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, Interprocessor arbitration, Interprocessor communication and synchronization, Cache Coherence.

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
- 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.

CISC Characteristics:

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

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

Parallel Processing

- 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.
- 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.
- Shift registers operate in serial fashion one bit at a time, while registers with parallel load operate with all the bits of the word simultaneously.
- Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.
- Parallel processing is established by distributing the data among the multiple functional units.
- **For example**, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

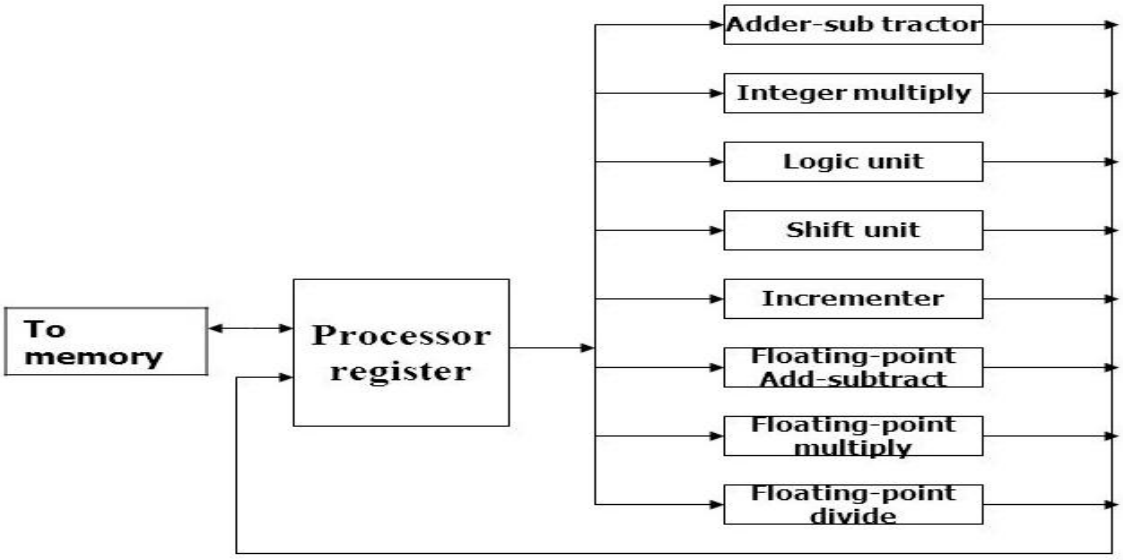


Figure 5-A: Processor with multiple functional units.

- Figure 5-A shows one possible way of separating the execution unit into eight functional units operating in parallel.
- 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.

- There are a variety of ways that parallel processing can be classified. It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.
- One classification introduced by **M. J. Flynn** considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

  - 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)

Parallel processing has the following main methods:

1. Pipeline processing
2. Vector processing
3. Array processors

Pipelining

- 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. The name "**pipeline"** implies a flow of information analogous to an industrial assembly line.
- The simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit. The register holds the data and the combinational circuit performs the sub operation in the particular segment. The output of the combinational circuit in a given segment is applied to the input register of the next segment. A clock is applied to all registers after enough time has elapsed to perform all segment activity.

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. 5-B.

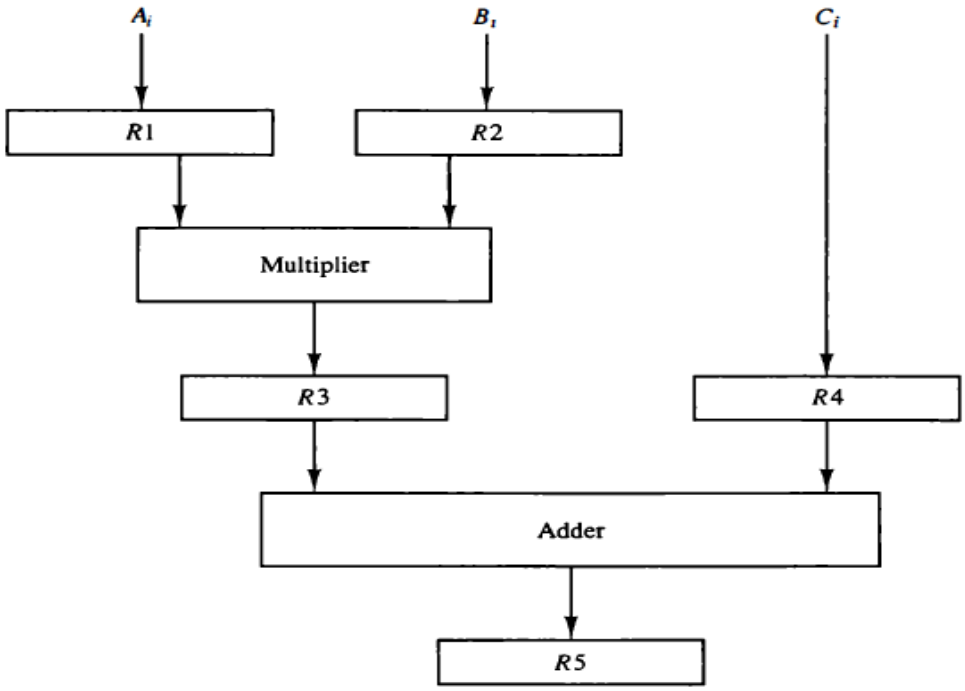


Figure 5-B: Example of pipeline processing.

- R1 through R5 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:

$R1 \leftarrow A_i$,	$R2 \leftarrow B_i$	Input Ai, and Bi,
$R3 \leftarrow R1 * R2$,	$R4 \leftarrow C_i$	Multiply and input C,
$R5 \leftarrow R3 + R4$		Add C_i to product

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table 5-C.

Table 5-C.: Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$

- The first clock pulse transfers A_1 and B_1 into R1 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 A_2 and B_2 into R1 and R2.
- The third clock pulse operates on all three segments simultaneously. It places A_3, and B_3 into R1 and R2, transfers the product of R1 and R2 into R3, transfers C_2, into R4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5.
- From there on, each dock produces a new output and moves the data one step down the pipeline.

Four-segment pipeline:

The general structure of a four-segment pipeline is illustrated in Fig. 5-D.

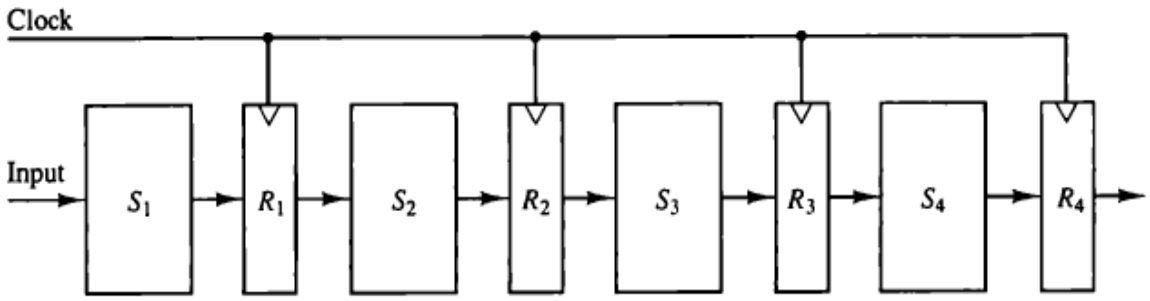


Figure 5-D: Four-segment pipeline

- The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers R_i 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 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

Segment		1	2	3	4	5	6	7	8	9
	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

→Clock pulses

Figure 5-E: Space- time diagram for pipeline.

- The horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.
- The diagram shows six tasks T_1 through T_6 executed in four segments. Initially, task T_1 is handled by segment 1.

- After the first clock, segment2 is busy with T_1, while segment 1 is busy with task T_2 .
- Continuing in this manner, the first task T_1 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

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.

Example: Floating-point addition and subtraction.

- The inputs to the floating-point adder pipeline are two normalized floating- point binary numbers.

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

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

- A and B are two fractions that represent the mantissas and a and b are the exponents.
- The floating-point addition and subtraction can be performed in four segments, as shown in Fig. 5-F.
- The registers labeled **R** are placed between the segments to store intermediate results. The sub operations that are performed in the four segments are:

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

- The exponents are compared by subtracting them to determine their difference.
- 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. This produces an alignment of the two mantissas.
- The two mantissas are added or subtracted in segment 3. The result is normalized in segment 4.
- 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.

