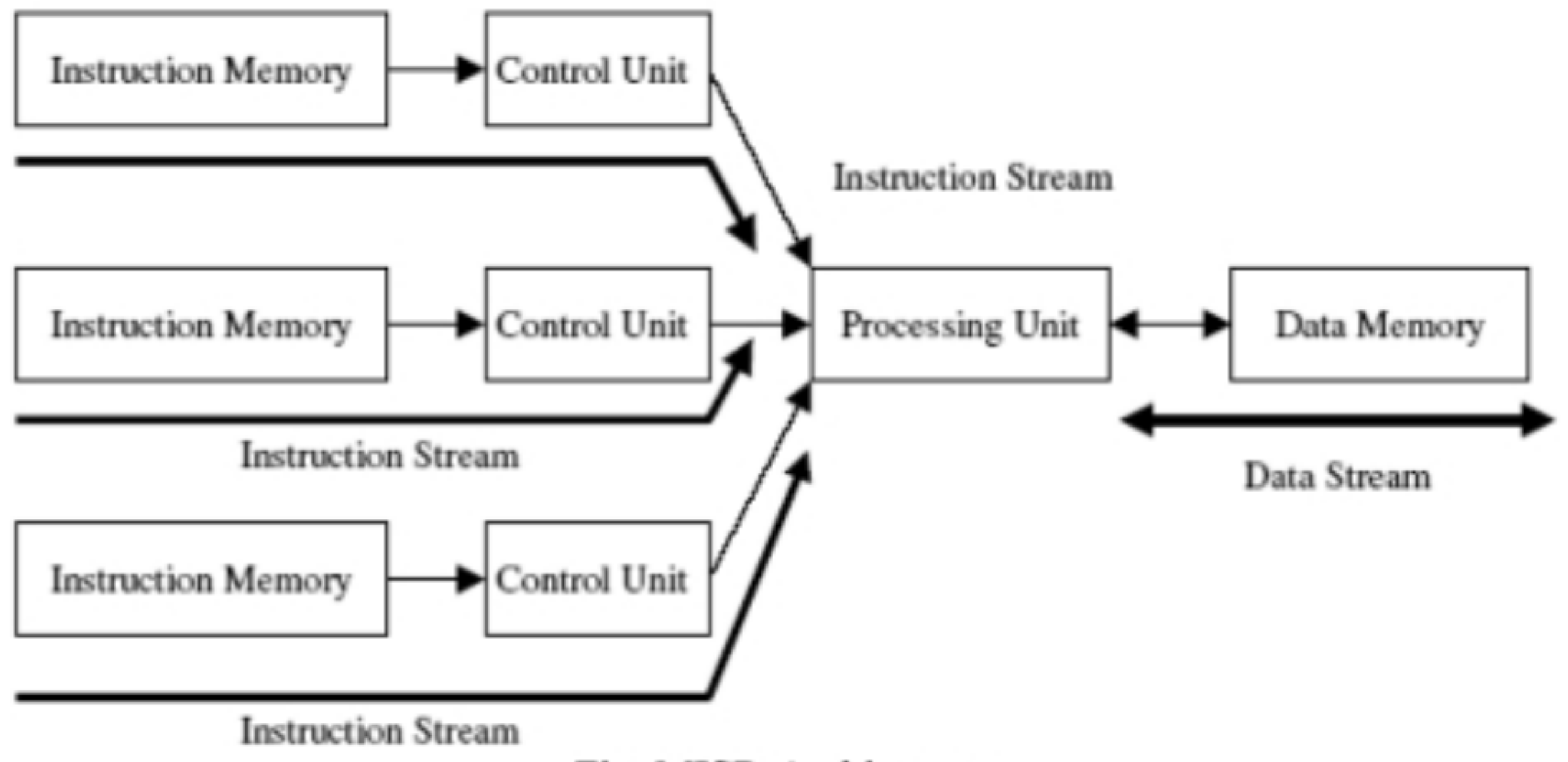


Fig: MISD Architecture

Multiple Instructions, Multiple Data streams (MIMD)
Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

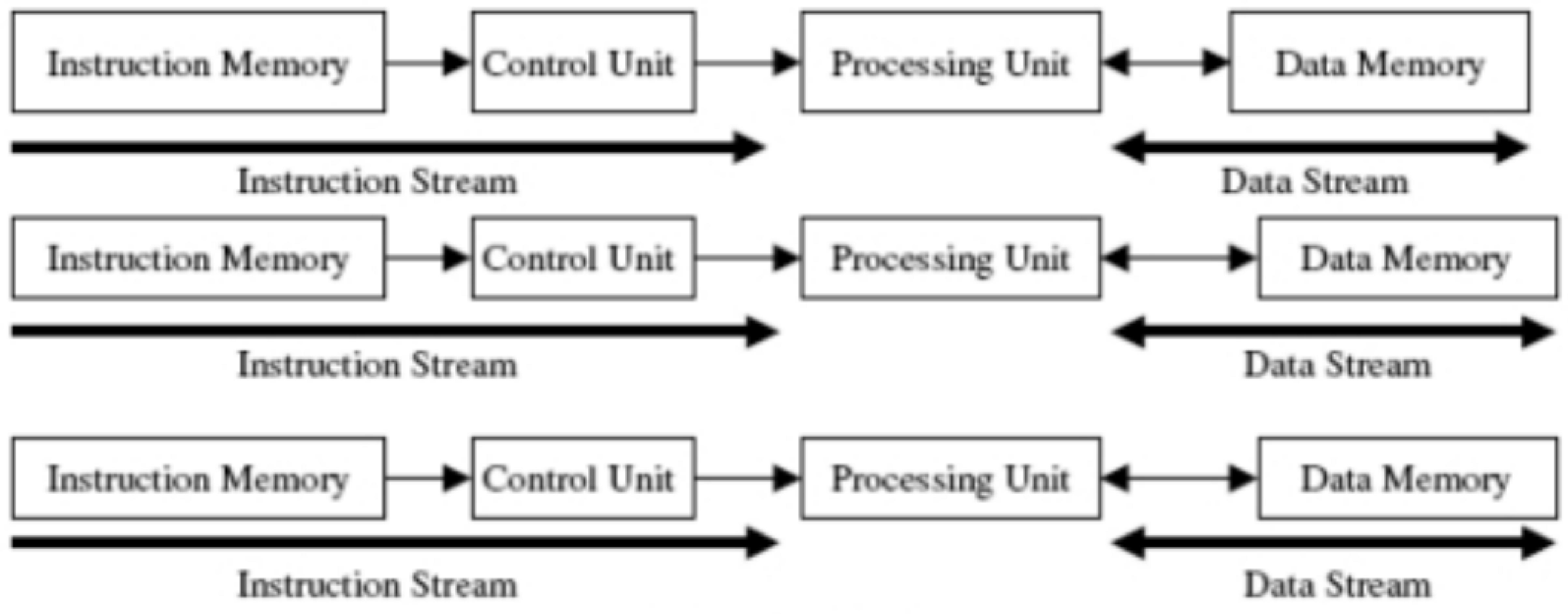


Fig: MIMD Architecture

2. Implement the data routing logic of SIMD architecture to compute
$s(k)=\sum_{i=0}^{k} Ai$ **for** $k=0,1,2...N-1$. **[WBUT 2008]**

OR,

Why do we need masking mechanism in SIMD array processors? [WBUT 2015]
In an SIMD array processor of 8 PEs, the sum S(k) of the first k components in a vector A is desired for each k from 0 to 7.
Let $A=(A_0,A_1,......,A_7)$. **We need to compute the following and throughput.**

$$S(k)=\sum_{i=0}^{k} A_i; for\ k=0,1,.....,7$$

Discuss how data-routing and masking are performed in the processor.
[WBUT 2015]

Answer:
Masking technique for a SIMD processor is capable of masking a plurality of individual machine operations within a single instruction incorporating a plurality of operations. To accomplish this each different machine operation within the instruction includes a number of masking bits which address a specific location in a mask register. The mask register includes a mask bit bank. The mask location selected within the mask register is bit-wise ANDed with a mask context bit in order to establish whether the processing element will be enabled or disabled for a particular conditional sub-routine which is called.

We show the execution details of the following vector instruction in an array of N processing elements (PEs) to illustrate the necessity of data routing in an array processor. Here the sum S(k) of the first k components in a vector A is preferred for each k from 0 to n-1.
Now, A = (A_0 , A_1 , …, A_{n-1})

So, the following n summations are,
$s(k)=\sum_{i=0}^{k} Ai$ for $k=0,1,2...N-1$.

These n vector summations can be computed recursively by going through the following n-1 iterations:
S(0) = A_0
S(k) = S(k-1) + A_k for k= 1, 2 ,3…, n-1
The above recursive summations for the case of n = 8 are implemented in an array processor with N = 8 PEs in $\log_2 n = 3$ steps as shown in the figure below. At first each A_i is transfer to the R_i register in PE $_i$ for I = 0, 1,…n-1. Now in step 1, A_i is routed from R_i to R_{i+1} and added to A_{i+1} with the resulting sum $A_i + A_{i+1}$in R_{i+1} for I = 0, 1,…,6
In step 2, the intermediate sums in R_i are routed to R $_{i+2}$ for I = 0 to 5. In step 3, the intermediate sums in R_i are routed to R_{i+4} for i = 0 to 3. Similarly, PE_k has the final value of S(k) for k = 0,1,2,…,7 in the last column of the figure below.

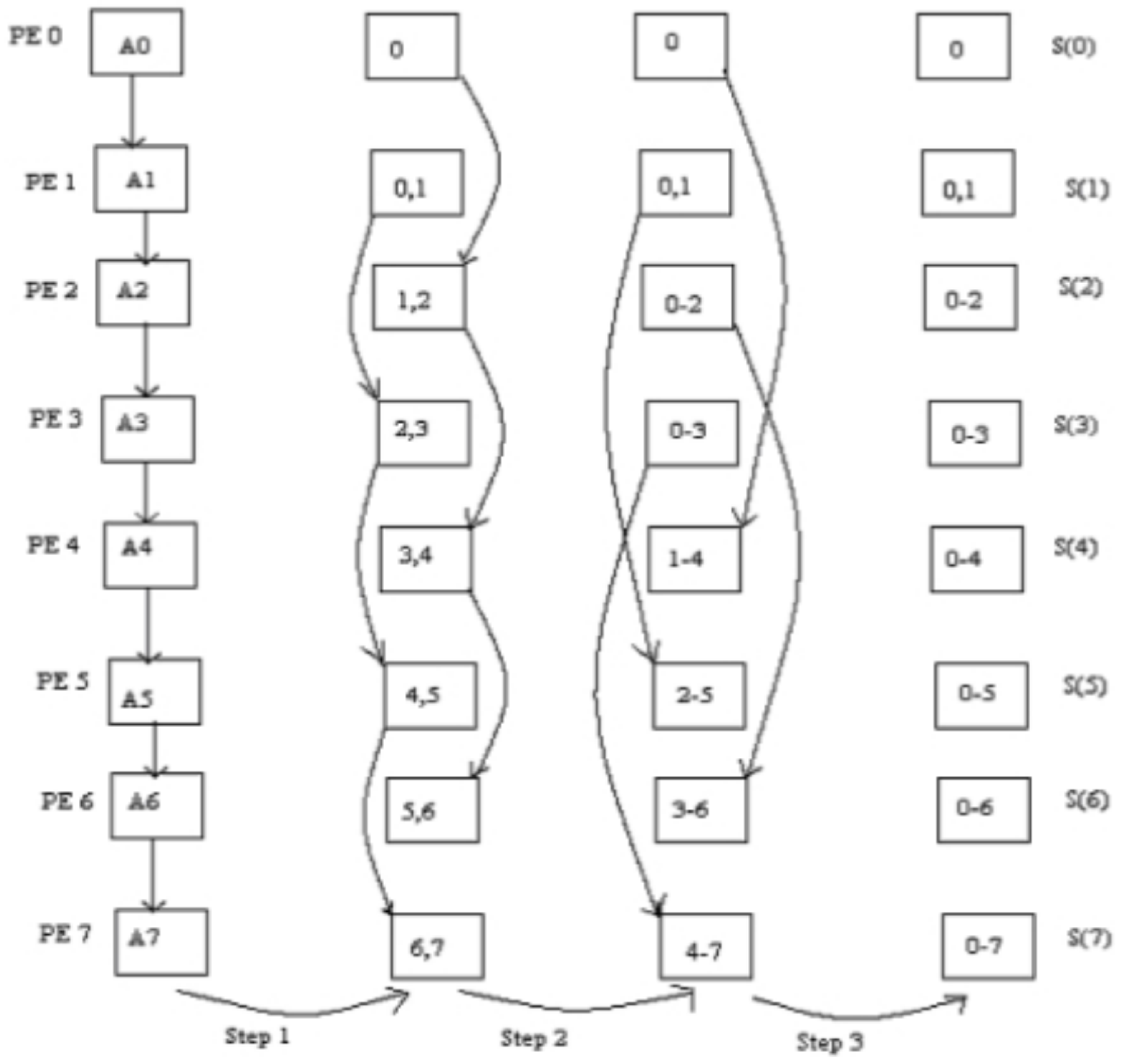


Fig: The calculation of the summation $s(k)=\sum_{i=0}^{k} Ai$ for $k=0,1,2...N-1$ in an SIMD machine

3. A 50 MHz processor was used to execute a program with the following instruction mix and clock cycle counts:

Instruction Type	**Instruction Count**	**Clock Cycle Count**
Integer Arithmetic	**50000**	**2**
Data Transfer	**70000**	**3**
Floating point arithmetic	**25000**	**1**
Branch	**4000**	**2**

Calculate the effective CPI, MIPS rate for this program. [WBUT 2011]

Answer:

We know,

CPU time = Instruction Count (IC) * Clock pre Instruction (CPI) * Clock Cycle Time (CCT)

$$=\sum_{i=1}^{n} CPI_i * I_i * CCT$$

where, I_i = Number of times the ith instruction is executed in a program.

CPI_i = Number of clock cycles for the ith instruction.

The average value of clock Per Instruction (CPI_{av}) is given by,

$$CPI_{av} = \frac{\sum_{i=1}^{n} CPI_i * I_i}{IC} = \sum_{i=1}^{n}\left(CPI_i * \frac{I_i}{IC}\right)$$

where $\frac{I_i}{IC}$ = frequency of occurrence of ith instruction in the program.

$$\text{CPU time} = IC * CPI_{av} + CCT$$

$$MIPS = \frac{IC}{CPU\ \text{time} \times 10^6} = \frac{1}{CPI_{av} \times CCT \times 10^6} = \frac{\text{Clock rate}}{CPI_{av} \times 10^6}$$

Given, Clock rate = 50 MHz

CPI_{av} = .500 * 2 + .700*3 + .250*1 + .40*2 = 4.15

MIPS = $(50 * 10^6) / (4.15 * 10^6)$ = 12.05

4. What is the instruction level parallelism? [WBUT 2014]

OR,

Discuss the techniques to achieve instruction level parallelism. [WBUT 2018]

Answer:

Instruction-level parallelism (ILP) is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism. 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.

Long Answer Type Questions

1. Differentiate between multiprocessors and multicomputer based on their structures, resource sharing and inter processor communication. [WBUT 2007]

Answer:

A multicomputer comprises a number of von Neumann computers, or nodes, linked by an interconnection network. Each computer executes its own program. This program may access local memory and may send and receive messages over the network. Messages are used to communicate with other computers or, equivalently, to read and write remote memories. In the idealized network, the cost of sending a message between two nodes is independent of both node location and other network traffic, but does depend on message length.

A defining attribute of the multicomputer model is that accesses to local (same-node) memory are less expensive than accesses to remote (different-node) memory. That is, read and write are less costly than send and receive. Hence, it is desirable that accesses to local data be more frequent than accesses to remote data. This property, called locality, is a third fundamental requirement for parallel software, in addition to concurrency and scalability.

Another important class of parallel computer is the multiprocessor or shared-memory MIMD computer. In multiprocessors, all processors share access to a common memory, typically via a bus or a hierarchy of buses. In the idealized Parallel Random Access Machine (PRAM) model, often used in theoretical studies of parallel algorithms, any processor can access any memory element in the same amount of time. In practice, scaling this architecture usually introduces some form of memory hierarchy; in particular, the frequency with which the shared memory is accessed may be reduced by storing copies of frequently used data items in a cache associated with each processor. Access to this cache is much faster than access to the shared memory.

2. a) Describe the distribution and shared memory model of SIMD architecture.
b) Draw the block diagram and explain the functionality of processing element.
[WBUT 2008]

Answer:
a) There are two types of SIMD computer models are described below based on the memory distribution and addressing scheme used. One is Distributed-Memory Model and another is Shared-Memory Model. Most SIMD computers use a single control unit and distributed memories, except for a few that use associative memories. The instruction set of an SIMD computer is decoded by the array control unit. The pr*ocessing elements* (PEs) in the SIMD array are passive ALUs executing instructions broadcast from the control unit.

Distributed-Memory Model: A distributed-memory SIMD computer consists of an array of PEs which are controlled by the same array control unit, as shown in Fig: 1.

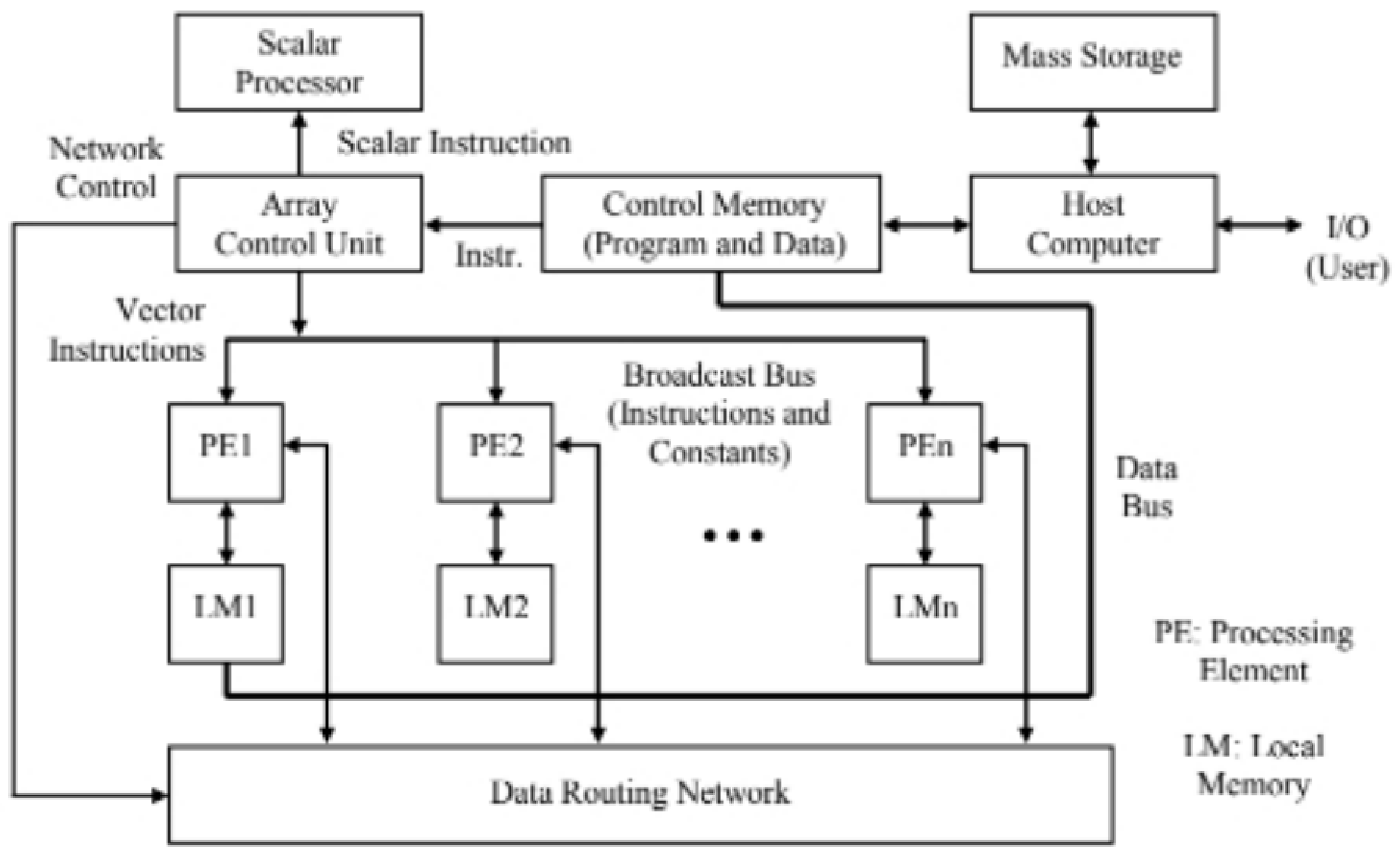


Fig: 1 Distributed –Memory Model SIMD architecture

Program and data are loaded into the control memory through the host computer.
An instruction is sent to the control unit for decoding. If it is a scalar or program control operation, it will be directly executed by a scalar processor attached to the control unit. If the decoded instruction is a vector operation, it will be broadcast to all the PEs for parallel execution.

Partitioned data sets are distributed to all the local memories attached to the PEs through a vector data bus. The PEs are interconnected by a data-routing network which performs inter-PE data communications such as shifting, permutation, and other routing operations. The data-routing network is under program control through the control unit. The PEs are synchronized in hardware by the control unit. Almost all SIMD machines built today are based on the distributed-memory model. Illiac IV, CM-2 are examples of Distributed Memory SIMD architecture.

Shared-Memory Model: In Fig: 2 we show a variation of the SIMD computer using shared memory among the PEs. An alignment network is used as the inter-PE memory communication network. Again this network is controlled by the control unit. The alignment network must be properly set to avoid access conflicts. Some SIMD computers use bit-slice PEs i.e. Shared-Memory Model. Example, DAP610 and CM / 200.

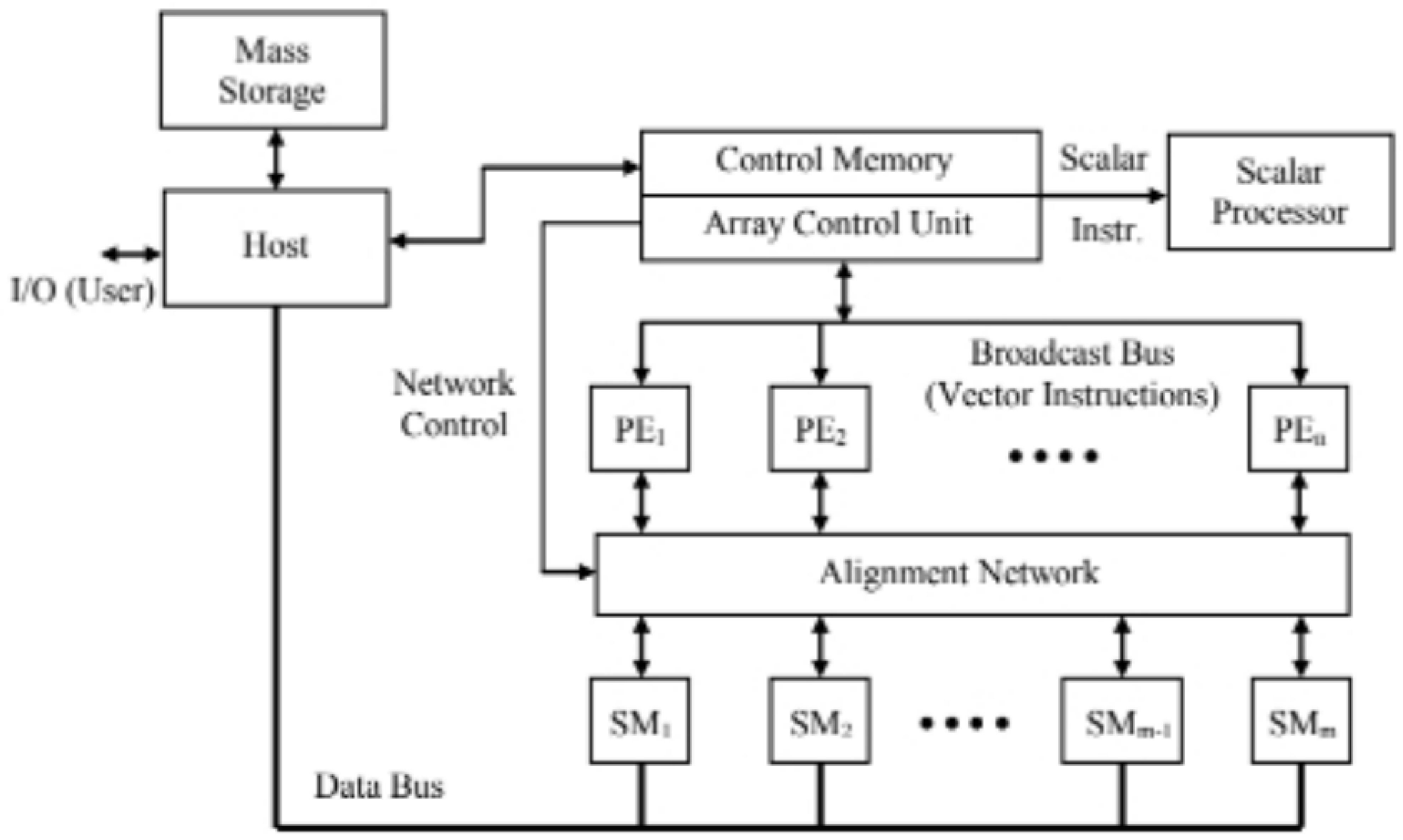


Fig: 2 Shared –Memory Model SIMD architecture

b) An array processor is a synchronous parallel computer with multiple arithmetic logic units, called processing elements (PE). The PEs are synchronized to perform the same function at same time.

