COMPUTER ORGANIZATION AND ARCHITECTURE

MODULE 5

REDUCED INSTRUCTION SET COMPUTER

INTRODUCTION OF RISC & CISC: In the early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. This type of computer is classified as a reduced instruction set computer or RISC. In this section we introduce the major characteristics of CISC and RISC architectures and then present the instruction set and instruction format of a RISC processor.

5.1 CISC CHARACTERISTICS

- The design of an instruction set for a computer must take into consideration not only machine language constructs, but also the requirements imposed on the use of high-level programming languages.

- The translation from high-level to machine language programs is done by means of a compiler program. One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance.

- The task of a compiler is to generate a sequence of machine instructions for each high-level language statement. The task is simplified if there are machine instructions that implement the statements directly.

- The essential goal of a CISC architectures to attempt to provide a single machine instruction for each statement that is written in a high-level language. Examples of CISC architectures are the Digital Equipment Corporation VAX computer and the IBM 370 computer.

- Another characteristic of CISC architecture is the incorporation of variable-length instruction formats. Instructions that require register operands may be only two bytes in length, but instructions that need two memory addresses may need five bytes to include the entire instruction code. If the computer has 32-bit words (four bytes), the first instruction occupies half a word, while the second instruction needs one word in addition to one byte in the next word.

- Packing variable instruction formats in a fixed-length memory word requires special decoding circuits that count bytes within words and frame the instructions according their byte length. The instructions in a typical CISC processor provide direct manipulation of operands residing in memory. For example, an ADD instruction may specify one operand in memory through index addressing and a second operand in memory through a direct addressing. Another memory location may be included in the instruction to store the sum. This requires three memory references during execution of the instruction.

- Although CISC processors have instructions that use only processor registers, the availability of other modes of operations tend to simplify high-level language compilation. However, as more instructions and addressing modes are incorporated into a computer, the more hardware logic is needed to implement and support them, and this may cause the computations to slow down. In summary

The major characteristics of CISC architecture are:

1) A large number of instructions-typically from 100 to 250 instructions
2) Some instructions that perform specialized tasks and are used in frequently
3) A large variety of addressing modes-typically from 5 to 20 different modes
4) Variable-length instruction formats.
5) Instructions that manipulate operands in memory.

Reduced Instruction Set Computer (RISC)

- An important aspect of computer architecture is the design of the instruction set for the processor. The instruction set chosen for a particular computer determines the way that machine language programs are constructed. Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware used to implement them. As digital hardware became cheaper with the advent of integrated circuits, computer instructions tended to increase both in number and complexity.

- Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions. These computers also employ a variety of data types and a large

number of addressing modes. The trend into computer hardware complexity was influenced by various factors, such as upgrading existing models to provide more customer applications, adding instructions that facilitate the translation from high-level language into machine language programs, and striving to develop machines that move functions from software implementation into hardware implementation. A computer with a large number of instruction is classified as a complex instruction set computer, abbreviated CISC.

5.2 RISC CHARACTERISTICS

The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

**The major characteristics of a RISC processor are:**
1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction format
6. Single-cycle instruction execution
7. Hardwired rather than micro programmed control

The small set of instructions of a typical RISC processor consists mostly of register-to- register operations, with only simple load and store operations for memory access. Thus each operand is brought into a processor register with a load instruction. All computations are done among the data stored in processor registers. Results are transferred to memory by means of store instructions. This architectural feature simplifies the instruction set and encourages the optimization of register manipulation. The use of only a few addressing modes results from the fact that almost all instructions have simple register addressing. Other addressing modes may be included, such as immediate operands and relative mode.

5.3 PIPELINE AND VECTOR PROCESSING
5.3.1 PARALLEL PROCESSING:

➢ Parallel processing is a term used to denote 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.

➢ The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.

➢ The amount of hardware increases with parallel processing, and with it, the cost of the system increases.

➢ Parallel processing can be viewed from various levels of complexity.

➢ At the lowest level, we distinguish between parallel and serial operations by the type of registers used. e.g. shift registers and registers with parallel load

➢ At a higher level, it can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

➢ Fig. below shows one possible way of separating the execution unit into eight functional units operating in parallel.

➢ A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.
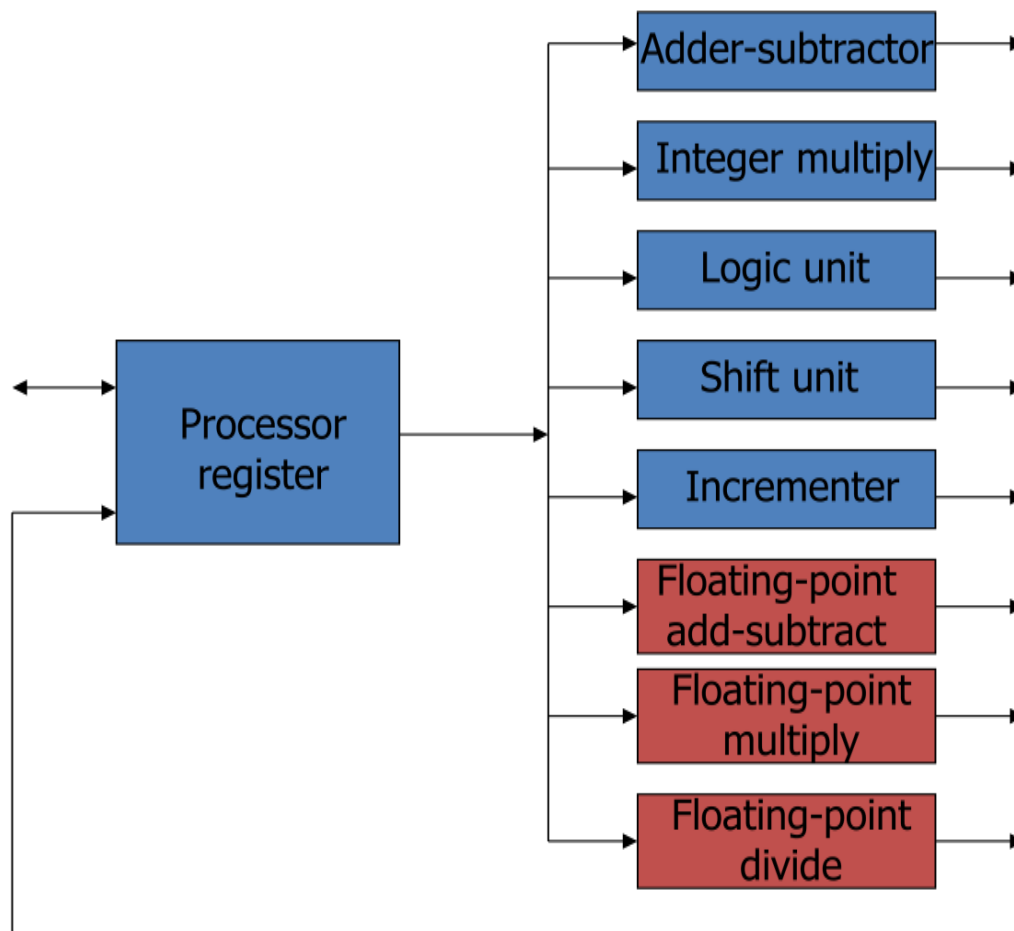
Figure 5.1 - Parallel Processing

The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands. The operation performed in each functional unit is indicated in each block of the diagram. The adder and integer multiplier perform the arithmetic operations with integer numbers. The floating-point operations are separated into three circuits operating in parallel. The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented. A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.

There are a variety of ways that parallel processing can be classified. it can be considered from the

- • Internal organization of the processors
- • Interconnection structure between processors
- • The flow of information through the system

One classification introduced by M. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

The sequence of instructions read from memory constitutes an *instruction stream.* The operations performed on the data in the processor constitute a *data stream*. Parallel processing may occur in the instruction stream, in the data stream, or in both. Flynn's classification divides computers into four major groups as follows:

- ➢ **Single instruction stream, single data stream (SISD)**
- ➢ **Single instruction stream, multiple data stream (SIMD)**
- ➢ **Multiple instruction stream, single data stream (MISD)**
- ➢ **Multiple instruction stream, multiple data stream (MIMD)**
- ➢ **Single instruction stream, single data stream (SISD)**
    - • Represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
    - • Instructions are executed *sequentially* and the system may or may not have internal parallel processing capabilities.
    - • parallel processing may be achieved by means of *multiple functional units* or by
    - • *pipeline processing*.

- ➢ **Single instruction stream, multiple data stream (SIMD)**
    - • Represents an organization that includes many processing units under the supervision of a common control unit.
    - • All processors receive the same instruction from the control unit but operate on different items of data.
    - • The shared memory unit must contain *multiple modules* so that it can communicate with all the processors simultaneously.

➢ **Multiple instruction stream, single data stream (MISD)**

- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

➢ **Multiple instruction stream, multiple data stream (MIMD)**

- MIMD organization refers to a computer system capable of processing several programs at the same time.

e.*g. Multiprocessor and multicomputer system*

We consider parallel processing under the following main topics:

- **Pipeline processing:** Is an implementation technique where arithmetic sub operations or the phases of a computer instruction cycle overlap in execution.
- **Vector processing: Deals** with computations involving large vectors and matrices.
- **Array processing:** Perform computations on large arrays of data.