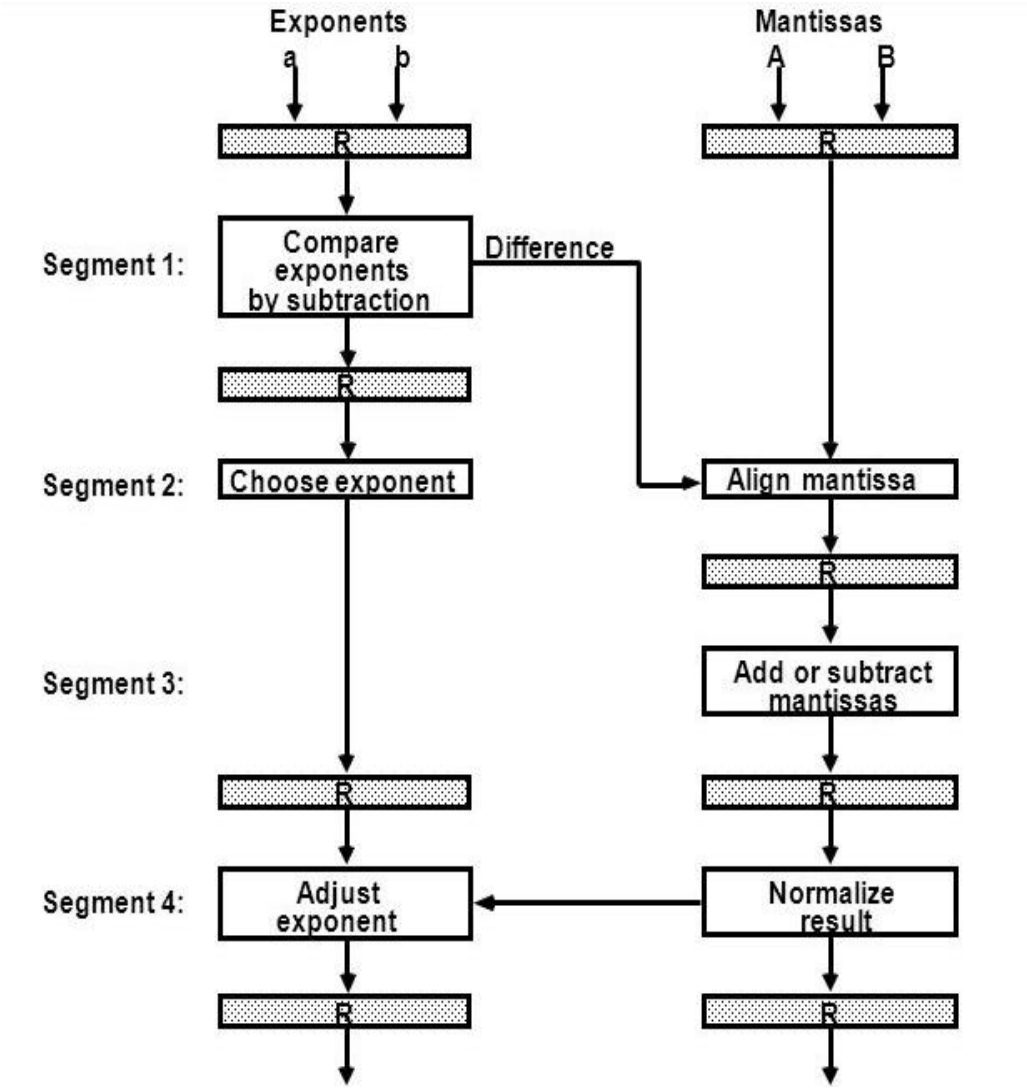


Figure 5-F: Pipeline Floating-point addition and subtraction

The following numerical example may clarify the sub operations performed in each segment. For simplicity, we use decimal numbers, although Fig. 5-F refers to binary numbers.

Consider the two normalized floating-point numbers:

$$X = 0.9504 \times 10^{2}$$

$$Y = 0.8200 \times 10^{2}$$

The two exponents are subtracted in the first segment to obtain 3 - 2 = 1. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^{3}$$

$$Y = 0.0820 \times 10^{3}$$

This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^{3}$$

The sum is adjusted by normalizing the result so that it has a fraction with a nonzero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^{4}$$

The comparator, shifter, adder-subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits.

Instruction Pipeline:

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- These causes the instruction fetches and executes phases to overlap and perform simultaneous operations. One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence.
- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.

- Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely. The following steps are

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.

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. Assume further that most of the instructions place the result into a processor registers so that the instruction execution and storing of the result can be combined into one segment. This reduces the instruction pipeline into four segments.

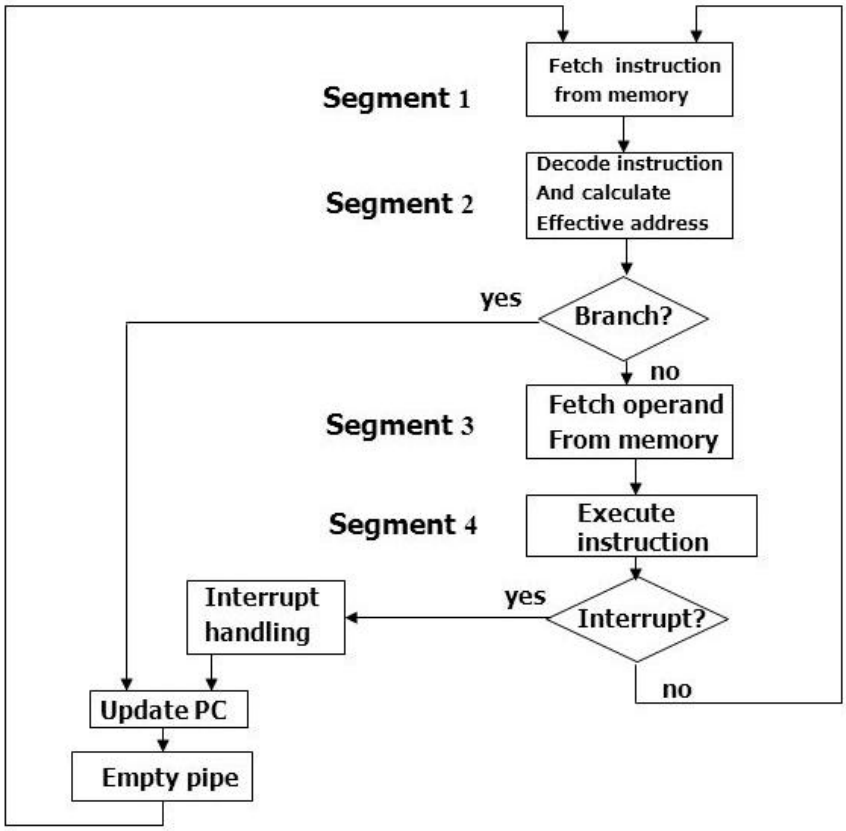


Figure 5-G: Four-segment CPU pipeline

- While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.
- The effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions being processed at the same time.
- Once in a while, an instruction in the sequence may be a program control type that 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. The pipeline then restarts from the new address stored in the program counter.
- Similarly, an interrupt request, when acknowledged, will cause the pipeline to empty and start again from a new address value.

Timing of instruction pipeline:

	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	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

Figure 5-H: Timing of instruction pipeline.

- Figure 5-H 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.

    1. Fl 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.

- It is assumed that the processor has separate instruction and data memories so that the operation in Fl and FO 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 FI to DA of the other instructions is halted until the branch instruction is executed in step 6.
- If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.

RISC (Reduced instruction set computer) 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.
- 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: one for storing the instructions and the other for storing the data.

Example: Three-Segment Instruction Pipeline:

- 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) that performs the necessary arithmetic, logic, and shift operations.
- 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

  A: ALU operation

  E: Execute instruction

- The **I** segment fetches the instruction from program memory.
- The instruction is decoded and an ALU operation is performed in the **A** segment.
- The ALU is used for three different functions, depending on the decoded instruction. It performs an operation for a data manipulation instruction, it evaluates the effective address for a load or store instruction, or it calculates the branch address for a program control instruction.
- The **E** segment 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, or it transfers the branch address to the program counter.