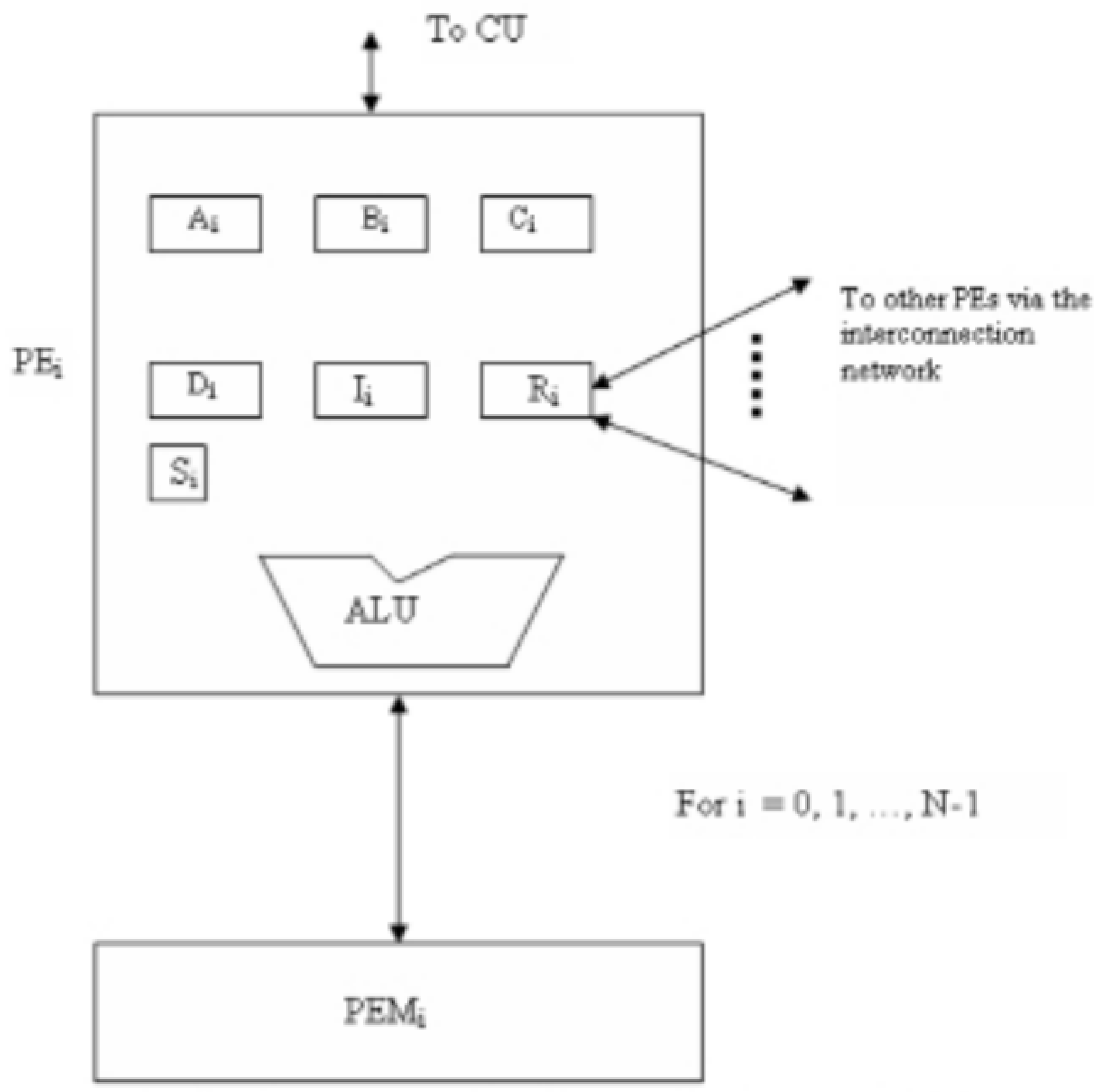


Fig: 3 Components of a Processing Element (PE)

PEs can establish an appropriate data routing mechanism. Each PE consists of an ALU with registers and local memory. The PEs are interconnected by a data-routing network. There are set of local registers and flags, A_i, B_i, C_i and S_i are present in a PE. The data routing register is R_i, address register is D_i and a local index register is I_i. When data transfer process occurs in PEs, then contains of the data routing register is transferred.

3. What is the main difference and similarities between multi-computer and Multiprocessor? Give the architecture for a typical MIMD processor? Explain the shared memory modes of MIMD. [WBUT 2011]

OR,

Briefly discuss MIMD architecture. [WBUT 2012, 2014]

OR,

What is the difference and similarities between multi-computer and multiprocessor? [WBUT 2014]

OR,

What are the differences between shared memory multiprocessor system and message passing multi-computer system? [WBUT 2018]

Answer:

A parallel machine model is called the multicomputer system. A multicomputer comprises a number of von Neumann computers, or nodes, linked by an interconnection network. Each computer executes its own program. This program may access local memory and may send and receive messages over the network. Messages are used to communicate with other computers or, equivalently, to read and write remote memories.
A shared-memory MIMD computer is called the multiprocessor computer. In multiprocessors, all processors share access to a common memory, typically via a bus or a hierarchy of buses and any processor can access any memory element in the same amount of time. Examples of this class of machine include the Silicon Graphics Challenge, Sequent Symmetry, and the many multiprocessor workstations.

MIMD (multiple instruction, multiple data) is a technique to achieve parallelism. Machines using MIMD have a number of processors that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. MIMD machines can be of either shared memory or distributed memory categories. These classifications are based on how MIMD processors access memory. Shared memory machines may be of the bus-based. Distributed memory machines may have hypercube or mesh interconnection schemes. MIMD machines with shared memory have processors which share a common, central memory. In the simplest form, all processors are attached to a bus which connects them to memory. MIMD machines with hierarchical shared memory use a hierarchy of buses to give processors access to each other's memory. Processors on different boards may communicate through inter-nodal buses. Buses support communication between boards. With this type of architecture, the machine may support over a thousand processors.

4. Why do we need parallel processing? What are different levels of parallel processing? Explain. [WBUT 2015]

Answer:

In computers, parallel processing is the processing of program instructions by dividing them among multiple processors with the objective of running a program in less time. In the earliest computers, only one program ran at a time. A computation-intensive program that took one hour to run and a tape copying program that took one hour to run would take a total of two hours to run. An early form of parallel processing allowed the interleaved execution of both programs together. The computer would start an I/O operation, and while it was waiting for the operation to complete, it would execute the processor-intensive program. The total execution time for the two jobs would be a little over one hour.

Levels of parallel processing:
We can have parallel processing at four levels.
i) Instruction Level: Most processors have several execution units and can execute several instructions (usually machine level) at the same time. Good compilers can reorder instructions to maximize instruction throughput. Often the processor itself can do this. Modern processors even parallelize execution of micro-steps of instructions within the same pipe.

ii) Loop Level: Here, consecutive loop iterations are candidates for parallel execution. However, data between subsequent iterations may restrict parallel execution of instructions at loop level. There is a lot of scope for parallel execution at loop level.

iii) Procedure Level: Here parallelism is available in the form of parallel executable procedures. Here the design of the algorithm plays a major role. For example each thread in Java can be spawned to run a function or method.

iv) Program Level: This is usually the responsibility of the operating system, which runs processes concurrently. Different programs are obviously independent of each other.
So parallelism can be extracted by the operating system at this level.

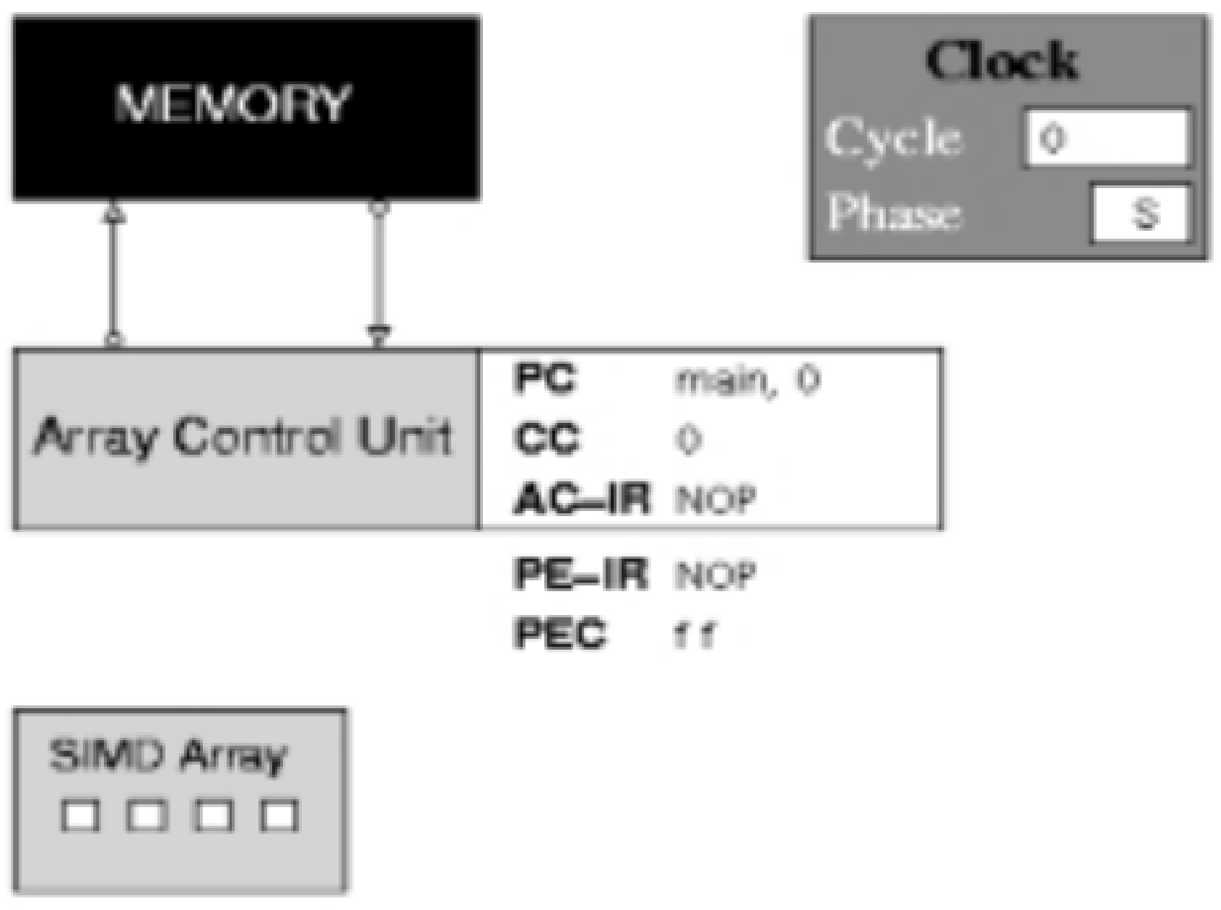
5. Write short notes on the following:

a) Array processor	**[WBUT 2005, 2007, 2010]**
b) MMX Technology	**[WBUT 2005, 2006, 2007]**
c) CM-2 machine	**[WBUT 2008]**
d) Flynn's classification	**[WBUT 2011]**

Answer:

a) Array processor:

The SIMD-1 Array Processor consists of a Memory, an Array Control Unit (ACU) and the one-dimensional SIMD array of simple processing elements (PEs). The figures show a 4-processor array. The figures shows the initial image seen when the model is loaded.



The system operates on a two phase clock. In clock cycles in which they are active, each unit executes its internal actions in the first phase of the clock and sends out a result packet in the second phase. The Memory, for example, reads an instruction or operand in the first phase and sends its output to the ACU in the second phase.

The ACU is a simple load/store, register-register arithmetic processor. It has 16 general purpose registers, a Program Counter (PC), a Condition code Register (CC) and an Instruction Register (AC-IR). The Program Counter has two fields: label and offset. The label field is initially set to "main" and the offset to zero. The ACU also uses two other registers, the Processing Element Instruction Register (PE-IR) and the Processing Element Control register (PEC) which are global registers used to communicate with the SIMD Array. The Processing Elements operate in lock step, *i.e.* each active PE (determined by the state of its PEC bit) obeys the same instruction at the same time. Whenever a PE ACC is updated by a PE instruction, the PE sends the new ACC value to each of its neighbors.

When first loaded, the model contains a program which reverses the order of the values held in memory locations 0 and 2 of the Processing Elements (initially in locations 0 and 2 of each of their memories) and leaves the results in location 1 and 3 of each of their memories.

b) MMX Technology:
MMX technology is an extension to the Intel Architecture (IA) designed to improve performance of multimedia and communication algorithms. The Pentium processor with MMX Technology is the first microprocessor to implement the new instruction set.
MMX consists of two main processor architectural improvements.

Operation of MMX Technology
The MMX technology consists of several improvements over the non-MMX Pentium microprocessors:
1. There are 57 new microprocessor instructions added those have been designed to handle video, audio, and graphical data more efficiently. Programs can use MMX instructions without changing to a new mode or operating-system visible state.

2. New 64-bit integer data type is also added to MMX Technique.
3. A new process, Single Instruction Multiple Data (SIMD), makes it possible for one instruction to perform the same operation on multiple data items.
4. The memory cache on the microprocessor has increased to 32 KB, meaning fewer accesses to memory that is off the microprocessor.

All MMX chips have a larger internal L1 cache than their non-MMX counterparts. This improves the performance of any software running on the chip, regardless of whether it actually uses the MMX-specific instructions or not.

The Pentium processor with MMX implementation was the design of a new, dedicated, high- performance MMX pipeline, which was able to execute two MMX instructions with minimal logic changes in the existing units. In addition, the design goal was to stay on the microprocessors' performance curve. With the addition of new instructions, the instruction decode logic had to be modified to decode, schedule and issue the new instructions at a rate of up to two instructions per clock.

Frequency Speedup

To simplify the design and to meet the core frequency goal, the pipeline of the Pentium processor w/MMX was extended with a new pipeline stage (length decode). In order to maintain and improve the CPI (Clock per Instruction) of MMX technology is due to modifications that increase the Clock Rate.
As we know,

Execution Time = (**No. of instructions**). (**CPI**). (**Clock Cycle Time**)
i.e., increasing the Clock Rate decreases the Clock Cycle Time, which in turn decreases Execution Time. So, in order to increase **Clock Rate**, the MMX Pentium designers need to find and eliminate some bottlenecks. The two major bottlenecks were the instruction decoder and the data cache access. So they tried to fix the decoder bottleneck first. Here is an instruction that uses old 5-stage pipe like **Fetch, Decode1, Decode2, Execute, Write-Back**.

To speed things up, a 6th stage was added to the pipe i.e. **Prefetch**. A queue was also added between Fetch and Decode1 to decouple freezes. So now an instruction looks like: **Prefetch, Fetch, Decode1, Decode2, Execute, Write-Back** as shown in the figure below. After adding this new stage, machine timing is rebalanced to take advantage of the extra clock cycle.

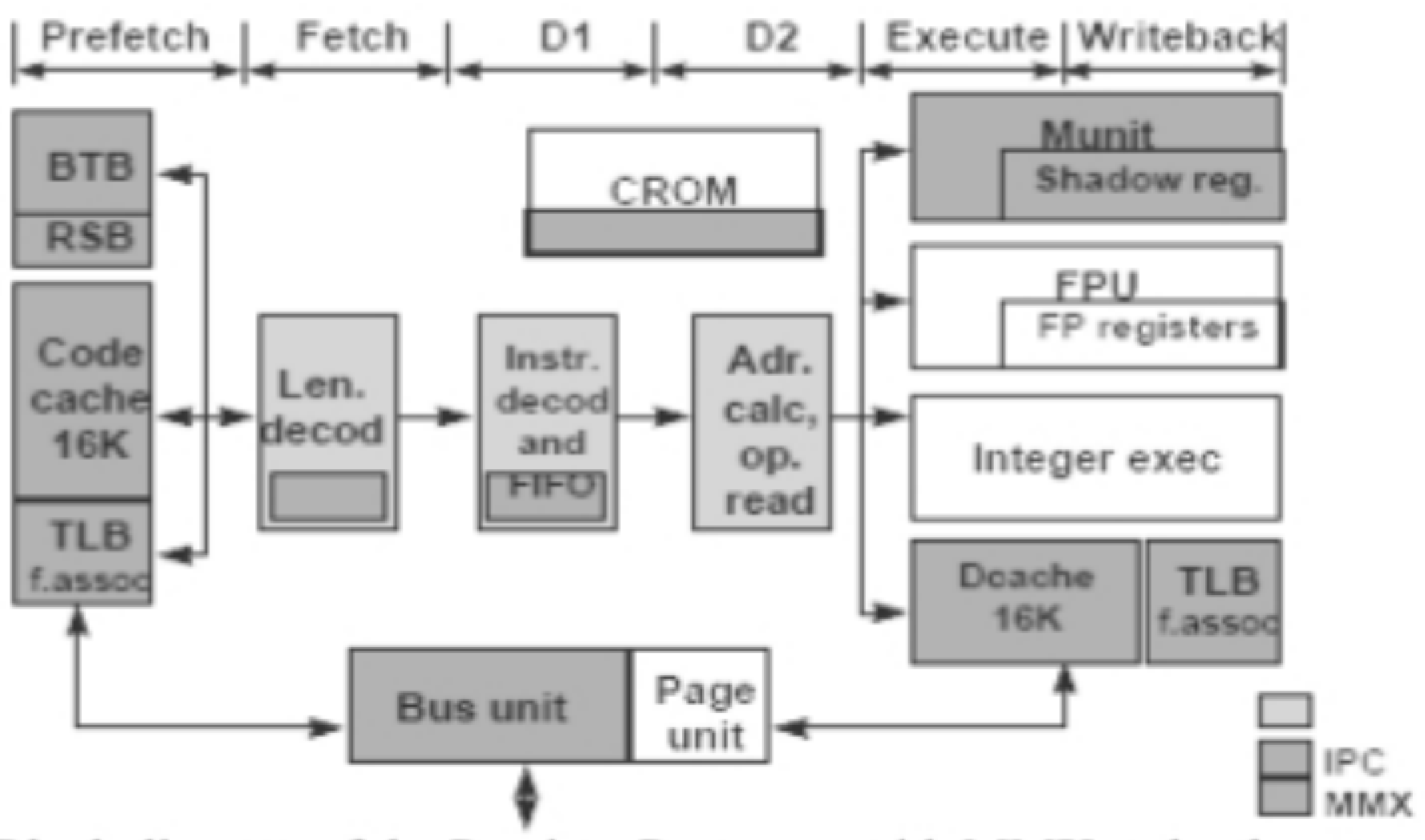


Fig: Block diagram of the Pentium Processor with MMX technology

Although adding a pipeline stage improves frequency, it decreases CPI performance, i.e., the longer the pipeline, the more work done speculatively by the machine and therefore more work is being thrown away in the case of branch miss-prediction. The additional pipeline stage costs decreased the CPI performance of the processor by 5-6%.

c) CM-2 machine:
The CM-2 was SIMD architecture based machine. The PEs in the CM-2 was capable of performing bit-serial arithmetic. The control processor, or sequencer could decompose an 8-bit operation, for example, into 8 PE nano-instructions. The CM-2 provides the mechanism for the programmer to assign PEs to groups that will execute at different times. This functionality is achieved through the use of PE instruction masking.

Although the PEs and the PE module floating point accelerator provide extensive processing capability, the programming paradigm was still limited to SIMD. The CM-2 distinguishes itself from its predecessors through the use of systematic inclusion of error-detecting and error-correcting circuits within the memories and communication networks. The CM-2 is capable of achieving a peak processing speed of around 10GFlops.

The CM-2 machine provides the hypercube connections between different processing elements (PEs). The PEs were organized into modules each having 32 PEs. Within a given module, the PEs were organized into to 16-PE sets with each set having its own router node. All the PEs within a given set use shared memory to communicate with one another by writing values into their respective local memories. Each router node represented a vertex in the hypercube. One interesting feature of the routers was that they provided special circuitry for message combining for messages with the same destination. In addition to the communication via local memories of PEs within a given module, the CM-2 also supports patterned communications directly across the wires of the hypercube.

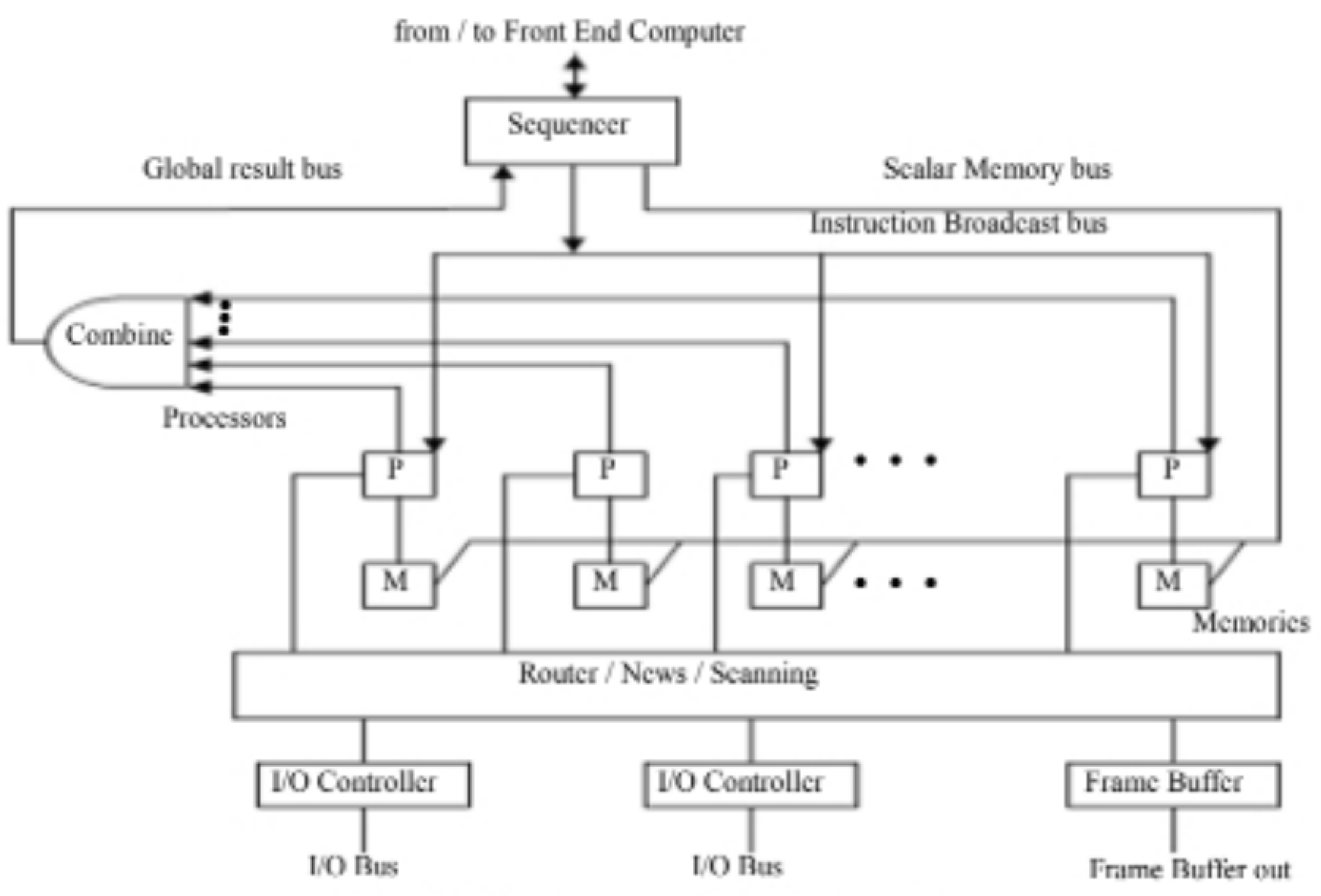


Fig: Block diagram of CM-2 machine

d) Flynn's classification: ***Refer to Question No.1 of Short Answer Type Questions.***

RISC & CISC ARCHITECTURES

☞ Chapter at a Glance

Non von Neumann architecture characteristics

Any computer architecture in which the underlying model of computation is different from what has come to be called the standard von Neumann model. A non von Neumann machine may thus be without the concept of sequential flow of control (i.e. without any register corresponding to a "program counter" that indicates the current point that has been reached in execution of a program) and/or without the concept of a variable (i.e. without "named" storage locations in which a value may be stored and subsequently referenced or changed). Examples of non von Neumann machines are the dataflow machines and the reduction machines. In both of these cases there is a high degree of parallelism, and instead of variables there are immutable bindings between names and constant values.

Cluster Computer

A cluster computer consists of a set of loosely connected computers that work together so that in many respects they can be viewed as a single system. The components of a cluster are usually connected to each other through fast local area networks, each node (computer used as a server) running its own instance of an operating system. Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low cost microprocessors, high speed networks, and software for high performance distributed computing. Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability. Computer clusters have a wide range of applicability and deployment, ranging from small business clusters with a handful of nodes to some of the fastest supercomputers in the world.

Multiple Choice Type Questions

1. Overlapped register windows are used to speed-up procedure call and return in **[WBUT 2007, 2011]**

a) RISC architectures **b) CISC architectures**
c) both (a) and (b) **d) none of these**

Answer: (a)

2. What is a main advantage of classical vector systems (VS) compared with RISC based systems (RS)? **[WBUT 2008, 2009]**

a) VS have significantly higher memory bandwidth than RS
b) VS have higher clock rate than RS
c) VS are more parallel than RS
d) None of these

Answer: (a)

3. Difference between RISC and CISC is **[WBUT 2010]**

a) RISC is more complex **b) CISC is more effective**
c) RISC is better optimizable **d) none of these**

Answer: (a)

4. The advantage of RISC over CISC is that **[WBUT 2011]**

a) RISC can achieve pipeline segments, requiring just one clock cycle
b) CISC uses many segments in its pipeline with the longest segment requiring two or more clock cycle
c) both (a) & (b)
d) none of these

Answer: (d)

5. Which of the following is not RISC architecture characteristic? **[WBUT 2012, 2018]**

a) simplified and unified format of code of instructions
b) no specialized register
c) no storage / storage instruction
d) small register file

Answer: (d)

6. Which of the following architectures correspond to von-Neumann architecture? **[WBUT 2012]**

a) MISD **b) MIMD** **c) SISD** **d) SIMD**

Answer: (c)

7. The CPI value for RISC processors is **[WBUT 2015]**

a) 1 **b) 2** **c) 3** **d) more**

Answer: (a)

8. In which of the following shared memory multiprocessor models the time to access shared memory is same? **[WBUT 2019]**

a) NORMA **b) COMA** **c) UMA** **d) NUMA**

Answer: (c)

Short Answer Type Questions

1. Compare between RISC and CISC. **[WBUT 2010, 2012, 2014, 2015, 2018]**

OR,

Compare RISC and CISC architecture in brief. **[WBUT 2019]**

Answer:

Characteristics	CISC	RISC
Instruction set size and instruction formats	Instruction set is very large and instruction format is variable (16 – 64 bit per instruction)	Instruction set is small and instruction format is fixed.
Addressing mode	12 - 24	3- 5
General purpose registers and cache design	8-24 general purpose registers present. Unified cache is used for instruction and data	Though most instructions are register base so large numbers of registers (32 – 192) are used and cache is split in data cache and instruction cache.
CPI	CPI is between 2 to 15	In most cases CPI is 1 but average CPI is less than 1.5
CPU control	CPU is controlled by control memory (ROM) using microprograms.	CPU is controlled by hardware without control memory

2. What are multiprocessor, multi-computer and multi-core systems? **[WBUT 2012, 2014]**

Answer:

In Multiprocessor system there are more than one processor that works simultaneously. In this system there is one master processor and other are the Slave. If one processor fails then master can assign the task to other slave processor. But if Master will be fail than entire system will fail. Central part of Multiprocessor is the Master. All of them share the hard disk, memory and other devices.