5.3.2 <u>PIPELINING</u>

- Pipelining is 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 pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segmenting the pipeline.
- The final result is obtained after the data have passed through all segments. The name "pipeline" implies a flow of information analogous to an industrial assembly line. It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline isolation between each segment so that each can operate on distinct data simultaneously
- **The pipeline organization will be demonstrated by means of a simple**

- Example. Suppose that we want to perform the combined multiply and add operations with a stream of numbers

$$A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7$$

- Each sub operation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as **shown in Fig.** R 1 through RS are registers that receive new data with every clock pulse.

- The multiplier and adder are combinational circuits. The sub operations performed in each segment of the pipeline are as follows:

$$R1 \leftarrow A_i, \quad R2 \leftarrow B_i \qquad \text{Input } A_i \text{ and } B_i$$
$$R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i \qquad \text{Multiply and input } C_i$$
$$R5 \leftarrow R3 + R4 \qquad \text{Add } C_i \text{ to product}$$

- Input A, and B, Multiply and input C, Add C; to product The five registers are loaded with new data every clock pulse. The effect of each clock is shown in **Table 5.1**. The first clock pulse transfers A1 and B1 into R 1 and R2. The second dock pulse transfers the product of R 1 and R2 into R3 and C1 into R4. The same clock pulse transfers A2 and B2 into R 1 and R2. The third clock pulse operates on all three segments simultaneously. It places A, and B, into R1 and R2, transfers the product of R1 and R2 into R3, transfers C, into R4, and places the sum of R3 and R4 into RS. It takes three clock pulses to fill up the pipe and retrieve the first output from RS. From there on, each dock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.

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	—	—	—	—	$A_7 * B_7 + C_7$

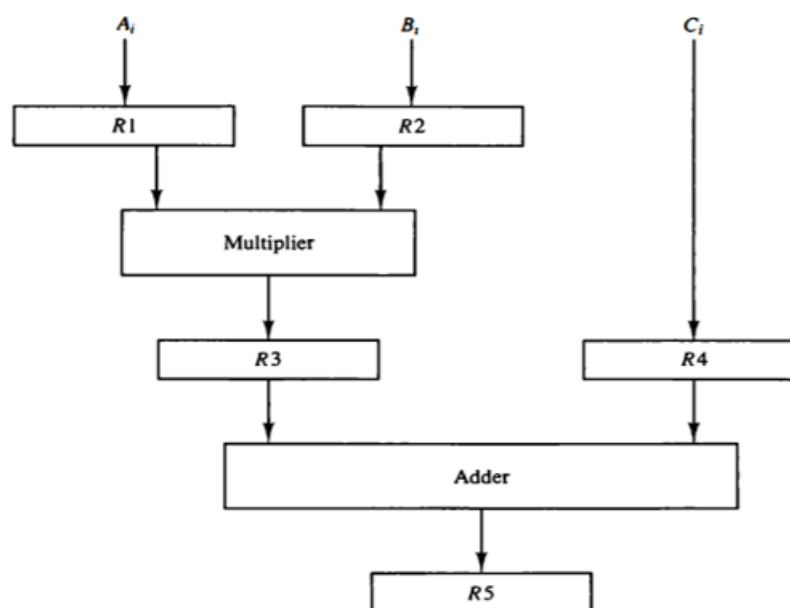**Table 5.1 – Content of Registers in Pipeline Example**

Figure 5.2 - Example of Pipeline Processing

5.3.3 FOUR-SEGMENT PIPELINE

The general structure of a four-segment pipeline is illustrated in Fig. 9-3. The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S; that performs a sub operation over the data stream flowing through the pipe. The segments are separated by registers R; that hold the intermediate results between the stages. Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously. We task define a task as the total operation performed going through all the segments in the pipeline The behaviour of a pipeline can be illustrated with a

space-time diagram. This is a diagram that shows the segment utilization as a function of time. The space-time diagram of a four-segment pipeline is demonstrated in **Figure 5.3**. The horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.
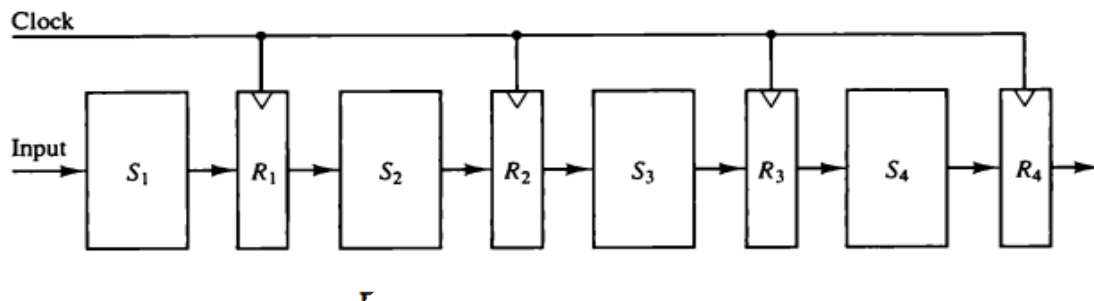
Figure 5.3 - Four-Segment Pipeline

The diagram shows six tasks T1 through T6 executed in four segments. Initially, task 1i is handled by segment 1. After the first clock, segment 2 is busy with T_1 while segment 1 is busy with task T2• Continuing in this manner, the first task T1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock cycle. No matter how many segments there are in the system, once the pipeline is full, it takes only one clock period to obtain an output.

Now consider the case where a k-segment pipeline with a clock cycle time t, is used to execute n tasks. The first task T1 requires a time equal to k_{tp}, to complete its operation since there are k segments in the pipe. The remaining n - 1 tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n - 1)t,. Therefore, to complete n tasks using a k-segment pipeline requires k + (n - 1) clock cycles. For example, the diagram of **Figure 5.4** shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.

Next consider a non-pipeline unit that performs the same operation and takes a time equal to t. to complete each task. The total time required for n tasks is n_{tn}. The speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio
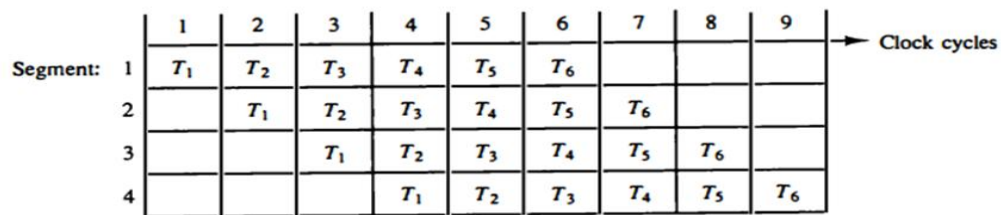
$$S = \frac{nt_n}{(k + n - 1)t_p}$$

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Figure 5.4 – Space Time Diagram of Pipeline

As the number of tasks increases, n becomes much larger than k - 1, and k + n - 1 approaches the value of n. Under this condition, the speedup becomes

$$S = \frac{t_n}{t_p}$$

If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits, we will have t, = kt,. Including this assumption, the speedup reduces to

$$S = \frac{kt_p}{t_p} = k$$

This shows that the theoretical maximum speedup that a pipeline can provide is k, where k is the number of segments in the pipeline.
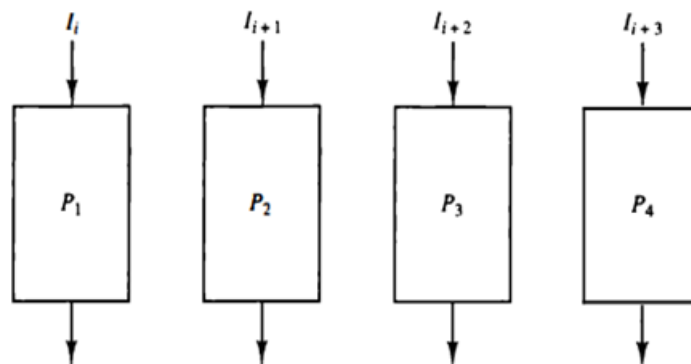

Figure 5.5 - Multiple Functional Units in Parallel

5.3.4 ARITHMETIC PIPELINE:

Pipeline arithmetic units are usually found in very high-speed computers.

They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.

There are various reasons why the pipeline cannot operate at its maximumtheoretical 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 two areas of computer design where the pipeline organization is applicable.
- Arithmetic pipeline
- Instruction pipeline

Arithmetic Pipeline: Introduction

- 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 sub operations as demonstrated in Sec. 10-5.
- An example of a pipeline unit for floating-point addition and subtraction is showed in the following:

o The inputs to the floating-point adder pipeline are two normalized floating- point binary number

$X \square A \square 2^a$

$Y \square B \square 2^b$

- 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 **Figure 5.6**.

