UNIT V

Reduced Instruction Set Computer: CISC Characteristics, RISC Characteristics. Pipeline and Vector Processing: Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline, RISC Pipeline, Vector Processing, Array Processor. Multi Processors: Characteristics of Multiprocessors, Interconnection Structures, Inter-processor arbitration, Inter-processor communication and synchronization, Cache Coherence.

CISC Characteristics, RISC Characteristics.

No.	RISC	CISC
1.	Simple instructions taking one cycle	Complex instructions taking multiple cycles.
2.	Very few instructions refer memory	Most of instructions may refer memory.
3.	Instructions are executed by hardware.	Instructions are executed by microprogram.
4.	Fixed format instructions	Variable format instructions.
5.	Few instructions	Many instructions.
6.	Few addressing mode, and most instructions have register to register addressing mode.	Many addressing modes.
7.	Complexed addressing modes are synthesized in software.	Supports complex addressing modes
8.	Multiple register sets.	Single register set.
9.	Highly pipelined.	Not pipelined or less pipelined.
10.	Complexity is in the compiler	Complexity is in the microprogram
11.	Conditional jump can be based on a bit anywhere in memory.	Conditional jump is usually based on status register bit.

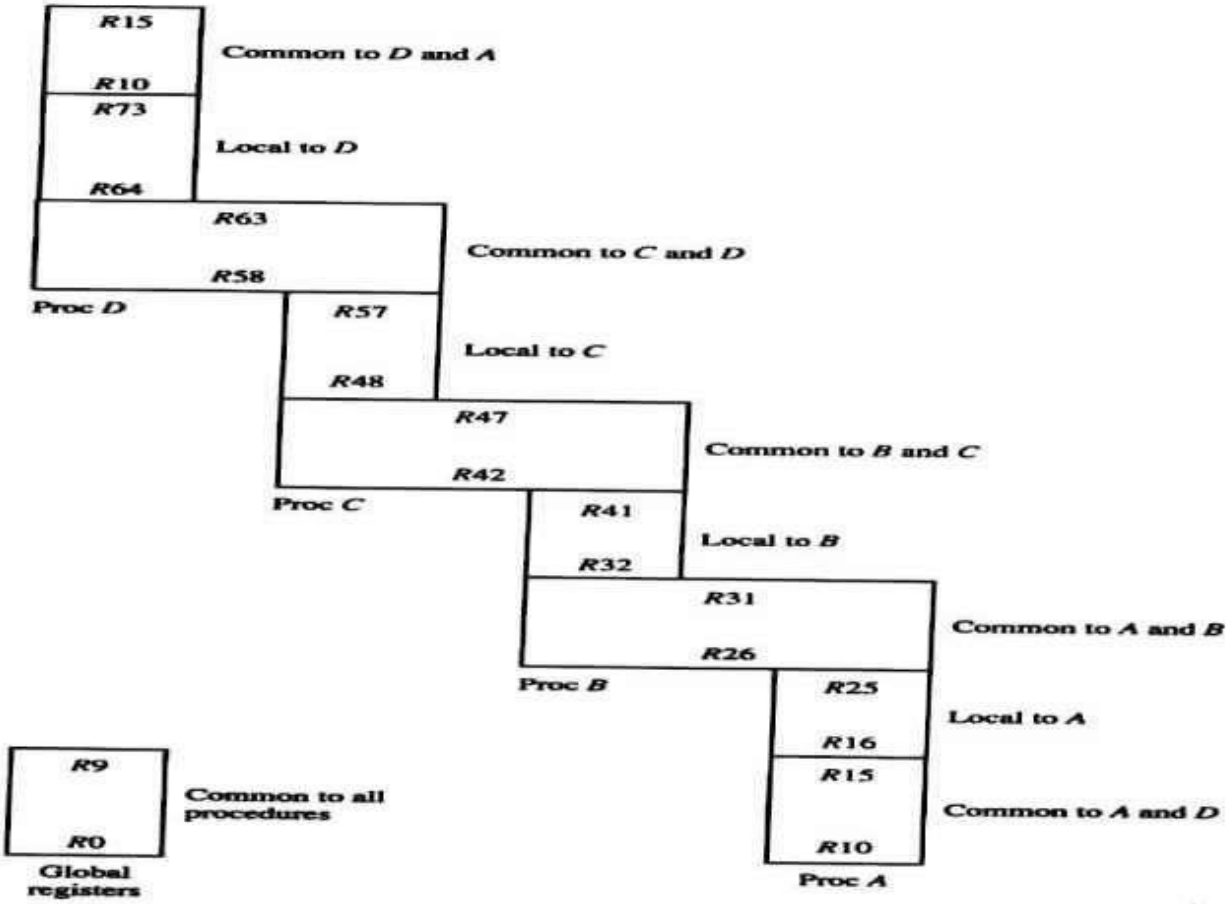


Figure 8-9 Overlapped register windows.

Parallel Processing:

Parallel processing is a term used for a large class of techniques that are used to provide **simultaneous data-processing tasks** for the purpose of increasing the computational speed of a computer system.

It refers to techniques that are used to provide simultaneous data processing.

The system may have two or more ALUs to be able to execute two or more instruction at the same time.

The system may have two or more processors operating concurrently.

It can be achieved **by having multiple functional units that perform same or different operation simultaneously.**

Parallel processing is done by distributing the data among multiple functional Units.

Processor with Multiple function units:

The following figure shows one possible way of separating the execution unit into 8 functional units operating in parallel

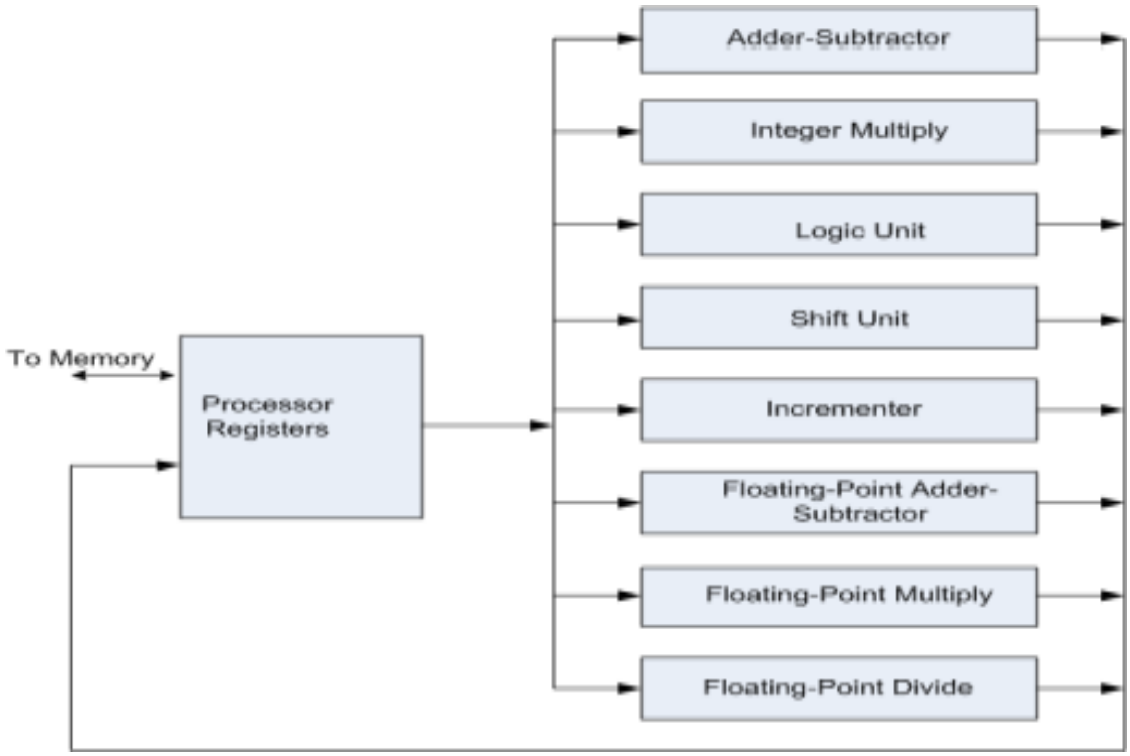


Fig: Processor with Multiple functional units

The operation performed in each functional unit is indicated in each block of the diagram.

The Adder and integer multiplier perform arithmetic operation with Integer numbers.

The floating point operations are separated into 3 circuits operating in parallel.

The logic, shift, and increment operation can be performed concurrently on different data.

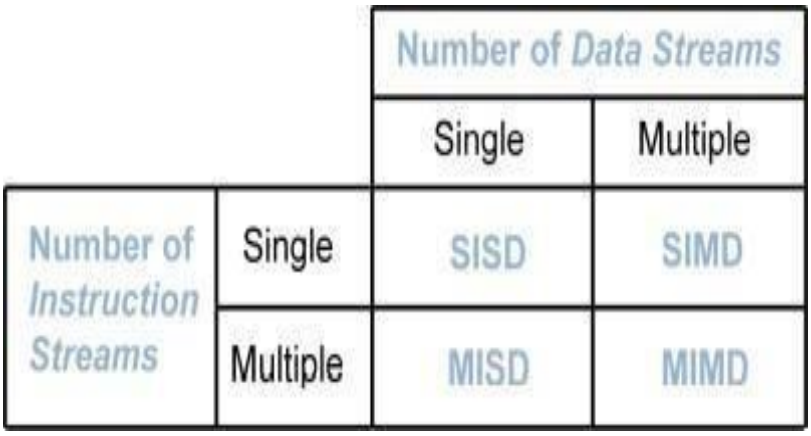
All units are independent, so one number can be shifted while another number is being activated.