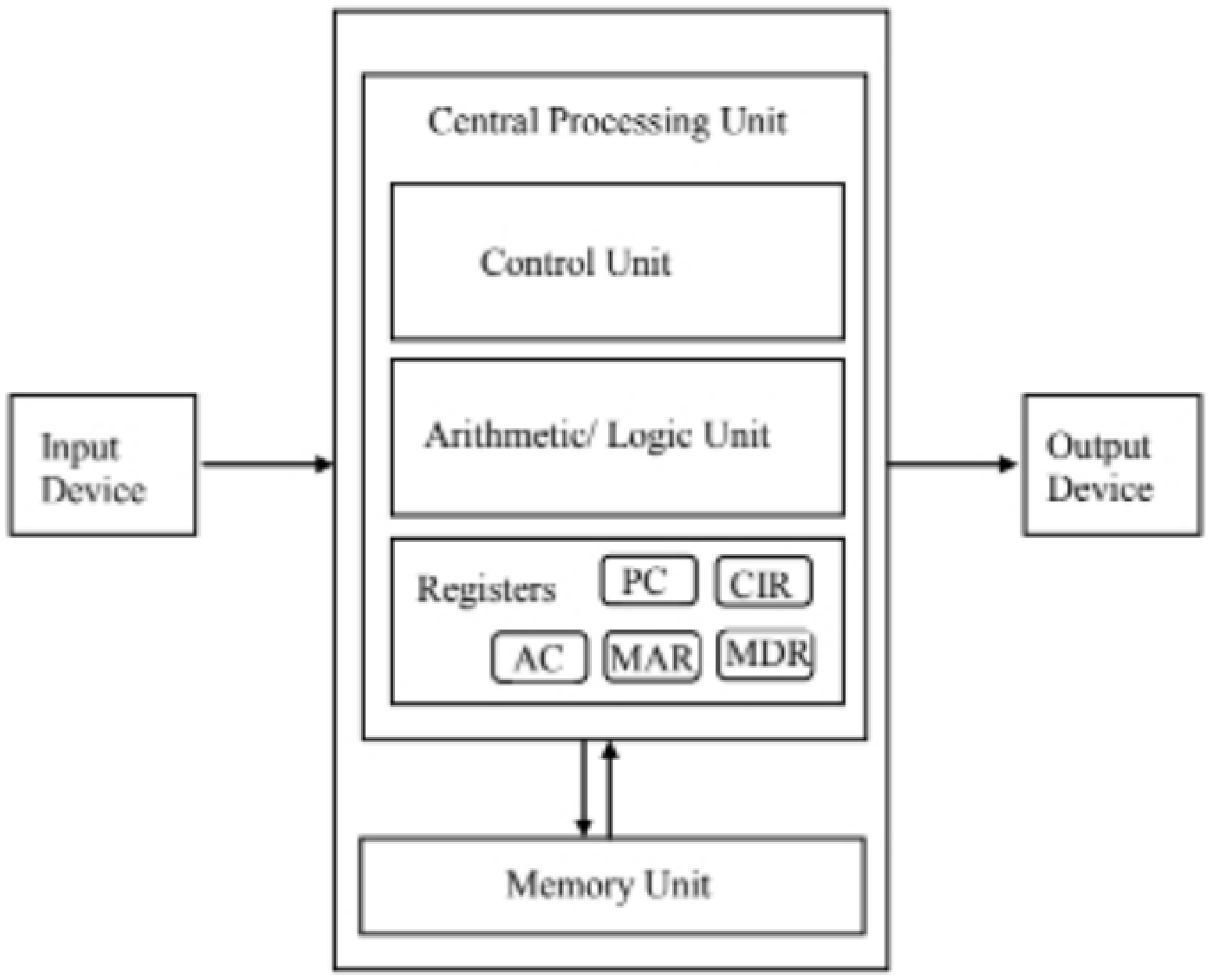
A multicomputer system consisting of more than one computer, usually under the supervision of a master computer, in which smaller computers handle input/output and routine jobs while the large computer carries out the more complex computations.

A multi-core processor is a single computing component with two or more independent actual central processing units (called "cores"), which are the units that read and execute program instructions. The instructions are ordinary CPU instructions such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing. Manufacturers typically integrate the cores onto a single integrated circuit die or onto multiple dies in a single chip package.

3. What is Von-Neumann architecture? What is a Von-Neumann bottleneck? How can this be reduced? [WBUT 2019]

Answer:

Von Neumann architecture was first published by John von Neumann in 1945. His computer architecture design consists of a Control Unit, Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/Outputs. Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today. The Central Processing Unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program. It is sometimes referred to as the microprocessor or processor. The CPU contains the ALU, CU and a variety of registers. Registers are high speed storage areas in the CPU. All data must be stored in a register before it can be processed. Arithmetic and Logic Unit (ALU) allows arithmetic (add, subtract etc.) and logic (AND, OR, NOT etc.) operations to be carried out. The control unit controls the operation of the computer's ALU, memory and input/output devices, telling them how to respond to the program instructions it has just read and interpreted from the memory unit. The control unit also provides the timing and control signals required by other computer components. Buses are the means by which data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory. A standard CPU system bus is comprised of a control bus, data bus and address bus.



CPUs processing speed is much faster in comparison to the main memory (RAM) as a result the CPU needs to wait longer to obtain data-word from the memory. The CPU and memory speed disparity is known as Von Neumann bottleneck. This problem can be solved in two ways:

1. Use of cache memory between CPU and main memory
2. Using RISC computers

This performance problem can be reduced by introducing a cache memory (special type of fast memory) in between the CPU and the main memory. This is because the speed of the cache memory is almost same as that of the CPU. So there is no waiting time for CPU and data-word to come to it for processing. Another way of solving the problem is by using special type of computer known as Reduced Instruction Set Computers (RISC).
The main intention of the RISC is to reduce the total number of memory references made by the CPU; instead it uses large number of registers for the same purpose.

Long Answer Type Questions

1. a) What is SPEC rating? Explain. **[WBUT 2015]**
b) A 50 MHz processor was used to execute a program with the following instruction mix and clock cycle counts:

Instruction type	Instruction count	Clock cycle count
Integer arithmetic	50000	1
Data transfer	35000	2
Floating point arithmetic	20000	2
Branch	6000	3

Calculate the effective CPI, MIPS rate and execution time for this program.

Answer:

a) The Standard Performance Evaluation Corporation (SPEC) is an American non-profit organization that aims to "produce, establish, maintain and endorse a standardized set" of performance benchmarks for computers. SPEC was founded in 1988. SPEC benchmarks are widely used to evaluate the performance of computer systems; the test results are published on the SPEC website. Results are sometimes informally referred to as "SPECmarks" or just "SPEC". SPEC evolved into an umbrella organization encompassing four diverse groups; Graphics and Workstation Performance Group (GWPG), the High Performance Group (HPG), the Open Systems Group (OSG) and the newest, the Research Group (RG).

b) Total instruction count= 111000
CPI=(50000 x 1+ 35000 × 2 +20000 × 2 + 6000 × 3) / 111000=1.6
MIP= (clock frequency)/ (CPI × 1000000) = (50 × 1000000) / (1.6 × 1000000) = 31.25
Execution time = CPI × Instruction count × Clock time
= 1.6 × 111000 × (1/ (50 × 1000000)) = 0.003ms

2. a) Explain different types of addressing modes?
b) What are the advantages of Relative addressing mode over Direct addressing mode? **[WBUT 2017]**

Answer:

a) The term addressing modes refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. Following are the main addressing modes that are used on various platforms and architectures.

1) Immediate Mode: The operand is an immediate value is stored explicitly in the instruction.

2) Index Mode: The address of the operand is obtained by adding to the contents of the general register (called index register) a constant value. The number of the index register and the constant value are included in the instruction code.

3) Indirect Mode: The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses. The register or memory location that contains the address of the operand is a pointer. When an execution takes place in such mode, instruction may be told to go to a specific address.

4) Direct Mode: The address of the operand is embedded in the instruction code.

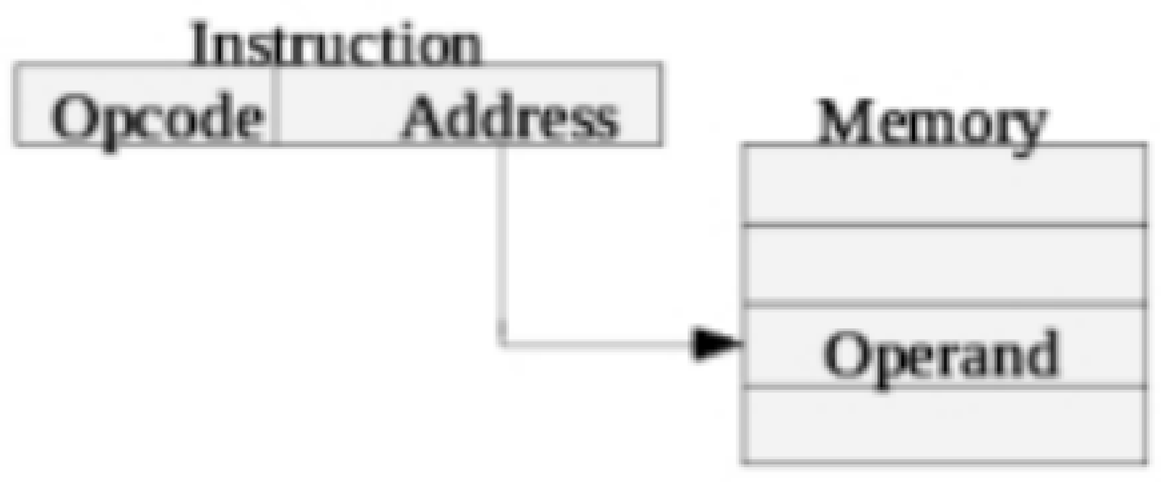
5) Register Mode: The name of the CPU register is embedded in the instruction. The register contains the value of the operand. The number of bits used to specify the register depends on the total number of registers from the processor set.

b) Relative Addressing Mode: In this mode, the content of the program counter (PC) is added to the address part of the instruction to obtain the effective address. When the number is added to the content of the PC, the result is an effective address whose position in memory is relative to the address of the next instruction.

Effective Address (EA) = PC + A

Direct Addressing Mode: In this mode, the address of the memory location which holds the operand is included in the instruction. The operand resides in memory and its address is given by the address field of the instruction.

For example LDA 4000H



Advantage: Simple and provides more flexibility compared to immediate mode.

Disadvantage: Limited address field.

3. Write short notes on the following:

a) Power PC	**[WBUT 2007, 2010]**
b) Non von Neumann architecture characteristics	**[WBUT 2012]**
c) Cluster Computer	**[WBUT 2012, 2018]**
d) RISC	**[WBUT 2019]**

Answer:

a) Power PC:

The PowerPC microprocessor is a highly integrated single-chip processor that combines a powerful RISC architecture, a superscalar machine organization, and a versatile high-performance bus interface. The processor contains a 32KB unified cache and is capable of dispatching, executing, and completing up to 3 instructions per cycle. The bus interface configurations provide a wide range of system bus interfaces, including pipelined, non-pipelined, and split transactions. The result is a cost effective, general purpose microprocessor solution that offers very competitive performance.

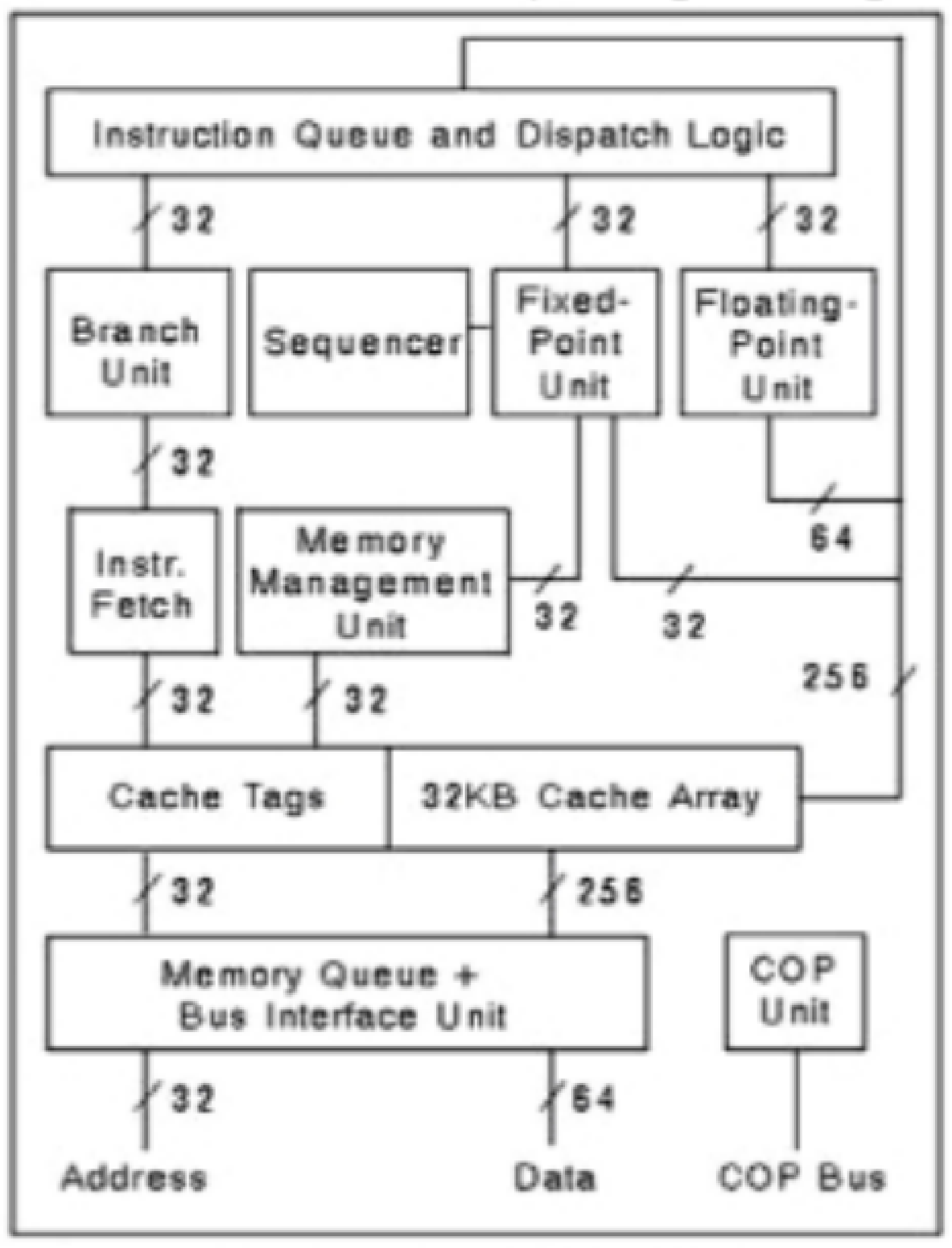


Fig: Power PC Architecture

As shown in the above figure, it is a superscalar design with three pipelined execution units. The processor can dispatch up to three 32-bit instructions each cycle - one each to the Fixed-Point Unit (FXU), the Floating-Point Unit (FPU), and the branch unit (BPU). The 32KB unified cache provides a 32-bit interface to the FXU, a 64-bit interface to the FPU, and a 256-bit interface to both the instruction queue and the memory queue. The chip I/Os include a 32-bit address bus and a 64-bit data bus. The designers optimized the 601 pipeline structure for high performance and concurrent instruction processing in each of the execution units as shown below.

- The fixed-point pipeline performs all integer arithmetic logic unit (ALU) operations and all processor load and store instructions, including floating-point loads and stores.
- The branch instruction pipeline has only two stages. The first stage can dispatch, decode, evaluate, and, if necessary, predict the direction of a branch instruction in one cycle. On the next cycle, the resulting fetch can be accessing new instructions from the cache.

- The floating-point instruction pipeline contains six stages and has been optimized for fully pipelined execution of single-precision operations.

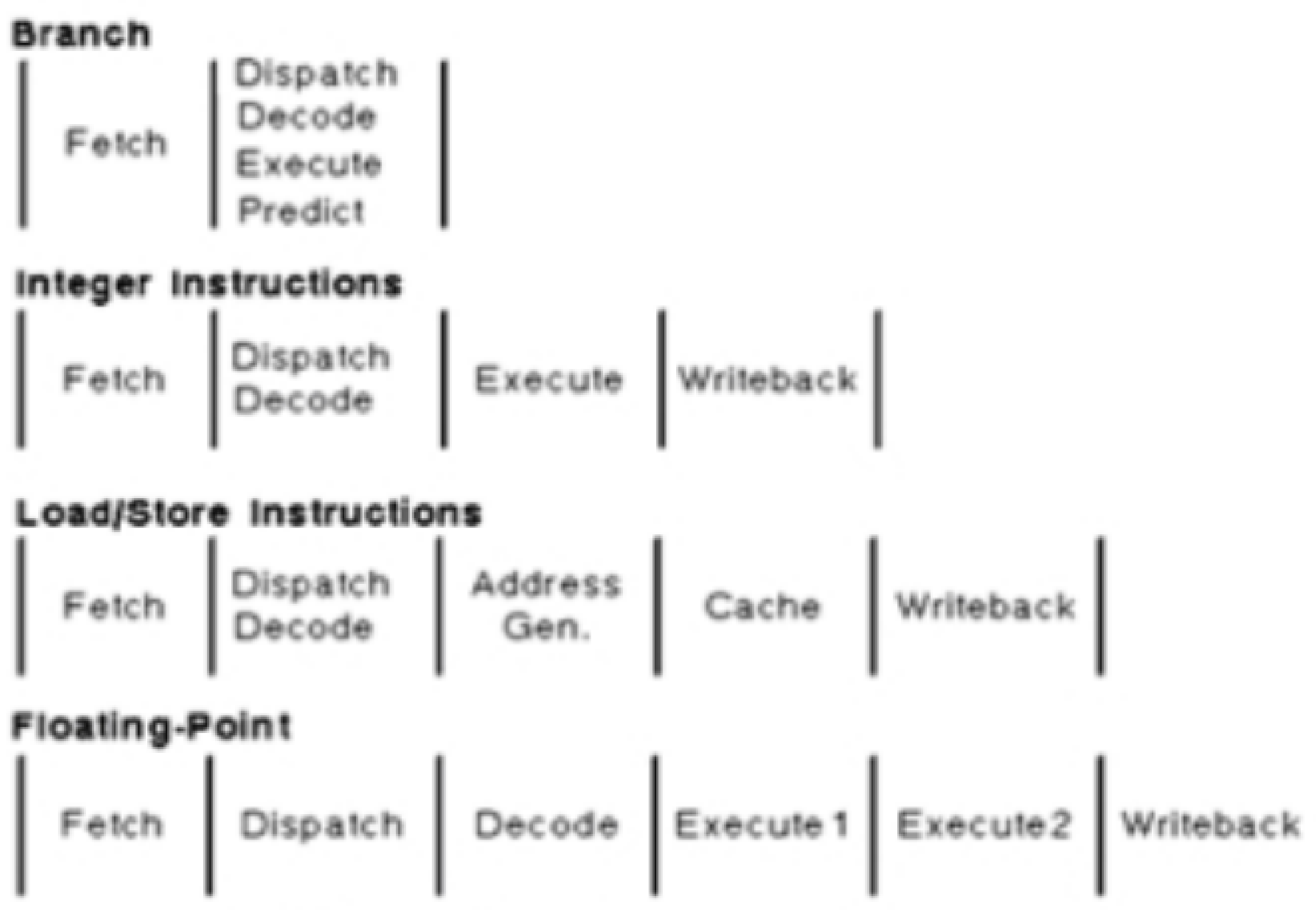


Fig: PowerPC 601 pipeline Architecture

b) Non von Neumann architecture characteristics:
Any computer architecture in which the underlying model of computation is different from what has come to be called the standard von Neumann model. A non von Neumann machine may thus be without the concept of sequential flow of control (i.e. without any register corresponding to a "program counter" that indicates the current point that has been reached in execution of a program) and/or without the concept of a variable (i.e. without "named" storage locations in which a value may be stored and subsequently referenced or changed).
Examples of non von Neumann machines are the dataflow machines and the reduction machines. In both of these cases there is a high degree of parallelism, and instead of variables there are immutable bindings between names and constant values.
Note that the term non von Neumann is usually reserved for machines that represent a radical departure from the von Neumann model, and is therefore not normally applied to multiprocessor or multicomputer architectures, which effectively offer a set of cooperating von Neumann machines.

c) Cluster Computer:
A cluster computer consists of a set of loosely connected computers that work together so that in many respects they can be viewed as a single system. The components of a cluster are usually connected to each other through fast local area networks, each node (computer used as a server) running its own instance of an operating system. Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low cost microprocessors, high speed networks, and software for high performance distributed computing. Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability. Computer

clusters have a wide range of applicability and deployment, ranging from small business clusters with a handful of nodes to some of the fastest supercomputers in the world. Computer clusters may be configured for different purposes ranging from general purpose business needs such as web-service support, to computation-intensive scientific calculations. In either case, the cluster may use a high-availability approach. Note that the attributes described below are not exclusive and a "compute cluster" may also use a high-availability approach, etc. "Load-balancing" clusters are configurations in which cluster-nodes share computational workload to provide better overall performance. For example, a web server cluster may assign different queries to different nodes, so the overall response time will be optimized. However, approaches to load-balancing may significantly differ among applications, e.g. a high-performance cluster used for scientific computations would balance load with different algorithms from a web-server cluster which may just use a simple round-robin method by assigning each new request to a different node.

d) RISC:

RISC, or Reduced Instruction Set Computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures. The first RISC projects came from IBM, Stanford, and UC-Berkeley in the late 70s and early 80s. The IBM 801, Stanford MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristic of most RISC processors:

- One cycle execution time: RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called ;
- Pipelining: a technique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions.
- Large number of registers: the RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory.

Characteristics of RISC

- Simpler instruction, hence simple instruction decoding.
- Instructions come under size of one word.
- Instructions take single clock cycle to get executed.
- More number of general purpose register.
- Simple Addressing Modes.
- Less Data types.
- Pipeline can be achieved.

INTERPROCESS COMMUNICATION

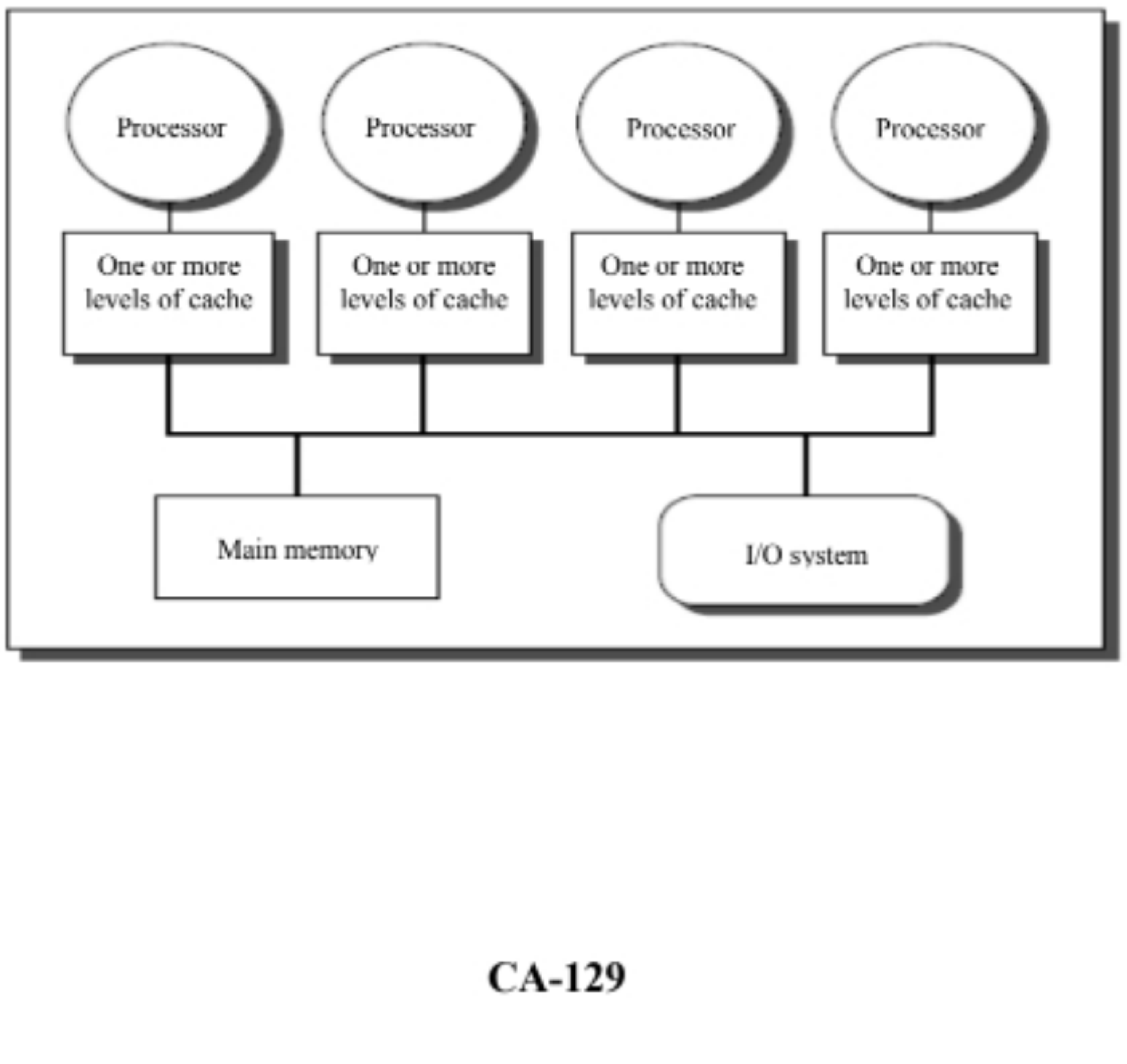
☞ Chapter at a Glance

Centralized shared memory Architecture

A centralized shared memory design can be scaled to a few dozen processors. Although scaling beyond that is technically conceivable, sharing a centralized memory, even organized as multiple banks, becomes less attractive as the number of processors sharing it increases. Because there is a single main memory that has a symmetric relationship to all processors and a uniform access time from any processor, these multiprocessors are often called symmetric (shared-memory) multiprocessors (SMPs), and this style of architecture is sometimes called UMA(uniform memory access). This type of centralized shared-memory architecture is currently by far the most popular organization.