- The sub operations that are performed in the four segments are:
  - *Compare the exponents*
  - The larger exponent is chosen as the exponent of the result.
  - *Align the mantissas*

- The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right.
  - *Add or subtract the mantissas*
  - *Normalize the result*

- 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 number that must be subtracted from the exponent.
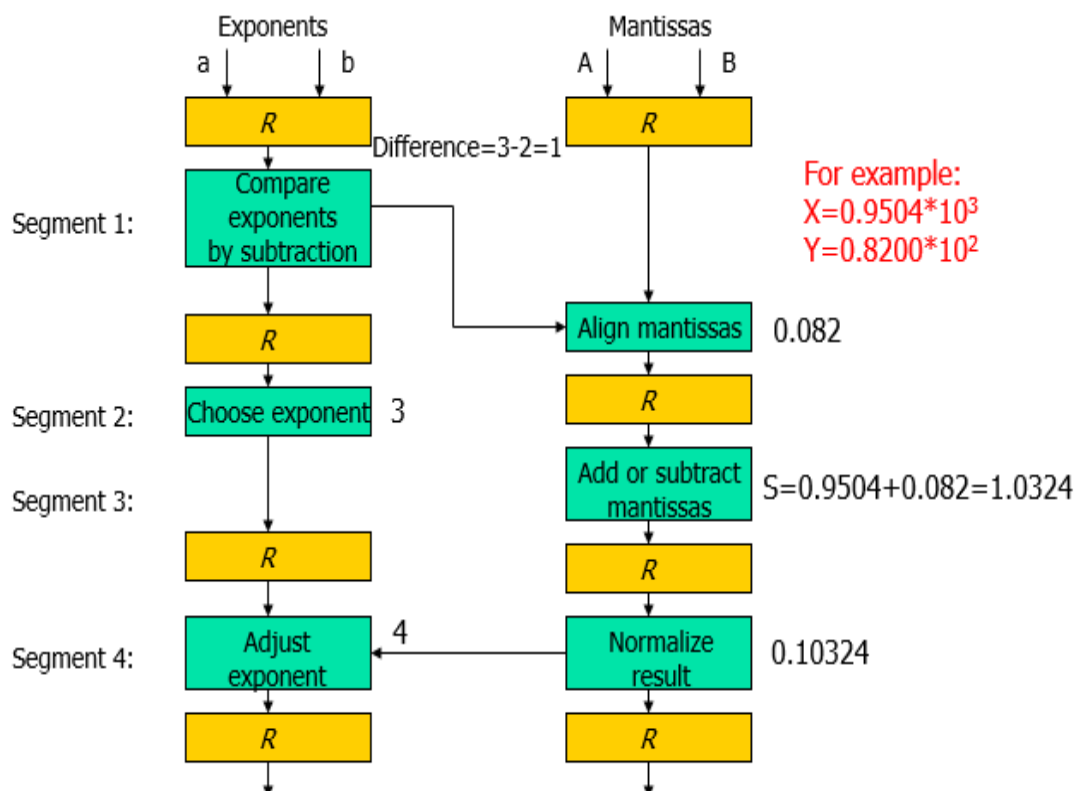

Figure 5.6 - Arithmetic Pipeline

5.3.5 <u>INSTRUCTION PIPELINE</u>

- Pipeline processing can occur not only in the *data stream* but in the *instruction* as well.

- Consider a computer with an instruction fetch unit (FIFO)and an instruction execution unit (PC) 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.

- Fetch the instruction from memory.

- Decode the instruction.

- Calculate the effective address.
- Fetch the operands from memory.
- Execute the instruction.
- Store the result in the proper place.
- There are certain difficulties that will prevent the instruction pipeline from operatingat 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.
- The instruction execution and storing of the result can be combined into one segment. **Figure** shows how the instruction cycle in the CPU can be processed with a four- segment pipeline.
- Thus, up to four sub operations in the instruction cycle can overlap and up to four different

instructions can be in progress of being processed at the same time.
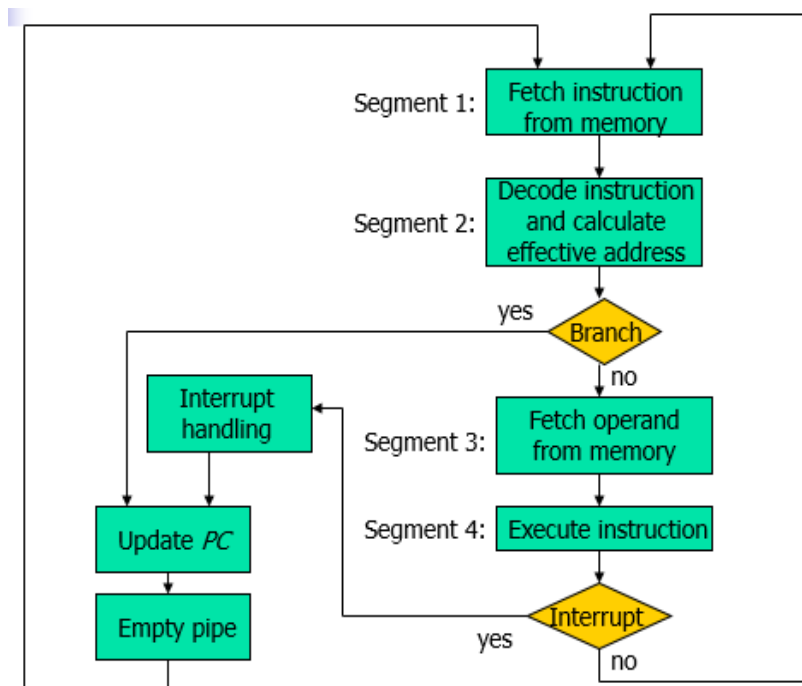
Figure 5.7 - Four-Segment Instruction Pipeline

- 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. Figure shows the operation of the instruction pipeline
- The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.
  - F1 is the segment that fetches an instruction.
  - DA is the segment that decodes the instruction and calculates the effective address.
  - Fo is the segment that fetches the operand.
  - EX is the segment that executes the instruction.

Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	—	—	FI	DA	FO	EX			
5					—	—	—	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

Instruction: (Branch)

FI: the segment that fetches an instruction
DA: the segment that decodes the instruction
    and calculate the effective address
FO: the segment that fetches the operand
EX: the segment that executes the instruction

It is assumed that the processor has separate instruction and data memories so that the operation in F1 and PC can proceed at the same time. In the absence of a branch instruction, each segment operates on different instructions. Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

Assume now that instruction 3 is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from F1 to DA of the other instructions is halted until the branch instruction is executed in step6. If the branch is taken, a new instruction is fetched in step 7. If the branches not taken, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.

Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand. In that case, segment FO must wait until segment EX has finished its operation.

- In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.
- *Resource conflicts* caused by access to memory by two segments at the same time.
- Can be resolved by using separate instruction and data memories

- *Data dependency conflicts* arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- *Branch difficulties* arise from branch and other instructions that change the value of PC.

Data dependency: A difficulty that may cause a degradation of performance in an instruction pipeline is due to possible collision of data or address. A data dependency occurs when an instruction needs data that are not yet available.

- An address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available. Pipelined computers deal with such conflicts between data dependencies in a variety of ways.

- *Hardware interlocks***:** an interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. This approach maintains the program sequence by using hardware to insert the required delays.

- Operand forwarding: uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

- *Delayed load***:** the *compiler* for such computers is designed to detect a dataconflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions.

Handling of branch instructions

- One of the major problems in operating an instruction pipeline is the occurrenceof branch instructions.
    - o An unconditional branch always alters the sequential program flow byloading the program counter with the target address.
    - o In a conditional branch, the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition isnot satisfied.

- Pipelined computers employ various hardware techniques to minimize the performance degradation caused by instruction branching.

- **Prefetch target instruction:** To prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed.

- **Branch target buffer (BTB):** The BTB is an associative memory included in the fetch segment of the pipeline.

  - Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch.
  - It also stores the next few instructions after the branch target instruction.

- **Loop buffer:** This is a small very high-speed register file maintained by the instruction fetch segment of the pipeline.

- **Branch prediction:** A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed.

- **Delayed branch:** in this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by *inserting useful instructions* that keep the pipeline operating without interruptions.

  - A procedure employed in most RISC processors.
  - e.g. no-operation instruction

5.3.4 RISC PIPELINE

Among the characteristics attributed to RISC is its ability to use an efficient instruction pipeline. The simplicity of the instruction set can be utilized to implement an instruction pipeline using a small number of sub operations, with each being executed in one clock cycle. Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection. All data manipulation instructions have register-to-register operations. Since all operands are in registers, there is no need for calculating an effective address or fetching of operands from memory. Therefore, the instruction pipeline can be implemented with two or three segments. One segment fetches the instruction from program memory, and the other segment executes the instruction in the ALU. A third segment may be used to store the result of the ALU operation in a destination register.

- The data transfer instructions in RISC are limited to load and store instructions.

- ➢ These instructions use register indirect addressing. They usually need three or four stages in the pipeline.
- ➢ 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.
- ➢ Cache memory: operate at the same speed as the CPU clock

- • One of the major advantages of RISC is its ability to execute instructions at the rate of one per clock cycle.
- ➢ In effect, it is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction execution.
- ➢ RISC can achieve pipeline segments, requiring just one clock cycle.

- • *Compiler* supported that translates the high-level language program into machine language program.
- ➢ Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties.
- ➢ RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Example: Three-Segment Instruction Pipeline

- • A typical set of instructions for a RISC processor are discussed earlier.
- • These are three types of instructions:
  - ➢ The data manipulation instructions: operate on data in processor registers
  - ➢ The data transfer instructions:
  - ➢ The program control instructions:

Now consider the hardware operation for such a computer.

- • The *control section* fetches the instruction from program memory into an instruction register.
- • The instruction is decoded at the same time that the registers needed for the execution of the instruction are selected.
- • The processor unit consists of a number of registers and an arithmetic logic unit (ALU).
- • A data memory is used to load or store the data from a selected register in the register file.