- *Architectural Classification: –*

• Flynn's classification

- Considers the organization of a computer system by number of instructions and data items that are manipulated simultaneously.
- Based on the multiplicity of Instruction Streams and Data Streams
- **Instruction Stream**-Sequence of Instructions read from memory
- **Data Stream** - Operations performed on the data in the processor
- Parallel processing may occur in the instruction stream, in the data stream or in both.
- Flynn’s classification divides computer into 4 major groups:

1. SISD (Single Instruction stream, Single Data stream)
2. SIMD (Single Instruction stream, Multiple Data stream)
3. MISD (Multiple Instruction stream, Single Data stream)
4. MIMD (Multiple Instruction stream, Multiple Data stream



- SISD represents the organization containing single control unit, a processor unit and a memory unit.
- Instruction are executed sequentially and system may or may not have

internal parallel processing capabilities.

- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- MISD structure is of only theoretical interest since no practical system has been constructed using this organization.
- MIMD organization refers to a computer system capable of processing several programs at the same time.

  The main difference between **multicomputer system and multiprocessor system** is that the multiprocessor system is controlled by one operating system that provides interaction between processors and all the component of the system cooperate in the solution of a problem
- Parallel Processing can be discussed under following topics:
- **Pipeline Processing**
- **Vector Processing**
- **Array Processors**

PIPELINING

- A technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipelinig is a **collection of processing segments.**
- Each segment performs partial processing dictated by the way task is partitioned.
- The result obtained from each segment is transferred to next segment.
- The final result is obtained when data have passed through all segments.
- Suppose we have to perform the following task:
- Each sub operation is to be performed in a segment within a pipeline.

- Each segment has one or two registers and a combinational circuit.
- The **register holds the data**. The **combinational circuit performs the suboperation** in the particular segment.
- A clock is applied to all registers after enough time has elapsed to perform all segment activity.
- A clock is applied to all registers after enough time has elapsed to perform all segment activity.
- The pipeline organization will be demonstrated by means of a simple example.
- To perform the combined multiply and add operations with a stream of numbers

  Ai * Bi + Ci for i = 1, 2, 3, …, 7
- Each suboperation is to be implemented in a segment within a pipeline.

  R1 ← Ai , R2 ← Bi Input Ai and Bi

  R3 ← R1 * R2, R4 ← Ci Multiply and input Ci

  R5 ← R3 + R4 Add Ci to product
- Each segment has one or two registers and a combinational circuit as shown in Fig.

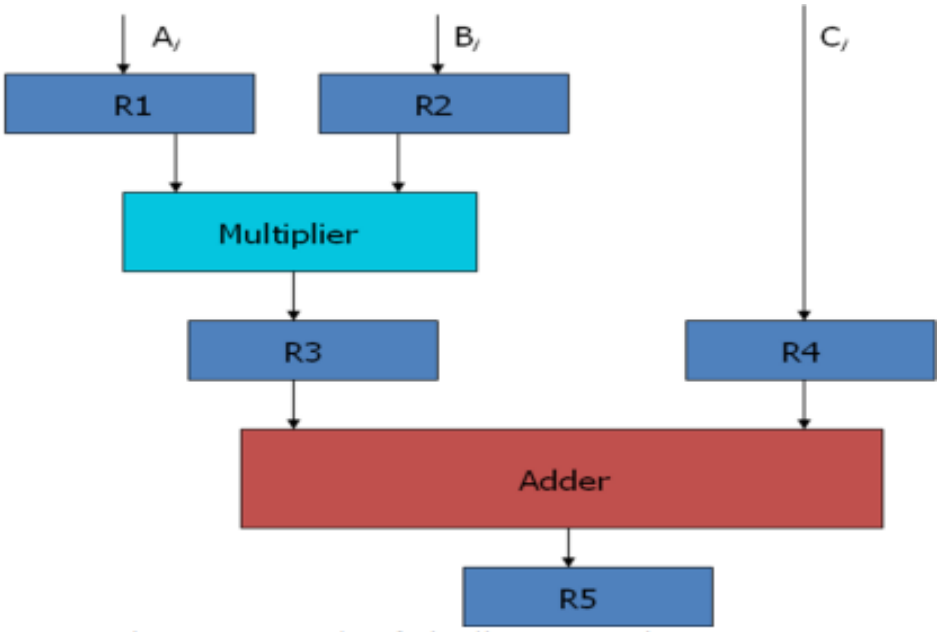


Fig 4-1: Example of pipeline processing

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	--	--	--
2	A_2	B_2	A_1*B_1	C_1	--
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	--	--	A_7*B_7	C_7	$A_6*B_6+C_6$
9	--	--	--	--	$A_7*B_7+C_7$

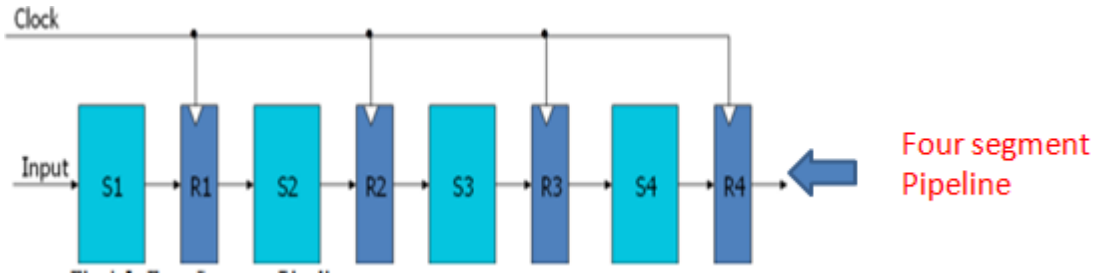
Table 4-1: Content of Registers in Pipeline Example

General Considerations:

- Any operation that can be decomposed into a sequence of suboperations of

about the same complexity can be implemented by a pipeline processor.

- The general structure of a **four-segment pipeline** is illustrated in Fig. 4-2. We define a task as the total operation performed going through all the segments in the pipeline.



- The behavior of a pipeline can be illustrated with a space-time diagram. o It shows the segment utilization as a function of time
- The space-time diagram of a four-segment pipeline is demonstrated in Fig

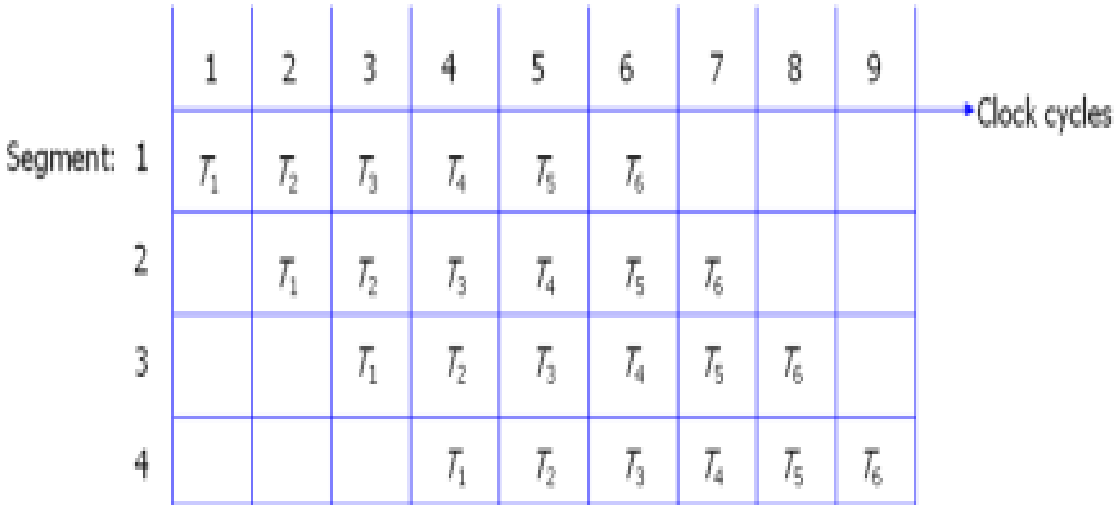


Fig 4-3: Space-time diagram for pipeline

- Where a k-segment pipeline with a clock cycle time tp is used to execute n tasks.
- The first task T1 requires a time equal to ktp to complete its operation.

- The remaining n-1 tasks will be completed after a time equal to (n-1)tp
- Therefore, to complete n tasks using a k-segment pipeline requires k+(n-1) clock cycles.
- Consider a nonpipeline unit that performs the same operation and takes a time equal to tn to complete each task.
- The total time required for n tasks is ntn.
- The speedup of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

  S = ntn/(k+n-1)tp .
- If n becomes much larger than k-1, the speedup becomes

  S = tn/tp.
- If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits, i.e.,tn = ktp, the speedup reduces to **S=ktp/tp=k.**
- This shows that the theoretical maximum speedup that a pipeline can provide is k, where k is the number of segments in the pipeline.
- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct k identical units that will be operating in parallel.
- This is illustrated in Fig. below, where four identical circuits are connected in parallel.
- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time

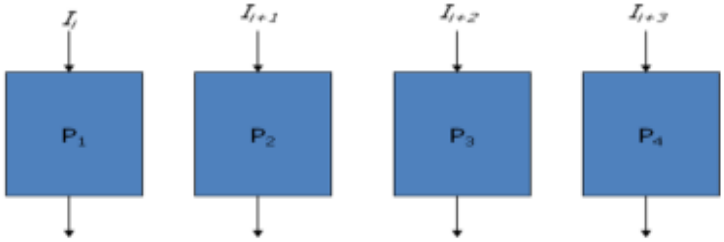


Fig 4-4: Multiple functional units in parallel

- There are various reasons why the pipeline cannot operate at its maximum theoretical rate.
- Different segments may take different times to complete their sub operation.
- It is not always correct to assume that a nonpipe circuit has the same time delay as that of an equivalent pipeline circuit.
- There are three areas of computer design where the pipeline organization is applicable.