Delayed Load:

Example: Consider now the operation of the following four instructions:

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

- If the three-segment pipeline proceeds without interruptions, 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. 5-I (a).
- The E segment in clock cycle 4 is in a process of placing the memory data into R2. The A segment in clock cycle 4 is using the data from R2, but the value in R2 will not be the correct value since it has not yet been transferred from memory

Clock cycles:	1	2	3	4	5	6
1. Load *R*1	I	A	E			
2. Load *R*2		I	A	E		
3. Add *R*1 + *R*2			I	A	E	
4. Store *R*3				I	A	E

(a) Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load *R*1	I	A	E				
2. Load *R*2		I	A	E			
3. No-operation			I	A	E		
4. Add *R*1 + *R*2				I	A	E	
5. Store *R*3					I	A	E

(b) Pipeline timing with delayed load

Figure 5-I: Example of delayed Load.

- It is up to the compiler to make sure that the instruction following the load instruction uses the data fetched from memory.

- If the compiler cannot find a useful instruction to put after the load, it inserts a no-op (no-operation) instruction.
- This is a type of instruction that is fetched from memory but has no operation, thus wasting a clock cycle. This concept of delaying the use of the data loaded from memory is **referred to as delayed load.**
- Figure 5-I (b) shows the same program with a no-op instruction inserted after the load to R2 instruction. The data is loaded into R2 in clock cycle 4.
- The add instruction uses the value of R2 in step 5. Thus the no-op instruction is used to advance one clock cycle in order to compensate for the data conflict in the pipeline.

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 for a processor that uses delayed branches is designed to analyze the instructions before and after the branch and rearrange the program sequence by inserting useful instructions in the delay steps.
- For example**,** the compiler can determine that the program dependencies allow one or more instructions preceding the branch to be moved into the delay steps after the branch.
- These instructions are then fetched from memory and executed through the pipeline while the branch instruction is being executed in other segments.
- The effect is the same as if the instructions were executed in their original order, except that the branch delay is removed.
- 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.

Example: Consider five instructions:

1. Load from memory to R 1
2. Increment R 2
3. Add R3 t o R4
4. Subtract R5 from R6
5. Branch to address X

- In Fig. 5-J (a) the compiler inserts two no-op instructions after the branch. The branch address X is transferred to PC in clock cycle 7. The fetching of the instruction at X is delayed by two clock cycles by the no-op instructions.
- The instruction at X starts the fetch phase at clock cycle 8 after the program counter PC has been updated.

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

(a) Using no-operation instructions

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

(b) Rearranging the instructions

Figure 5-J: Example of delayed branch.

- The program in Fig. 5-J (b) is rearranged by placing the add and subtract instructions after the branch instruction instead of before as in the original program.
- Inspection of the pipeline timing shows that PC is updated to the value of X in clock cycle 5, but the add and subtract instructions are fetched from memory and executed in the proper sequence.

❖ The advantage of the delayed load approach is that the data dependency is taken care of by the compiler rather than the hardware. This results in a simpler hardware segment since the segment does not have to check if the content of the register being accessed is currently valid or not.