- The instruction cycle can be divided into three sub operations 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.

5.4 VECTOR PROCESSING

- 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. e.g.

- *Long-range weather forecasting*

- *Petroleum explorations*

- *Seismic data analysis*

- *Medical diagnosis*

- *Artificial intelligence and expert systems*

- *Image processing*

- *Mapping the human genome*

- To achieve the required level of high performance it is necessary to utilize the *fastest and*

most *reliable hardware* and apply innovative procedures from *vector and parallel processing techniques*.

- **Vector Operations**
- Many scientific problems require arithmetic operations on large arrays of numbers.
- A vector is an ordered set of a one-dimensional array of data items.
- A vector V of length n is represented as a row vector by V=[v1,v2,…,Vn].
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

DO 20 I = 1, 100

20   C(I) = B(I) + A(I)

- This is implemented in machine language by the following sequence of operations.

```
      Initialize I = 0
 20   Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = I + 1
      If I ≤ 100 go to 20
      Continue
```

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.

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

- A possible instruction format for a vector instruction is shown in Fig. .
  o This assumes that the vector operands reside in *memory*.

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

- It is also possible to design the processor with a large number of *registers* and store all operands in registers prior to the addition operation.
  o The base address and length in the vector instruction specify a group of CPU registers.

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

$$c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$$

- 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.

$$\begin{aligned} C = \ & A_1 B_1 + A_5 B_5 + A_9 B_9 + A_{13} B_{13} + \cdots \\ & + A_2 B_2 + A_6 B_6 + A_{10} B_{10} + A_{14} B_{14} + \cdots \\ & + A_3 B_3 + A_7 B_7 + A_{11} B_{11} + A_{15} B_{15} + \cdots \\ & + A_4 B_4 + A_8 B_8 + A_{12} B_{12} + A_{16} B_{16} + \cdots \end{aligned}$$

- **The inner product calculation on a pipeline vector processor is shown in Figure.**
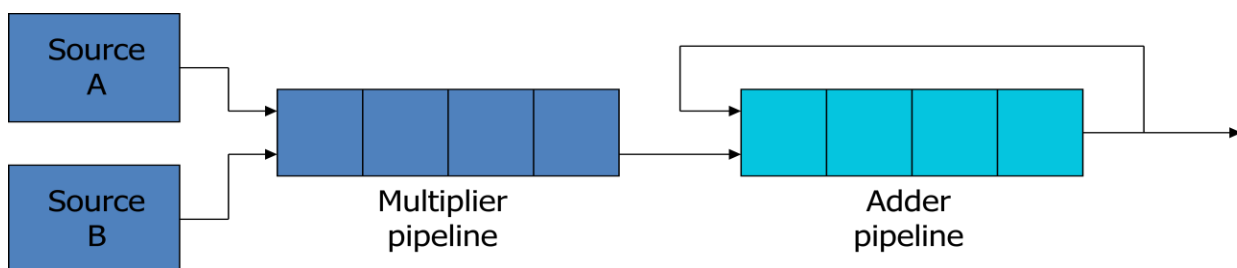
Memory Interleaving

- *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.

  - **Figure 5.8** shows a memory unit with four modules.
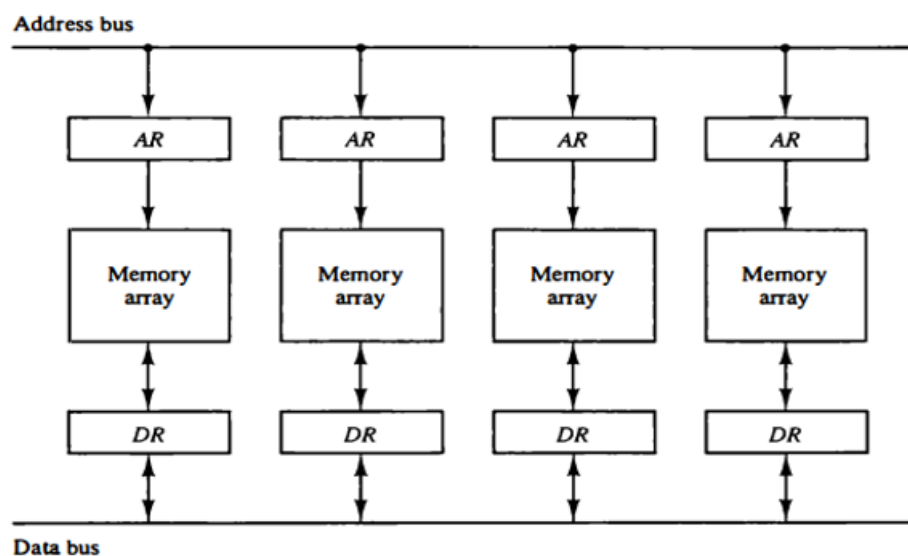
Figure 5.8 - Multiple Module Memory Organization

- 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*.

Supercomputers

- A commercial computer with vector instructions and pipelined floating-point arithmetic operations is referred to as a *supercomputer*.

- To speed up the operation, the components are *packed tightly* together to minimize the

distance that the electronic signals have to travel.

- This is augmented by instructions that process vectors and combinations of scalars and vectors.

- A supercomputer is a computer system best known for its high computational speed, fast and large memory systems, and the extensive use of parallel processing.

- It is equipped with *multiple functional units* and each unit has its own *pipeline* configuration.

- It is specifically optimized for the type of numerical calculations involving vectors and matrices of floating-point numbers.

- They are limited in their use to a number of scientific applications, such *as numerical weather forecasting, seismic wave analysis, and space research*.

- A measure used to evaluate computers in their ability to perform a given number of floating-point operations per second is referred to as *flops*.

- A typical supercomputer has a basic cycle time of 4 to 20 ns.

- The examples of supercomputer: Cray-1: it uses vector processing with 12 distinct functional units in parallel; a large number of registers (over 150); multiprocessor configuration (Cray X- MP and Cray Y-MP)

- Fujitsu VP-200: 83 vector instructions and 195 scalar instructions; 300 megaflops.

5.4 <u>ARRAY PROCESSORS:</u>

Introduction

- 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:
  - ➤ 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

- **Figure 5.9** shows the interconnection of an attached array processor to a 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.
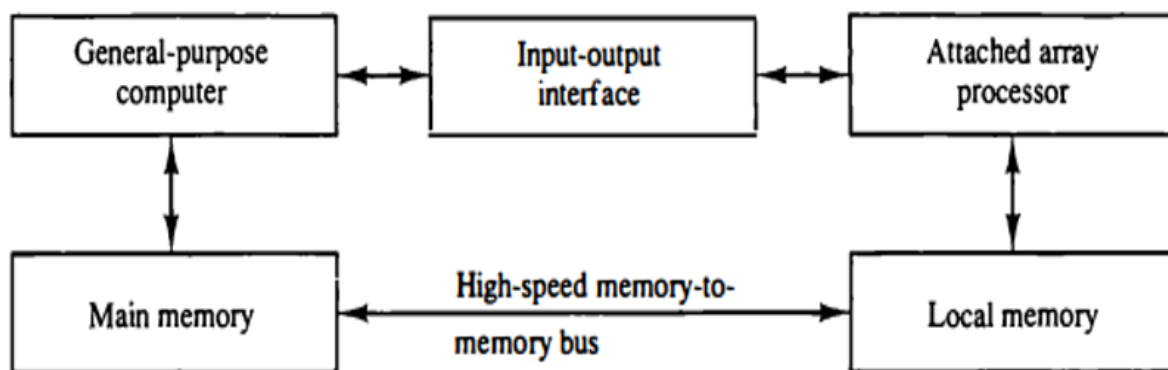
Figure 5.9 - Attached Array Processor with Host Computer

- 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

- A SIMD array processor is a computer with multiple processing units operating in parallel.

- A general block diagram of an array processor is shown in Figure 5.10.

- 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 Compare highly specialized computers. They are suited primarily for numerical problems that can be expressed invector or matrix form.
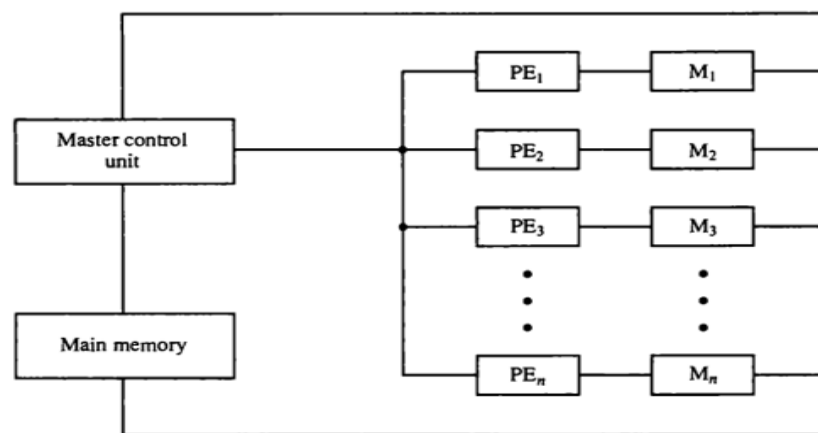

Figure 5.10 - SIMD Array Processor Organization

5.5 <u>MULTI PROCESSORS:</u>

5.5.1 <u>CHARACTERISTICS OF MULTIPROCESSOR:</u>

- 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 (MIMO) systems.