Arithmetic pipeline

Instruction pipeline

RISC pipeline

Arithmetic pipeline:

- Pipeline arithmetic units are usually found in very high speed computers
- Floating–point operations, multiplication of fixed-point numbers, and similar computations in scientific problem
- Floating–point operations are easily decomposed into suboperations as demonstrated in Sec. 10-5.
- An example of a pipeline unit for floating-point addition and subtraction is showed in the following:
- The inputs to the floating-point adder pipeline are two normalized floating point binary number

$$X = A \times 2^a$$

$$Y = B \times 2^b$$

- A and B are two fractions that represent the mantissas, a and b are the exponents.
- The floating-point addition and subtraction can be performed in four segments, as shown in Fig. 9-6.
- The suboperations that are performed in the **four segments** are:

1. **Compare the exponents**
2. **Align the mantissa**
3. **Add or subtract the mantissas**
4. **Normalize the result**

Example: Consider two floating point numbers binary addition

$X = 0.9504 * 10^3$

Y = = 0.8200 * 10^2

1. Compare exponents by subtraction:

- The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.
- The difference of the exponents, i.e., 3 - 2 = 1 determines how many times the mantissa associated with the smaller exponent must be shifted to the right.

2. Align the mantissas:

- The next segment shifts the mantissa of Y to the right

$X = 0.9504 * 10^3$

$$Y = 0.08200 * 10^3$$

3. Add mantissas:

- The two mantissas are added in segment three.

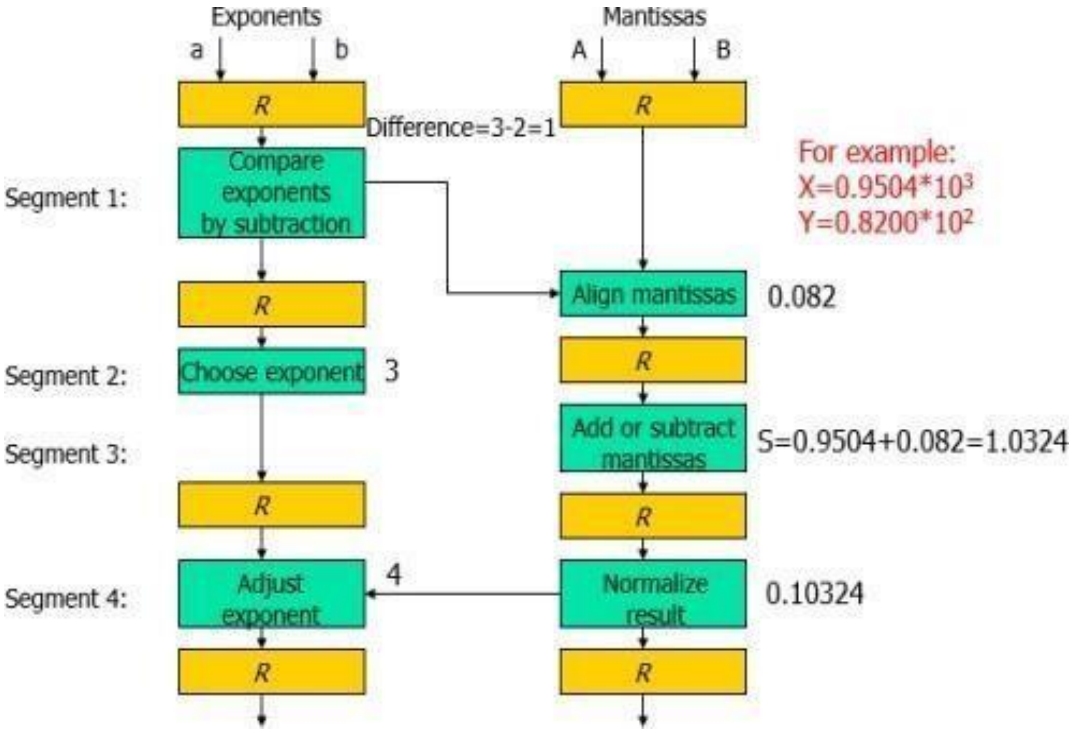
$$Z = X + Y = 1.0324 * 10^3$$

4. Normalize the result:

- After normalization, the result is written as:

$$\mathbf{Z = 0.1324 * 10^4}$$

Flow chart for floating point addition and subtraction using Pipelining



Pipelining for Floating point Addition and Subtraction

- The larger exponent is chosen as the exponent of the result
- The exponent difference determines how many times the mantissa associated

with the smaller exponent must be shifted to the right.

- When an **overflow occurs**, the mantissa of the sum or difference is **shifted right** and the exponent incremented by one.
- If an **underflow occurs**, the number of leading zeros in the mantissa determines the number of **left shifts** in the mantissa and the the exponent decremented by one.

Instruction Pipeline:

- Pipeline processing can occur not only in the data stream but in the instruction as well.
- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.
- Computers with complex instructions require other phases in addition to above phases to process an instruction completely.
- In the most general case, the computer needs to process each instruction with the following sequence of steps.

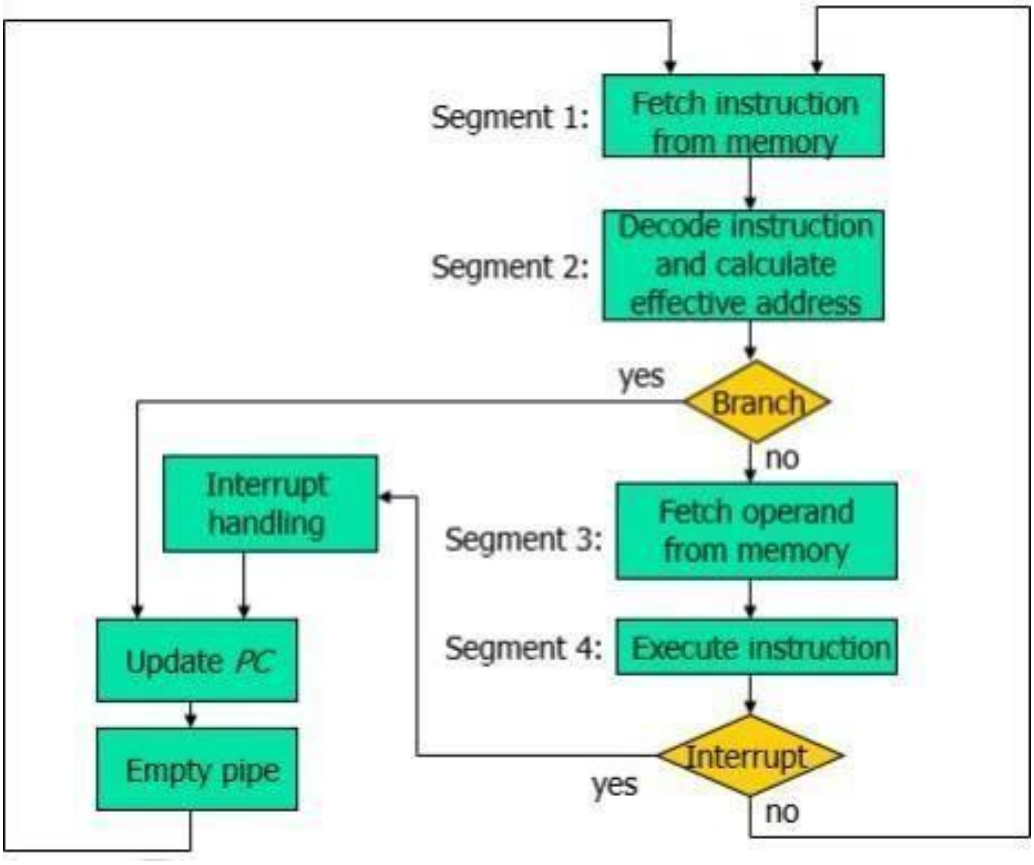
1. **Fetch the instruction from memory.**
2. **Decode the instruction.**
3. **Calculate the effective address.**
4. **Fetch the operands from memory.**
5. **Execute the instruction.**
6. **Store the result in the proper place.**

- There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate.
- Different segments may take different times to operate on the incoming information.

- Some segments are skipped for certain operations.
- Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Example: four-segment instruction pipeline:

- Assume that:
- The decoding of the instruction can be combined with the calculation of the effective address into one segment (**DA in segment 2 and FI in segment 1**).
- The instruction execution and storing of the result can be combined into one **segment(FO in segment 3 and IE in segment 4)**
- Fig 9-7 shows how the instruction cycle in the CPU can be processed with a four segment pipeline.



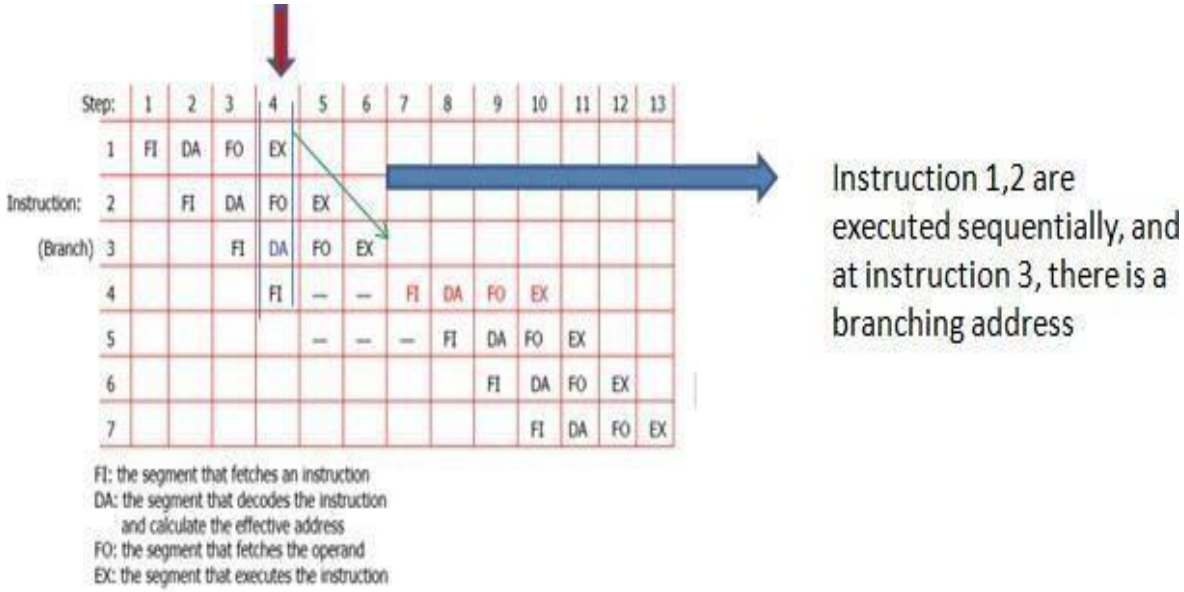
- **Thus up to four suboperations in the instruction cycle can overlap** and

up to four different instructions can be in progress of being processed at the same time.