So, communication with shared memory can be defined as follows:

- Allows a familiar programming style. Sometimes it is straightforward to make an existing program run on a parallel machine (with a small number of processors).
- Requires synchronization with critical regions or semaphores for shared data.
- The cache can help reduce the amount of communication.
- Complex hardware is needed to keep the caches correct.


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O system

Multiple Choice Type Questions

1. In general an n input Omega network requires .... stages of 2*2* switches. **[WBUT 2011, 2016]**

a) 2 b) 4 c) 8 d) 16

Answer: (a)

2. The time to access shared memory is same in which of the following shared memory multiprocessor models? **[WBUT 2012, 2015, 2018]**

a) NUMA b) UMA c) COMA d) ccNUMA

Answer: (b)

3. Example of a recirculating network is **[WBUT 2013]**

a) 3 cube network b) ring network
c) tree network d) mess connected Illiac network

Answer: (b)

4. In general 64 input Omega network requires .... stages of 2×2 switches. **[WBUT 2013]**

a) 6 b) 64 c) 8 d) 7

Answer: (a)

5. The UMA multiprocessor system is best suited **[WBUT 2015]**

a) when the degree of interaction among different modules in program is large
b) when the degree of interaction among different modules in program is less
c) when there is no interaction among different modules in program
d) when different programs are to be executed concurrently

Answer: (d)

6. Which of the following is a recursive network? **[WBUT 2017]**

a) Benes network b) Baseline network
c) Cross bar network d) none of these

Answer: (d)

Short Answer Type Questions

1. With architecture and timing diagram explain S-access memory organization. **[WBUT 2005, 2007]**

Answer:

There is a way to arrange the low-order interleaved memory which is called simultaneous access, or S-access, as illustrated in figure below. In this case all memory modules are accessed simultaneously in a synchronized manner. Again the high-order (n- a) bits select the same offset word from each module. At the end of each memory cycle as shown in the figure below, m = 2a consecutive words are latched in the data buffers simultaneously. The low-order a bits are then used to multiplex the m words out, one per each minor cycle. If the minor cycle is chosen to be 1/m of the memory cycle, then it takes two memory cycles to access m consecutive words. If the access phase of the last access is overlapped with the fetch phase of the current access, effectively m words take only one memory cycle to access. The throughput is decreases if stride is greater than 1.

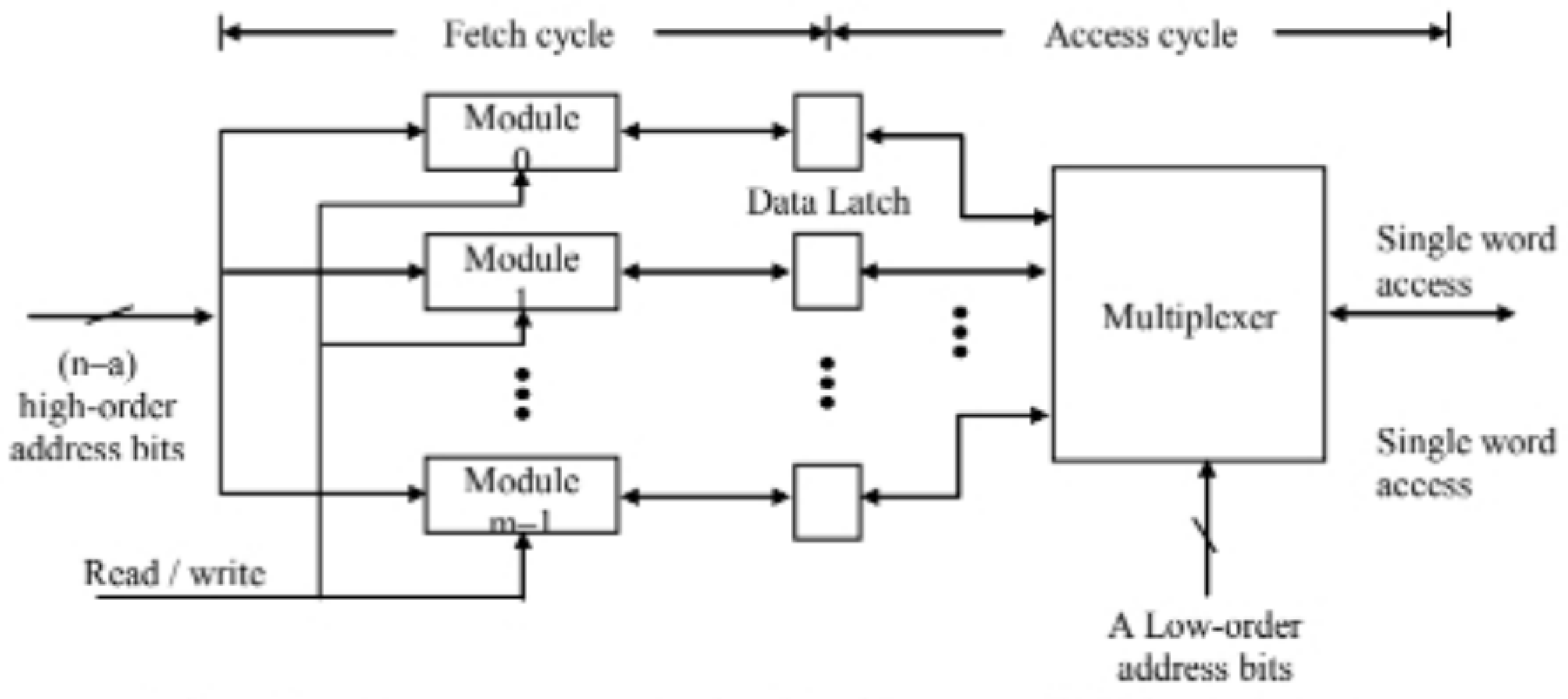


Fig: S-Access memory organization for m-way interleaved memory

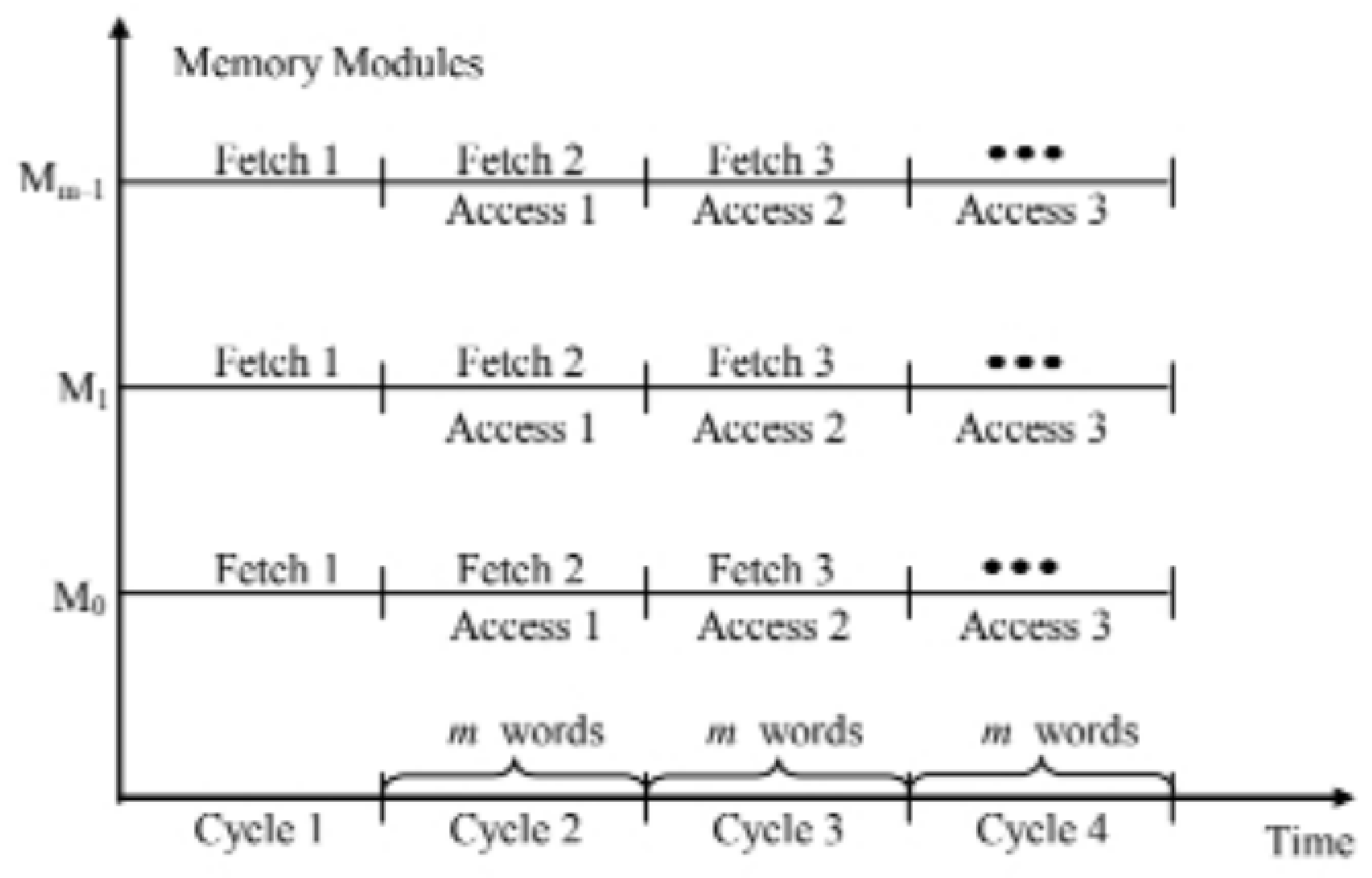


Fig: Successive memory access operation using overlapped fetch and access cycle

2. Develop $3^2 \times 4^2$ delta network. **[WBUT 2008]**

Answer:

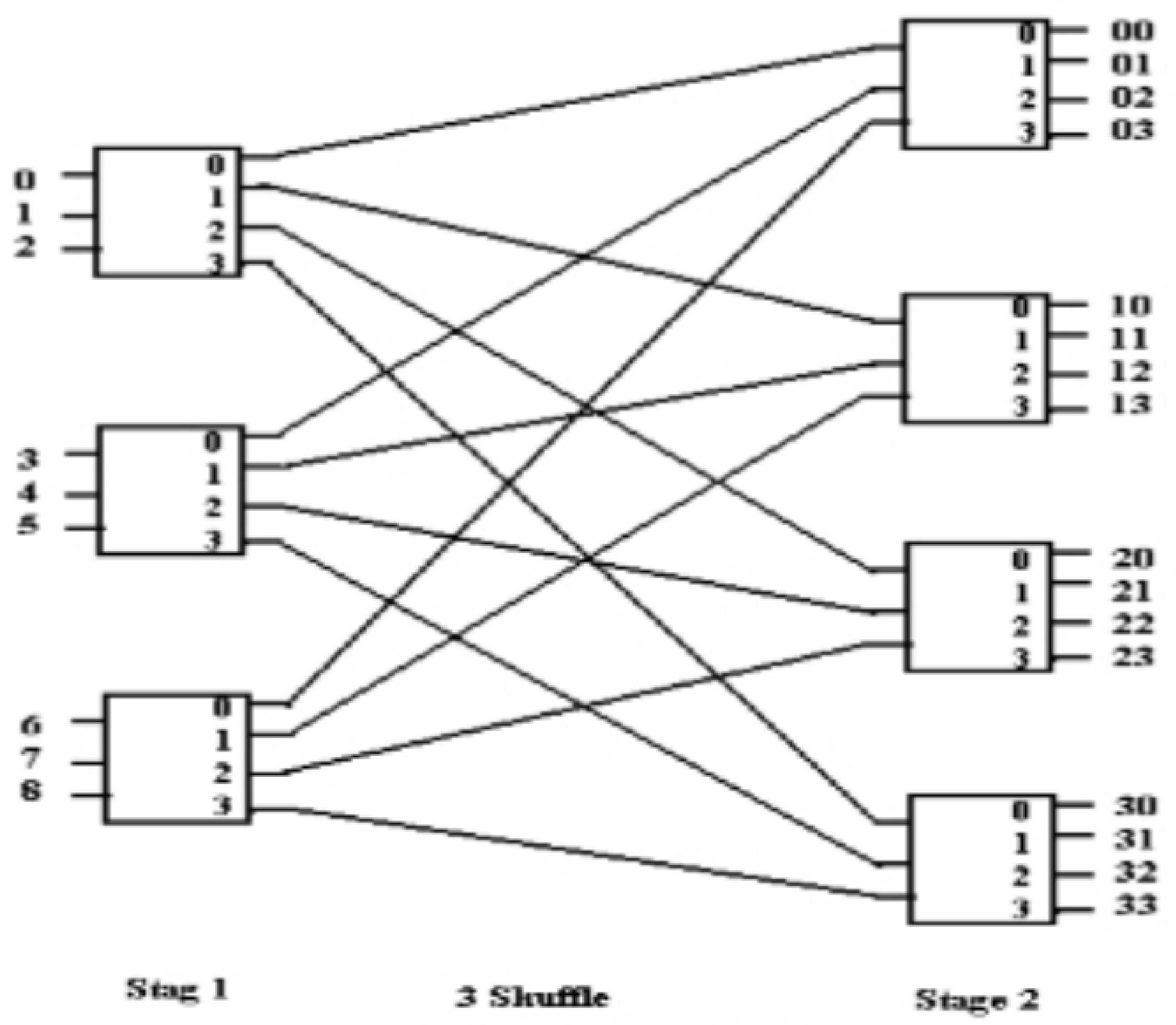


Fig: $3^2 \times 4^2$ delta network

In the delta network, there are a^n x b^n switches with n stages consisting of a x b crossbar modules. There is a unique interconnection path of constant length between the stages of the network. In delta network no input or output terminal of any crossbar module is left unconnected. To construct an a^n x b^n delta network there are *a* shuffle as the link pattern between every two consecutive stages of the network. In an a^n x b^n delta network, there are a^n sources and b^n destinations. Numbering the stages of the network as 1,2,…,n, starting at the source side of the network requires that there be a^{n-1} crossbar modules in the first stage. So there is a^{n-1} b output terminal in the first stage and this is the input terminals in second stage. So, the i-th stage has a^{n-I} b^{i-1} crossbar modules.

Now, in the above example, a = 3, b = 4 and n = 2. So, there are 2 stages in $3^2 \times 4^2$ delta network and the inter stage link pattern is a – shuffle i.e.3 –shuffle.

3. What is Multistage Switching Network? **[WBUT 2008]**

Answer:

A multistage Switching network capable of supporting massive parallel processing, including point-to-point and multicast communications between processor modules (PMs) which are connected to the input and output ports of the network. The network supports a circuit-switching model of communication and can provide parallel paths among input and output ports. It uses an address-based routing algorithm for path setup which makes this network suitable for designing high-speed switching systems. These switches are commonly used in telephony and multiprocessor systems. The network is built using interconnected switch nodes arranged in 2 $|\log_b N|$ stages, wherein b is the number of switch node input/output ports, N is the number of network input/output ports and $|\log_b N|$ indicates a ceiling function providing the smallest integer not less than $\log_b$

N. The additional stages provide additional paths between network input ports and network output ports, thereby enhancing fault tolerance and lessening contention.

4. Draw the block diagram of C-access memory function. Why is it necessary and how does it improve the memory access time? [WBUT 2008]

Answer:

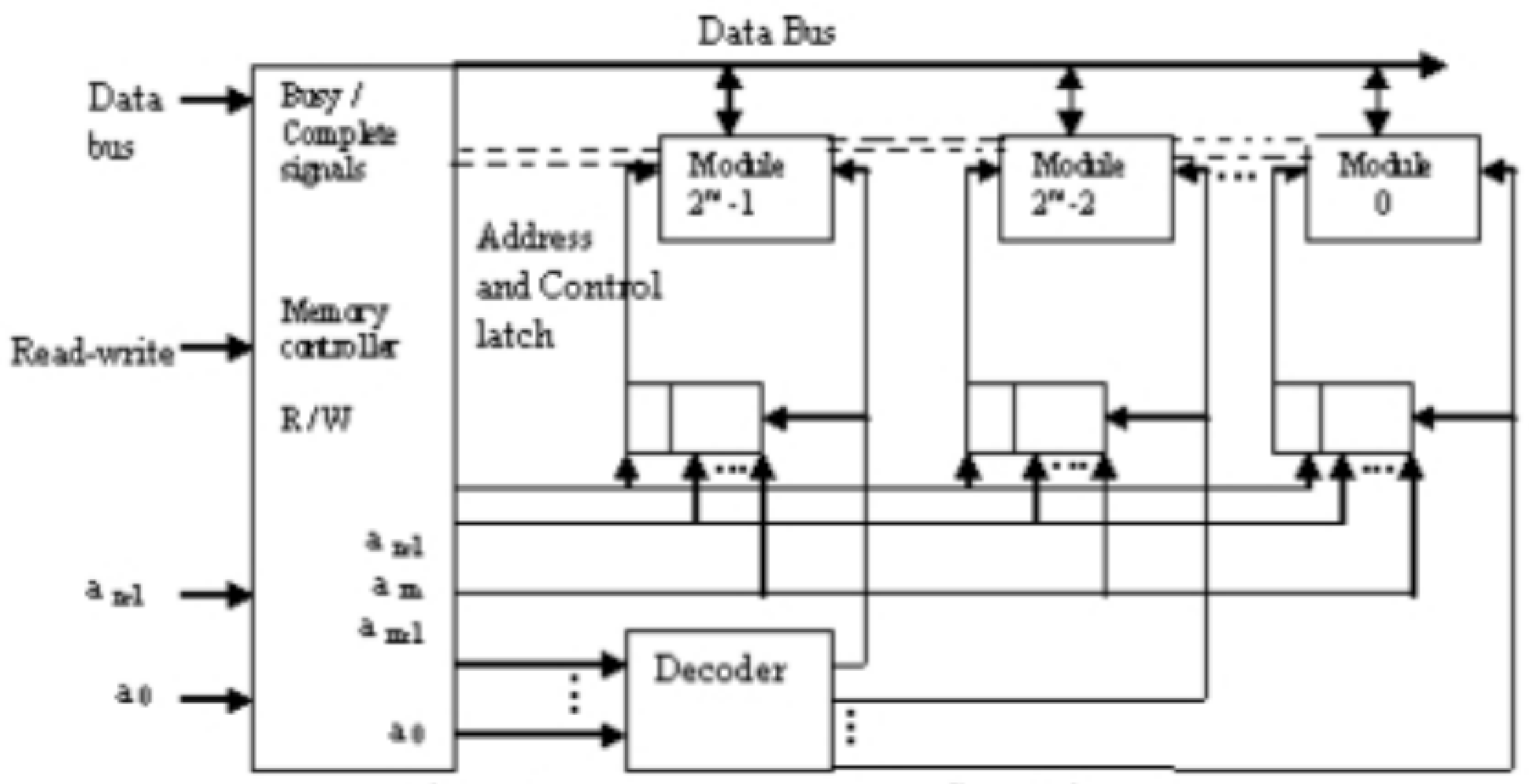


Fig.: C-access memory Configuration

In C-access memory configuration, the access of main memory modules is overlapped in a pipeline fashion. So, the memory cycle i.e. the major cycle is sub- divided into *m* minor cycles.

Let, θ be the major cycle and τ is the minor cycle, then we can write $\tau = \theta / m$, where *m* is the degree of interleaving. The major cycle θ is the total time required to complete the access of a single word from a module. The minor cycle τ is the actual time needed to produce one word, i.e. the overlap access of successive memory modules separated in every minor cycle.

Here we give an example in the figure below of the timing of the pipelined access of the eight contiguous memory words in a C-access memory organization. The pipelined access of the block of eight contiguous words is merged between other pipelined block access before and after the present block. Even though the total block access time is 2θ, the effective access time of each word is reduce to τ as the memory is contiguously accessed in a pipelined fashion.

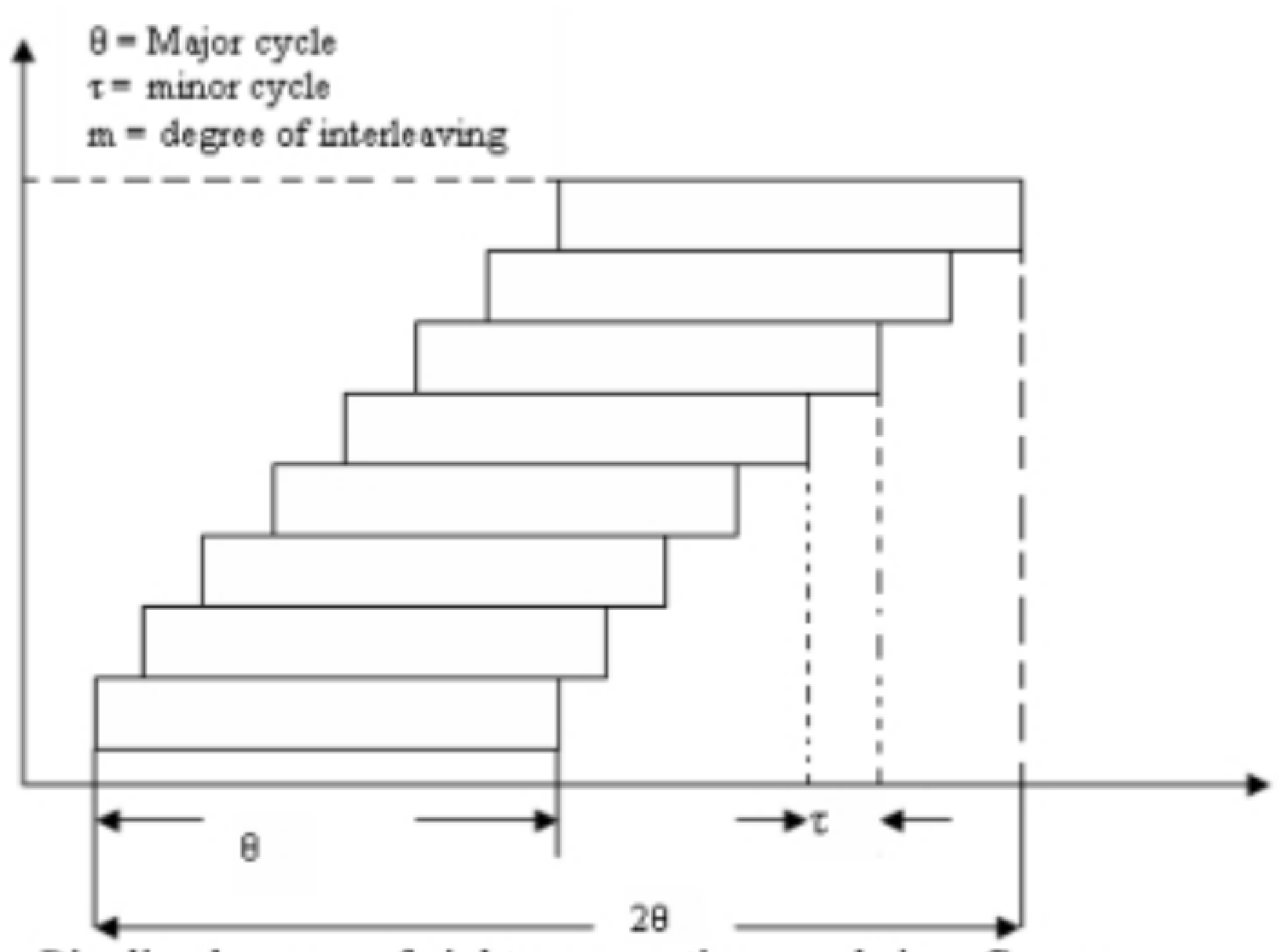


Fig: Pipelined access of eight consecutive words in a C-access memory

5. Differentiate between C-access and S-access memory organizations.
[WBUT 2010]

Answer:

C-access	S-access
More than one module can share a memory bank. It increase the bank utilization and reducing the bank cost. The low order m bit memory address are used to select the module and remaining (n-m) bits address the desired element within the module. The effectiveness of this memory configuration is revealed by its ability to access the elements of a vector.	More than one module can share a memory bank. It increase the bank utilization and reducing the bank cost. The low order m bit memory address are used to select the module and remaining (n-m) bits address the desired element within the module. The single access returns M consecutive words of information from the M memory module. S-access configuration is ideal for accessing a vector of data elements or for pre-fetching sequential instructions for a pipeline process.

6. What are the differences between loosely coupled and tightly coupled architecture? What do you mean by non-uniform memory access, uniform memory access and memory bandwidth? **[WBUT 2011]**

OR,

a) What are the differences between loosely coupled and tightly coupled architecture? **[WBUT 2013, 2014, 2016, 2018]**
b) Compare & contrast between UMA & NUMA with examples. What is Dumb memory? **[WBUT 2013]**

Answer:

MIMD computers with shared memory are known as **tightly coupled machines**. Examples are ENCORE, MULTIMAX. Tightly-coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory (SMP or UMA), or may participate in a memory hierarchy with both local and shared memory (NUMA).

MIMD computers with an interconnection network are known as **loosely coupled machines**. Examples are INTEL iPSC, NCUBE/7. Loosely-coupled multiprocessor systems referred to as clusters are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system. A Linux Beowulf cluster is an example of a loosely-coupled system. Tightly-coupled systems perform better and are physically smaller than loosely-coupled systems, but have historically required greater initial investments and may deflate rapidly; nodes in a loosely-coupled system are usually inexpensive commodity computers and can be recycled as independent machines upon retirement from the cluster.