- Computers are interconnected with each other by means of communication lines to form a computer network. The network consists of several autonomous computers that may or may not communicate with each other.

- A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.

- Multiprocessing 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 functions of the disabled processor.

● Multiprocessing can improve performance by decomposing a program into parallel executable tasks. This can be achieved in one of two ways.

● The user can explicitly declare that certain tasks of the program be executed in parallel.

● The other is a compiler with multiprocessor software that can automatically detect parallelism in a user's program.

● Multiprocessors are classified by the way their memory is organized.

● A multiprocessor system with common shared memory is classified as a shared memory or tightly coupled multiprocessor. This does not preclude each processor from having its own local memory. In fact, most commercial tightly coupled multiprocessors provide a cache memory with each CPU.

● An alternative model of microprocessor is the distributed-memory or loosely coupled system. Each processor element in a loosely coupled system has its own private local memory. The processors are tied together by a switching scheme designed to route information from one processor to another through a message-passing scheme.

● Loosely coupled systems are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

5.6 <u>INTERCONNECTION STRUCTURES:</u>

● The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit that may be partitioned into a number of separate modules.

● The interconnection between the components can have different physical configurations, depending on the number of transfer paths and memory in a shared memory system.

There are several physical forms available for establishing an interconnection network. Some of these schemes are presented in this section:

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

5. Hypercube system

Time-Shared Common Bus: A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit. A time-shared common bus for five processors is shown in **Figure 5.11**. Only one processor can communicate with the memory or another processor at any given time.
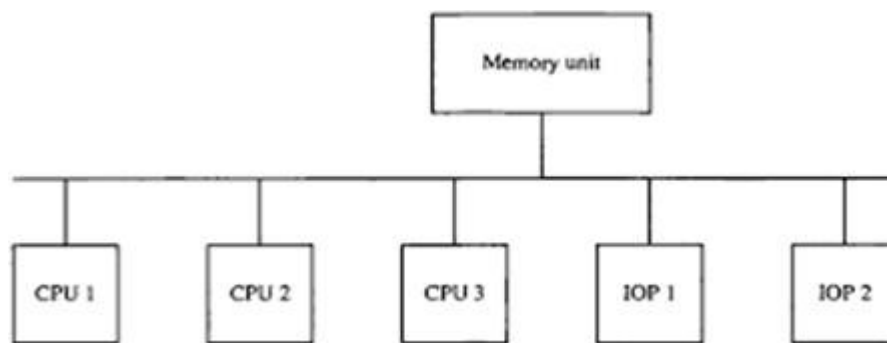
Figure 5.11: Time Shared Common Bus Organization

- Transfer operations are conducted by the processor that is in control of the bus at the time.
- If any other processor wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer.
- A command is issued to inform the destination unit what operation is to be performed.
- The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated.
- The system may exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.
- A single common-bus system is restricted to one transfer at a time. This means that when one processor is communicating with the memory, all other processors are either busy with internal operations or must be idle waiting for the bus. However, this increases the system cost and complexity.

A more economical implementation of a dual bus structure is depicted in Figure.
- In this method a number of local buses each connected to its own local memory and to one or more processors.

- Each local bus may be connected to a CPU, an lOP, or any combination of processors. A system bus controller links each local bus to a common system bus.

- The I/O devices connected to the local lOP, as well as the local memory, are available to the local processor.

- The memory connected to the common system bus is shared by all 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 VO devices. Part of the local memory may be designed as a cache memory attached to the CPU.

- In this way, inconsistent versions the average access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.
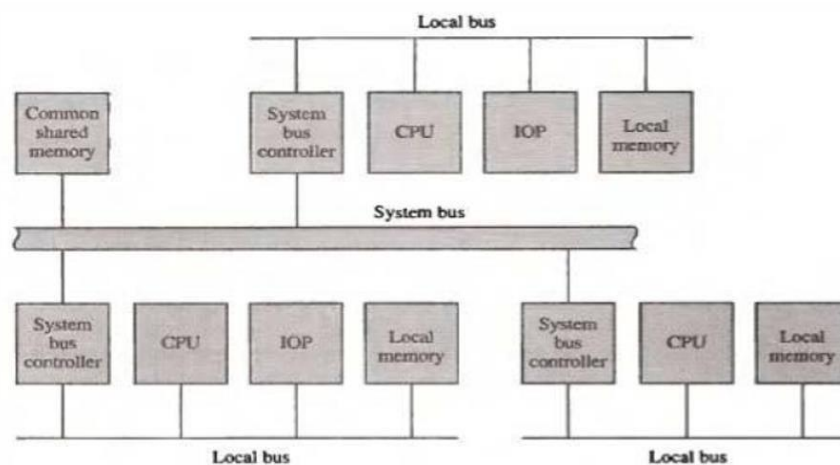

Figure 5.12: System Bus Structure for Multiprocessors

Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig .3 for four CPUs and four memory modules (MMs). 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. l

- The module must have internal control logic to determine which port will have access to memory at any given time. Memory access conflicts are resolved by assigning fixed priorities to each memory port.

- The priority for memory access associated with each processor may be established by the physical port position that its bus occupies in each module. Thus CPU 1 will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

- The advantage of the multiport memory organization is the high transfer rate that can be achieved because of the multiple paths between processors and memory.

- The disadvantage is that it requires expensive memory control logic and a large number of cables and connectors.
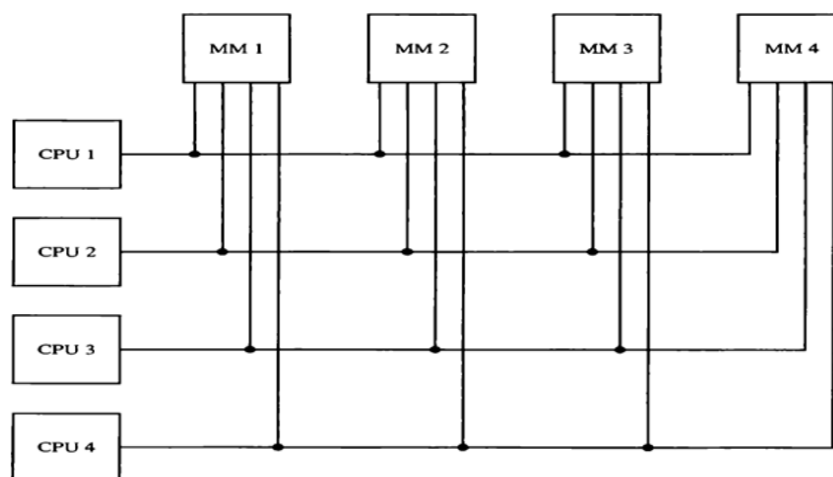
Figure 5.13 - Multiport Memory Organization

Crossbar Switch

- The crossbar switch organization consists of a number of cross points that are placed at intersections between processor buses and memory module paths.

- Figure 4 shows a crossbar switch interconnection between four CPUs and four memory modules.
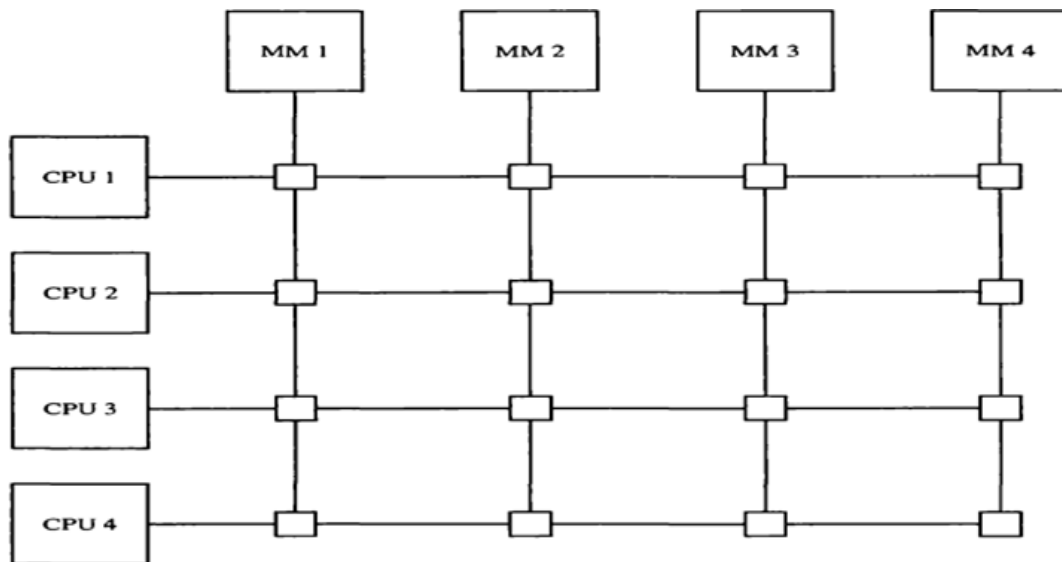
Figure 5.14 - Crossbar Switch Interconnection Between Four CPU