- An instruction in the sequence may be causes **a branch out of normal sequence.**
- In that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted.
- Similarly, an interrupt request will cause the pipeline to empty and start again from a new address value.
- Fig. above shows the operation of the instruction pipeline.
- **The four segments are represented in the diagram with an abbreviated symbol.**
    1. **FI is the segment that fetches an instruction.**
    2. **DA is the segment that decodes the instruction and calculates the effective address.**
    3. **FO is the segment that fetches the operand.**
    4. **EX is the segment that executes the instruction**

Timing of Instruction Pipeline

- The time in the horizontal axis is divided into steps of equal duration.



At step4, Instruction 1 is executed
Instruction 2 is fetching operands from memory
Instruction 3(Brach Address) is decoded and calculating effective Address
Instruction 4 is fetching Instruction from memory
After Decoding the Branch address in Instruction 3, the transfer of other instruction from FI to DA is halted until the **current Instruction is executed in step6**
If the Branch address condition is satisfied, **a new Instruction** is fetched in **step7.**

- **Pipeline Hazards**
- It is a conflict that prevents an instruction from executing during its designated clock cycles.
- In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. **Structural Hazards**
2. **Data Hazard**
3. **Control hazard**

1. Structural Hazards:

- These are the Resource conflicts caused by access to memory by two segments at the same time.

- Can be resolved by using separate instruction and data memories

2. **Data Hazard:**

   These conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3. **Control Hazard:**

   These conflicts arise when an Branch instruction arise and this branch instruction causes the change the value of PC.

RISC (Reduced Instruction Set Computer)Pipeline:

- The data transfer instructions in RISC are LOAD and STORE.
- To prevent conflicts between a memory access to fetch an instruction and to load or store an operand, most RISC machines use two separate buses with two memories
- One for storing information and other for storing data.
- Example: Three-Segment Instruction Pipeline
- There are three types of instructions:
- The data manipulation instructions: operate on data in processor registers
- The data transfer instructions(load and store)
- The program control instructions(branch instructions)
- The instruction cycle can be divided into three suboperations and implemented in three segments:

I: Instruction fetch

- Fetches the instruction from program memory

A: ALU operation

- The instruction is decoded and an ALU operation is performed. It performs an operation for a data manipulation instruction, It evaluates the effective address for a load or store instruction. It calculates the branch address for a

program control instruction.

E: Execute instruction

- Directs the output of the ALU to one of three destinations, depending on the decoded instruction. It transfers the result of the ALU operation into a destination register in the register file.
- It transfers the effective address to a data memory for loading or storing.
- It transfers the branch address to the program counter.

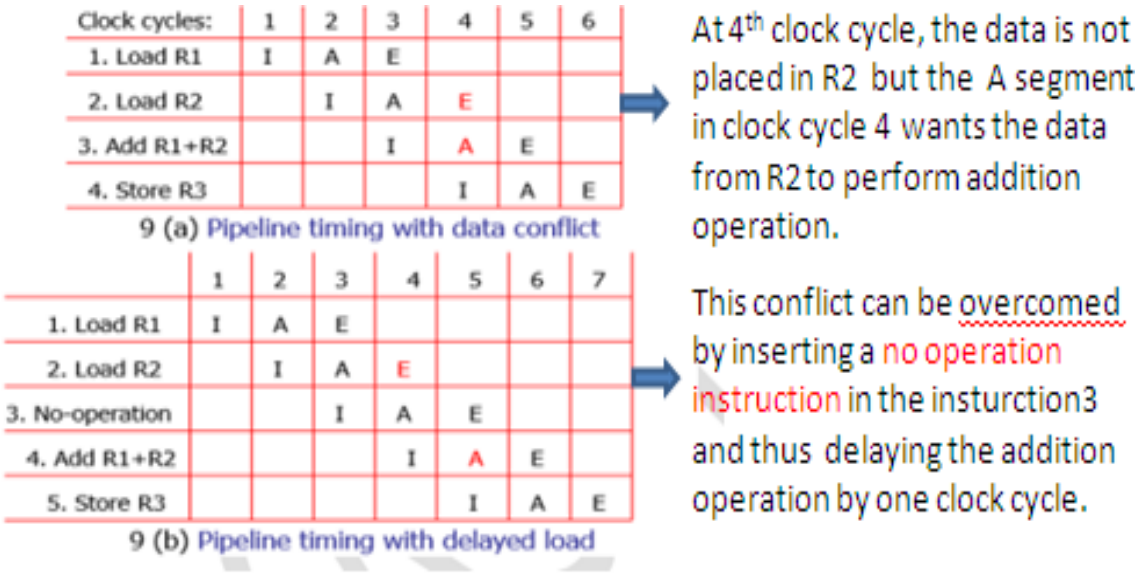
Delayed Load:

- Consider the operation of the following four instructions:

1. LOAD: R1 ← M[address 1]
2. LOAD: R2 ← M[address 2]
3. ADD: R3 ← R1 +R2
4. STORE: M[address 3] ← R3

- There will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment.
- This can be seen from the timing of the pipeline shown in Fig. 9-9(a).

Pipelining Timing with Delayed load:



This concept of delaying the data loaded into the memory is referred as " Delayed Load"

Delayed Branch

- The method used in most RISC processors is to rely on the compiler to redefine the branches so that they take effect at the proper time in the pipeline.
- This method is referred to as delayed branch.
- The compiler is designed to analyze the instructions before and after the branch and rearrange the program sequence by inserting useful instructions in the delay steps.
- It is up to the compiler to find useful instructions to put after the branch instruction. Failing that, the compiler can insert no-op instructions.
- **An Example of Delayed Branch**:
- The program for this example consists of five instructions.

    1. Load from memory to R1

2. Increment R2
3. Add R3 to R4
4. Subtract R5 from R6
5. Branch to address X

- In Fig. 9-10(a) the compiler inserts two no-op instructions after the branch.
- The branch address X is transferred to PC in clock cycle 7.
- The program in Fig. 9-10(b) is rearranged by placing the add and subtract instructions after the branch instruction.
- PC is updated to the value of X in clock cycle 5.

Pipelining Timing with Delayed Branch:

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in X								I	A	E

10 (a) Using no-operation instructions

The compiler inserts two no-op instructions at 6 &7 after the branch instruction (5).The branch address X is transferred to PC in clock cycle 7.so the fetching of instruction is delayed by 2 clock cycles and branch address x is fetched at instruction 8.

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in X						I	A	E

10 (b) Rearranging the instructions

The program is rearranged by placing the add and subtract instructions after the branch instruction Inspection of the pipeline timing shows that PC is updated to the value of X in clock cycle 5

Vector processing:

- Normal computational systems are not enough in some special processing requirements
- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.
- Computers with vector processing capabilities are in demand in specialized applications.

Examples:

- **Long-range weather forecasting**
- **Petroleum explorations**
- **Seismic data analysis**
- **Medical diagnosis**
- **Artificial intelligence and expert systems**
- **Image processing**
- **Mapping the human genome**

The term vector processing involves the data processing on the vectors of involving high amount of data.

- The large data can be classified as very big arrays.
- The vectors are considered as the large one dimensional array of data.
- The vector processing system can be understood by the example below.
- **EX: Consider a program which is adding two arrays A and B of length 100 to produce a vector C**
- Machine level program

Initialize I=0

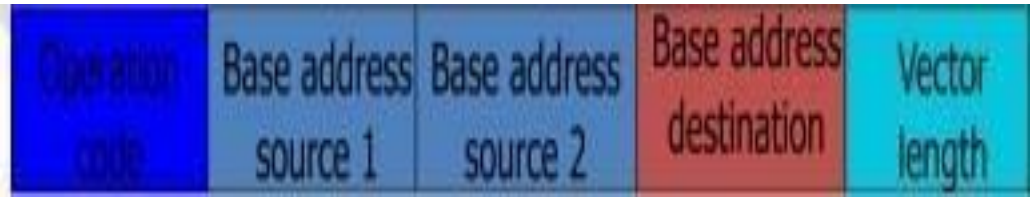
Read A(I)

```
                Read B(I)
20              Store C(I)=A(I)+B(I)
                Increment I=I+1
                If I<=100 go to 20 continue
```

- so in this above program we can see that the two arrays are being added in a loop format.
- First we are starting from the value of 0 and then we are continuing the loop with the addition operation until the I value has reached to 100.
- In the above program there are **5 loop statements** which will be executing 100 times.
- Therefore the total cycles of the CPU taken are **500 cycles**.
- But if we use the concept of vector processing then we can reduce the unnecessary fetch cycles.
- The same program written in the vector processing statement is given below:

C(1:100)=A(1:100)+B(1:100)

- In the above statement, when the system is creating a vector like this the original source values are fetched from the memory into the vector.
- Therefore the data is readily available in the vector.
- So when a operation is initiated on the data, naturally the operation will be performed directly on the data and will not wait for the fetch cycle.
- So the **total no of CPU Cycles** taken by the above instruction is only **100**
- **Instruction format of vector Instruction:**



Matrix Multiplication

- The multiplication of two n x n matrices consists of n2 inner products or n3 multiply-add operations.
- Consider, for example, the multiplication of two 3 x 3 matrices A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

The product matrix C is a 3×3 matrix whose elements are related to the elements of A and B by the inner product:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

For example, the number in the first row and first column of matrix C is calculated by letting $i = 1, j = 1$, to obtain

c11= a11b11+ a12b21+ a13b31

- This requires three multiplication and (after initializing c11 to 0) three additions.
- In general, the inner product consists of the sum of k product terms of the form

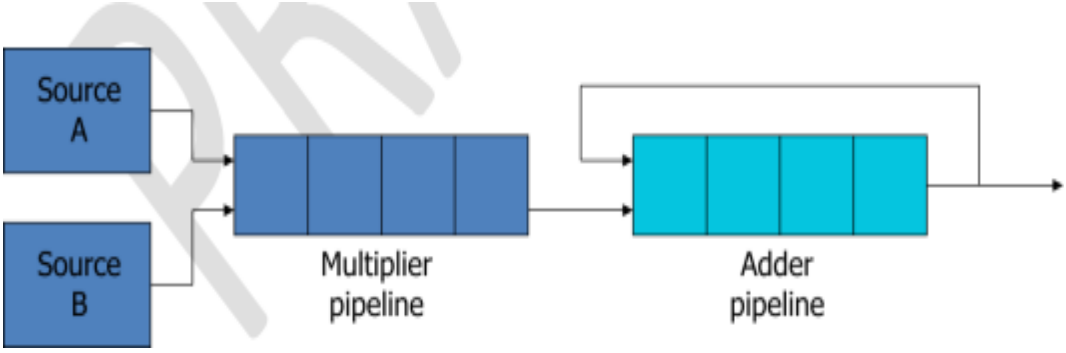
C = A1B1+A2B2+A3B3+…+AkBk.