Vector Processing

- Computers with vector processing capabilities are in demand in specialized applications. The following are representative application areas where vector processing is of the utmost importance.
    - Long-range weather forecasting
    - Petroleum explorations
    - Seismic data analysis
    - Medical diagnosis
    - Artificial intelligence and expert systems
    - Mapping the human genome
    - Image processing
- Without sophisticated computers, many of the required computations cannot be completed within a reasonable amount of time.
- 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. These numbers are usually formulated as vectors and matrices of floating-point 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 $V = [V_1\ V_2\ V_3 \ldots V_n]$. It may be represented as a column vector if the data items are listed in a column.
- A conventional sequential computer is capable of processing operands one at a time.
- Consequently, operations on vectors must be broken down into single computations with subscripted variables.
- The element V_i of vector V is written as V(I) and the index **I** refers to a memory address or register where the number is stored.
- **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 a program for adding two vectors A and B of length 100 to produce a vector C . 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 goto 20
      Continue
```

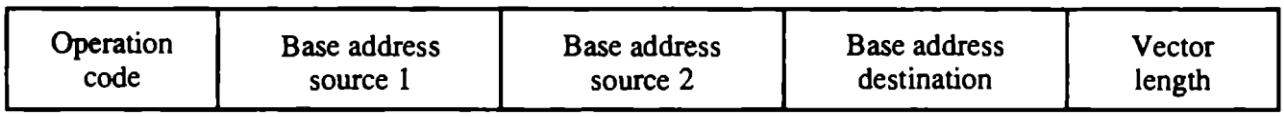
- This constitutes a program loop that reads a pair of operands from arrays A and B and performs a floating-point addition.
- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.

  It allows operations to be specified with a single vector instruction of the form.

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

- The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.
- A possible instruction format for a vector instruction is shown in **Fig. 5-K.**
- This is essentially a three-address instruction with three fields specifying the base address of the operands and an additional field that gives the length of the data items in the vectors. This assumes that the vector operands reside in memory.

Figure 5-K: Instruction format for vector processor



Matrix Multiplication:

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors. The multiplication of two n x n matrices consists of n^2 inner products.
- An n x m matrix of numbers has n rows and m columns and may be considered as constituting a set of **n** row vectors or a set of **m** column vectors.

Example: Consider 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 x 3 matrix whose elements are related to the elements of A and B by the inner product.
- The number in the first row and first column of matrix C is calculated by letting i = 1, j = 1, to obtain

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

- The inner product calculation on a pipeline vector processor is shown in **Fig. 5-K**. The values of A and B are either in memory or in processor registers.
- The floating-point multiplier pipeline and the floating-point adder pipeline are assumed to have four segments each.
- All segment registers in the multiplier and adder are initialized to 0. Therefore, the output of the adder is 0 for the first eight cycles until both pipes are full.
- A_i and B_i pairs are brought in and multiplied at a rate of one pair per cycle. After the first four cycles, the products begin to be added to the output of the adder.
- During the next four cycles 0 is added to the products entering the adder pipeline.
- At the end of the eighth cycle, the first four products $A_1 B_1$ through $A_4 B_4$ are in the four adder segments, and the next four products, $A_5 B_5$ through A_8, B_8 are in the multiplier segments.
- At the beginning of the ninth cycle, the output of the adder is $A_1 B_1$ and the output of the multiplier is $A_5 B_5$
- Thus the ninth cycle starts the addition $A_1 B_1 + A_5 B_5$ in the adder pipeline. The tenth cycle starts the addition $A_2 B_2 + A_6 B_6$, and so on.
- This pattern breaks down the summation into four sections as follows:

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

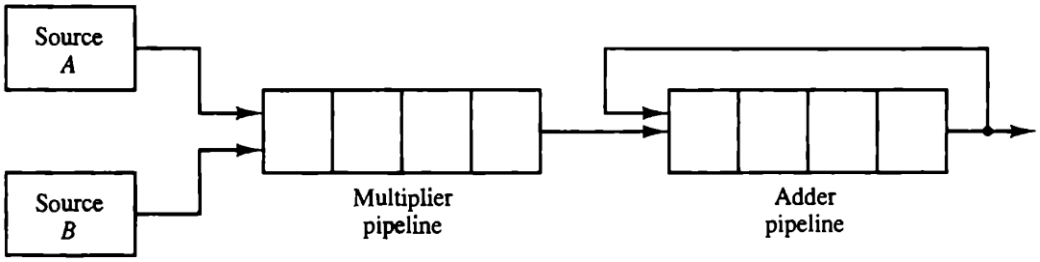


Figure 5-K: Pipeline for calculating an inner product

- When there are no more product terms to be added, the system inserts four zeros into the multiplier pipeline.
- The adder pipeline will then have one partial product in each of its four segments, corresponding to the four sums listed in the four rows in the above equation. The four partial sums are then added to form the final sum.

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.
- An **attached array processor** is an auxiliary processor attached to a general-purpose computer. It is intended to improve the performance of the host computer in specific numerical computation tasks.
- An **SIMD array processor** is a processor that 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:

- An attached array processor is designed as a peripheral for a conventional host computer, and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