Shared memory does not mean that there is a single, centralized memory. The symmetric shared-memory multiprocessors are known as UMAs (uniform memory access). Uniform Memory Access (UMA) is a computer memory architecture used in parallel computers having multiple processors and probably multiple memory chips.

All the processors in the UMA model share the physical memory uniformly. Peripherals are also shared. Cache memory may be private for each processor. In UMA architecture, accessing time to a memory location is independent from which processor makes the request or which memory chip contains the target memory data. It is used in symmetric multiprocessing (SMP).

Uniform Memory Access computer architectures are often contrasted with Non-Uniform Memory Access (NUMA) architectures. UMA machines are, generally, harder for computer architects to design, but easier for programmers to program, than NUMA architectures.

Non-Uniform Memory Access or Non-Uniform Memory Architecture (NUMA) is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory.

NUMA architectures logically follow in scaling from symmetric multiprocessing (SMP) architectures. Modern CPUs operate considerably faster than the main memory they are attached to. Limiting the number of memory accesses provided the key to extracting high performance from a modern computer. The dramatic increase in size of the operating systems and of the applications run on them has generally overwhelmed these cache-processing improvements. NUMA attempts to address this problem by providing separate memory for each processor, avoiding the performance hit when several processors attempt to address the same memory. NUMA can improve the performance over a single shared memory by a factor of roughly the number of processors (or separate memory banks).

7. What is the significance of interconnection network in multiprocessor architecture? [WBUT 2012, 2014, 2017]

Answer:

The purpose of a network is to allow exchanging data between processors in the distributed system. Regarding this data exchange, two important terms need to be introduced: network switching and network routing. Network switching refers to the method of transportation of data between processors in the network.

There are roughly two classes of network switching:

- circuit switching and
- Packet switching

In circuit switching, a connection is established between the source and destination processors which is kept intact during the entire data transmission. During this communication, no other processors can use the allocated communication channel(s). This is like how a traditional telephone works. Some of the early parallel machines used this switching method, but nowadays they mainly use packet switching. In packet switching, data are divided into relatively small packets and a communication channel is allocated only for the transmission of a single packet. Thereafter, the channel may be freely used for another data transmission or for a next packet of the same transmission.

The processors in a parallel architecture must be connected in some manner. Interconnection networks carry data between processors and to memory and the interconnects are made of switches and links (wires, fiber).

Interconnects are classified as static or dynamic.

Static networks consist of point-to-point communication links among processing elements (PEs) and are also referred to as direct networks.

Dynamic networks are built using switches and communication links. Dynamic networks are also referred to as indirect networks. A variety of network topologies have been proposed and implemented. These topologies tradeoff performance for cost. Commercial machines often implement hybrid topologies for reasons of packaging, cost, and available components.

8. What do you mean by Program Flow Mechanism? [WBUT 2013]

Answer:

The Program-Flow Architecture is a Von Neumann or control flow computing model. Here a program is a series of addressable instructions, each of which either specifies an operation along with memory locations of the operands or it specifies conditional transfer of control to some other instruction.

9. Use Bernstein's conditions for determining the maximum parallelism in the following sequence of instructions: [WBUT 2015]

$$A = B \times C$$
$$B = D + E$$
$$C = A + B$$
$$E = F - D$$

Answer:
Bernstein has elaborated the work of data dependency and derived some conditions based on which we can decide the parallelism of instructions or processes. Bernstein conditions are based on the following two sets of variables: i) The Read set or input set R_1 that consists of memory locations read by the statement of instruction I_1. ii) The Write set or output set W_1 that consists of memory locations written into by instruction I_1. The sets R_1 and W_1 are not disjoint as the same locations are used for reading and writing by S_1.
The following are Bernstein Parallelism conditions which are used to determine whether statements are parallel or not:
1) Locations in R_1 from which S_1 reads and the locations W_2 onto which S_2 writes must be mutually exclusive. That means S_1 does not read from any memory location onto which S_2 writes. It can be denoted as: $R_1 \cap W_2 = \varphi$
2) Similarly, locations in R_2 from which S_2 reads and the locations W_1 onto which S_1 writes must be mutually exclusive. That means S_2 does not read from any memory location onto which S_1 writes. It can be denoted as: $R_2 \cap W_1 = \varphi$
3) The memory locations W_1 and W_2 onto which S_1 and S_2 write, should not be read by S_1 and S_2. That means R_1 and R_2 should be independent of W_1 and W_2. It can be denoted as : $W_1 \cap W_2 = \varphi$
To show the operation of Bernstein's conditions, consider the following instructions of sequential program:

I1: $A = B \times C$
I2: $B = D + E$
I3: $C = A + B$
I4: $E = F - D$

Now, the read set and write set of I1, I2, I3 and I4 are as follows:

R1 = {B, C}	W1 = {A}
R2 = {D, E}	W2 = {B}
R3 = {A, B}	W3 = {C}
R4 = {F, D}	W4 = {E}

Now let us find out whether I1 and I2 are parallel or not
$R1 \cap W2 \neq \varphi$
$R2 \cap W1 = \varphi$
$W1 \cap W2 = \varphi$
That means I1 and I2 are not independent of each other.

Similarly for I1 || I3,
$R1 \cap W3 \neq \varphi$
$R3 \cap W1 \neq \varphi$
$W1 \cap W3 = \varphi$
Hence I1 and I3 are not independent of each other.

Similarly for I1 || I4,
$R1 \cap W4 = \varphi$
$R4 \cap W1 = \varphi$
$W1 \cap W4 = \varphi$
Hence I1 and I4 are independent of each other.

For I2 || I3,
$R2 \cap W3 = \varphi$
$R3 \cap W2 \neq \varphi$
$W2 \cap W3 = \varphi$
Hence I2 and I are not independent of each other.

For I2 || I4,
$R2 \cap W4 \neq \varphi$
$R4 \cap W2 = \varphi$
$W2 \cap W4 = \varphi$
Hence I2 and I4 are not independent of each other.

For I3 || I4,
$R3 \cap W4 = \varphi$
$R4 \cap W3 = \varphi$
$W3 \cap W4 = \varphi$
Hence I3 and I4 are independent of each other.

10. Design $2^2 \times 3^2$ Delta network. **[WBUT 2015]**

Answer:

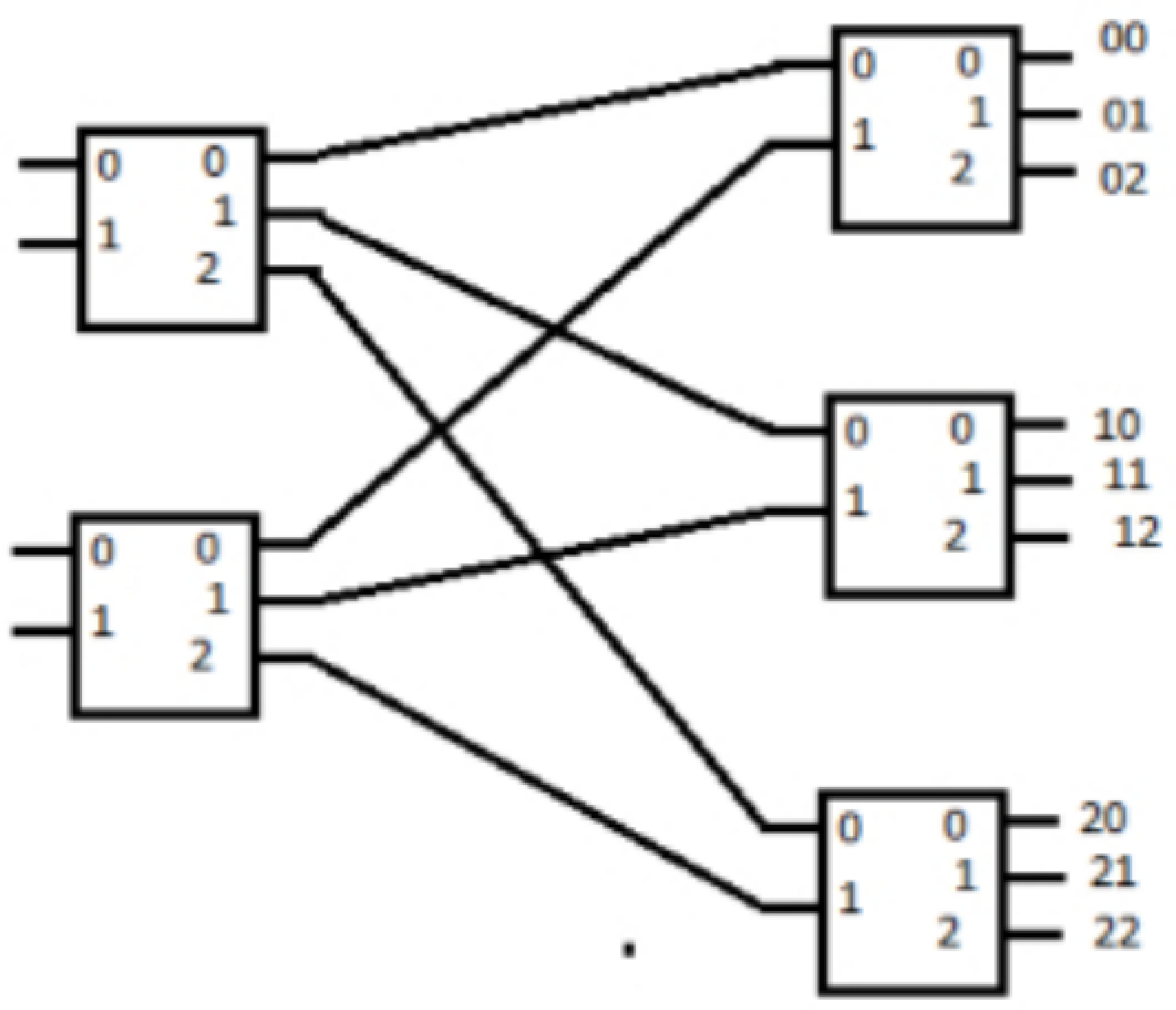


Fig: $2^2 \times 3^2$ Delta network

11. What is the difference between centralized shared memory and distributed shared memory? [WBUT 2016]

Answer:

Refer to Question No. 2(b) of Long Answer Type Questions.

12. Explain the C-access and S-access memory organizations for vector accesses. [WBUT 2018]

Answer:

C-access memory organizations: There are more than one modules can share a memory bank. It increases the bank utilization and reducing the bank cost. The low order m bit memory addresses are used to select the module and remaining (n-m) bits address the desired element within the module. The effectiveness of this memory configuration is revealed by its ability to access the elements of a vector. The vector access scheme from interleaved memory modules is used as m-way low-order interleaved memory structure where it allows m memory words to be accessed concurrently.

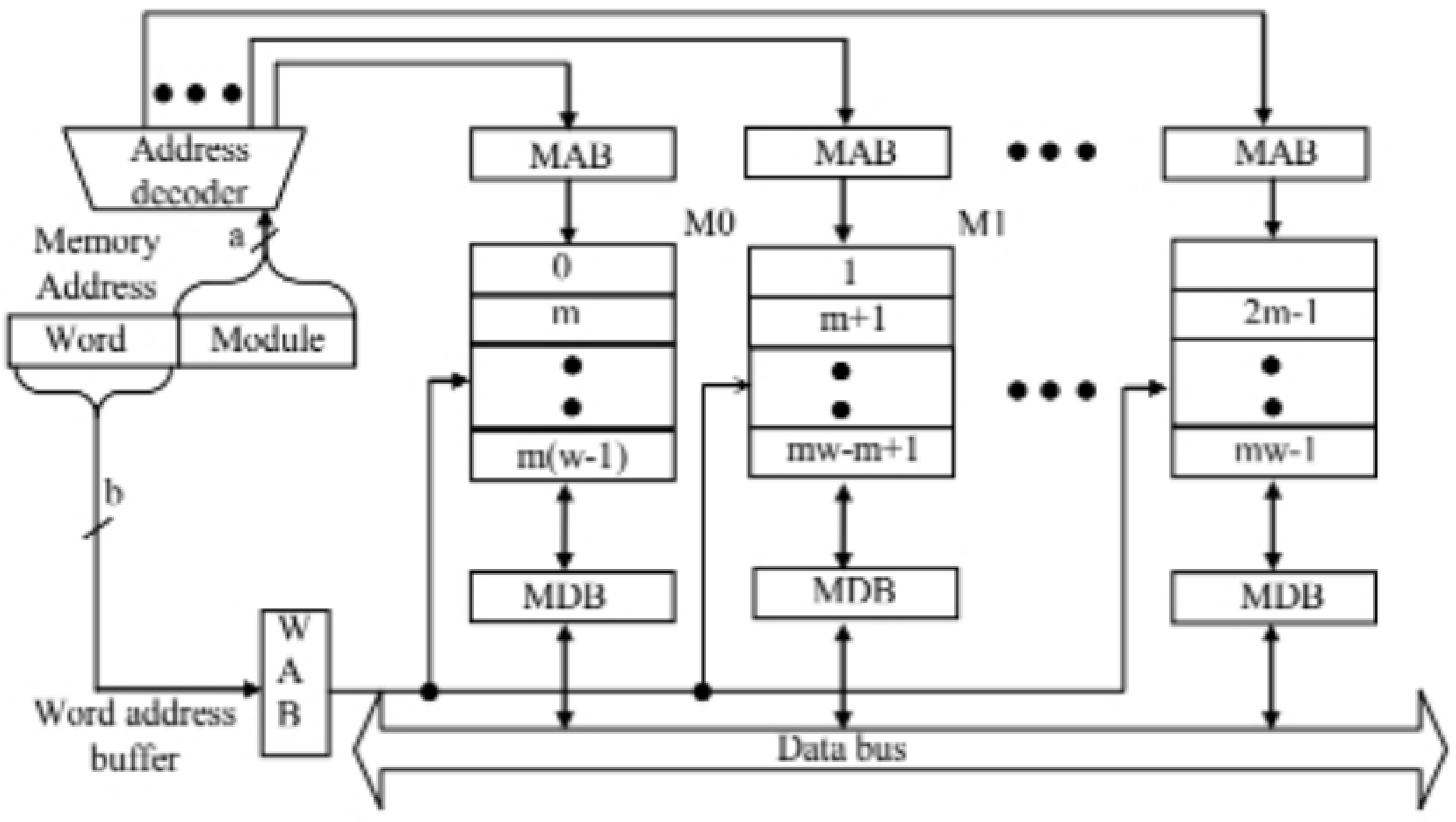


Fig: C-Access Memory Organization

S-access memory organizations: In s-access memory organizations, it is similar to the low-order interleaved memory structure. The high order bits select modules and the words from modules are latched at the same time. The low order bits select words from data latches. This is done through the multiplexed with higher speeds (minor cycles). This type of organization allows simultaneous access of memory.

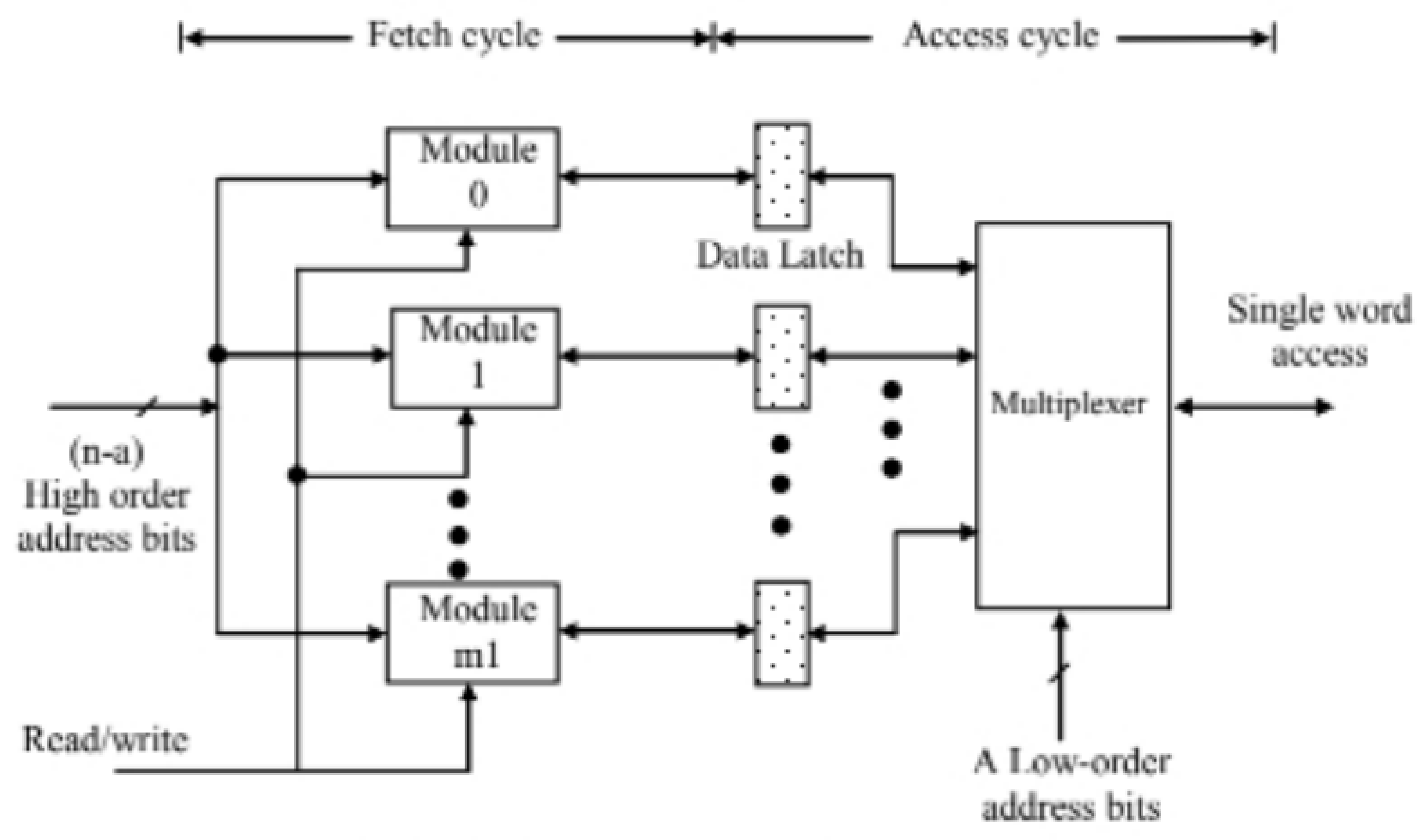


Fig: S-Access Memory Organization

Long Answer Type Questions

1. What is the basic purpose of data flow architecture? Compare it with control flow architecture. [WBUT 2005]

OR,

What is the basic objective of data flow architecture? Compare it with control flow architecture. [WBUT 2005, 2015]

Answer:

The data flow computers are based on a data driven mechanism. The operation of a conventional von Neumann machine, the fundamental difference is that instruction execution in a conventional computer is under program-flow control, whereas that in a data flow computer is driven by the data (operand) availability. There are three basic issues towards the development of an ideal architecture for future computers. The first is to achieve a high performance/cost ratio; the second is to match the ratio with technological progress; and the third is to offer better programmability in application areas. The data flow model offers an approach to meet these demands.

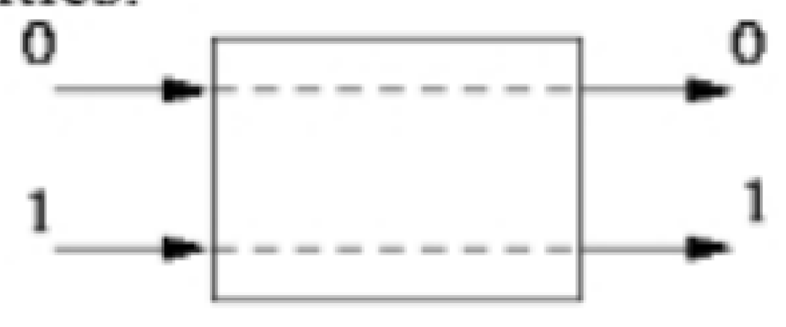
The control flow computers are either uniprocessor or parallel processors architecture. In uniprocessor system the instructions are executed sequentially and this is called control-driven mechanism. In parallel processors system control flow computers use shared memory. So, parallel executed instructions may cause side effects on other instructions and data due to shared memory. In control flow computer the sequence of execution of instructions is controlled by program counter register.

Shared memory cells are the means by which data is passed between instructions. Data (operands) are referenced by their memory addresses (variables). In the traditional sequential control flow model, there is a single thread of control, which is passed from instruction to instruction. There is more than one thread of control to be active at an instant and provide means for synchronizing these threads.

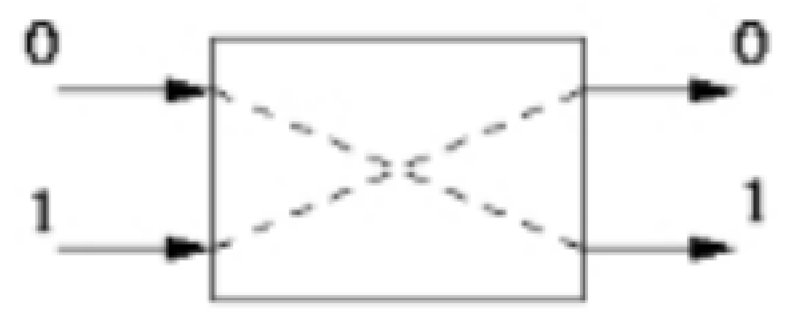
2. a) Compare dynamic connection networks such as multistage interconnection networks and crossbar switch networks in terms of the following characteristics: Bandwidth and Hardware complexity such as switching, arbitration, wires etc.
b) Compare between centralized and distributed shared memory architecture. Which is the best architecture among them and why? [WBUT 2007]

Answer:

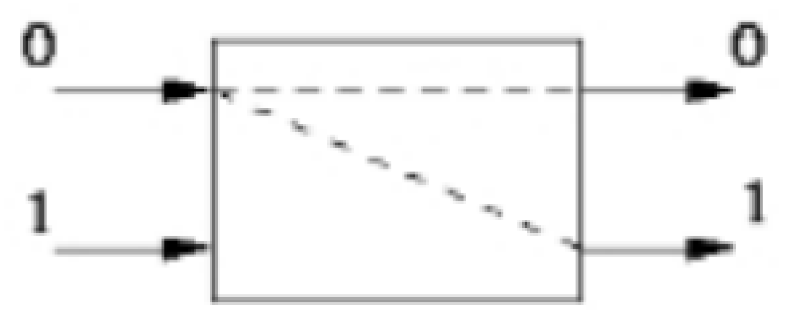
a) Multi-stage networks connect n processors with n memory blocks (so-called modules) with each other. The connection is established over several stages. Each stage consists of a number of switches, where each input must be connected to an output. Normally, a 2x2 switch box is used to build up a multi-stage network. Figure shows four different switching possibilities.



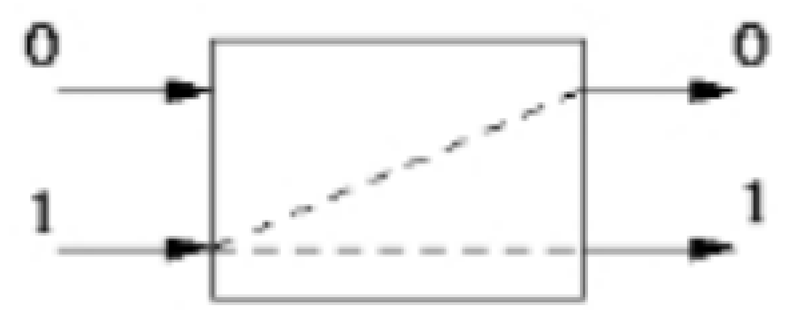
(a) Straight-through



(b) Criss-cross



(c) Upper broadcast



(d) Lower broadcast

A multistage interconnection network may capable for performing highly reliable communications with less hardware. In the multistage interconnection network for interconnecting a plurality of nodes, the first and final stages each have switches two times as large as the number of switches at an intermediate stage. Two output ports of each node are connected to the input ports of different first stage switches, and two input ports are connected to the output ports of final stage different switches. The input ports of switches of the intermediate stage are connected to the output ports of first stage different switches, and the output ports are connected to the input ports of final stage different switches. At least one output port of each switch at the first stage is directly connected to at least one input port of an optional switch at the final stage.

A modified version of a switching/routing device commonly known as a crossbar switch which is optimized to switch and reroute very high speed synchronous data communication signals without interruptions and/or excessive transition time shifts. In a network, a cross-bar switch is a device that is capable of channeling data between any two devices that are attached to it up to its maximum number of ports. The paths set up between devices can be fixed for some duration or changed when desired and each device-to-device path (going through the switch) is usually fixed for some period.

Cross-bar topology can be contrasted with bus topology, an arrangement in which there is only one path that all devices share. Traditionally, computers have been connected to storage devices with a large bus. A major advantage of cross-bar switching is that, as the traffic between any two devices increases, it does not affect traffic between other devices.

In addition to offering more flexibility, a cross-bar switch environment offers greater scalability than a bus environment. However crossbar architecture has a small problem. When a crossbar switch serves multiple networks, and two frames enter the switch at the same time destined for different ports, one of the frames is blocked while the first frame is forwarded. This results in all frames being queued as they flow through the switch. If there is sufficient traffic and insufficient buffer space on the switch, packets are dropped.

b) Shared memory systems form a major category of multiprocessors. In this category, all processors share a global memory. Communication between tasks running on different processors is performed through writing to and reading from the global memory. All inter-processor coordination and synchronization are also accomplished via the global memory. A shared memory computer system consists of a set of independent processors, a set of memory modules, and an interconnection network. Two main problems need to be addressed when designing a shared memory system: performance degradation due to contention, and coherence problems. Performance degradation might happen when multiple processors are trying to access the shared memory simultaneously. A typical design might uses caches to solve the contention problem. However, having multiple copies of data, spread throughout the caches, might lead to a coherence problem. The copies in the caches are coherent if they all equal the same value. But, if one of the processors writes over the value of one of the copies, then the copy becomes inconsistent because it no longer equals the value of the other copies. In this chapter we study a variety of shared memory systems and their solutions of the cache coherence problem. The aspects studied include Uniform Memory Access (UMA), Non-uniform memory access (NUMA), Cache-only memory Architecture (COMA), Bus Based Symmetric Multiprocessors, Basic Cache Coherency Methods, Snooping Protocols, Directory Based Protocols, and Shared Memory Programming.
To support larger processor counts, memory must be distributed among the processors rather than centralized; otherwise the memory system would not be able to support the bandwidth demands of a larger number of processors without incurring excessively long access latency. With the rapid increase in processor performance and the associated increase in a processor's memory bandwidth requirements, the scale of multiprocessor for which distributed memory is preferred over a single, centralized memory continues to decrease in number (which is another reason not to use small and large scale). Of course, the larger number of processors raises the need for a high bandwidth interconnects. Both direct interconnection networks (i.e., switches) and indirect networks (typically multidimensional meshes) are used. So, we can say that distributed share memory architecture is better than centralize memory architecture.