- In a typical application k may be equal to 100 or even 1000.

$$\begin{aligned} C = {} & A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots \\ & + A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots \\ & + A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \cdots \\ & + A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots \end{aligned}$$

Implementation of the Vector Processing

- Below we can see the implementation of the vector processing concept on the following matrix multiplication.

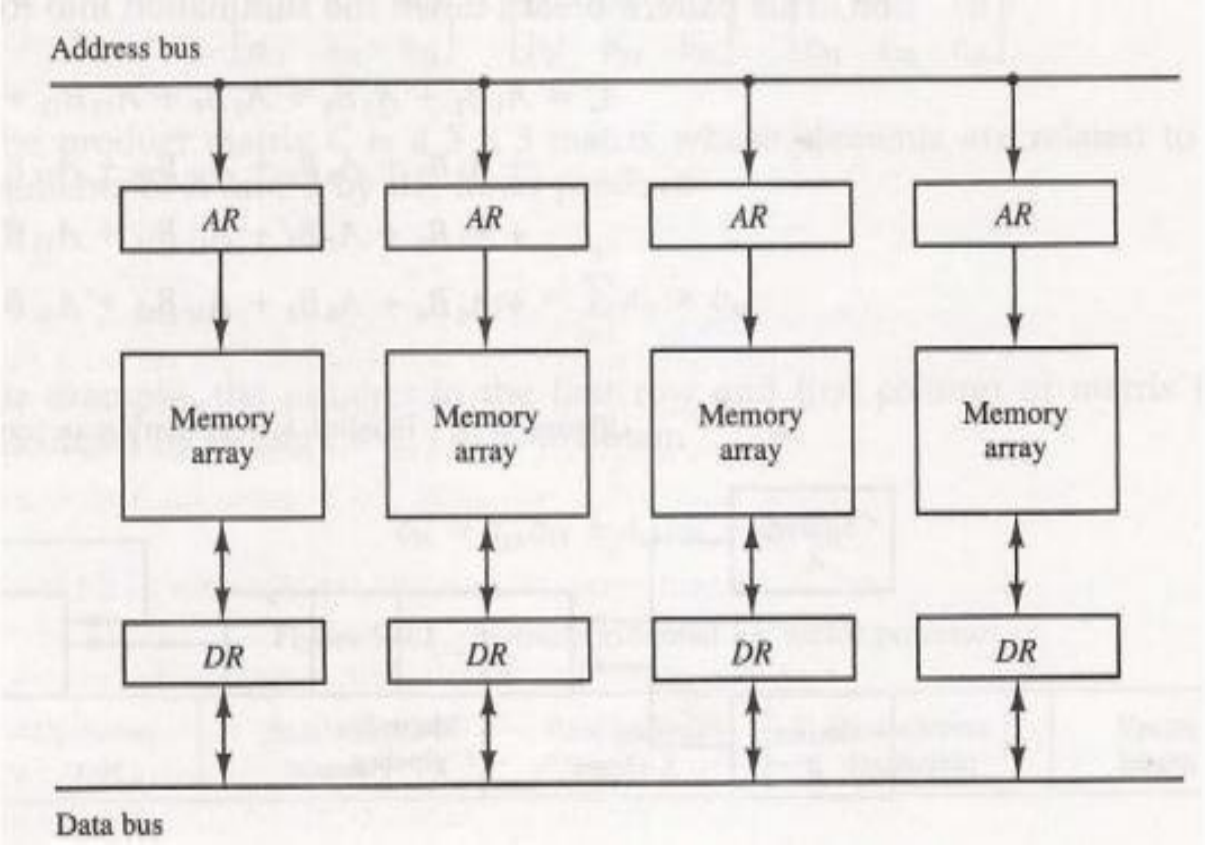


- In the above diagram we can see that how the values of A vector and B Vector which represents the matrix are being multiplied. Here we will be considering a 4x4 matrix A and B.
- When addition operation is taking place in the adder pipeline the next set of values will be brought into the multiplier pipeline, so that all the operations can be performed simultaneously using the parallel processing concepts by the implementation of pipeline.

Memory Interleaving:

- Pipeline and vector processors often require simultaneous access to memory from two or more sources.
- An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.

- An arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time.
- **Instead of using two memory buses for simultaneous access**, **the memory can be partitioned into a number of modules** connected to a common memory address and data buses.
- A memory module is a memory array together with its own address and data registers.
- Fig. 9-13 shows a memory unit with four modules.



Multiple module Memory Organization

- The advantage of a modular memory is that it allows the use of a technique called interleaving.
- In an interleaved memory, different sets of addresses are assigned to different memory modules.

- By staggering the memory access, the effective memory cycle time can be reduced by a factor close to the number of modules.

Array Processors:

- An array processor is a processor that performs computations on large arrays of data.
- The term is used to refer to two different types of processors.

Attached array processor:

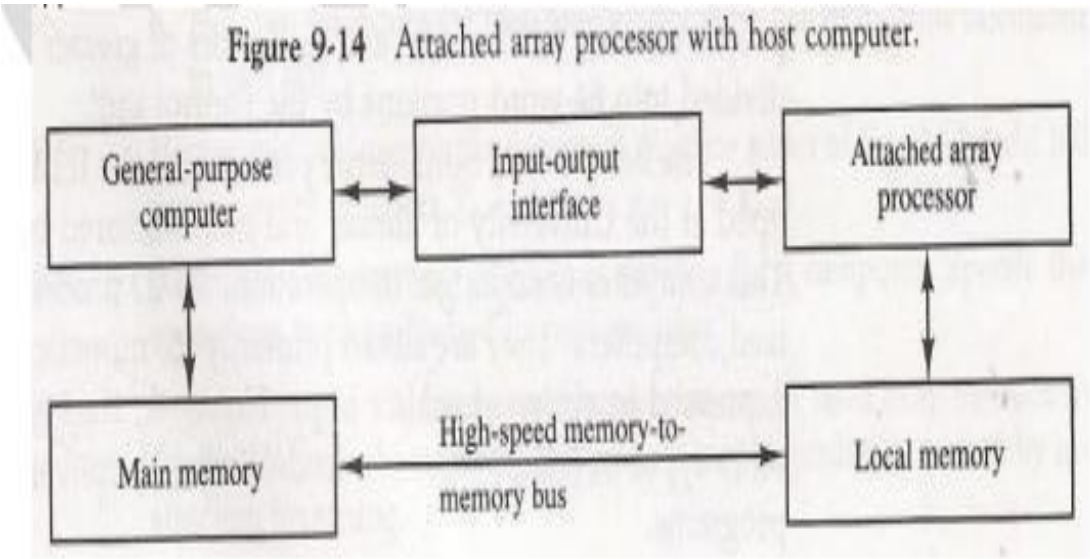
It is an auxiliary processor. It is intended to improve the performance of the host computer in specific numerical computation tasks.

SIMD array processor:

Has a single-instruction multiple-data organization. It manipulates vector instructions by means of multiple functional units responding to a common instruction

Attached Array Processor

- Its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
- Parallel processing with multiple functional units
- Fig. 9-14 shows the interconnection of an attached array processor to a host computer.



Attached Array Processor with host computer

- The host computer is a general-purpose commercial computer and the attached processor is a back-end machine driven by the host computer.
- The array processor is connected through an input-output controller to the computer and the computer treats it like an external interface.
- The data for the attached processor are transferred from main memory to a local memory through a high-speed bus.
- The general-purpose computer without the attached processor serves the users that need conventional data processing.
- The system with the attached processor satisfies the needs for complex arithmetic applications.
- For example, when attached to a VAX 11 computer, the FSP-164/MAX from Floating Point Systems increases the computing power of the VAX to 100megaflops.
- The objective of the attached array processor is to provide vector manipulation capabilities to a conventional computer at a fraction of the cost of supercomputer.

SIMD Array Processor:

- An SIMD array processor is a computer with multiple processing units operating in parallel.
- A general block diagram of an array processor is shown in Fig. 9-15.