- It achieves high performance by means of parallel processing with multiple functional units. It includes an arithmetic unit containing one or more pipelined floating point adders and multipliers.
- The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.
- **Figure 5-L** 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.

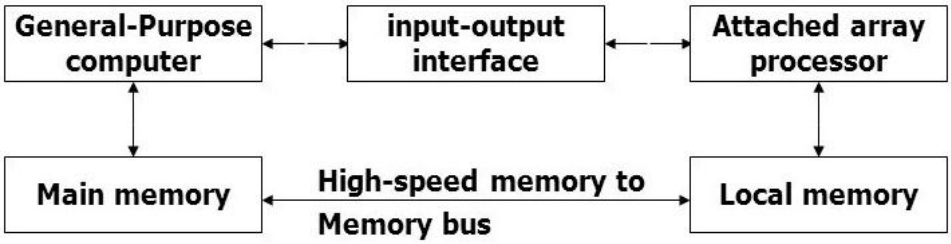


Figure 5-L: Attached array processor with host computer.

SIMD Array Processor:

- An SIMD array processor is a computer with multiple processing units operating in parallel.
- The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.
- A general block diagram of an array processor is shown in **Fig. 9-M**. It contains a set of identical **processing elements (PEs),** each having a local memory M.
- Each processor element includes an ALU, a floating-point arithmetic unit, and working registers. The master control unit controls the operations in the processor elements.
- The main memory is used for storage of the program. The function of the master control unit is to decode the instructions and determine how the instruction is to be executed.
- Vector instructions are broadcast to all PEs simultaneously. Each PE uses operands stored in its local memory. Vector operands are distributed to the local memories prior to the parallel execution of the instruction.

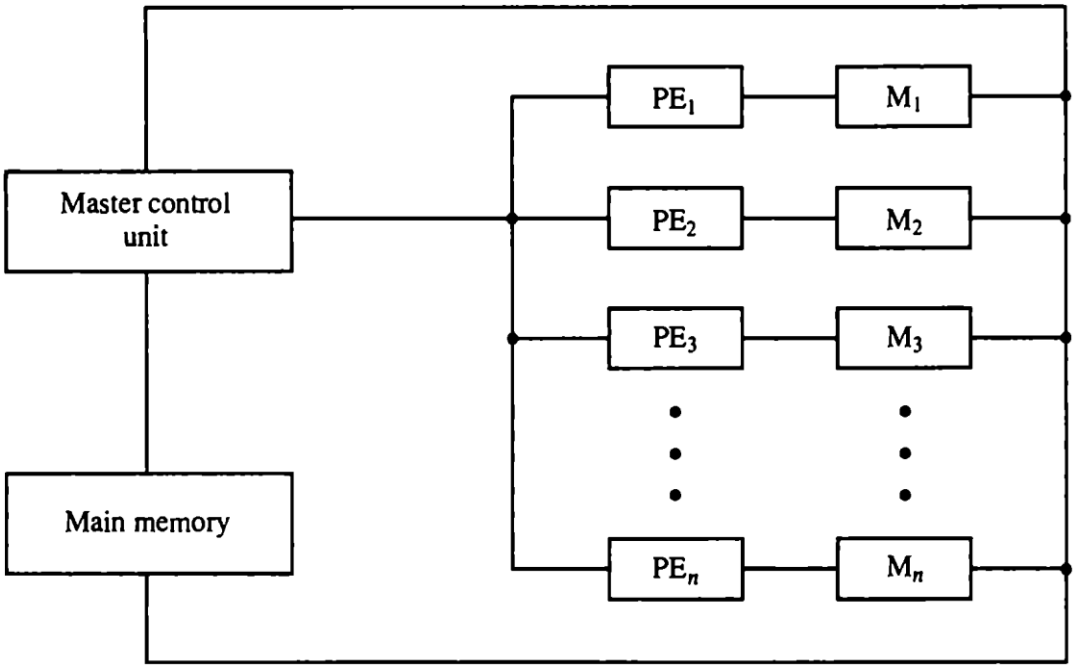


Figure 5-M: SIMD array processor organization.

Example: Consider the vector addition C = A + B. The master control unit first stores the ith components a_i and b_i of A and B in local memory M_i for i = 1, 2, 3. . . n.

- It then broadcasts the floating-point add instruction $c_i = a_i + b_i$ to all PEs, causing the addition to take place simultaneously.
- The components of c_i are stored in fixed locations in each local memory. This produces the desired vector sum in one add cycle.
- 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.
- This ensures that only those PEs that need to participate are active during the execution of the instruction.
- For example, suppose that the array processor contains a set of 64 PEs. 1f a vector length of less than 64 data items is to be processed, the control unit selects the proper number of PEs to be active. Vectors length of greater than 64 must be divided into 64-word portions by the control unit.

Multi Processors

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 (lOP).
- However, a system with a single CPU and one or more lOPs is usually not included in the definition of a multiprocessor system unless the lOP has computational facilities comparable to a CPU.
- As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more lOPs as well. Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.
- 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. This must be done prior to loading the program by specifying the parallel.
  - The other, more efficient way is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program. The compiler checks for data dependency in the program.
- Multi processors are classified by the way their memory is organized.

1. **Tightly coupled multiprocessor System:**
   - A multiprocessor system with common shared memory is classified as a **shared memory** or tightly coupled multiprocessor
   - Tightly coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory.
2. **Loosely Coupled Multiprocessor System:**
   - A loosely coupled multiprocessor system is a type of multiprocessing where the individual processors are configured with their own memory and are capable of executing user and operating system instructions independent of each other
   - Loosely coupled multiprocessor systems are also known as **distributed memory**, as the processors do not share physical memory and have their own IO channels.

Interconnection Structures

The interconnection between the components (CPUs and IOPs) can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system.

There are several physical forms available for establishing an interconnection network.

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 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 Fig. 5-N. Only one processor can communicate with the memory or another processor at any given time.

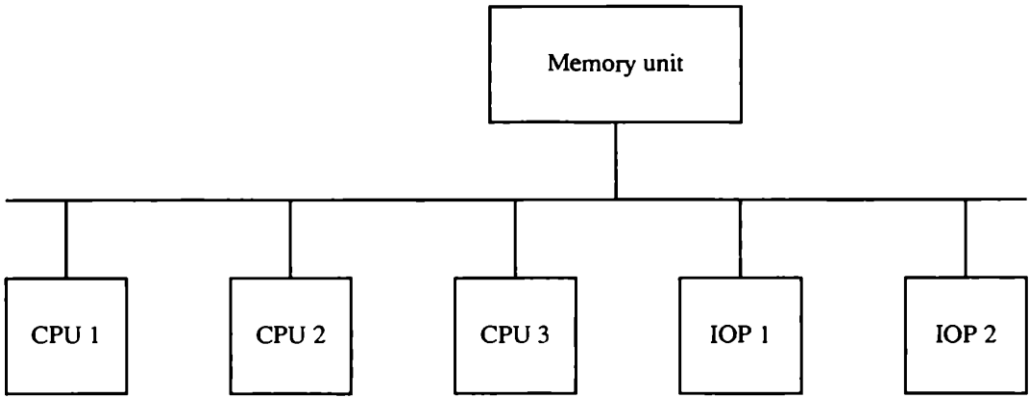


Figure5-N: Time-shared common bus organization.

- 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.

- 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.
- A more economical implementation of a dual bus structure is depicted in Fig. 5-O

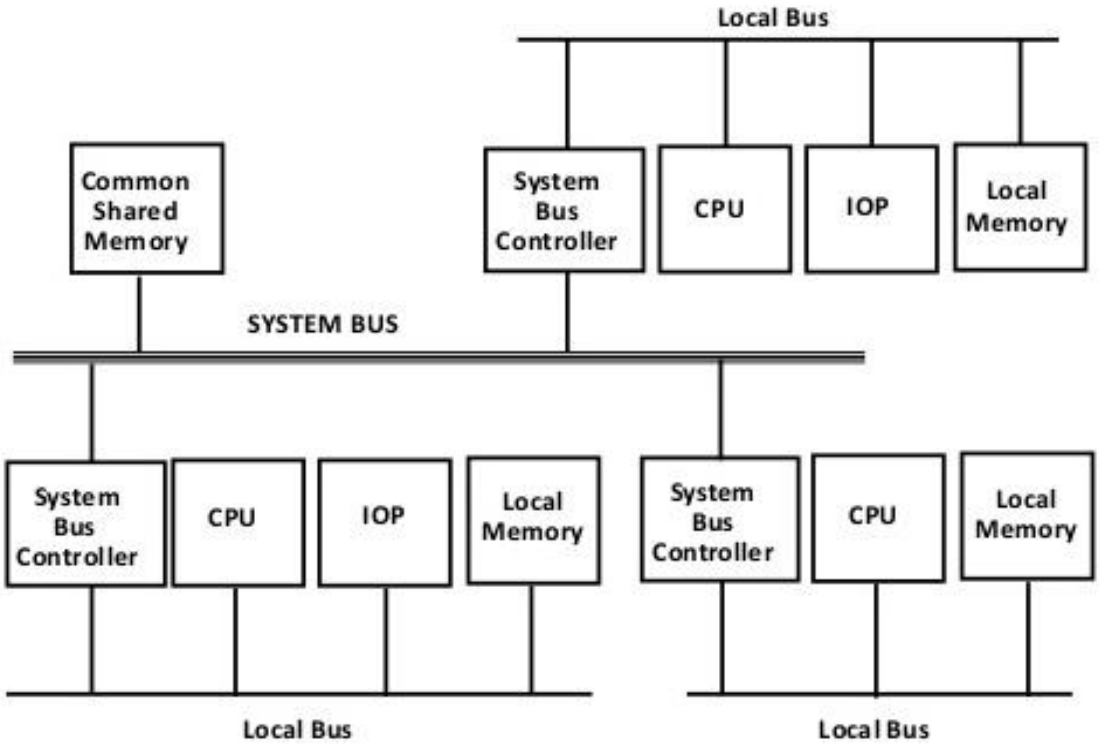


Fig 5-O: System bus structure for multi processors

- Here we have 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. If an lOP is connected directly to the system bus, the I/O devices attached to it may be made available to 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 I/O devices.

2. Multiport memory:

- A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig. 5-P 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.
- The memory module is said to have four ports 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

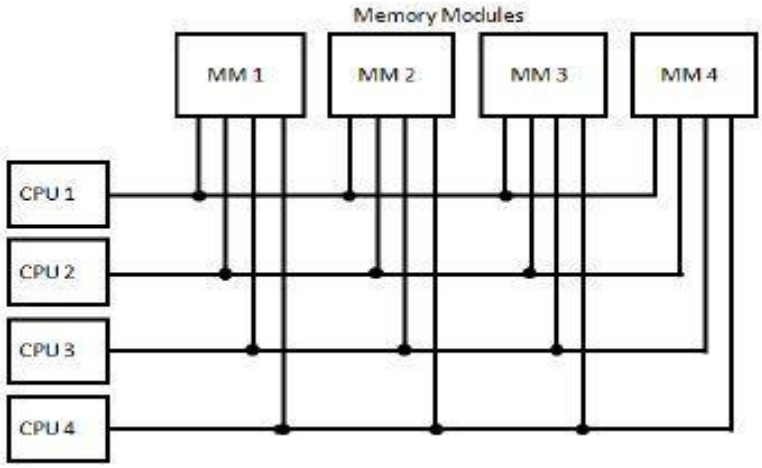


Figure 5-P: Multiport memory organization

- The advantage of the multi port 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