- The small square in each cross point is a switch that determines the path from a processor to a memory module.
- Each switch point has control logic to set up the transfer path between a processor and memory. It examines the address that is placed in the bus to determine whether its particular module is being addressed.
- It also resolves multiple requests for access to the same memory module on a predetermined priority basis.
- **Figure 5.15** shows the functional design of a crossbar switch connected to one memory module.
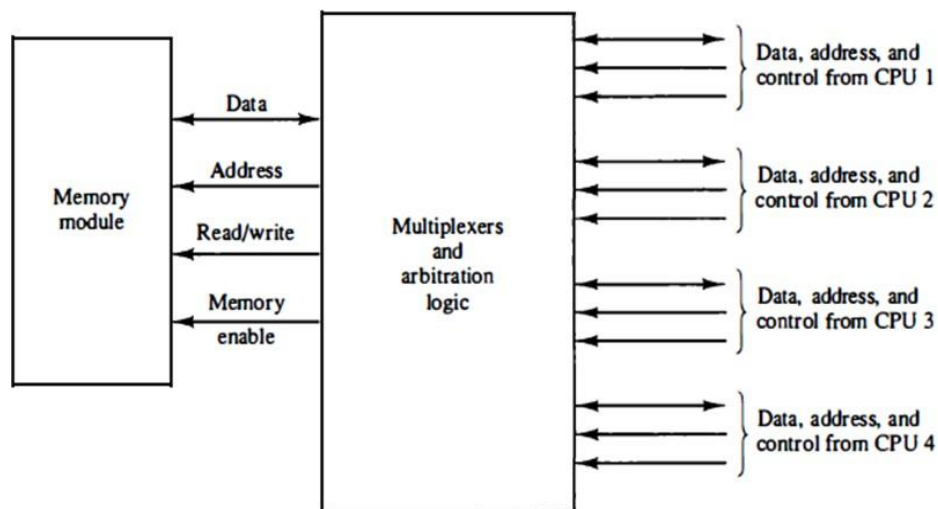

Figure 5.15: Functional Design of a Crossbar Switch

- The circuit consists of multiplexers that select the data, address, and control from one CPU

for communication with the memory module.

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

- The multiplexers are controlled with the binary code that is generated by a priority encoder within the arbitration logic.

- A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module.

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

Multistage Switching Network:

- The basic component of a multistage network is a two-input, two-output interchange switch.

- As shown in **Figure 5.16**. the 2 X 2 switch has two inputs labeled A and B, and two outputs, labeled 0 and 1.

- There are control signs (not shown) associated with the switch that establish the interconnection between the input and output terminals.

The switch has the capability connecting input A to either of the outputs. Terminal B of the switch behave in a similar fashion.
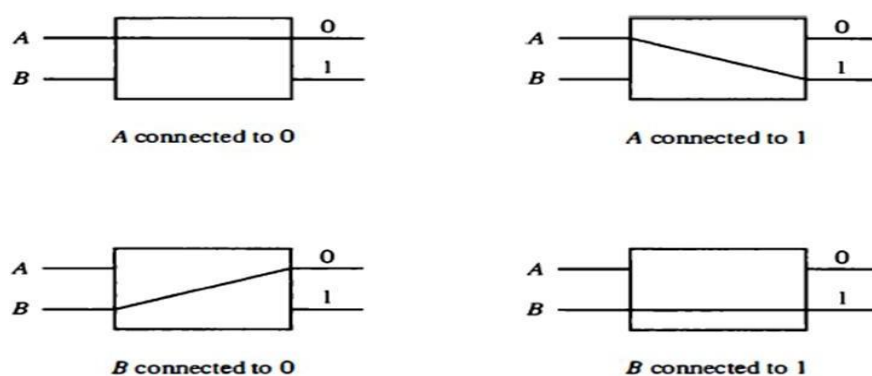

Figure 5.16: The 2 X 2 Switch has Two Inputs Labeled A and B

- The switch also has the capability to arbitrate between conflicting requests. If inputs A and B both request the same output terminal only one of them will be connected; the other will be blocked.

- Using the 2 x 2 switch as a building block, it is possible to build multistage network to control the communication between a number of souroce and destinations. To see how this is done, consider the binary tree shown in **Figure 5.17**.
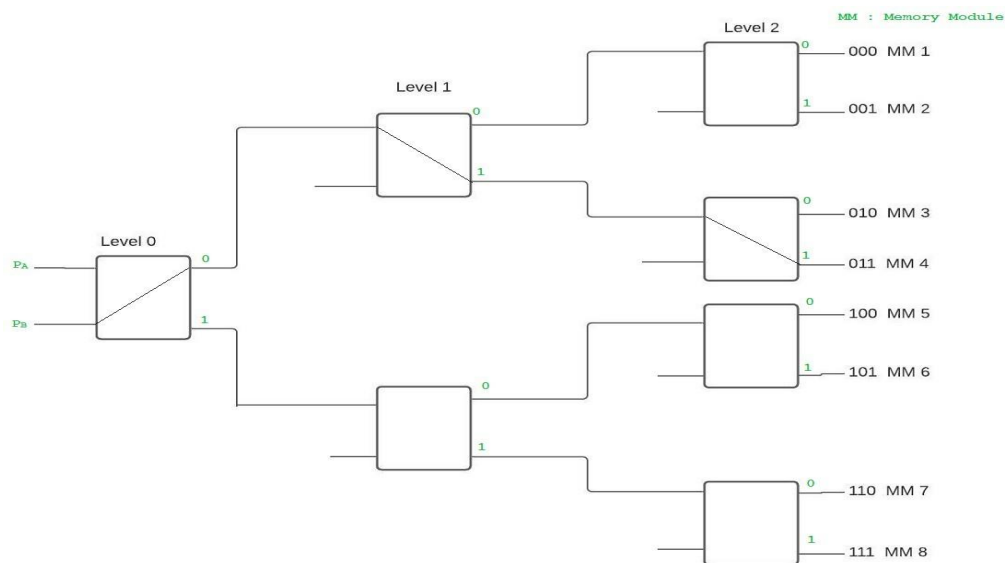
Fig 5.17: Binary Tree with 2 X 2 Switches

- The two processors P1 and P2 are connected through switches to eight memory modules marked in binary from 000 through 111. The path from source to a destination is determined from the binary bits of the destination number.
- The first bit of the destination number determines the switch output in the first level. The second bit specifies the output of the switch in the second level, and the third bit specifiesthe output of the switch in the third level.
- For example, to connect P1 to memory 101, it is necessary to form a path from P1 to output1 in the first-level switch, output 0 in the second-level switch, and output 1 in the third- level switch.

  Omega switching network: In this configuration, there is exactly one path from each source to any particular destination. For example, any two sources cannot be connected simultaneously to destinations 000 and 001.
- A particular request is initiated in the switching network by the 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 x 2 switch setting. Level 1 inspects the most significant bit, level 2 inspects the

middle bit, and level 3 inspects the least significant bit.

- When the request arrives on either Input of the 2 x 2 switch, it is routed to the upper output if the specified bit is 0 or to the lower output if the bit is 1.
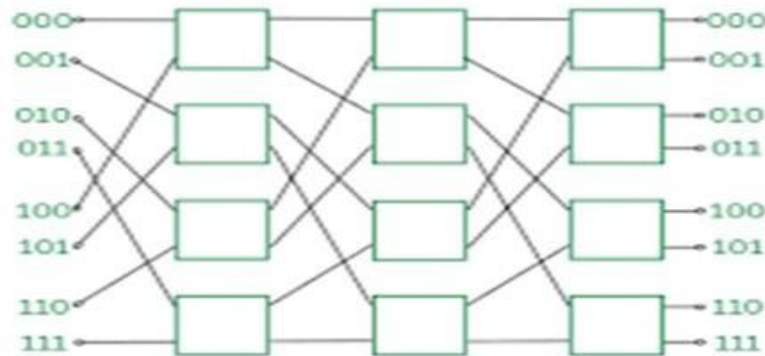

Figure 5.18 - 8 * 8 Omega Switching Network

- A particular request is initiated in the switching network by the 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 x 2 switch setting. Level 1 inspects the most significant bit, level 2 inspects the middle bit, and level 3 inspects the least significant bit.

- When the request arrives on either Input of the 2 x 2 switch, it is routed to the upper output if the specified bit is 0 or to the lower output if the bit is 1.

Hypercube Interconnection:

- Structure represents a loosely coupled system made up of $N=2^n$ processors interconnected in an n-dimensional binary cube.

- Each processor makes a node of the cube. Therefore, it is customary to refer to each node as containing a processor, in effect it contains not only a CPU but also local memory and I/O interface.

- Each processor has direct communication paths to n other neighbor processors. These paths correspond to the cube edges.

- There are 2 distinct n-bit binary addresses which can be assigned to the processors. Each processor address differs from that of each of its n neighbors by exactly one bit position.

- Hypercube structure for n= 1, 2 and 3.