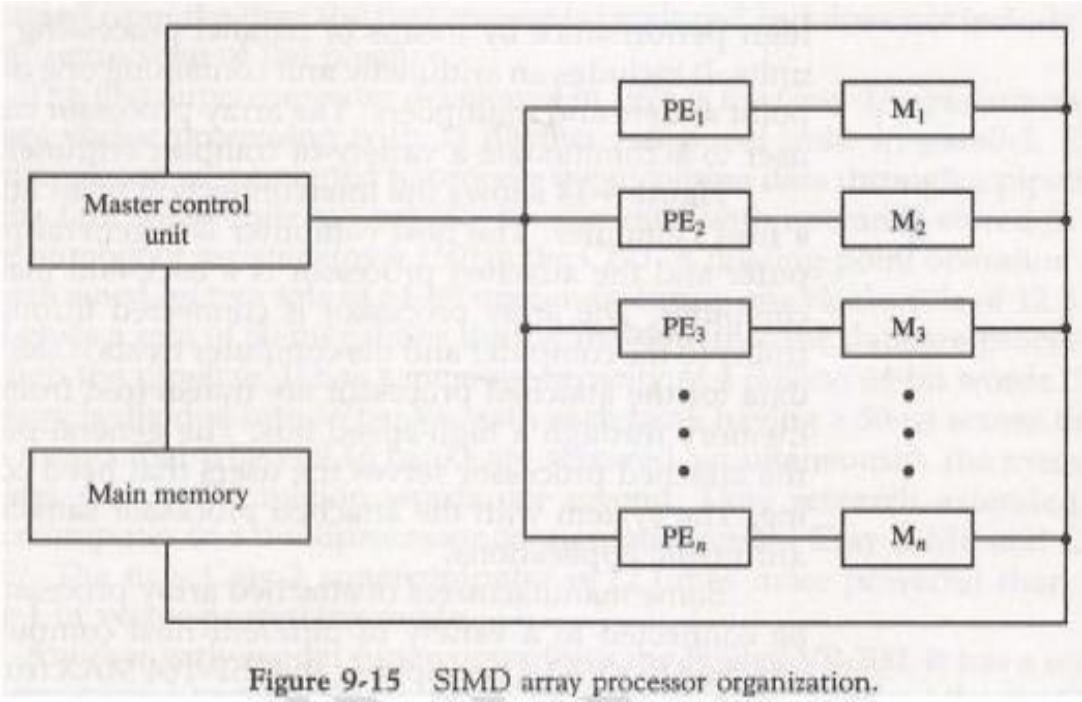


Figure 9-15 SIMD array processor organization.

- It contains a set of identical processing elements (PEs), each having a local memory M.
- Each PE includes an ALU, a floating-point arithmetic unit, and working registers.
- Vector instructions are broadcast to all PEs simultaneously.
- Masking schemes are used to control the status of each PE during the execution of vector instructions.
- Each PE has a flag that is set when the PE is active and reset when the PE is inactive.
- For example, the ILLIAC IV computer developed at the University of

Illinois and manufactured by the Burroughs Corp.

- Are highly specialized computers.
- They are suited primarily for numerical problems that can be expressed in vector or matrix form

Characteristics of multiprocessors

A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment. The term "processor" in multiprocessor can mean either a central processing unit (CPU) or an input-output processor (IOP). Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system. A multiprocessor improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the function of the disabled processor. This causes the loss in efficiency of the system but with no delay.

Multiprocessors can improve performance by decomposing the program into parallel executable tasks. This can be achieved in two ways.

*Multiple independent jobs can be made to operate in parallel.

*A single job can be partitioned into multiple parallel tasks.

Multiprocessors are classified by the way their memory is organized

1. Shared memory or tightly coupled multiprocessor:

A multiprocessor system with common shared memory is classified as shared memory. It provides a cache memory with each CPU. There will be a global common memory that all CPUs can access. Information

can therefore be shared among the CPUs by placing it in the common global memory.

2. Disturbed memory or loosely coupled system:

Each processor element in a loosely coupled system has its own private local memory. The processors relay programs and data to other processors in packets, which consists of an address, the data content, and error detecting code.

Interconnection Structures

There are several physical forms available for establishing an interconnection network. Some of these schemes are

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

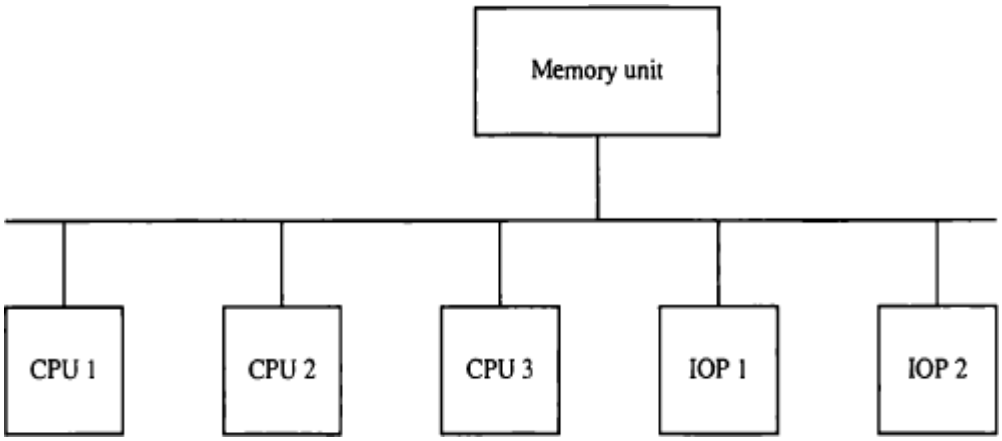
1. Time-shared common bus:

A common bus multi processor system consists of a number of processors connected through a common path to a memory unit. Only one processor can communicate with the memory or another processor at any given time. Any processor wishing to initiate a transfer must first determine the availability of the bus. A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address and responds to the control signals from the sender, after this the transfer will be initiated.

The system may exhibit transfer conflicts since all processors share one common bus. These conflicts must be resolved by putting a bus controller that establishes priorities among the requesting units.

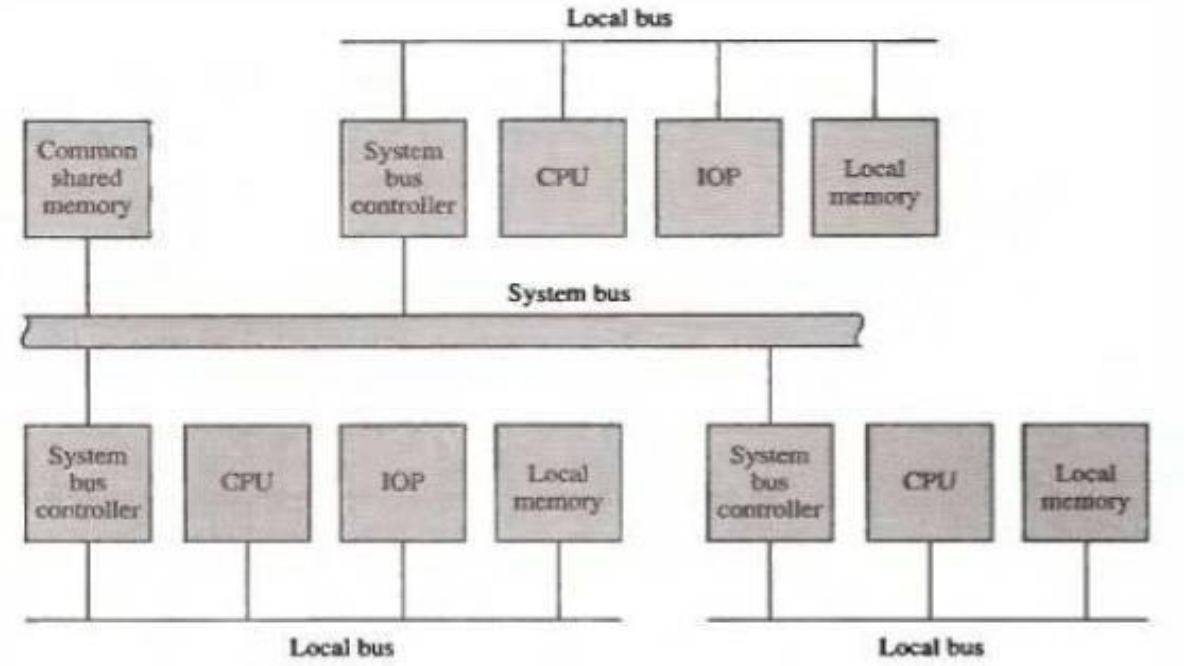
The processors in the system are kept busy through the implementation

of two or more buses to permit multiple simultaneous bus transfer. It increases the system cost and complexity.



Time shared common bus organization

Consider implementation of dual bus structure. In this a number of local buses connected to its own local memory and to one or more processors. Each local bus is connected to a CPU, an IOP or any combination of processors. A system bus controller links each local bus to common system bus. I/O devices connected to local memory as well as to local IOP are available to local processor



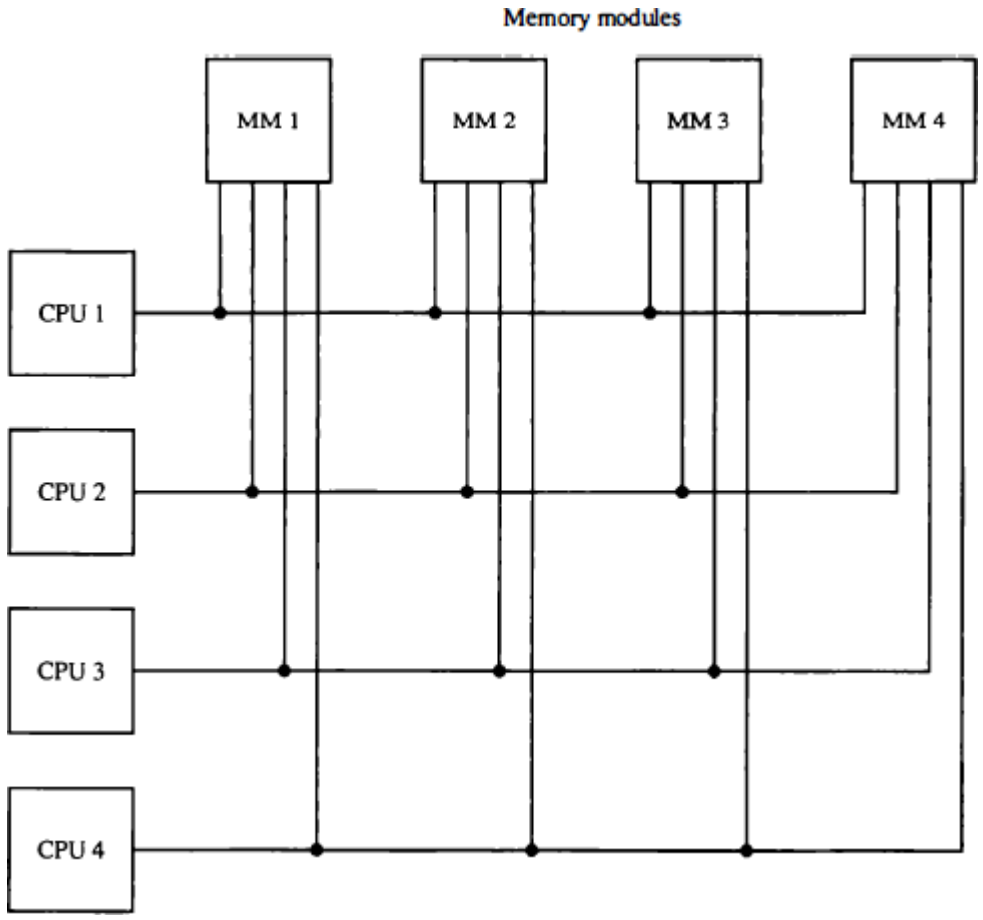
If an IOP is connected directly to the system bus, the I/O devices attached to it may be made available to the processors. Only one processor can communicate with the shared memory and other common resources through the system bus at any given time.