3. Draw a 16-input Omega network using 2×2 switches as building blocks:

i) Show the switching setting for routing a message from node 1011 to node 0101 and from node 0111 to node 1001 simultaneously. Does blocking exist in this case?

ii) Determine how many permutations can be implemented in one-pass through this Omega network. What is the percentage of one-pass permutations among all permutations?

iii) What is the maximum number of passes needed to implement any permutation through the network? [WBUT 2008]

Answer:

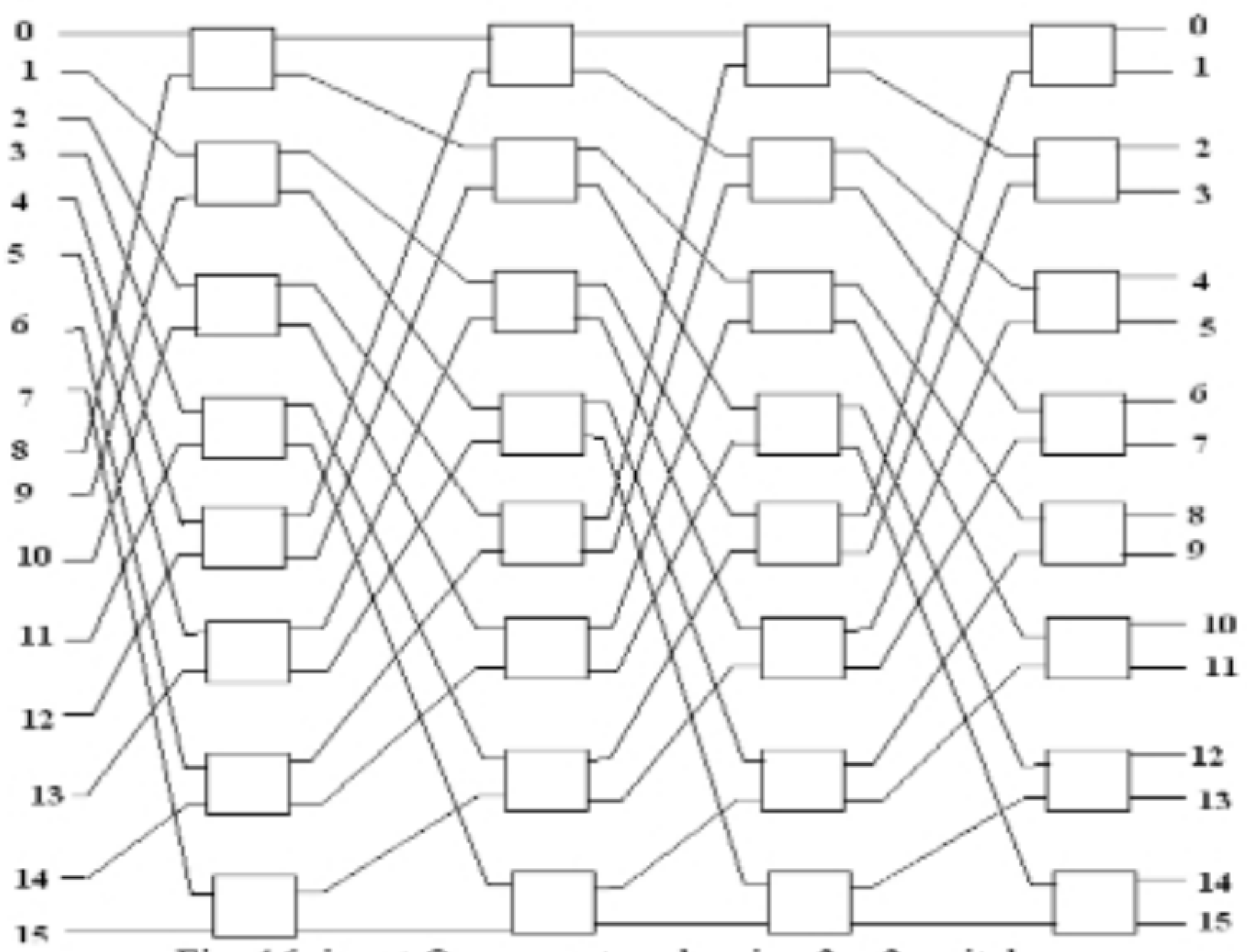


Fig: 16–input Omega network using 2 x 2 switches

i) Now, we have shown the switching setting for routing a message from node 1011= 11 to node 0101= 5 and from node 0111= 7 to node 1001 = 9 simultaneously in the figure below.

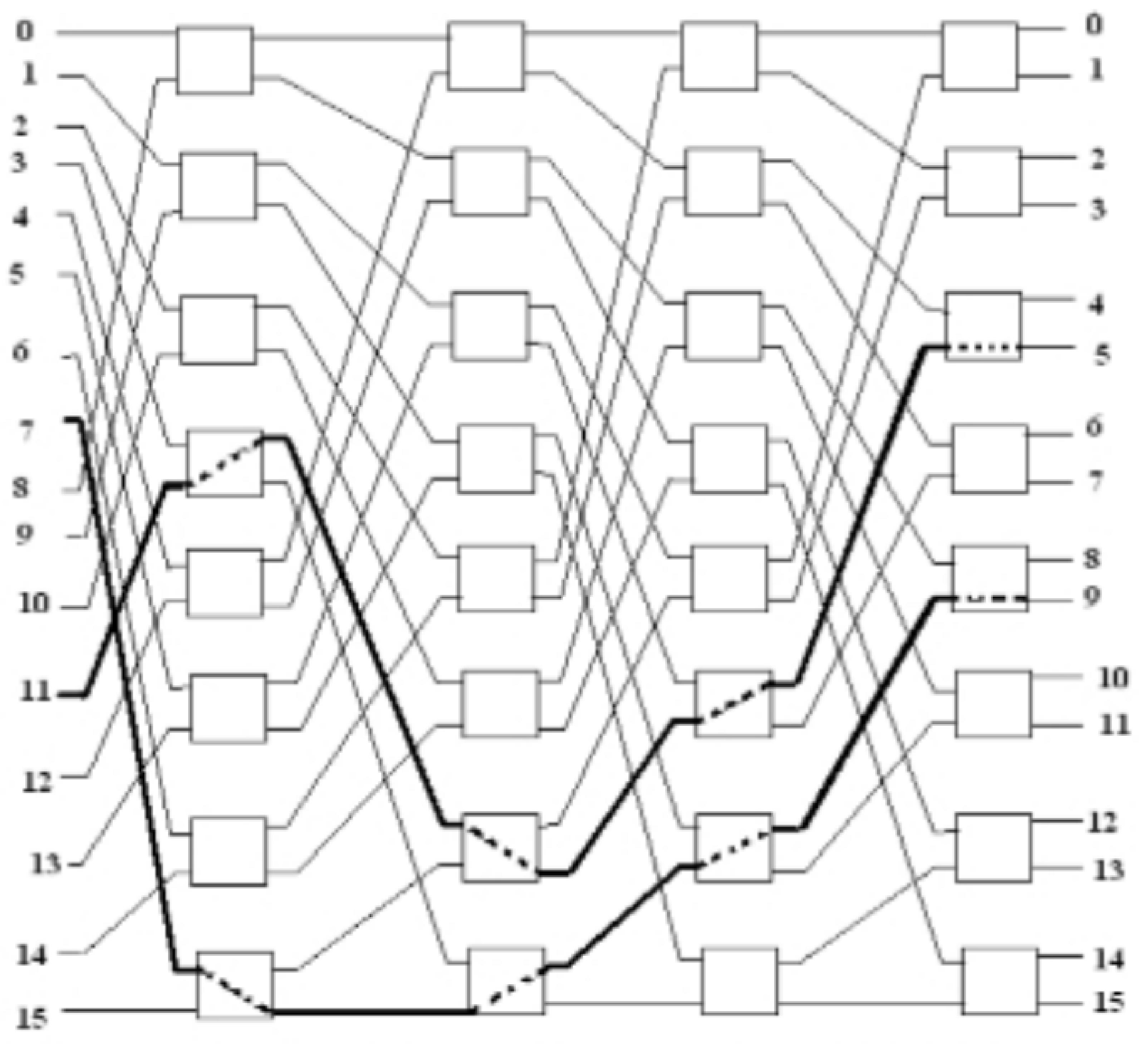


Fig: data routing from node 1011 to node 0101 and 0111 to 1001

There is no blocking in the above problem as is shown in the above picture.

ii) An n-input Omega network can implement $n^{n/2}$ permutations in a single pass and there are total n! Permutation is occurred.
Here, n = 16, so, there are $16^{16/2} = 16^{8}$ permutation is occurred in first pass.
There are total 16! Permutations happen.
So, the percentage of one-pass permutations among all permutations = $16^8/16!$ = .000205 = 0.0205 %

iii) In general, maximum number of passes needed to implement any permutation through the network is $\log_2 n$, where n is the number of inputs.

4. a) What do you mean by multiprocessor system? What are the similarities and dissimilarities between the multiprocessor system and multiple computer system? [WBUT 2010]

OR,

Differentiate between multiprocessors and multi computers based on their structures, resource sharing and inter processor communication. [WBUT 2019]
b) What are the different architectural models for multiprocessors? Explain each of them with example. [WBUT 2010]

OR,

What is a fundamental difference in interprocessor coordination mechanism between multiprocessor & multicomputer systems? Explain with reference to their architectural differences. [WBUT 2013]

c) Distinguish between loosely coupled and tightly coupled multiprocessor architectures. Which architecture is better and why? [WBUT 2010]

OR,

What are the differences between loosely coupled and tightly coupled architectures? [WBUT 2017]

Answer:

a) 1st part

A multiple processor system consists of two or more processors that are connected in a manner that allows them to share the simultaneous (parallel) execution of a given computational task. Parallel processing has been advocated as a promising approach for building high-performance computer systems. Two basic requirements are inevitable for the efficient use of the employed processors. These requirements are (1) low communication overhead among processors while executing a given task and (2) a degree of inherent parallelism in the task. A number of communication styles exist for multiple processor networks. These can be broadly classified according to (1) the communication model (CM) or (2) the physical connection (PC). According to the CM, networks can be further classified as (1) multiple processors (single address space or shared memory computation) or (2) multiple computers (multiple address space or message passing computation). According to PC, networks can be further classified as (1) bus-based or (2) network-based multiple processors.

2nd part:

In a multiprocessors system all CPUs share a common physical memory, as illustrated in Figure below. A system based on shared memory, like this one, is called a **multiprocessor** or sometimes just a **shared memory system**.

The multiprocessor model extends into software. All processes working together on a multiprocessor can share a single virtual address space mapped onto the common memory. Any process can read or write a word of memory by just executing a LOAD or STORE instruction. Nothing else is needed. Two processes can communicate by simply having one of them write data to memory and having the other one read them back. Each of the 16 CPUs runs a single process, which has been assigned one of the 16 sections to analyze. Some examples are the Sun Enterprise 10000, Sequent NUMA-Q, SGI Origin 2000, and HP/Convex Exemplar.

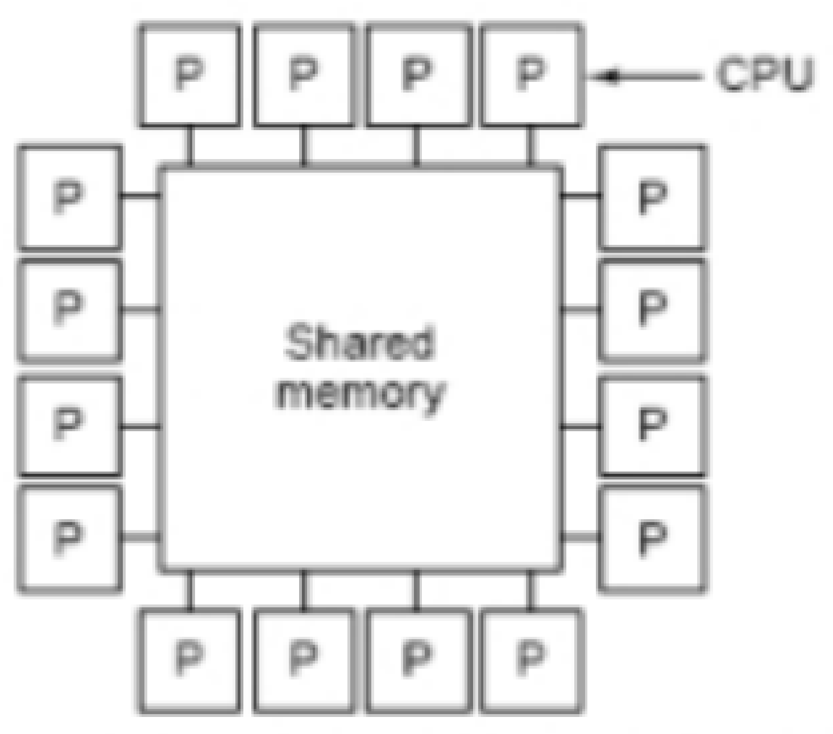


Fig: A multiprocessor with 16 CPUs sharing a common memory

In multicomputer design for a parallel architecture is one in which each CPU has its own private memory, accessible only to itself and not to any other CPU. Such a design is called a **multicomputer** or sometimes a **distributed memory system** and is illustrated in Figure below. Multicomputers are frequently loosely coupled. The key aspect of a multicomputer that distinguishes it from a multiprocessor is that each CPU in a multicomputer has its own private, local memory that it can access by just executing LOAD and STORE instructions, but which no other CPU can access using LOAD and STORE instructions. Thus multiprocessors have a single physical address space shared by all the CPUs whereas multicomputers have one physical address space per CPU.

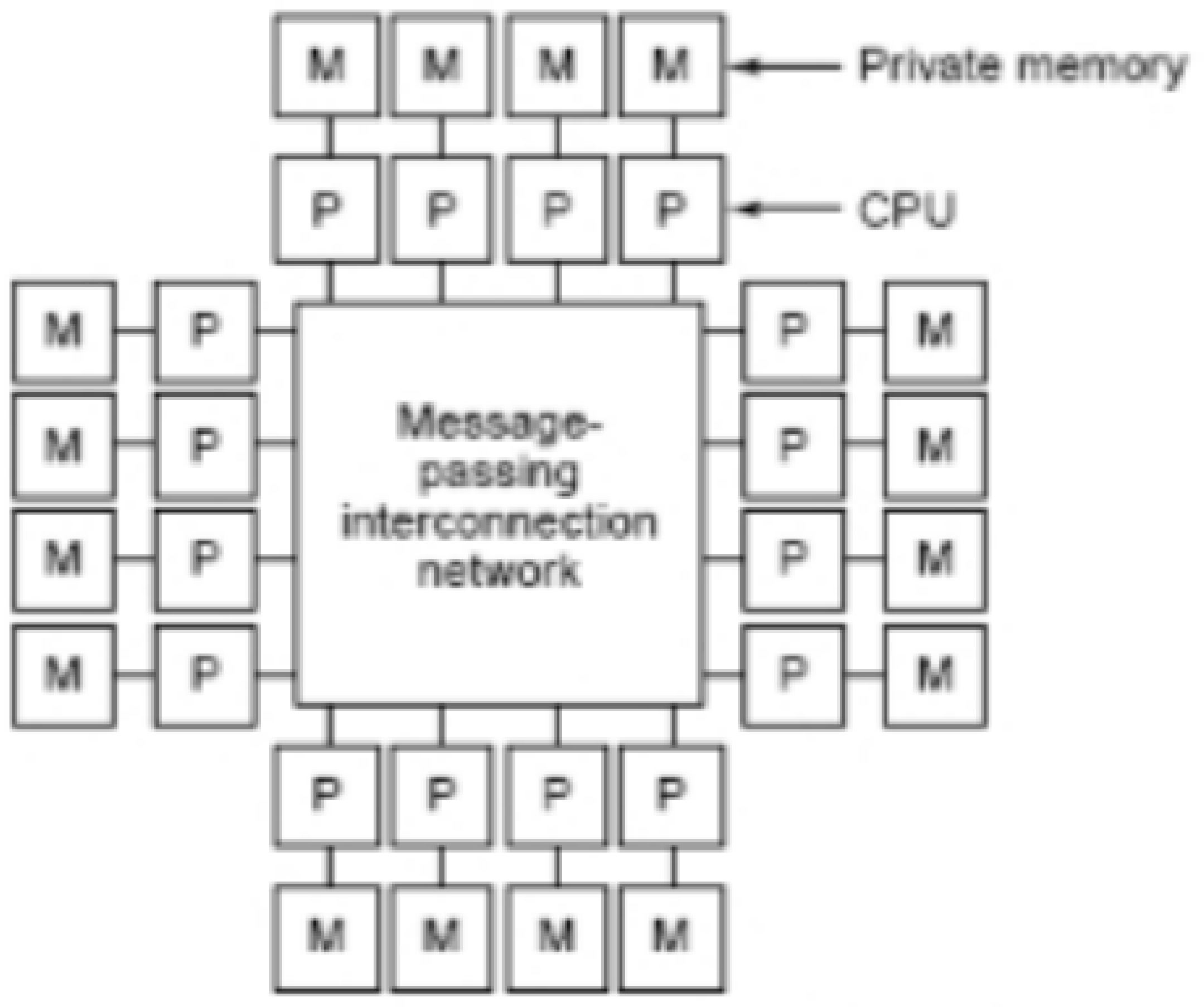


Fig: A multicomputer with 16 CPUs, each with each own private memory

Since the CPUs on a multicomputer cannot communicate by just reading and writing the common memory, they need a different communication mechanism. Examples of multicomputers include the IBM SP/2, Intel/Sandia Option Red, and the Wisconsin COW(Cluster of Workstations).

b) There are two different architectural models of multiprocessor system. First we discuss about centralized shared-memory architectures. For multiprocessors with small processor counts, it is possible for the processors to share a single centralized memory and to interconnect the processors and memory by a bus. With large caches, the bus and the single memory, possibly with multiple banks, can satisfy the memory demands of a small number of processors. By replacing a single bus with multiple buses, or even a switch, a centralized shared memory design can be scaled to a few dozen processors. Although scaling beyond that is technically conceivable, sharing a centralized memory, even organized as multiple banks, becomes less attractive as the number of processors sharing it increases. Because there is a single main memory that has a symmetric relationship to all processors and a uniform access time from any processor, these multiprocessors are often called symmetric (shared-memory) multiprocessors (SMPs), and this style of architecture is sometimes called UMA(uniform memory access). This type of centralized shared-memory architecture is currently by far the most popular organization. Figure 1 shows what these multiprocessors look like.

So, communication with shared memory can be defined as follows:

- Allows a familiar programming style. Sometimes it is straightforward to make an existing program run on a parallel machine (with a small number of processors).
- Requires synchronization with critical regions or semaphores for shared data.
- The cache can help reduce the amount of communication.

- Complex hardware is needed to keep the caches correct.

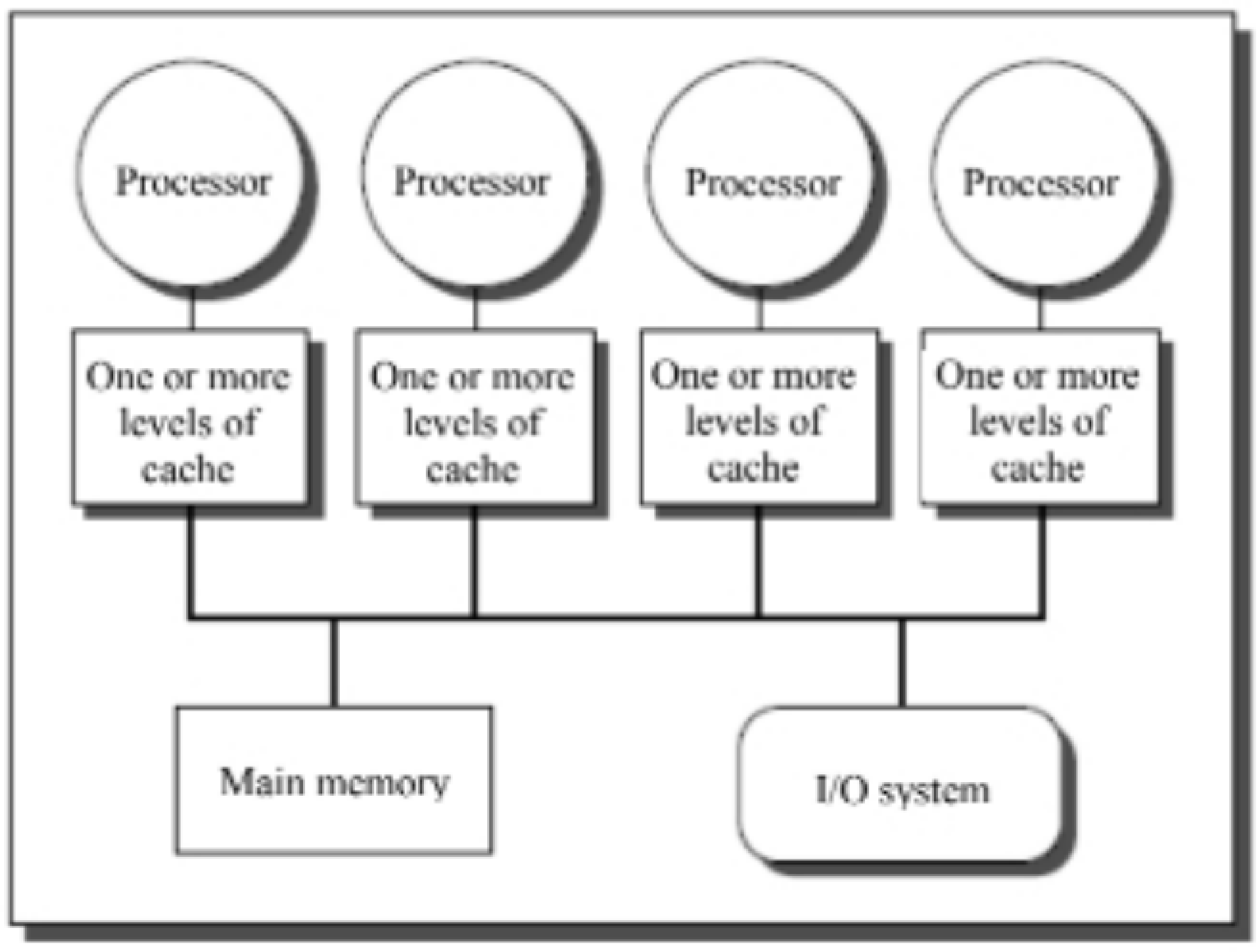

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O system

Fig: Basic structure of a centralized shared-memory multiprocessor

The second group consists of multiprocessors with physically distributed memory. To support larger processor counts, memory must be distributed among the processors rather than centralized; otherwise the memory system would not be able to support the bandwidth demands of a larger number of processors without incurring excessively long access latency. With the rapid increase in processor performance and the associated increase in a processor's memory bandwidth requirements, the scale of multiprocessor for which distributed memory is preferred over a single, centralized memory continues to decrease in number (which is another reason not to use small and large scale). Of course, the larger number of processors raises the need for a high bandwidth interconnects. Both direct interconnection networks (i.e., switches) and indirect networks (typically multidimensional meshes) are used. Figure below shows what these multiprocessors look like.

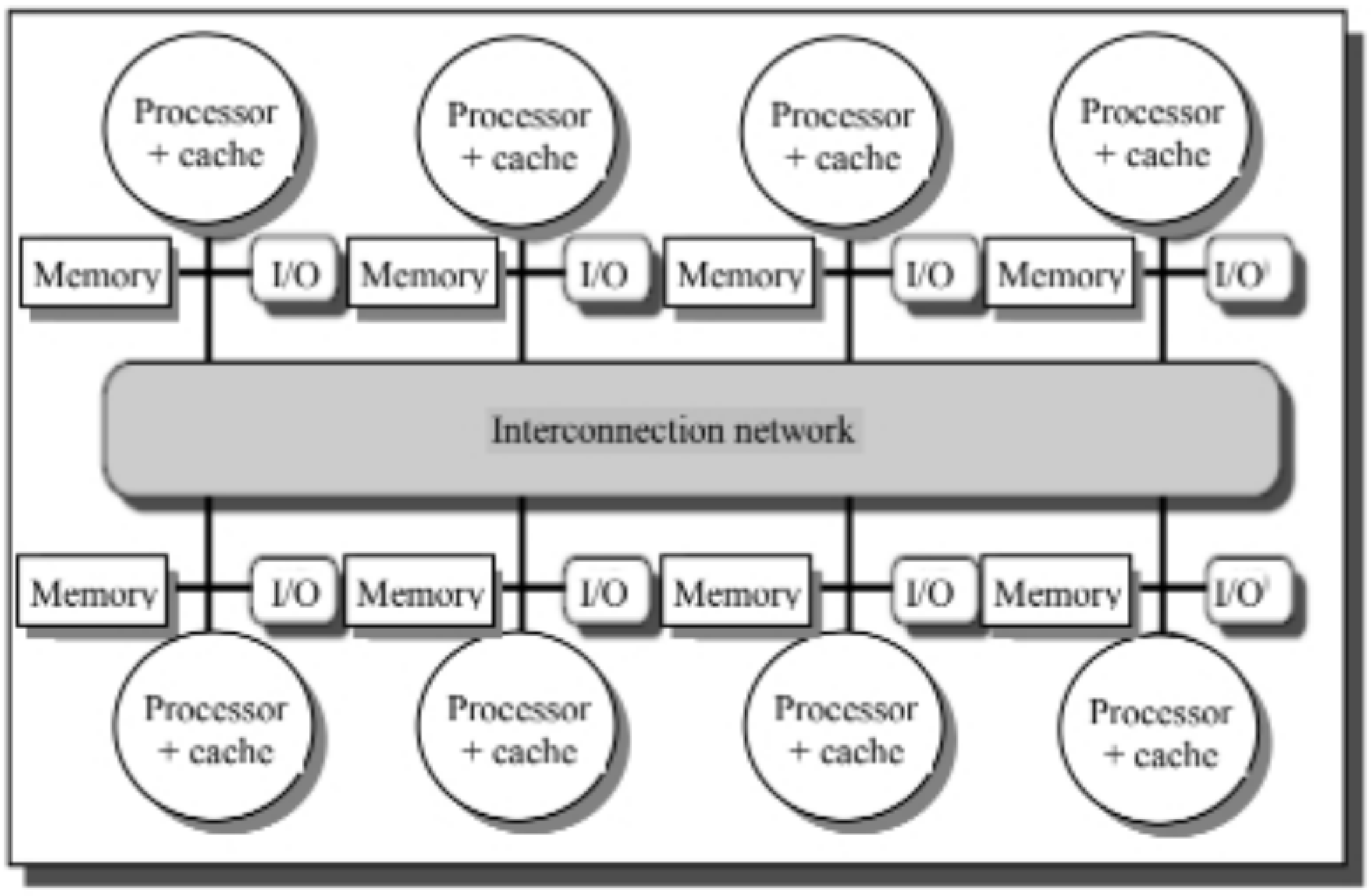


Fig: The basic architecture of a distributed-memory multiprocessor

The basic architecture of a distributed-memory multiprocessor consists of individual nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all the nodes. Individual nodes may contain a small number of processors, which may be interconnected by a small bus or a different interconnection technology, which is less scalable than the global interconnection network. Distributing the memory among the nodes has two major benefits. First, it is a cost-effective way to scale the memory bandwidth, if most of the accesses are to the local memory in the node. Second, it reduces the latency for accesses to the local memory. These two advantages make distributed memory attractive at smaller processor counts as processors get ever faster and require more memory bandwidth and lower memory latency. The key disadvantage for distributed memory architecture is that communicating data between processors becomes somewhat more complex and has higher latency, at least when there is no contention, because the processors no longer share a single centralized memory. As we will see shortly, the use of distributed memory leads to two different paradigms for interprocess communication

c) In many cases, each processor executes a different process. A process is a segment of code that may be run independently, and that the state of the process contains all the information necessary to execute that program on a processor. In a multiprogrammed environment, where the processors may be running independent tasks, each process is typically independent of the processes on other processors. MIMD computers with shared memory are known as **tightly coupled machines**. Examples are ENCORE, MULTIMAX. Tightly-coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory (SMP or UMA), or may participate in a memory hierarchy with both local and shared memory (NUMA).