3. 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 5-Q shows a crossbar switch interconnection between four CPUs and four memory modules.
- 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.

Memory modules

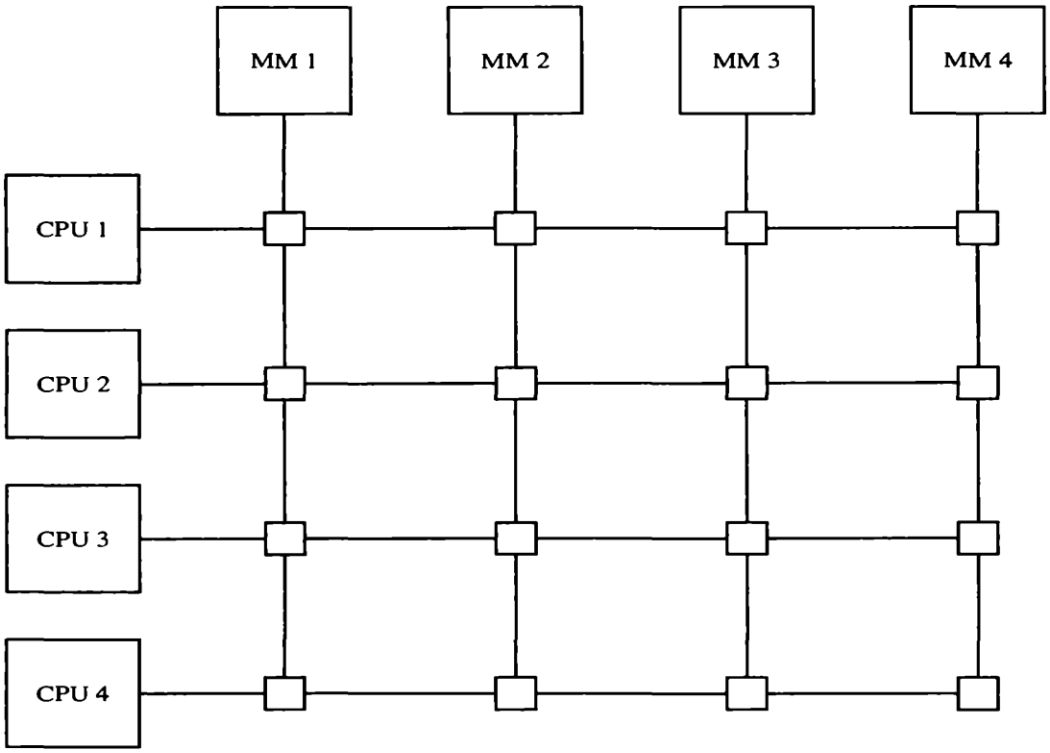


Fig 5-Q: Crossbar switch.

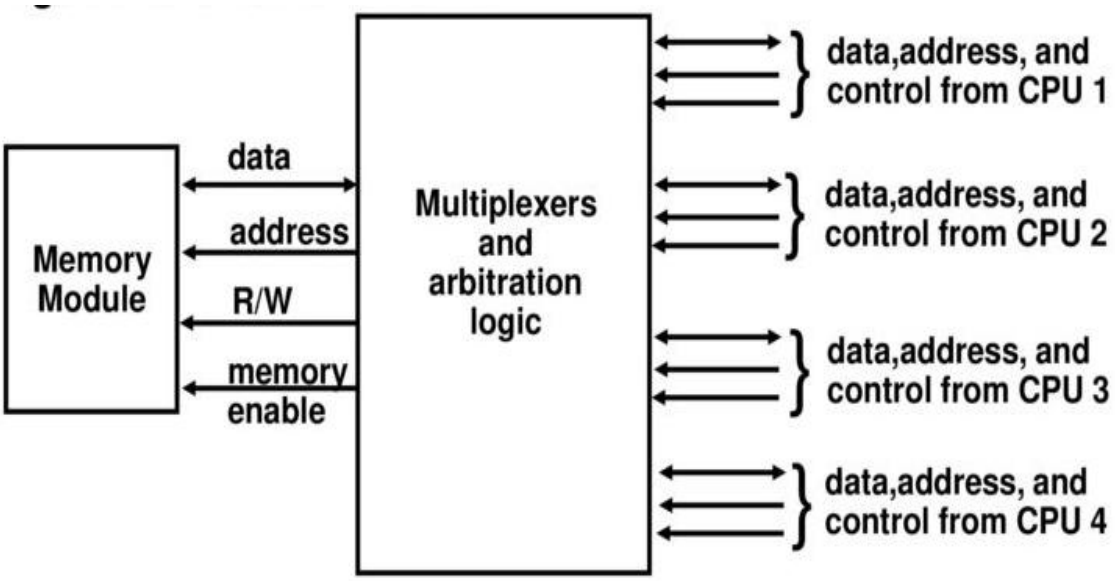


Figure 5-R: Block diagram of crossbar switch.

- Figure 5-R shows the functional design of a crossbar switch connected to one memory module.
- The circuit consists of multiplexers that select the 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.

- Crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module. However, the hardware required to implement the switch can become quite large and complex.

4. Multistage Switching Network:

- The basic component of a multistage network is a two-input, two-output, interchange switch. As shown in Fig. 5-S.
- The 2 X 2 switch has two input labeled A and B, and two outputs, labeled 0 and 1. There are control sign (not shown) associated with the switch that establish the interconnection between the input and output terminals.

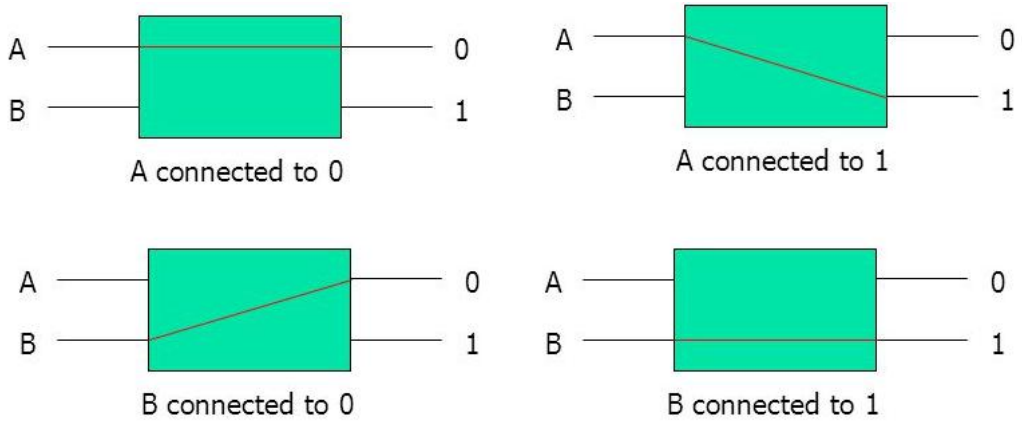


Figure 5-S: Operation of a 2 X 2 interchange switch.

- The switch has the capability connecting input A to either of the outputs. Terminal B of the switch behaves in a similar fashion.
- 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 source and destinations. To see how this is done, consider the binary tree shown Fig. 5-T.

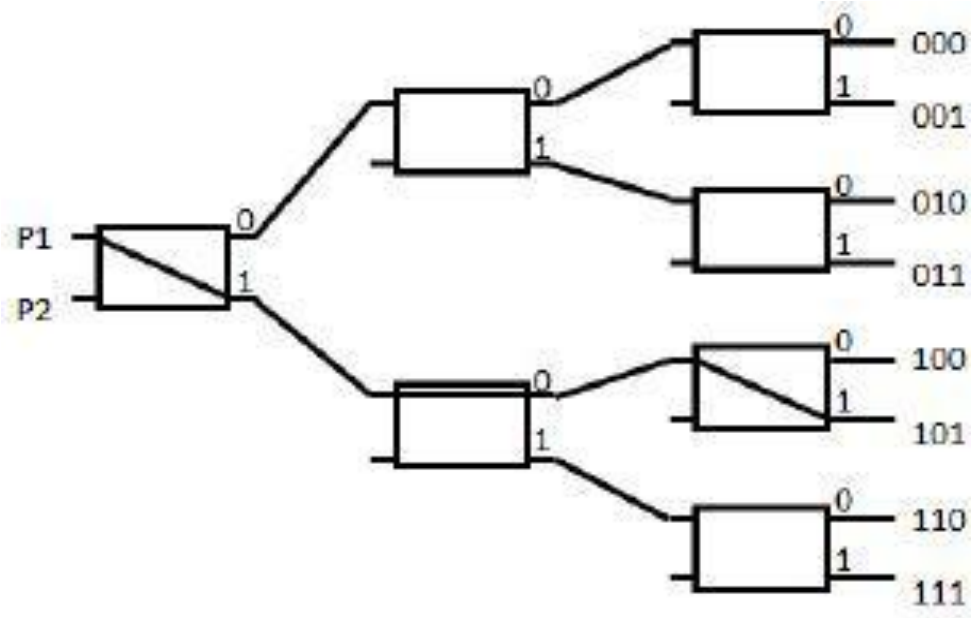


Figure 5-T: Binary tree with 2 X 2 switches.

- 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.
- 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 specifies the 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 output 1 in the first-level switch, output 0 in the second-level switch, and output 1 in the third-level switch.

❖ Many different topologies have been proposed for multistage switching networks to control processor-memory communication in a tightly coupled multiprocessor system or to control the communication between the processing elements in a loosely coupled system. One such topology is the **omega switching network** shown in Fig. 5-U.

8 X 8 Omega switching network:

- In this configuration, there is exactly one path from each source to any particular destination. Some request patterns, however, cannot be connected simultaneously.
- 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.

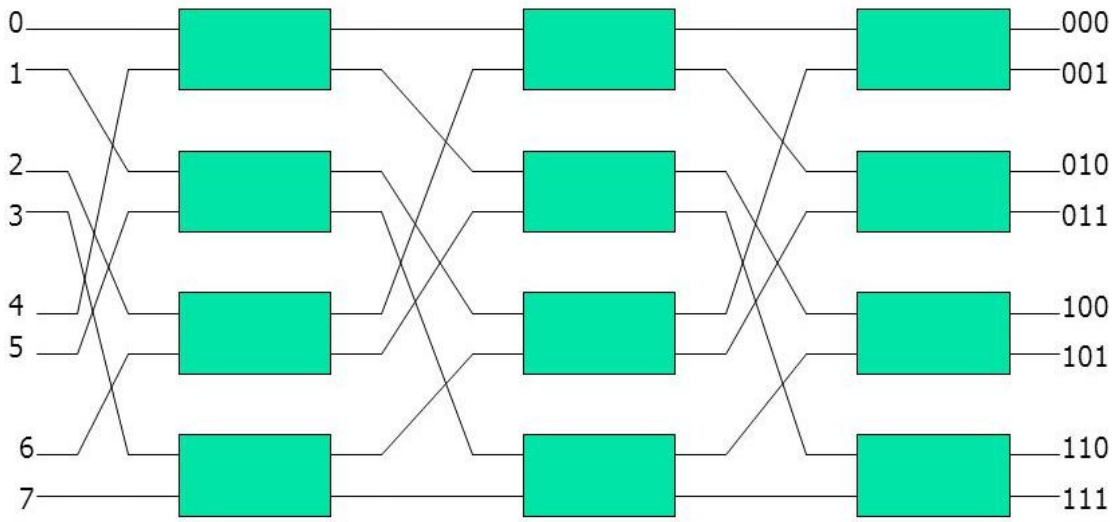


Fig. 5-U: 8 X 8 Omega switching network

- In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.
- The first pass through the network sets up the path. Succeeding passes are used to transfer the address into memory and then transfer the data in either direction, depending on whether the request is a read or a write.
- In a loosely coupled multiprocessor system, both the source and destination are 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 $N = 2^n$ processors interconnected in an n –dimension binary cube.
- Each processor forms a node of the cube. Each processor has direct communication paths to n other neighbor processors.
- These paths correspond to the edges of the cube. There are 2" distinct n-bit binary addresses that can be assigned to the processors.
- Each processor address differs from that of each of its n neighbors by exactly one bit position.
- Figure 5-V shows the hypercube structure for n = 1, 2, and 3.

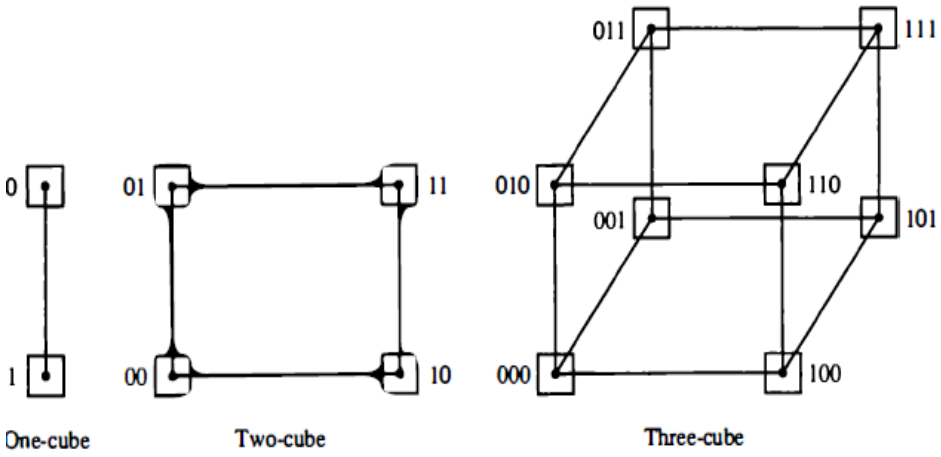


Figure 5-V: Hypercube structures for n = 1 , 2 ,3.

- A one-cube structure has n = 1 and $2^n = 2$. It contains two processors interconnected by a single path. A two-cube structure has n = 2 and $2^n = 4$. It contains four nodes interconnected as a square.
- A three-cube structure has eight nodes interconnected as a cube. An n -cube structure has 2^n 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.
- **For example,** the three neighbors of the node with address 100 in a three-cube structure are 000, 110, and 101.
- 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 (from 000 to 001 to 011 or from 000 to 010 to 011). It is necessary to go through at least three links to communicate from node 000 to node 111.
- A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address. The resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ. The message is then sent along any one of the axes.
- **For example**, in a three-cube structure, a message at 010 going to 001 produces an exclusive-OR of the two addresses equal to 01 1 . The message can be sent along the second axis to 000 and then through the third axis to 001.

Inter processor Arbitration

- Computer systems contain a number of buses at various levels to facilitate the transfer of information between components.
- 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 multiprocessor system, such as CPUs, lOPs, and memory, is called a system bus

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 similar to the priority interrupt logic.
- 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-W 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.

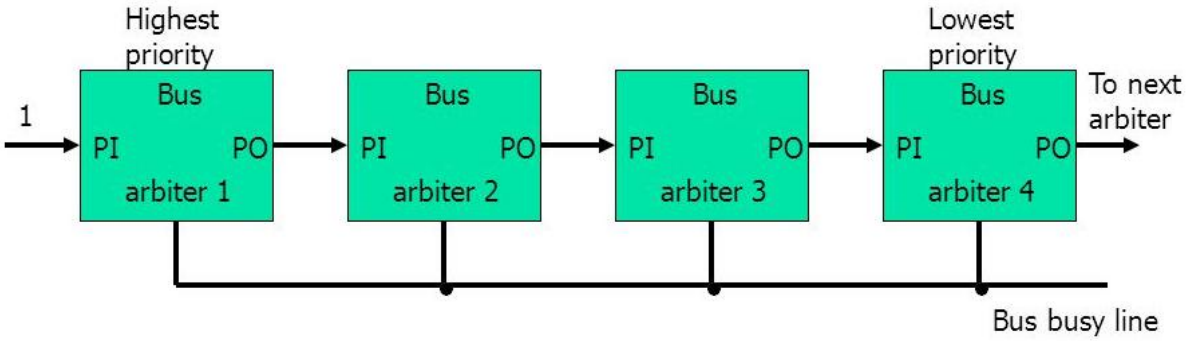


Figure 5-W: Serial (daisy-chain) arbitration