The other processors are kept busy communicating with their local memory and I/O devices. Part of local memory may be designed as a cache memory attached to CPU. In this way average access time of local memory can be made to approach the cycle time of CPU to which it is attached.

2. Multiport memory:

A multiport memory system employs separate buses between each memory module (MM) and each CPU. Each processor bus is connected to each memory module. A processor bus consists of the address, data and control lines required to communicate with memory. MM is said to have 4ports and each port accommodates one of the buses. The module must have internal control logic to determine which port will have access to memory at any given time.

The advantage of multiport memory organization is the higher transfer rate because of multiple paths between processor and main memory. The disadvantage is it requires expensive control logic and a large number of cables and connectors. This structure is appropriate for a system with large number of processors.



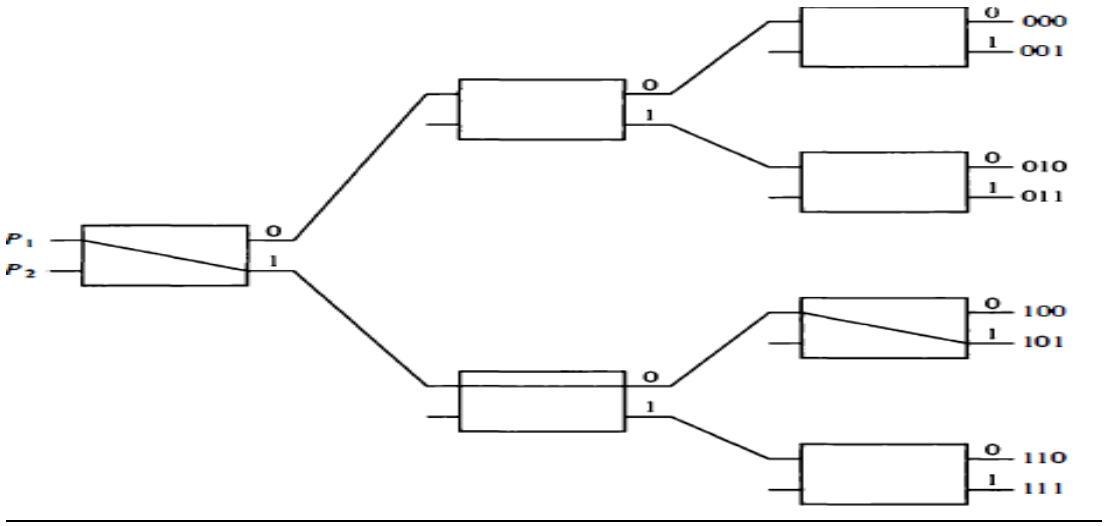
Multiport memory organization

4. Multistage Switching Network:

The basic component of a multistage network is a two-input, two-output interchange switch. A 2*2 switch has two inputs, labeled A and B and two- outputs labeled 0 and 1. The switch has the capability of connecting inputs A or B to either of the outputs. If inputs A and B both requests for the same output only one of them will be connected other will be blocked.

The two processors P1 and P2 are connected through switches to eight memory modules marked in binary from 000 through 111. The first bit of destination number determines the switch output in the first level.

For example to connect P1 to memory 101,it is necessary to form paths from P1 to o/p 1 in t he 1st level switch, o/p 0 in the 2nd level switch and o/p in the 3rd level.



Omega Network:

Omega switching network have been proposed for multistage switching network to control processor-memory communication. In this configuration there is exactly one path from each source to any particular destination. A particular request is initiated in the switching network by source, which sends a 3-bit pattern representing the destination number. As the binary pattern moves through the network, each level examines a different bit to determine the 2*2 switch setting. When the request arrives on either input of 2*2 switch, it is routed to the upper output if the specified bit is 0 or to the lower output if it is 1.

Omega Network in a tightly coupled multiprocessor:

In this the source is a memory module and the destination is a memory module. The first pass through the network sets up the path. Succeeding paths are used to transfer the address into memory and then transfer the data in read or write direction.

Omega Network in loosely coupled multiprocessor:

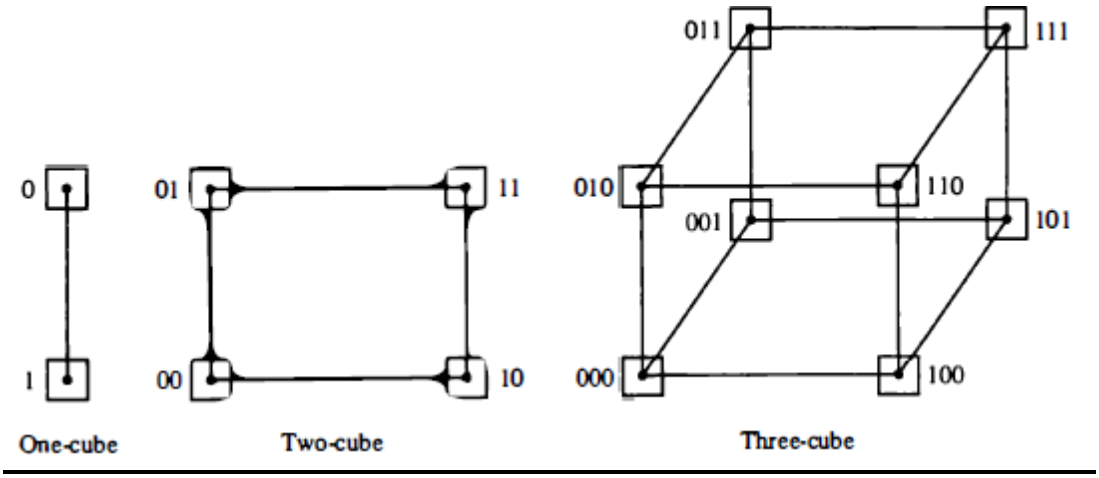
In this both the source and the destination are the processing elements. After the path is established the source processor transfers a

message to the destination processor.

5. *Hypercube Interconnection*

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of 2 power of n processors interconnected in an n-dimensional binary cube. Each processor forms a node of the cube. Each processor has direct communication paths to n other neighbor processors. Each processor address differs from that of its neighbors by exactly one bit position.

Routing messages through an n-cube structure may take from one to n links from a source node to a destination node. For example, in a three-cube structure node 000 can communicate directly with node 001. It must cross at least two links to communicate with 011. Computing the ex-or of the source node address with the destination node address can develop a routing procedure.



5.1. Interprocessor Arbitration

Computer systems contain a number of buses at various levels to facilitate the transfer of information between components. A bus that connects the major components in a multiprocessor system such as CPUs, IOPs and memory is called a system bus. The processors in a shared memory

multiprocessor system request access to common memory through the system bus. The requesting processor may wait if another processor is currently utilizing the system bus. Arbitration must then be performed to resolve this multiple contention for the shared resources. Arbitration logic is a part of system bus controller placed between the local bus and the system bus.

System Bus:

A typical system bus consists of approximately 100 signal lines. These lines are divided into three functional groups data, address and control lines.

Six bus arbitration signals

These are used for Interprocessor arbitration.

Bus request	BREQ
Common Bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO

Functions of the Bus Arbitration Signals

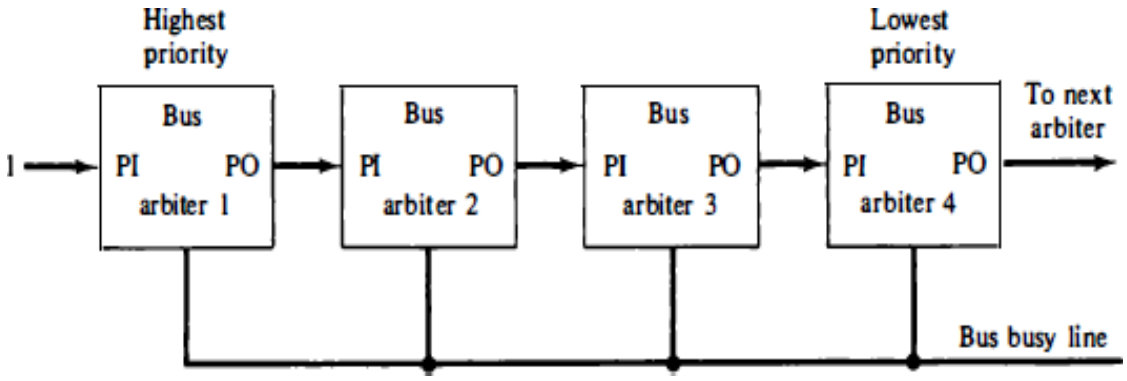
The bus priority –in BPRN and the bus priority-out BPRO are used for a daisy-chain connection of bus arbitration circuits. The bus BUSY signal is an open collector output to instruct all arbiters when the bus is busy. The common bus request CBRQ is also an open collector output used to instruct the arbiter if there are any other arbiters of lower-priority requesting the use of bus. The signals used to connect parallel arbitration procedure are bus request BREQ and priority-in BRPN for request and ack signals respectively. The bus clock BCLK is used to synchronize all bus transactions

Serial Arbitration Procedure:

A hardware bus priority resolving techniques can be established by

means of a serial or parallel connection of units requesting control of the system bus.