- A one cube structure contains n = 1 and 2^n = 2. It has two processors interconnected by a single path.
- A two-cube structure contains n=2 and 2^n=4. It has four nodes interconnected as a cube.
- An n-cube structure contains 2n nodes with a processor residing in each node.
- Each node is assigned a binary address in such a way that the addresses of two neighbors differ in exactly one bit position.
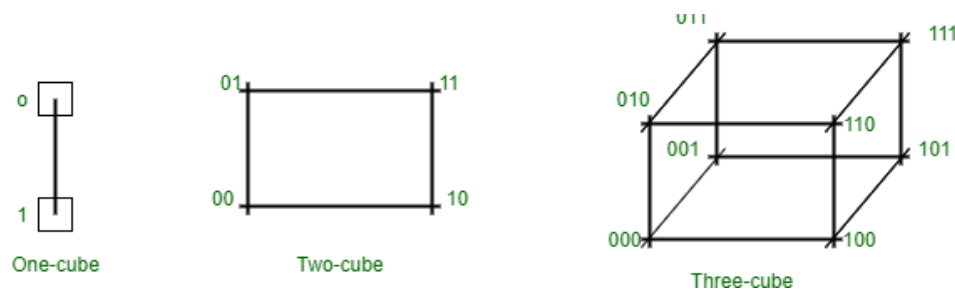

Figure 5.18 - Hypercube Structures for n=1,2,3

5.7 INTERPROCESSOR ARBITRATION:

- The CPU contains a number of internal buses for transferring information between processor registers and ALU.
- A memory bus consists of lines for transferring data, address, and read/write information.
- An I/O bus is used to transfer information to and from input and output devices. A bus that connects major components in a multisystem bus processor 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 or other common resources through the system bus.
- If no other processor is currently utilizing the bus, the requesting processor may be granted access immediately. However, the requesting processor must wait if another processor is currently utilizing the system bus. Furthermore, other processors may request the system bus at the same time.
- Arbitration must then be performed to resolve this multiple contention for the shared resources.
- The arbitration logic would be part of the 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. In addition, there are power distribution lines that supply power to the components.

- The data lines provide a path for the transfer of data between processors and common memory. The number of data lines is usually a multiple of 8, with 16 and 32 being most common.

- The address lines are used to identify a memory address or any other source or destination, such as input or output ports. The number of address lines determines the maximum possible memory capacity in the system.

- For example, an address of 24 lines can access up to 224 (16 mega) words of memory.

- Data transfers over the system bus may be synchronous or asynchronous.

- In a synchronous bus, each data item is transferred during a time slice known in advance to both source and destination units.

- In an asynchronous bus, each data item being transferred is accompanied by handshaking control signals to indicate when the data are transferred from the source and received by the destination.

- The control lines provide signals for controlling the information transfer between units. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed.

- Typical control lines include transfer signals such as memory read and write, acknowledge of a transfer, interrupt requests, bus control signals such as bus request and bus grant, and signals for arbitration procedures.

- Table 13-1 lists the 86 lines that are available in the IEEE standard 796 multibus. It includes16 data lines and 24 address lines.

IEEE Standard 796 Multibus Signals

	Signal name
Data and address	
Data lines (16 lines)	DATA0–DATA15
Address lines (24 lines)	ADRS0–ADRS23
Data transfer	
Memory read	MRDC
Memory write	MWTC
IO read	IORC
IO write	IOWC
Transfer acknowledge	TACK
Interrupt control	
Interrupt request (8 lines)	INT0–INT7
Interrupt acknowledge	INTA
Miscellaneous control	
Master clock	CCLK
System initialization	INIT
Byte high enable	BHEN
Memory inhibit (2 lines)	INH1–INH2
Bus lock	LOCK
Bus arbitration	
Bus request	BREQ
Common bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO
Power and ground (20 lines)	

Serial Arbitration Procedure:

Arbitration procedures service all processor requests on the basis of established priorities.

● A hardware bus priority resolving technique can be established by means of a serial or parallel connection of the units requesting control of the system bus.

● The serial priority resolving technique is obtained from a daisy-chain connection of bus arbitration circuits.

● The processors connected to the system bus are assigned priority according to their position along the priority control line.

● The device closest to the priority line is assigned the highest priority. When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it.
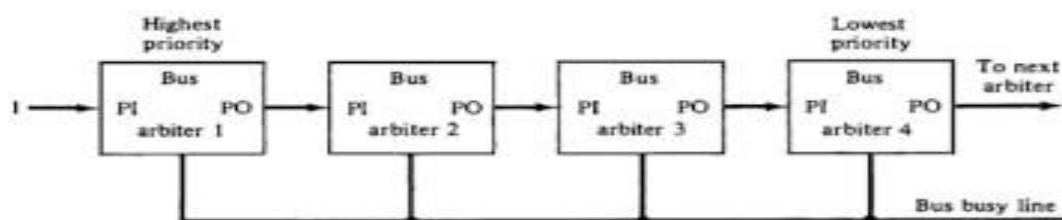

Figure 5.18 - The Daisy-Chain Connection of Four Arbiter

Figure shows the daisy-chain connection of four arbiters. It is assumed that each processor has its own bus arbiter logic with priority-in and priority-out lines.

- The priority out (PO) of each arbiter is connected to the priority in (PI) of the next-lower-priority arbiter. The PI of the highest priority unit is maintained at logic 1 value.

- The highest-priority unit in the system will always receive access to the system bus when it requests it. The Po output for a particular arbiter is equal to 1 if its PI input is equal to 1 and the processor associated with the arbiter logic is not requesting control of the bus.

- This is the way that priority is passed to the next unit in the chain. It the processor requests control of the bus and the corresponding arbiter finds its PI input equal to 1, it sets its PO output to 0.

- Lower priority arbiters receive a 0 in PI and generate a 0 in Po. Thus, the processor whose arbiter has a PI = 1 and Po = 0 is the one that is given control of the system bus.

- A processor may be in the middle of a bus operation when a higher-priority processor requests the bus. The lower-priority processor must complete its bus operation before it relinquishes control of the bus.

- The busy line comes from open-collector circuits in each unit and provides a wired-OR logic connection.

- When an arbiter receives control of the bus (because its PI = 1 and Po = 0) it examines the busy line. If the line is inactive, it means that no other processor is using the bus.

- The arbiter activates the busy line and its processor takes control of the bus. However, if the arbiter finds the busy line active, it means that another processor is currently using the bus.

- The arbiter keeps examining the busy line while the lower-priority processor that lost control of the bus completes its operation.

- When the bus busy line returns to its inactive state, the higher- priority arbiter enables the busy line, and its corresponding processor can then conduct the required bus transfers.

Parallel Arbitration Logic:

The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in **Figure 5.19.**
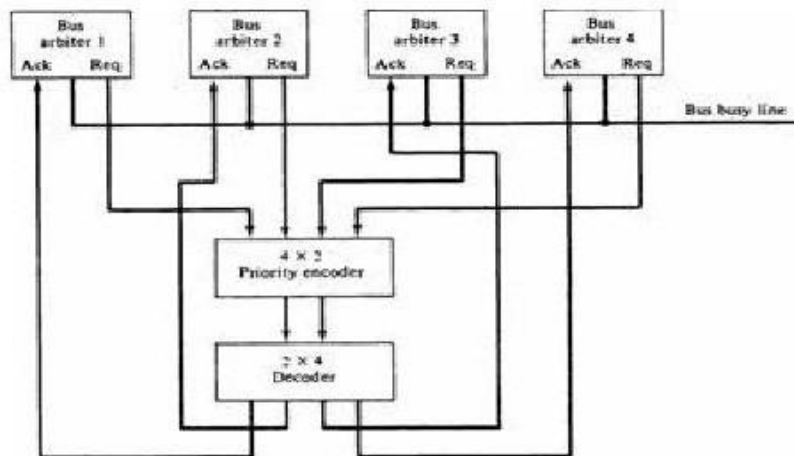
Figure 5.19 - Parallel Arbitration

- Each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.

- Each arbiter enables the request line when its processor is requesting access to the system bus. The processor takes control of the bus if it acknowledges input line is enabled.

- The bus busy line provides an orderly transfer of control, as in the daisy-chaining case.

- Figure 11 shows the request lines from four arbiters going into a 4 X 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 X 4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

Dynamic Arbitration Algorithms

- The two bus arbitration procedures just described use a static priority algorithm since the priority of each device is fixed by the way it is connected to the bus.

- In contrast, a dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation.

- **Time slice**: The time slice algorithm allocates a fixed-length time slice of bus time that is offered sequentially to each processor, in round-robin fashion.

- The service given to each system component with this scheme is independent of its location along the bus. No preference is given to any particular device since each is allotted the same amount of time to communicate with the bus.

- **Polling**: In a bus system that uses polling, the bus grant signal is replaced by a set of lines called poll lines which are connected to all units. These lines are used by the bus controller to define an address for each device connected to the bus. The bus controller sequences through the addresses in a prescribed manner. When a processor that requires access recognizes its address, it activates the bus busy line and then accesses the bus. After a number of bus cycles, the polling process continues by choosing a different processor. The polling sequence is normally programmable, and as a result, the selection priority can be altered under program control.

- **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. The priorities are adjusted after a number of bus cycles according to the LRU algorithm.

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

- **Rotating Daisy-Chain** The rotating daisy-chain procedure is a dynamic extension of the 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. Whichever device has access to the bus serves as a bus controller for the following arbitration. Each arbiter priority for a given bus cycle is determined 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 the lowest priority.

5.8 INTERPROCESSOR COMMUNICATION AND SYNCHRONIZATION:

- The various processors in a multiprocessor system must be provided with a facility for communicating with each other. A communication path can be established through common input output channels.

- In a shared memory multiprocessor system, the most common procedure is to setaside a

portion of memory that is accessible to all processors.

- The primary use of the 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 mailbox.

- The receiving processor can check the mailbox periodically to determine if there are valid messages for it.

- A more efficient procedure is for the sending processor to alert the receiving processor directly by means of an interrupt signal.