MIMD computers with an interconnection network are known as **loosely coupled machines**. Examples are INTEL iPSC, NCUBE/7. Loosely-coupled multiprocessor systems referred to as clusters are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system. A Linux Beowulf cluster is an example of a loosely-coupled system.
Tightly-coupled systems perform better and are physically smaller than loosely-coupled systems, but have historically required greater initial investments and may deflate rapidly; nodes in a loosely-coupled system are usually inexpensive commodity computers and can be recycled as independent machines upon retirement from the cluster.

5. What are the common data routing functions among the Processing Elements and how are they implemented? Explain the main factors that can influence the Performance of interconnection networks. What are the different types of Multi-stage interconnection networks? [WBUT 2011]

OR,

What is the significance of interconnection network in multiprocessor architecture? [WBUT 2014]

OR,

Explain the main factors that can influence the performance of the interconnection networks. [WBUT 2018]

Answer:

There are some common data routing functions among the Processing Elements (PEs):
Permutation: If there are n numbers of objects present, we can arrange a n! Connection between them. A permutation π = (a1, a2, a3) (a4, a5), where a1 ... a5 are nodes of the network. Then there is a bisection mapping a1→ a2, a2→a3, a3→a1 and a4→a5, a5→a4 in a circular fashion.
Perfect-Shuffle Routing Function: This is one type of permutation function. Let us consider a function $x = \{a_n, a_{n-1} \dots a_2, a_1\}$ then the perfect-shuffle routing function P(x) is given below.
$P(x) = \{a_{n-1}, \dots, a_2, a_1, a_n\}$
Where, $a_n, a_{n-1} \dots a_2, a_1$ are the respective node address bit of the network.
Let us consider an example, where X = 110. Then the function is P(x) = 101
Exchange Routing Function: This Exchange Routing Function is also one type of permutation function. Suppose, for a given $x = \{a_n, a_{n-1} \dots a_2, a_1\}$, the Exchange Routing Function is $E_i(x) = \{a_n, a_{n-1}, \dots, \overline{a}_{n-i+1}, a_2, a_1\} \ \forall i, 1 \le i \le n$
Butterfly Routing Function: There is also another type of routing function is known as Butterfly Routing Function. Here for a given node with address bit, x = {an, an-1… a2, a1}, the Butterfly Routing Function will be
B(x) = {a1, an-1… a2, an}

Multistage interconnection networks are a class of high-speed computer networks usually composed of processing elements (PEs) on one end of the network and memory elements (MEs) on the other end, connected by switching elements (SEs). The switching elements themselves are usually connected to each other in stages, hence the name. Such networks include omega networks, delta networks and many other types. These are typically used in high-performance or parallel computing as a low-latency interconnection though they could be implemented on top of a packet switching network. Though the network is typically used for routing purposes, it could also be used as a co-processor to the actual processors for such uses as sorting.

Multistage Networks
Crossbars switches have excellent performance scalability but poor cost scalability. Buses have excellent cost scalability, but poor performance scalability. Multistage interconnects strike a compromise between these extremes. A number of p x q switches present in every stage of this network. There is a fixed inter stage connections present between the switches in adjacent stages as shown in the figure below.

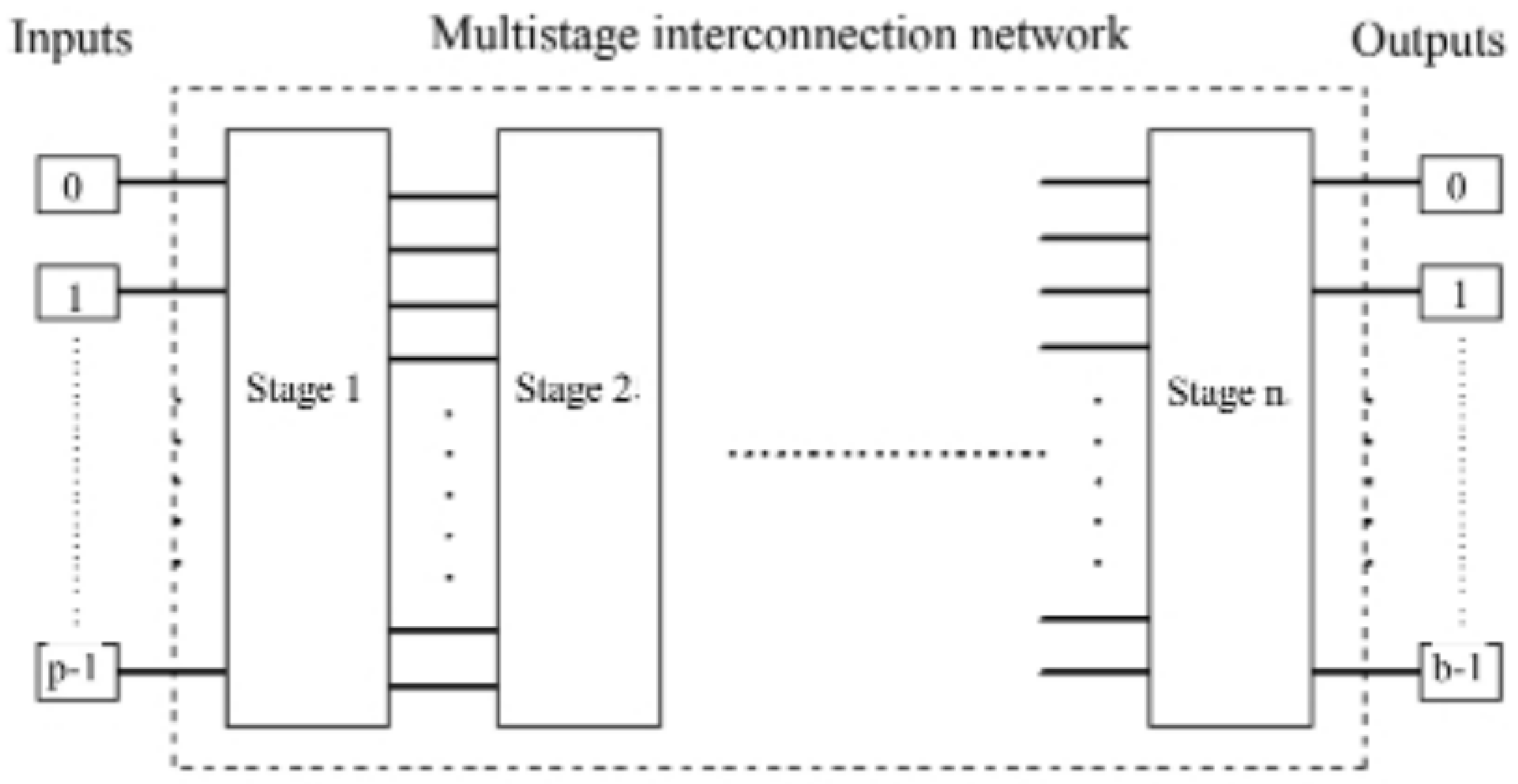


Fig: The schematic of a typical multistage interconnection network

Multistage Omega Network
One of the most commonly used multistage interconnects is the Omega network. This network consists of *log p* stages, where *p* is the number of inputs/outputs. At each stage, input *i* is connected to output *j* if:

$$j = \begin{cases} 2i, & 0 \le i \le p/2 - 1 \\ 2i + 1 - p, & p/2 \le i \le p - 1 \end{cases}$$

Each stage of the Omega network implements a perfect shuffle as follows:
A complete Omega network with the perfect shuffle interconnects and switches can now be illustrated:

- Let *s* be the binary representation of the source and *d* be that of the destination processor.
- The data traverses the link to the first switching node. If the most significant bits of *s* and *d* are the same, then the data is routed in pass-through mode by the switch else, it switches to crossover.
- This process is repeated for each of the *log p* switching stages.

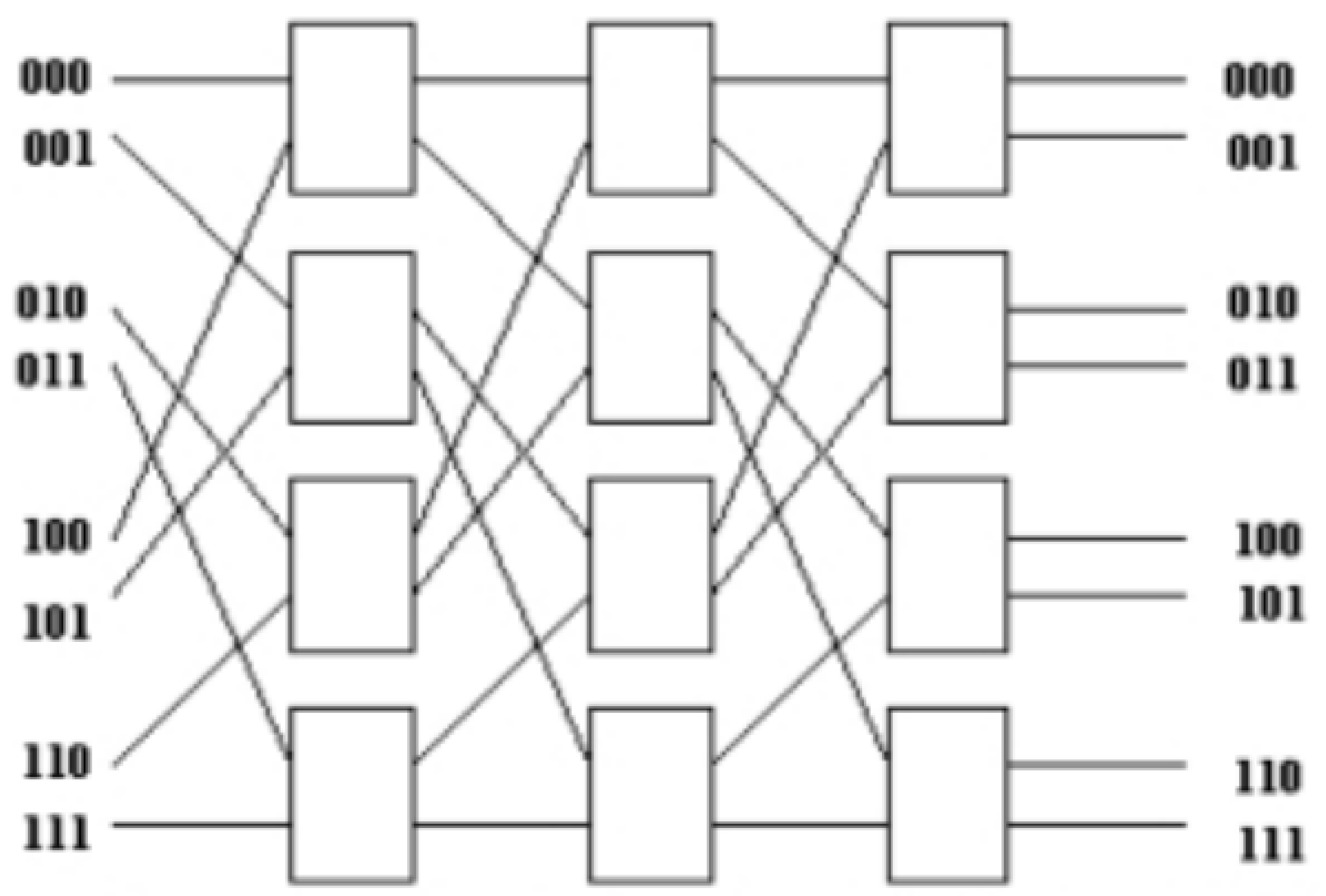


Fig: A complete omega network connecting eight inputs and eight outputs

6. Describe the different types of interconnection networks in computer systems. What is multistage switching networks? [WBUT 2013]

Answer:

In static networks, there are point-to-point connections between neighboring nodes. These networks typically are static, which implies that the point-to-point connections are fixed. Static networks use direct links which are fixed once built. This type of network is suitable for building computers where communication patterns are predictable. Well-known examples of static networks are linear array, rings, meshes, torus and cubes.

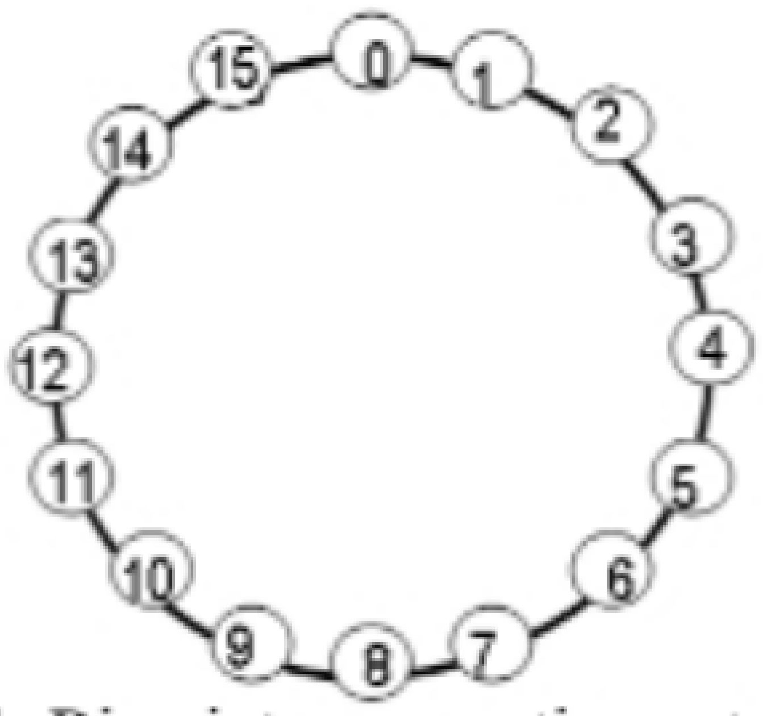


Fig 1: Ring interconnection network

Now we consider ring as a static interconnection network. Ring is obtained by connecting two terminal nodes of linear array with one extra link. In a linear array, each internal node has two neighbors, one to its left and one to its right. The ring is like the linear array, but the diameter of the network is cut in half if the links are bidirectional.

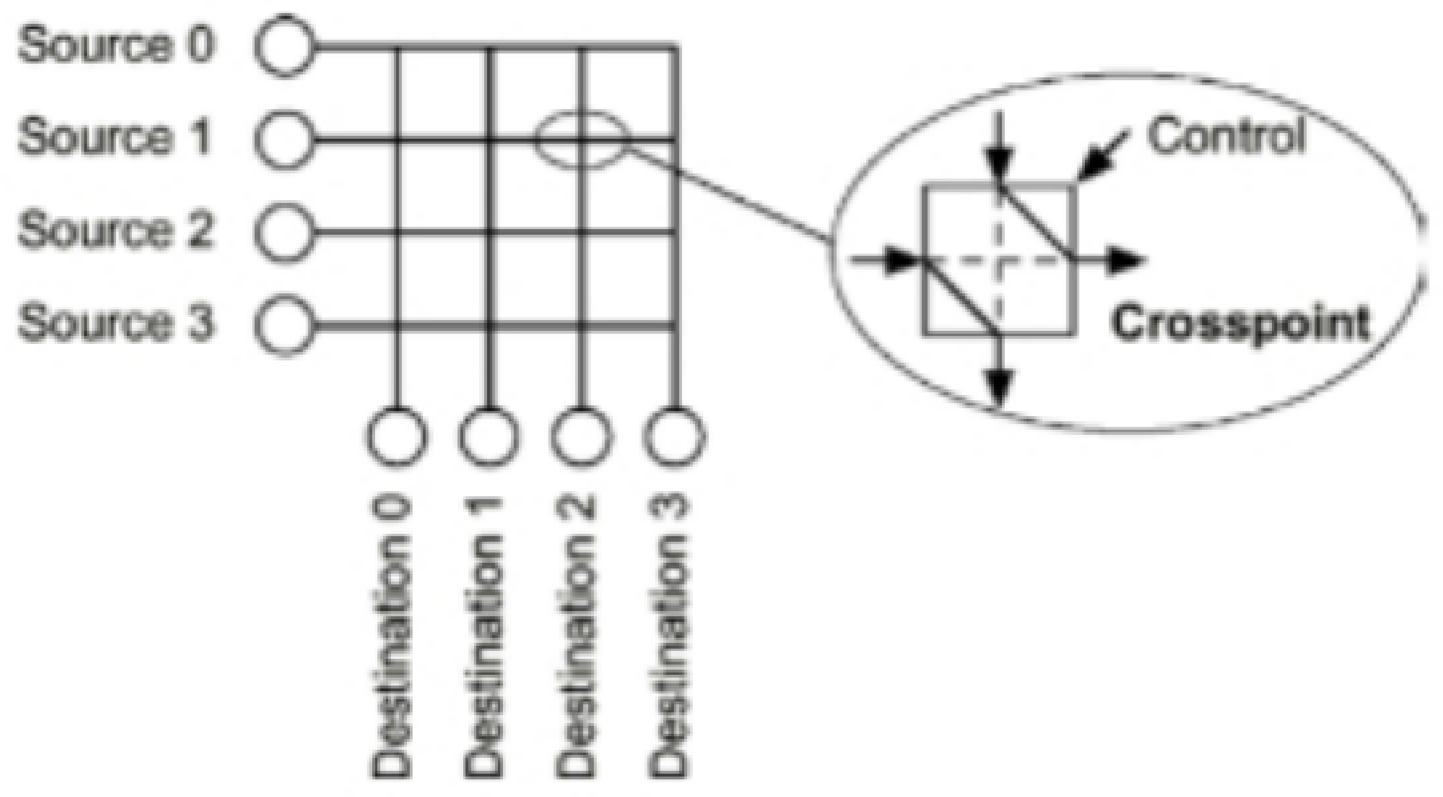


Fig 2: Crossbar switches

Dynamic networks are implemented with switched channels, which are dynamically configured to match communication demand in user program. Examples of dynamic interconnection network are Bus, Multistage switches, crossbar switch etc.

Let us consider, crossbar switch as dynamic interconnection network. The crossbar switch corresponds to an N x M array. Semiconductor switches are located at each of the cross points where inputs and output wires cross. We connect an input to an output by closing a cross-point at the intersection of the appropriate row and column. The complexity of a crossbar has two cost components, one which grows in proportion to the number of inputs and outputs and the other that grows as their product. The product term is often called the cross-point count because it is directly related to the number of simple 2 x 2 cross-points required to implement it. A crossbar requires N2 cross-points for N pairs of terminals.

7. Describe different access methods of the memory system? What will be the maximum capacity of a memory, which uses an address bus of size 8 bit?
[WBUT 2013]

Answer:
There are two types of memory access, i.e. C-access and S-access memory organization.

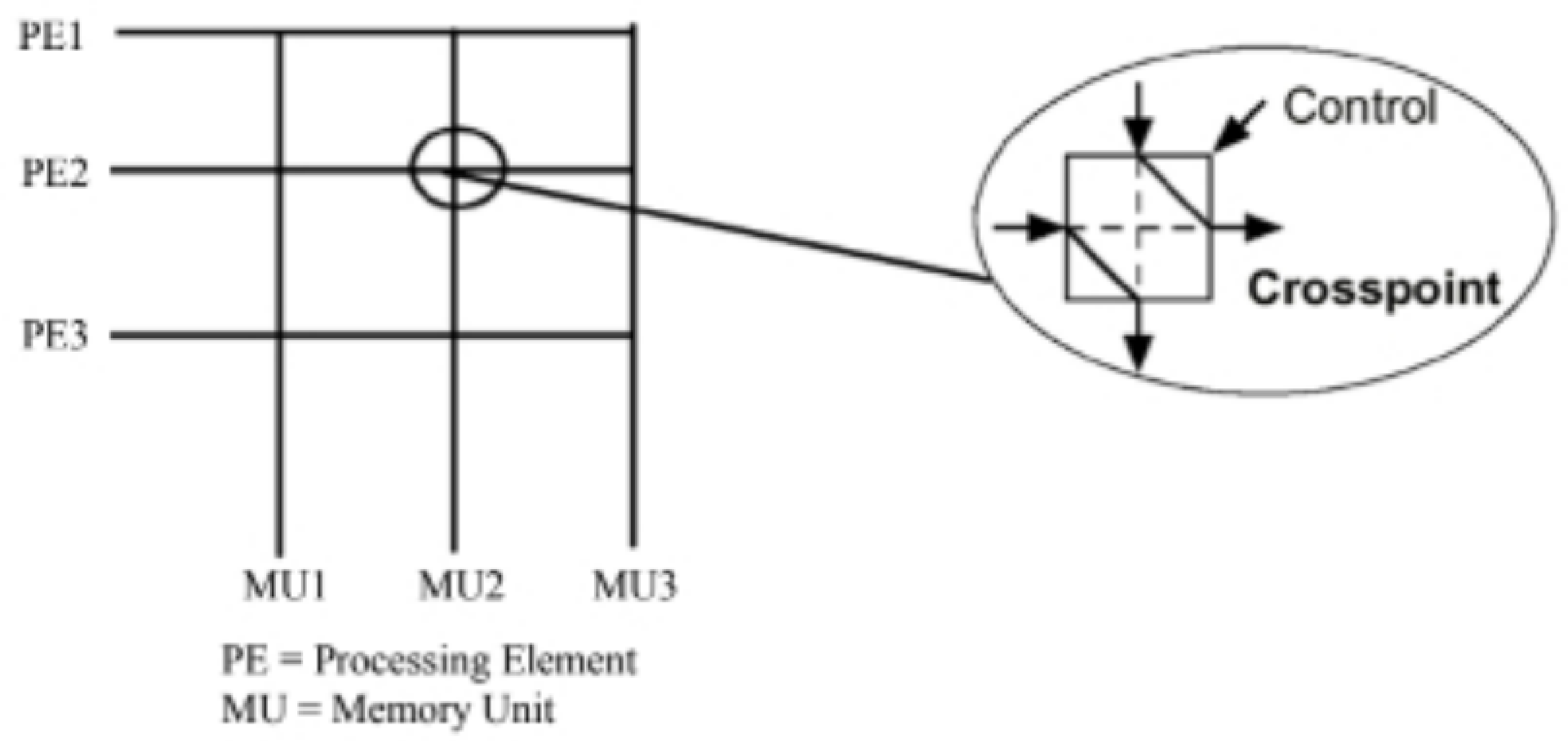
Refer to Question No. 1 and 5 of Short Answer Type Questions for C-access memory and S-access memory organization.

The maximum capacity of a memory, which uses an address bus of size 8 bit is 2^8 =256 bytes.