- The PI of the highest-priority unit is maintained at a 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. If 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.

Parallel Arbitration Procedure(Logic):

- The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in Fig. 5-X.
- 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 its acknowledge input line is enabled. The bus busy line provides an orderly transfer of control, as in the daisy-chaining case
- Figure 5-X 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.

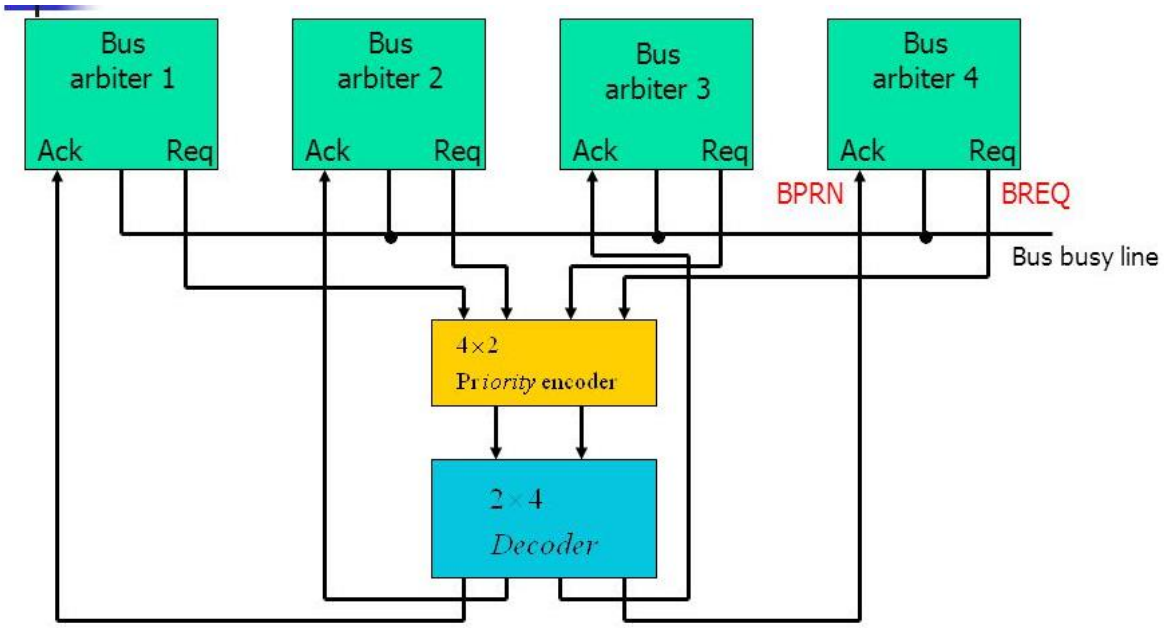


Figure 5-X: Parallel arbitration.

- The bus priority-in BPRN and bus priority-out BPRO are used for a daisy-chain connection of bus arbitration circuits.
- The bus busy signal BUSY is an open-collector output used to instruct all arbiters when the bus is busy conducting a transfer.

Inter processor 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 set aside 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.

- 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.
- This type of organization is also referred to as a floating operating system since the routines float from one processor to another and the execution of the routines 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 communication between processors is by means of message passing through I/O channels.

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.
- **For example**, parameters passed to a procedure in a different processor constitute inter processor communication. 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. The **hardware mechanism is mutual exclusion have been developed**.

Mutual Exclusion with a Semaphore:

- 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 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 to 1, 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.

Cache Coherence Problem:

- 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.
- The same information may reside in a number of copies in some caches and main memory. To ensure the ability of the system to execute memory operations correctly, the multiple copies must be kept identical. This requirement imposes a **cache coherence problem**.

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.

Example: Consider the three-processor configuration with private caches shown in Fig.5-Y.

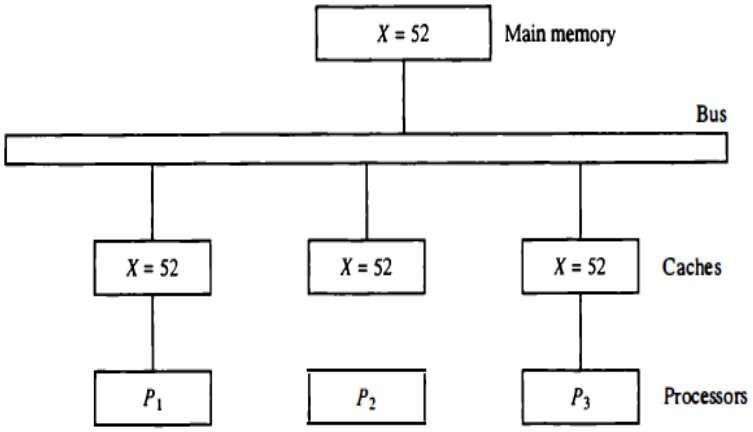


Figure 5-Y: Cache configuration after a load on X.

- ➢ 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 simplicity, 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.

- 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. 5-Z.

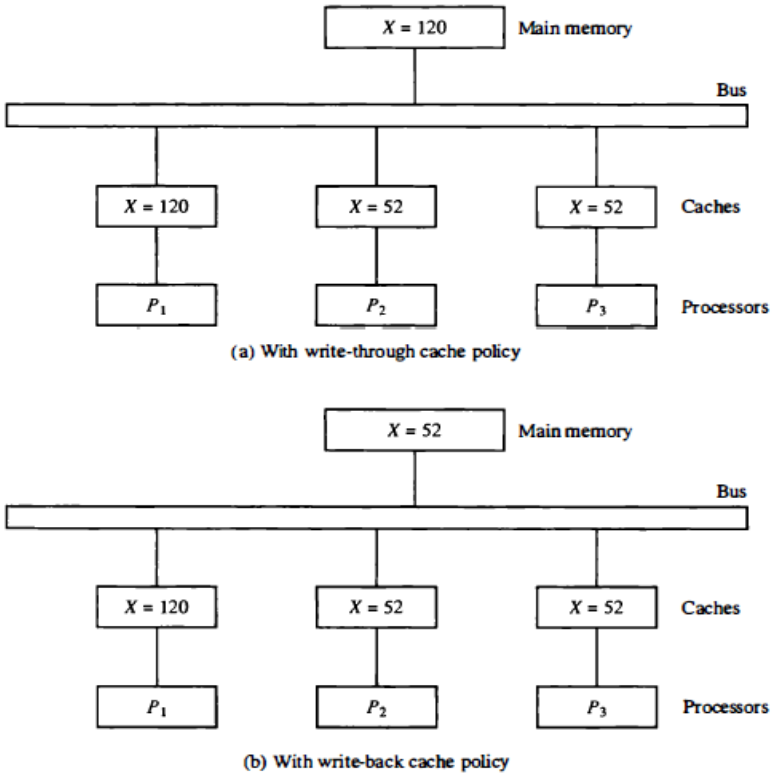


Figure 5-Z: Cache configuration after a store to X by processor P 1

- 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. The copies in the other two caches and main memory are inconsistent.
- Memory is updated eventually when the modified data in the cache are copied back into memory.

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.
- **This method violates** the principle of closeness of CPU to cache and increases the average memory access time. 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 noncachable. 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. This method restricts the type of data stored in caches .

UNIT-I

SHORT ANSWER QUESTIONS:

1. Define digital computer?
2. What is Common Bus System
3. List various registers in a computer along with their purpose (basic computer registers)
4. What is direct and indirect address explains?
5. What is a control unit?
6. What is pipeline register? What is the use of it? Explain in detail?
7. How do you map micro operation to a micro instruction address?
8. List the register-reference instructions?
9. List the of memory reference instructions.
10. Draw the block diagram of control unit?

LONG ANSWER QUESTIONS:

1. Explain various arithmetic micro-operations with hardware configuration.
2. Discuss about arithmetic circuit with an examples.
3. Explain about Logic and Shift Micro Operations with suitable Diagrams.
4. Design arithmetic Logic Unit.
5. List and explain different types of computer instructions. Also provide their formats.
6. What is instruction cycle?

(OR)

Draw and explain about the instruction cycle state diagram.

7. List and explain the register-reference instructions.
8. With the help of examples, explain in detail various types of memory reference instructions.
9. Briefly explain instruction cycle and interrupt cycle.
10. Explain briefly about input-output configuration.
11. Draw the interconnection structure of commonly used register connected to common bus.
12. Explain the block diagram of a control unit.

UNIT-II

SHORT ANSWER QUESTIONS:

1. Explain immediate addressing mode with example?
2. Explain about the condition field of micro operation?
3. Explain about the branch field of micro operation?
4. Discuss the significance of micro program sequencer?
5. List different types of addressing modes.
6. Describe One address Instruction Format.
7. Describe two address Instruction Format.
8. Describe Three address Instruction Format.
9. Describe Zero address Instruction Format.