Serial priority resolving technique is obtained from a daisy-chain interconnection of bus arbitration circuits. Each processor has its own bus arbitration logic with priority in and priority out lines. The priority out (PO) of each arbiter is connected to priority in (PI) of next lower-priority. The PI of highest priority unit is maintained at logic 1.



Serial Arbitration

The processor whose arbiter has a PI=1 and PO=0 is one that is given the control of the system bus. The PO output for a particular arbiter is equal to 1 f its PI input is equal to 1 and the processor associated with the arbiter logic is not requesting the control of the bus.

Parallel Arbitration Logic:

The parallel arbitration logic uses an external priority encoder and a decoder. Each bus arbiter in this parallel scheme has a bus request output line and a bus ack input line. The processor takes control of the bus if its ack input line is enabled. First the request lines from the four arbiters 4*2 priority encoder. The output of the encoder generates a 2-bit code, which represents the highest priority unit among those requesting the bus. The 2-bit code from the encoder output drives a 2*4 decoder, which enables the proper ack line to grant

bus access to the highest priority unit.

Dynamic Arbitration Algorithms:

In a dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation. The few arbitration procedures that follow dynamic priority algorithms are:

Time slice: It allocates a fixed length time slice of bus time that is offered sequentially to each processor. The service given to each component is independent of its location along the bus. No preference is given to any particular since each is given same amount of time to communicate with the bus.

Polling: In a bus system that uses polling, poll lines replace bus grant signal. The bus controller to define an address for each device connected to the bus uses Poll lines. After a number of bus cycles, the polling process continues by choosing different processor.

LRU: The least recently used (LRU) algorithm gives the highest priority to the requesting device that has not used the bus for the longest interval. With this procedure no processor is favored over any other since the priorities are dynamically changed to give every device an opportunity to access the bus.

FIFO: In the first-come, first-serve scheme, requests are served in the order received. For this the bus controller establishes a queue according to the time that the bus requests arrive. Each processor must wait for its turn to use the bus on FIFO basis.

Rotating daisy-chain: This procedure is a dynamic extension of daisy-chain algorithm. In this scheme there is no central bus controller and the priority line is connected from the priority-out of the last device back to the priority-in of the first device in a closed loop. Each arbiter priority for a given bus cycle is given by its position along the bus priority line from the arbiter whose

processor is currently controlling the bus. Once, an arbiter releases the bus it has lowest priority.

Interprocessor Communication

The various processors in a multiprocessor system must be provided with a facility for communicating with each other.

In a shared memory multi processor system, the most common procedure is to set aside a portion of memory that is accessible to all processors. The primary use of common memory is to act as a message center similar to a mailbox, where each processor can leave messages for other processors and pick up messages intended for it. The sending processor structures a request, a message or a procedure and places it in the memory mail box, whether it has meaningful information, and which processor it is intended. The receiving processor can check the mailbox periodically if there are valid messages for it.

In addition to shared memory, a multiprocessor system may have other shared resources. To prevent conflict use of shared resources by several processors there must be a provision for assigning resources to these processors. This task is given to the operating system.

There are three organizations that have been used in the design of the operating system for multiprocessors:

1. **Master-slave configuration**: one processor designated the master, always executes the operating system functions. The remaining processors as slaves do not perform the operating system functions. If a slave processor needs an operating system service, it must request it by interrupting the master and waiting until the current program can be interrupted.

2. **Separate operating system:** Each processor can execute the operating system routines its needs. This is more suitable for loosely coupled systems where every processor may have its own copy of the entire os.

3. Distributed operating system: In this each particular os has one processor at a time. This type of organization is also called floating os since the routine floats from one processor to another and the execution of routine may be assigned to different processors at different times.

In a loosely coupled multiprocessor system the memory is distributed among the processors and there is no shared memory for passing information. The communications between processors is by means of message passing through I/O channels. When the sending processor and the receiving processor name each other as source and destination, a channel of communication is established.

Interprocessor Synchronization

Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data. A number of hardware mechanisms for mutual exclusion have been developed. One of the most popular methods is binary semaphore.

Mutual exclusion with a Semaphore: It is necessary to protect data from being changed simultaneously by two or more processors. This mechanism has been termed mutual exclusion. Mutual exclusion must be provided in a multiprocessor system to enable one processor to exclude or lock out access to a shared resource by other processors when it is in a mutual section. It is a program sequence that, once begun, must complete execution before another processor accesses the same-shared resource.

A binary variable called a semaphore is often used to indicate whether or not a processor is executing a critical section. It is a software-controlled flag that is stored in a memory location that all processors can access. When it is equal to 1, it means that a processor is executing a critical program, so that the shared memory is not available to other processors. When it is equal to 0, the shared memory is available to any requesting processor.

Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation. This action would allow simultaneous execution of a critical section at the same time, which can result in erroneous initialization of control parameters and a loss of essential information. A semaphore can be initialized by means of a test and set instruction in conjunction with a hardware lock mechanism. A hardware lock is a processor- generated signal that serves to prevent other processors from using the system bus as long as the signal is active. The other processor changes the semaphore between the time that the processor is testing it and the time that it is setting it.

Cache Coherence:

The primary advantage of cache is to its ability to reduce the average access time in uniprocessors. When a processor finds a writ operation in cache during write operation there are two commonly used procedures to update memory.

1. write-through policy:

Both cache and maim memory are updated with every write operation

2. Write back policy:

Only cache is updated and the location is marked so that it can be copied later into the main memory.

A memory scheme is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address. Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.

Conditions for cache incoherence

To illustrate the problem of cache coherence, consider three processor configurations with private caches. Sometime during the operation an element

X from main memory is loaded into the three processors P1, P2, P3. It is also copied into the private caches of the three processors. Consider X as 52. The load on X to the three processors results in consistent copies in the cache and main memory.

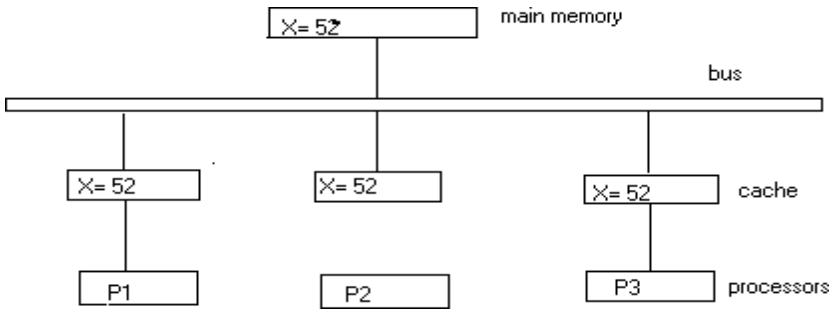
A store to X into the cache of processor P1 updates memory to the new value in a write through policy. A write through policy maintains consistency between memory and the originating cache, but the other two caches are inconsistent still they hold the same old value.

In writ back policy; main memory is not updated at the time of store. The copies in the other two caches and main memory are inconsistent. Memory is updated when the modified data is copied back into the main memory.

Solutions to the cache coherence problem:

Cache coherence problems can be solved by means of software and hardware combinations.

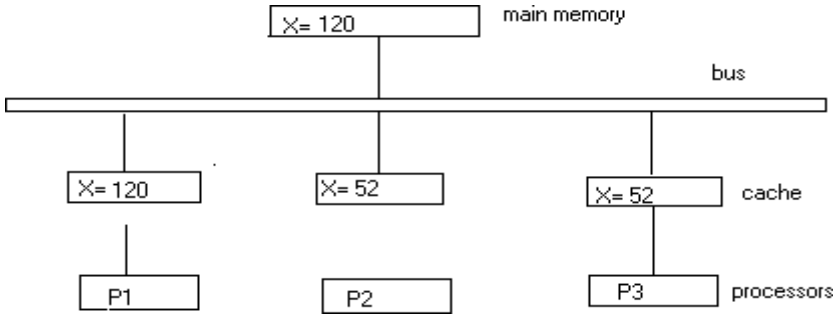
In the software solution, it is desirable to attach a private cache to each processor. If cache allows nonshared and read only data, such items are called cachable. Shared write only data are non cachable which remains in main memory. This method restricts the type of data stored in caches



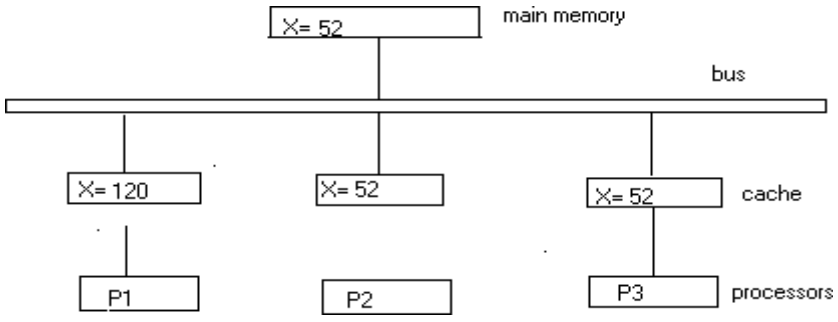
Cache configuration after a load X

Another scheme that allows writable data to exist in one cache is a method that employs centralized global table in its complier. Each block is identified as read only (RO) or write and read (WR). Only one copy of cache is RO. Thus if data is updated in RW block other caches are not affected because they do not have a copy of this block.

In the hardware solution, a cache controller called snoopy cache controller is specially designed to allow it to monitor all bus requests from CPUs and IOPs. All caches attached to the bus constantly monitor these network for possible write operations. It is basically a hardware unit designed to maintain a bus –watching mechanism over all the caches attaches to the bus. In this way inconsistent ways are prevented.



With write through cache policy



With write back cache policy