8. What is multi-processor system? Classify it with examples. [WBUT 2015]
Answer: ***Refer to Question No. 5 of Long Answer Type Questions.***

9. Construct a multiport network where three processing elements want to connect with three memory modules. Design a network where 9 inputs want to connect with 25 outputs. What is the difference between omega network and delta network? Construct an omega network for N = 8 where N represent no. of processors.
[WBUT 2016]

Answer:
1st part:
A multiport network where three processing elements want to connect with three memory modules:



PE = Processing Element
MU = Memory Unit

2nd part:

A network where 9 inputs want to connect with 25 outputs:

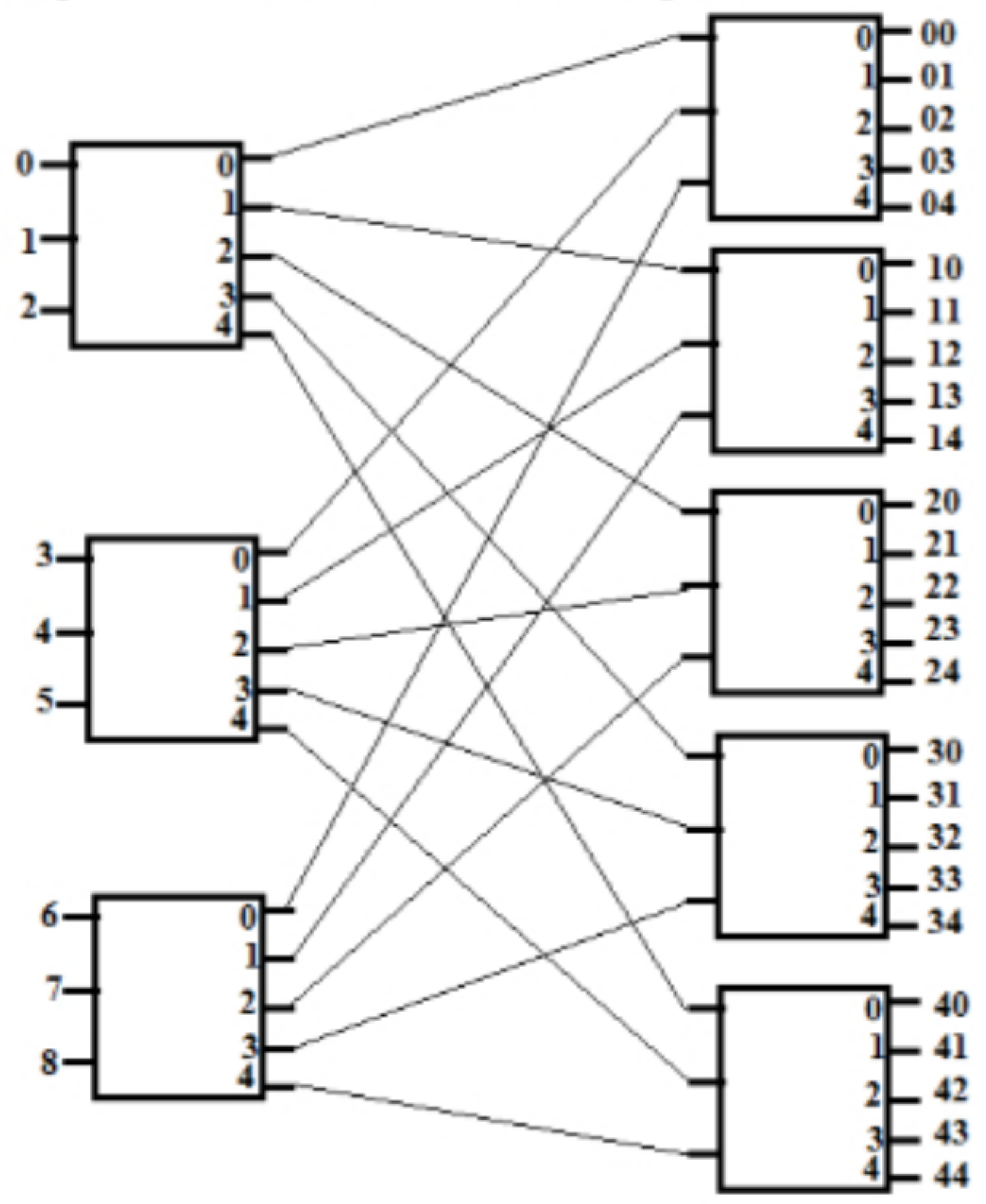


Figure: 9 inputs and 25 outputs Delta network

3rd part:

Difference between omega network and delta network: An NxN Omega network consists of $\log_2 N$ identical stages and between two stages there is a perfect shuffle interconnection. This network maintains a uniform connection pattern between stages. Every input terminal has a unique path to every output terminal.

In an a^n x b^n delta network, there are a^n sources and b^n destinations. There is a unique interconnection path of constant length between the stages of the network. Numbering the stages of the network as 1,2,…,n, starting at the source side of the network requires that there be a^{n-1} crossbar modules in the first stage.

An omega network for N = 8 where N represents no. of processors:

One of the most commonly used multistage interconnects is the Omega network. This network consists of *log p* stages, where *p* is the number of inputs/outputs. At each stage, input *i* is connected to output *j* if:

$$j = \begin{cases} 2i, & 0 \le i \le p/2-1 \\ 2i+1-p, & p/2 \le i \le p-1 \end{cases}$$

Each stage of the Omega network implements a perfect shuffle as follows:

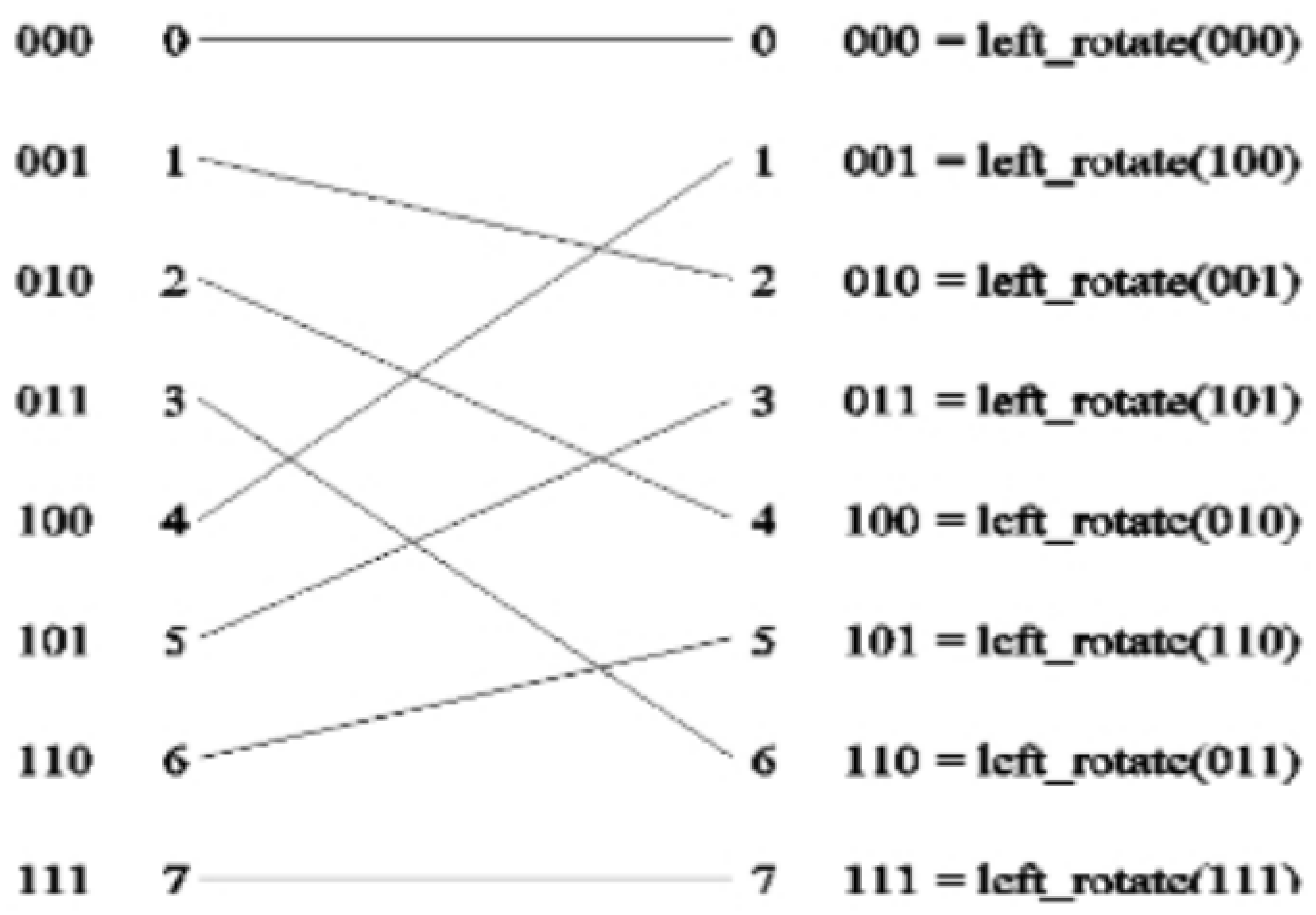

000 0 → 0 000 = left_rotate(000)
001 1 → 1 001 = left_rotate(100)
010 2 → 2 010 = left_rotate(001)
011 3 → 3 011 = left_rotate(101)
100 4 → 4 100 = left_rotate(010)
101 5 → 5 101 = left_rotate(110)
110 6 → 6 110 = left_rotate(011)
111 7 → 7 111 = left_rotate(111)

Fig 1: A perfect shuffle interconnection for eight inputs and outputs

A complete Omega network with the perfect shuffle interconnects and switches can now be illustrated:

- Let *s* be the binary representation of the source and *d* be that of the destination processor.
- The data traverses the link to the first switching node. If the most significant bits of *s* and *d* are the same, then the data is routed in pass-through mode by the switch else, it switches to crossover.
- This process is repeated for each of the *log p* switching stages.

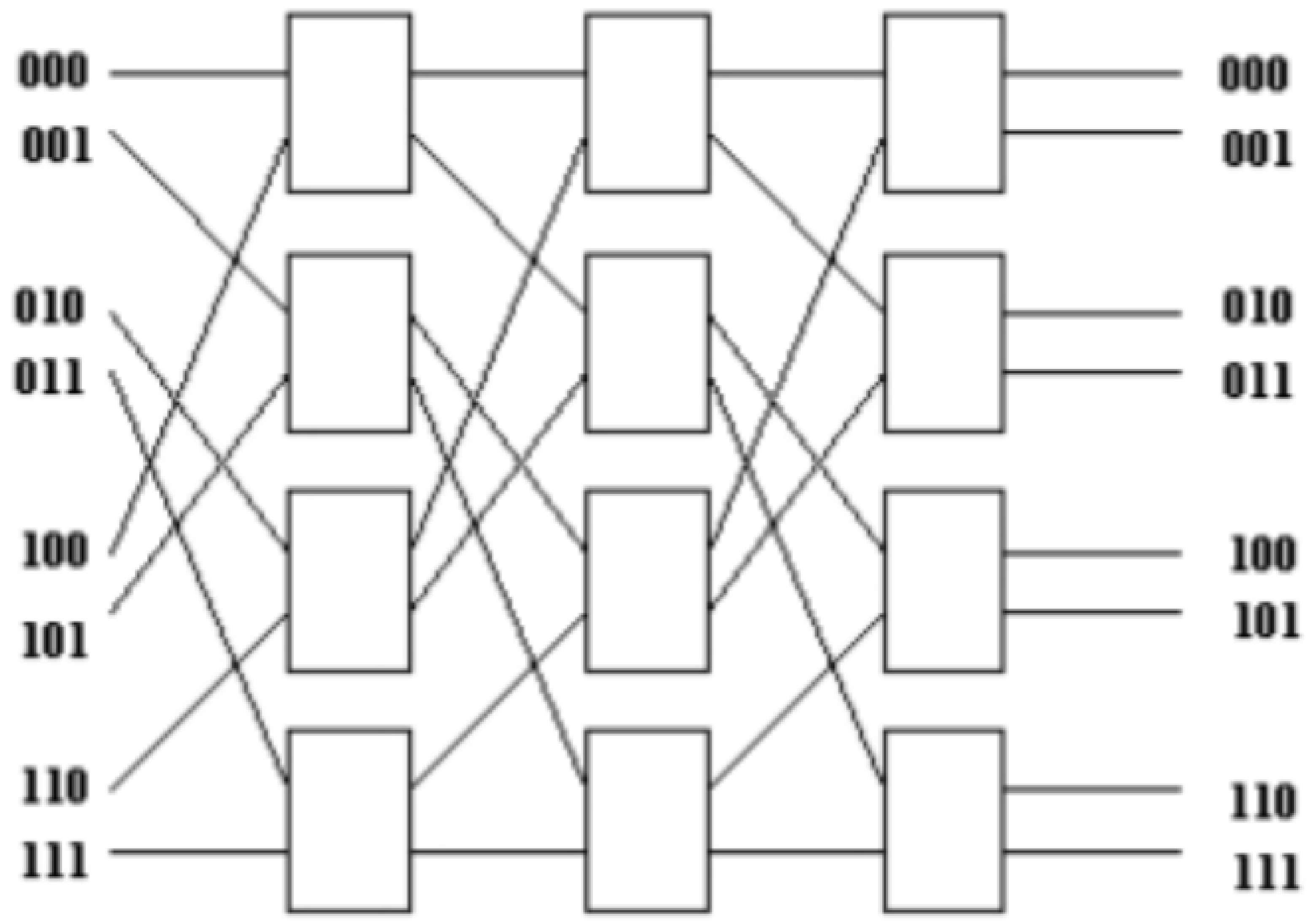


Fig 2: A complete omega network connecting eight inputs and eight outputs

10. a) What are the differences between the static network and the dynamic network?
b) What do you mean by bisection width and diameter of a network?
c) Explain software parallelism and hardware parallelism. [WBUT 2018]

Answer:

a) Interconnection network provides connections between processing nodes in a parallel processing system. Interconnection networks can be classified as static or dynamic. A static network is a point-to-point network between processing nodes. A processing node is connected to another processing node without use of any external switching elements. Static interconnection network can be also referred as direct interconnection network. Dynamic interconnection networks are built by using switches and cables between processing elements. Network switches and connections form an interconnection network and the processing units are separate from the network. Dynamic networks are called as indirect networks. Dynamic interconnection networks have scalability features in general.

b) Bisection width – The minimum number of edges between switch nodes that must be removed in order to divide the network into two equal halves. The high bisection width is desirable.
Diameter – The diameter of the network means the largest distance between two switch nodes. A low diameter is desirable for interconnection network.

c) Hardware Parallelism: This refers to the type of parallelism defined by the machine architecture and hardware multiplicity. Hardware parallelism is a function of cost and performance tradeoffs. It displays the resource utilization patterns of simultaneously executable operations. It can also indicate the peak performance of the processors. One way to characterize the parallelism in a processor is by the number of instruction issues per machine cycle.
Software Parallelism: It is defined by the control and data dependence of programs. The degree of parallelism is revealed in the program profile or in the program flow graph. Software parallelism is a function of algorithm, programming style, and compiler optimization.

11. With the help of neat sketches, explain the 10 subsystems in case of lightly coupled multiprocessor system. [WBUT 2019]

Answer:

A loosely coupled multiprocessor system is a type of multiprocessing where the individual processors are configured with their own memory and are capable of executing user and operating system instructions independent of each other. This type of architecture paves the way for parallel processing. Loosely coupled multiprocessor systems are connected via high-speed communication networks.
Loosely coupled multiprocessor systems are also known as distributed memory, as the processors do not share physical memory and have their own IO channels.

Techopedia explains Loosely Coupled Multiprocessor System

A multiprocessor system makes use of more than one CPU along with memory and IO channels. They are capable of processing multiple instruction, multiple data (MIMD) programming. Thus, they support concurrent operations. The configuration of processors in a multiprocessor system can be loosely coupled or tightly coupled. The major distinction between these two types of multiprocessors is the way memory is organized.

Tightly coupled systems share a single memory space and share information through the shared common memory. Loosely coupled multiprocessors consist of distributed memory where each processor has its own memory and IO channels. The processors communicate with each other via message passing or interconnection switching. Each processor may also run a different operating system and have its own bus control logic.

Loosely coupled systems are less costly than tightly coupled systems, but are physically bigger and have a low performance compared to tightly coupled systems. The individual nodes in a loosely coupled system can be easily replaced and are usually inexpensive. Loosely coupled systems also draw more power than tightly coupled systems. Loosely coupled systems are more robust and can resist failures. A single node failure does not break down the entire system and it is also easy to add more nodes to an existing system. But the need for extra hardware required to provide communication between the individual processors makes them complex and less portable.

Some of the characteristics of loosely coupled multiprocessors are:

- Distributed memory
- High scalability
- Low data rate
- Low cost
- Static interconnection
- Capable of running multiple OSs
- Low throughput
- Increased space requirements
- High power consumption
- Reusable and flexible components

12. Write short notes on the following:

a) Omega network **[WBUT 2005, 2006, 2008]**
b) Crossbar Switches **[WBUT 2008]**
c) Multiport Network **[WBUT 2008]**
d) Memory inclusion **[WBUT 2010]**
e) Memory interleaving **[WBUT 2010, 2014]**
f) Multiprocessor computer **[WBUT 2019]**

Answer:

a) Omega network:

Refer to Question No. 9 of Long Answer Type Questions.

b) Crossbar Switches:

Crossbar Switches allow any processor in the system to connect to any other processor or memory unit so that many processors can communicate simultaneously without contention. A new connection can be established at any time as long as the requested input and output ports are free. Crossbar switches are used in the design of high-performance small-scale multiprocessors, in the design of routers for direct networks. A crossbar can be defined as a switching network with *N* inputs and *M* outputs, which allows up to min {*N,M*} one-to-one interconnections without contention. Figure 1.1 shows an *N XM* crossbar network. Usually, *M* = *N* except for crossbars connecting processors and memory modules.

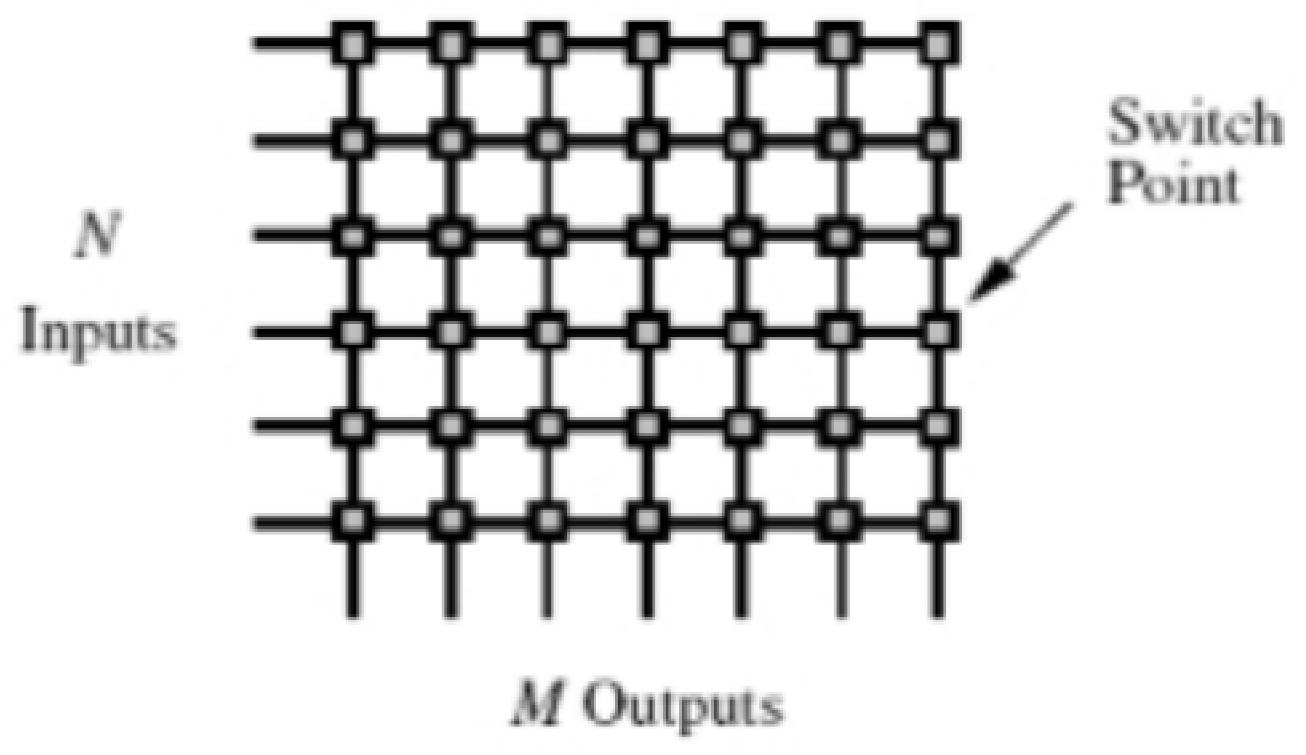


Fig 1.1 An N × M crossbar

The cost of such a network is *O(NM)*, which is prohibitively high with large *N* and *M*. For a crossbar network with distributed control, each switch point may have four states, as shown in Figure 1.2. In Figure 1.2 (a), the input from the row containing the switch point has been granted access to the corresponding output, while inputs from upper rows requesting the same output are blocked. In Figure 1.2 (b), an input from an upper row has been granted access to the output. The input from the row containing the switch point does not request that output and can be propagated to other switches. In Figure 1.2 (c), an input from an upper row has also been granted access to the output. However, the input from the row containing the switch point also requests that output and is blocked. The configuration in Figure 1.2 (d) is only required if the crossbar has to support multicasting (one-to-many communication).

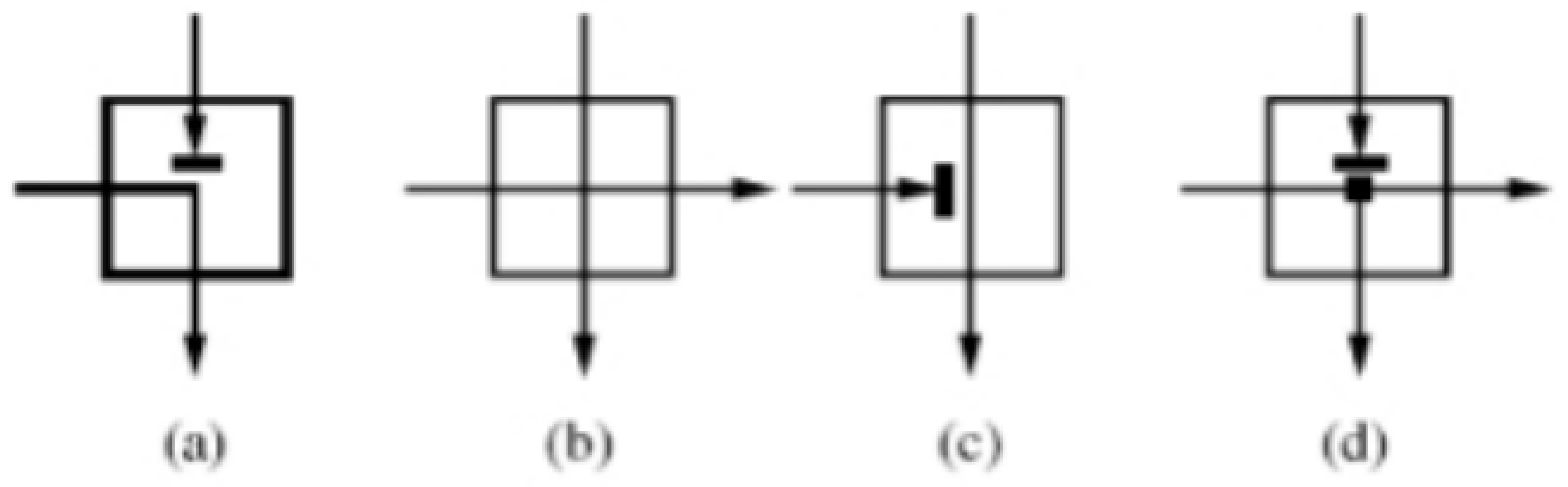


Fig 1.2: States of a switch point in a crossbar network

c) Multiport Network:

The bank-based multiport memory is an approach to realizing high access bandwidth than a conventional *N*-port memory cell approach. However, this method is unsuitable for large numbers of ports and banks because the hardware resources of the crossbar network which connects the ports and banks increase in proportion to the product of the numbers of ports and banks.

A parallel processor array with a two-dimensional crossbar switches architecture are configured as clusters of processors, wherein the individual processing elements within each cluster are interconnected by a two dimensional cluster network of crossbar switch elements. The clusters are interconnected via a two dimensional array network of crossbar switch elements. Input data is supplied directly into the array network of crossbar switch elements, which allows an optimal initial partitioning of the data set among the processing elements.

A parallel processor array, an interconnection network for interconnecting the processor clusters, the network including a two-dimensional mesh of multi-port crossbar switch elements arranged in rows and columns in a crossbar mesh network and wherein each processor cluster is connected to a port of a row crossbar switch element and to a port of a column crossbar switch element, and wherein an input data set to be processed is supplied directly into the network via crossbar switch element input ports for initial partitioning of the data set among the processing elements, said input data set being characterized as a three dimensional data cube, said three dimensional data cube being characterized as sensor data, a first data dimension represents a sensor channel dimension, a second data dimension represents a Doppler dimension, and a third data dimension represents a Range cell dimension, and wherein the interconnection network is configurable in an initial state such that the data set is initially distributed among the processing elements for processing in a first data dimension during a first processing function, and subsequently is configurable to perform a data dimension transposition of the data set for processing in a second data dimension by the processing elements during a second processing function.

d) Memory inclusion:

Memory hierarchy satisfies three important properties for information storing as inclusion, coherence, and locality. We consider that the cache memory is in the innermost level $M_1$and in the outermost level M_n represents the tape drive where all the information words stored. Processor can access all addressable words in M_n using the virtual address space of a computer.

- **Inclusion Property**

According to the inclusion property, the memory contains that present in the upper level of memory hierarchy must present also lower level of memory. So, we can state the inclusion property as $M_1 \subset M_2 \subset M_3 \subset ... \subset M_n$.

At the time of the processing, required portion of memory M_n are copied into M_{n-1}. Similarly, subsets of M_{n-1} are copied into M_{n-2}, and so on. So, if a word is found in memory M_i , then copies of the same word can be also found in all levels as M_{i+1}, M_{i+2}, ..., M_n. But, a word stored in M_{i+1} may not be found in M_i.

Many multiprocessors use multilevel cache hierarchies to reduce the latency of cache misses. If the cache also provides multilevel inclusion—every level of cache hierarchy is a subset of the level further away from the processor—then we can use the multilevel structure to reduce the contention between coherence traffic and processor traffic, as explained earlier. Thus most multiprocessors with multilevel caches enforce the inclusion property. This restriction is also called the subset property, because each cache is a subset of the cache below it in the hierarchy.

e) Memory interleaving:
Interleaving is a technique used to improve the memory performance. Memory interleaving increases bandwidth by allowing simultaneous access of more than one chunk of memory. This improves the performance of the processor because it can transfer more information to / from memory in the same amount of time. It also helps to alleviate the processor-memory bottleneck that is a major limiting factor in overall performance.
Interleaving works by dividing the system memory into multiple blocks. If there are m numbers of blocks then this is called the m-way memory interleaving. In general *two-way* or *four-way* interleaving technique is used. Each block of memory is accessed using different sets of control lines, which are merged together on the memory bus. When a read or write is begun to one block, a read or write to other blocks can be overlapped with the first one.
In an interleaved system, a main memory of size 2^{l} is divided into *m* modules, where *m* is a positive integer (usually, $m=2^{n}$ *for* some integer *n* such that $O<n < l$, *l* being the number of bits in a main memory address). Each main memory address is mapped to a module, and to an address within that module. Such a mapping is called a *hashing scheme*. Clearly, the mapping must be one-to-one.

f) Multiprocessor computer:
Refer to Question No. 4 (a) of Long Answer Type Questions.