10. List out Data Transfer Instructions.
11. Explain Status Bit Conditions

LONG ANSWER QUESTIONS:

1. Discuss about functioning of micro programmed control unit.
2. Diagrammatically, explain the process of selection of address for control memory.
3. Explain the following terms
   a) Control word
   b) Micro instruction
   c) Micro program
   d) Hardwired control
4. Explain the following terms
   1. Pipeline register
   2. Control address register
   3. Control memory
   4. Sequencer
5. With the help of a block diagram explain computer configuration
6. Explain about decoding of micro operation fields of micro instruction with diagram.
7. With the help of diagram, explain the organization of micro program sequencer for a control memory.
8. Discuss about various types of Instruction Formats.
9. What is Addressing Mode? Explain different types of addressing modes with example.
10. Explain about Data Transfer and Data Manipulation Instructions.

UNIT-III

SHORT ANSWER QUESTIONS:

1. What is floating point and fixed point representations?
2. Convert decimal 41. 6875 into binary
3. Explain about 9's and 10's Compliments.
4. Convert octal $(175)_8$ into Hexa –decimal
5. Convert octal 736.4 to decimal
6. Find (72532-03250) using 9's complements
7. Find the subtraction of 3250and 72532 using 10's complement
8. What is BCD?
9. Draw the flowchart for Addition and subtraction.
10. Draw BCD addition circuit and BCD addition circuit
11. What is overflow and underflow?

LONG ANSWER QUESTIONS:

1. Explain about fixed point representations with example.
2. Convert $(465.0647)_8$ into binary, decimal and hexadecimal equivalents.
3. Convert 1101101.1011 into its octal, decimal and hexadecimal equivalents.
4. Explain how a binary number can be converted to an octal and a hexadecimal number.
5. Convert $(19.625)_{10}$ into its binary, octal and hexadecimal equivalents.
6. Convert $(10A4.249)_{16}$ into its binary, octal and decimal equivalents
7. Convert the following numbers,
   a) 10101100111.0101 to Base 10
   b) $(153.513)_{10} = (\ )_8$
   c) Given that $(292)_{10} = (1204)_b$ determine 'b'.
8. What are the different types of complements? Explain.
9. Multiply 10111 with 10011 using Booth's algorithm.
10. Explain the addition and subtraction in BCD with hardware configuration.
11. Draw the flowchart for add and subtract operations and explain?
12. Explain the algorithm for adding and subtracting numbers in signed 2's complement representation?
13. Discuss the procedure for multiplication algorithm?
14. Discuss about Booth Multiplication algorithm with example.
15. Discuss about Decimal Arithmetic Circuit with examples.

UNIT-IV

SHORT ANSWER QUESTIONS:

1. Explain the memory hierarchy in a computer system?
2. What is meant by a bootstrap loader? Explain?
3. What is DMA?
4. What is the advantage of handshaking method over strobe control?
5. What is strobe control?
6. What is interrupt initiated mechanism in data transfer
7. What is daisy-chain method?
8. Explain I/O bus vs Memory bus
9. What is Isolated I/O vs Memory mapped I/O
10. What is priority interrupt?
11. Explain the memory address map for microcomputer
12. How many types of data transfer are there? What are their disadvantages?

LONG ANSWER QUESTIONS:

1. Explain the significance of input – output interface?
2. Explain the connection of I/O bus to input – output devices?
3. Explain the following

   a) I/O command b) data input command

4. What is the difference between isolated I/O and memory- mapped I/O? What are the advantages and disadvantages of each?
5. Explain the techniques used to achieve asynchronous data transfer?
6. Represent the block diagram and timing diagram for destination initiated strobe control for data transfer? Explain?
7. Explain the block diagram, timing diagram and sequence of events for destination initiated transfer using handshaking?
8. Explain daisy-chaining priority and parallel priority interrupt?
9. Discuss the block diagram of DMA controller with diagram?
10. Explain DMA transfer in a computer system?
11. Explain the procedure of writing into cache?

UNIT-V

SHORT ANSWER QUESTIONS:

1. Explain the Parallel Processing?
2. What is pipelining? Explain it?
3. Show space-time diagram for pipeline. Explain with an example?
4. Explain arithmetic pipeline?
5. Explain vector processing?
6. Explain Array Processors?
7. Describe the characteristics of multiprocessors?
8. Explain four segment pipelining?
9. What are the benefits of multiprocessor organization?
10. Give the timing diagram of instruction pipeline?
11. Define characteristics of CISC and RISC

LONG ANSWER QUESTIONS:

1. Explain Instruction Pipeline?
2. What is meant by instruction pipeline? Explain four segment Instruction Pipeline
3. Explain RISC Pipeline Vector Processing?
4. What is pipelining? Explain pipeline processing with an example?
5. Explain RISC pipeline in detail?
6. Differentiate b/w Arithmetic Pipeline & Instruction Pipeline ?
7. Diff b/w Parallel Processing and RISC Pipeline Vector Processing?
8. Explain how inter processor communication can be achieved?
9. Explain daisy-chain arbitration procedure?
10. Discuss the parallel arbitration procedure?

PREVIOUS YEARS QUESTION PAPERS