- This can be accomplished through a software-initiated interprocessor interrupt by means of an instruction in the program of one processor which when executed produces an external interrupt condition in a second processor.

- To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. This task is given to the operating system.

- There are three organizations that have been used in the design of operating system for multiprocessors: **master-slave configuration, separate operating system, and distributed operating system**. In a master-slave mode, one processor, designated the master, always executes the operating system functions. The remaining processors, denoted as slaves, do not perform 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.

- In the separate operating system organization, each processor can execute the operating system routines it needs.

- In the distributed operating system organization, the operating system routines are distributed among the available processors. However, each particular operating system function is assigned to only one processor at a time.

- In a loosely coupled multiprocessor system the memory is distributed among the processors and there is no shared memory for passing information.

- The communication between processors is by means of message passing through 1/0 channels.

- The communication is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate.

- When the sending processor and receiving processor name each other as a source and destination, a channel of communication is established.

- A message is then sent with a header and various data objects used to communicate between nodes.

- The operating system in each node contains routing information indicating the alternative paths that can be used to send a message to other nodes.

- The communication efficiency of the interprocessor network depends on the communication routing protocol, processor speed, data link speed, and the topology of the network.

Inter processor Synchronization:

- The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes. Communication refers to the exchange of data between different processes.

- Synchronization refers to the special case where the data used to communicate between processors is control information.

- Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data.

- Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources. Low-level primitives are implemented directly by the hardware.

- A number of hardware mechanisms for mutual exclusion have been developed. One of the most popular methods is through the use of a binary semaphore.

Mutual Exclusion with a Semaphore:

- 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 critical section it

is in a critical section.

- A critical section 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.

- A semaphore is a software- controlled flag that is stored in a memory location that all processors can access.

- When the semaphore is equal to1, it means that a processor is executing a critical program, so that the shared memory is not available to other processors.

- When the semaphore is equal to 0, the shared memory is available to any requesting processor. Processors that share the same memory segment agree by convention not to use the memory segment unless the semaphore is equal to 0, indicating that memory is available.

- They also agree to set the semaphore to 1 when they are executing a critical section and to clear it to 0 when they are finished.

- Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation.

- A semaphore can be initialized by means of a test and set instruction in hardware lock 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 test and-set instruction tests and sets a semaphore and activates the lock mechanism during the time that the instruction is being executed.

- This prevents other processors from changing the semaphore between the time that the processor is testing it and the time that it is setting it.

- Assume that the semaphore is a bit in the least significant position of a memory word whose address is symbolized by SEM.

- Let the mnemonic TSL designate the "test and set while locked" operation. The instruction

TSL SEM

will be executed in two memory cycles (the first to read and the second to write) without interference as follows:

<div align="center">

R←M [SEM] Test semaphore

M[SEM]←1 Set semaphore

</div>

- The semaphore is tested by transferring its value to a processor register R and then it is set to 1.

- The value in R determines what to do next. If the processor finds that R = 1, it knows that the semaphore was originally set. That means that another processor is executing a critical section, so the processor that checked the semaphore does not access the shared memory.

- If R = 0, it means that the common memory is available. The semaphore is set to 1 to prevent other processors from accessing memory.

- The processor can now execute the critical section. The last instruction in the program must clear location SEM to zero to release the shared resource to otherprocessors.

- The lock signal must be active during the execution of the test and set instruction.

5.9 CACHE COHERENCE:

- The primary advantage of cache is its ability to reduce the average access time in uniprocessors. When the processor finds a word in cache during a read operation, the main memory is not involved in the transfer.

- If the operation is to write, there are two commonly used procedures to update memory.

- In the **write-through policy**, both cache and main memory are updated with every write operation.

- In the **write-back policy**, only the cache is updated, and the location is marked so that it can be copied later into main memory.

Conditions for Incoherence:

- Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.

- Read-only data can safely be replicated without cache coherence enforcement mechanisms. Consider the three-processor configuration with private caches shown in **Figure 5.20**.

- Sometime during the operation an element X from main memory is loaded into the three processors, P1, P2, and P3. As a consequence, it is also copied into the private caches of the three processors.

- For example we assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory.
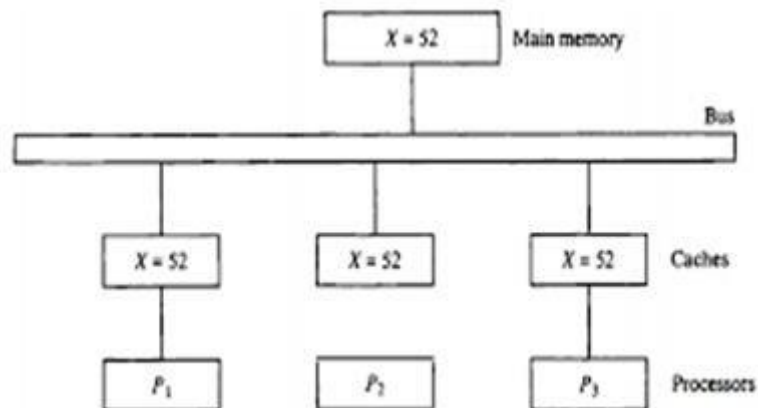

Figure 5.20: Cache Configuration after a Load on

- If one of the processors performs a store to X, the copies of X in the caches become inconsistent. A load by the other processors will not return the latest value.

- Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to the cache. This is shown in Fig.13. A store to X (of the value of 120) 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 since they still hold the old value.

- In a write-back policy, main memory is not updated at the time of the store. Thecopies in the other two caches and main memory are inconsistent.

- Another configuration that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an IOP connected to the system bus.

- In the case of input, the DMA may modify locations in main memory that also reside in cache without updating the cache.

- During a DMA output, memory locations may be read before they are updated from the cache when using a write-back policy.

- I/O-based memory incoherence can be overcome by making the IOP a participant in the cache coherent solution that is adopted in the system.
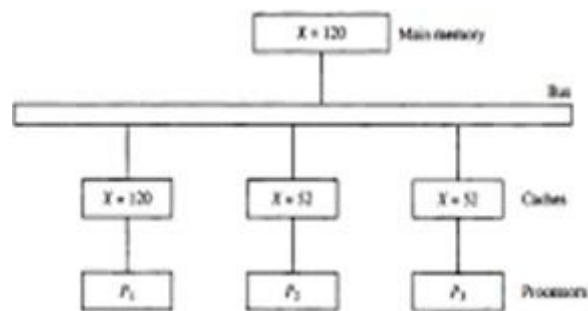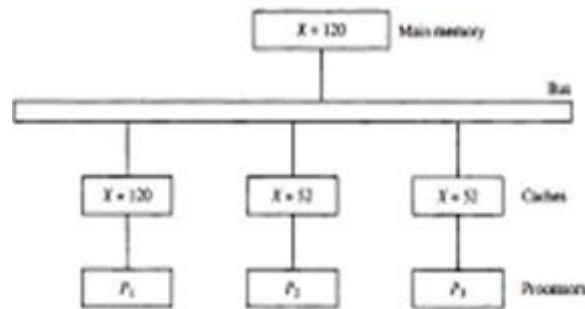
Figure 5.21(a) - With Write Through Cache Policy

Figure 5.21(b) - With Write Back Cache Policy

Solutions to the Cache Coherence Problem

- A simple scheme is to disallow private caches for each processor and have a shared cache memory associated with main memory. Every data access is made to the shared cache. In effect, this scheme solves the problem by avoiding it.

- For performance considerations it is desirable to attach a private cache to each processor. One scheme that has been used allows only nonshared and read-only data to be stored in caches. Such items are called cachable. Shared writable data are non cachable.

- The compiler must tag data as either cachable or noncachable, and the system hardware makes sure that only cachable data are stored in caches. The noncachable data remain in main memory.

- A scheme that allows writable data to exist in at least one cache is a method that employs a centralized global table in its compiler. The status of memory blocks is stored in the central global table.

- Each block is identified as read-only (RO) or read and write (RW). All caches can have copies of blocks identified as RO. Only one cache can have a copy of an RW block.

- Thus, if the data are updated in the cache with an RW block, the other caches are not affected because they do not have a copy of this block.

- The cache coherence problem can be solved by means of a combination of software and hardware or by means of hardware-only schemes.

- The two methods mentioned previously use software-based procedures that require the ability to tag information to disable caching of shared writable data.

- Hardware-only solutions are handled by the hardware automatically and have the advantage of higher speed and program transparency.

- In the hardware solution, the 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 the network for possible write operations.

- Depending on the method used, they must then either update or invalidate their own cache copies when a match is detected. The bus controller that monitors this action is referred to snoopy cache as a snoopy cache controller.

- This is basically a hardware unit designed to maintain a bus-watching mechanism over all the caches attached to the bus.

- All the snoopy controllers watch the bus for memory store operations.

- When a word in a cache is updated by writing into it, the corresponding location in main memory is also updated.

- The local snoopy controllers in all other caches check their memory to determine if they have a copy of the word that has been overwritten.

- If a copy exists in a remote cache, that location is marked invalid. Because all caches snoop on all bus writes, whenever a word is written, the net effect is to update it in the original cache and main memory and remove it from all other caches.

- In another variant of the snoopy cache coherence protocol. whenever a processor writes to a block in a write through scheme, the cache controllers at all processor s match the address you check if they have a copy of the block.

- If they have copy of the block, then they update the local cache.