Pipelining and Vector Processing

Parallel Processing:

- Parallel processing is a term used for a large class of techniques that are used to provide **simultaneous data-processing tasks** for the purpose of increasing the computational speed of a computer system.
- It refers to techniques that are used to provide simultaneous data processing.
- The system may have two or more ALUs to be able to execute two or more instruction at the same time.
- The system may have two or more processors operating concurrently.
- It can be achieved **by having multiple functional units that perform same or different operation simultaneously.**
- **Parallel processing is done by distributing the data among multiple functional Units.**

Processor with Multiple function units:

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

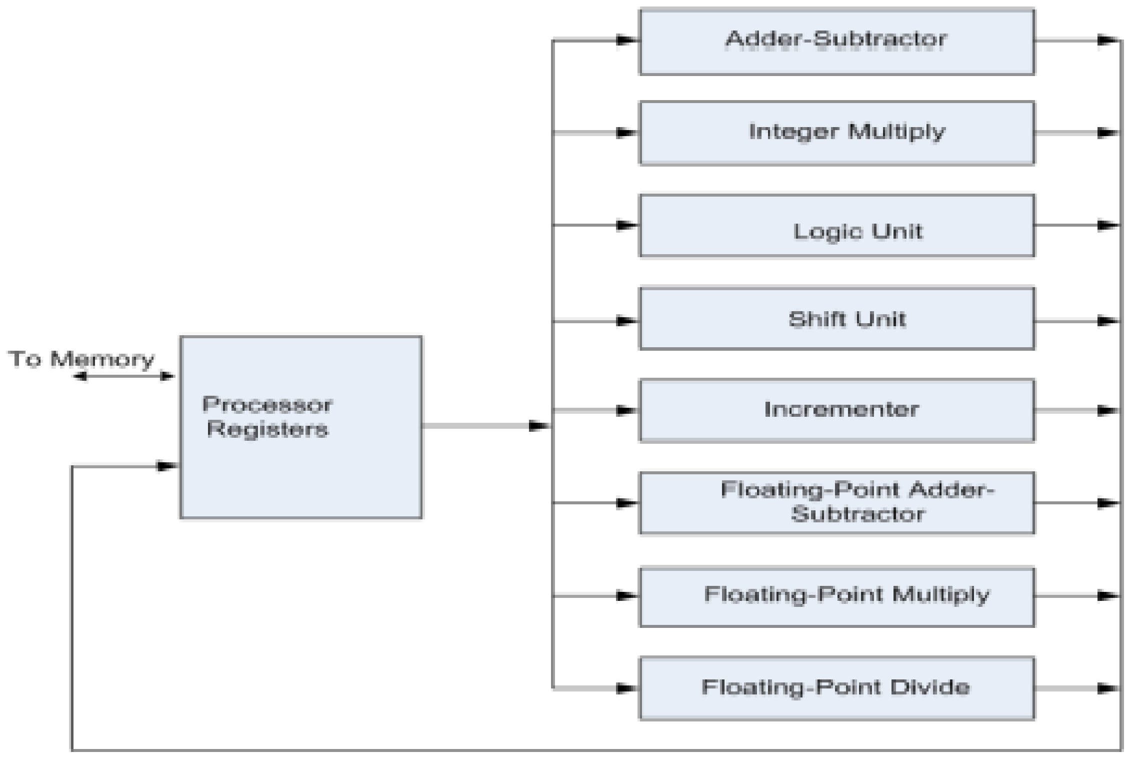
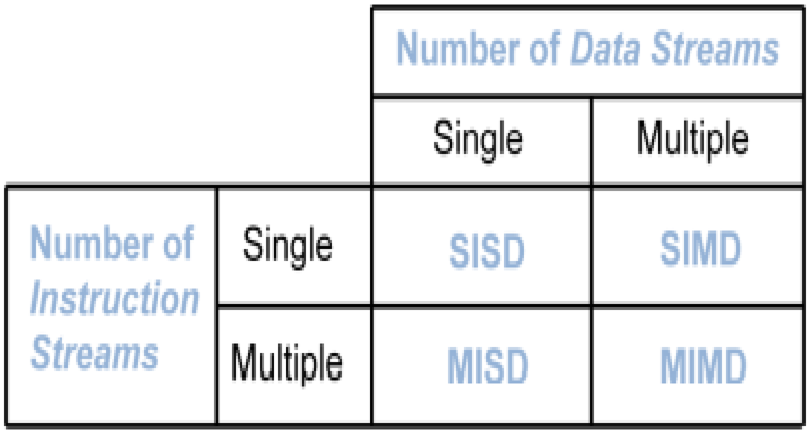


Fig: Processor with Multiple functional units

- The operation performed in each functional unit is indicated in each block of the diagram.
- The Adder and integer multiplier perform arithmetic operation with Integer numbers.
- The floating point operations are separated into 3 circuits operating in parallel.
- The logic, shift, and increment operation can be performed concurrently on different data.
- All units are independent, so one number can be shifted while another number is being activated.

- Architectural Classification: –
- Flynn's classification
- Considers the organization of a computer system by number of instructions and data items that are manipulated simultaneously.
- Based on the multiplicity of Instruction Streams and Data Streams
- **Instruction Stream**-Sequence of Instructions read from memory
- **Data Stream** - Operations performed on the data in the processor
- Parallel processing may occur in the instruction stream, in the data stream or in both.
- Flynn's classification divides computer into 4 major groups:
    1. SISD (Single Instruction stream, Single Data stream)
    2. SIMD (Single Instruction stream, Multiple Data stream)
    3. MISD (Multiple Instruction stream, Single Data stream)
    4. MIMD (Multiple Instruction stream, Multiple Data stream



		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

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

internal parallel processing capabilities.

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

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

PIPELINING

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

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

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

  R1 ← Ai , R2 ← Bi   Input Ai and Bi

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

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

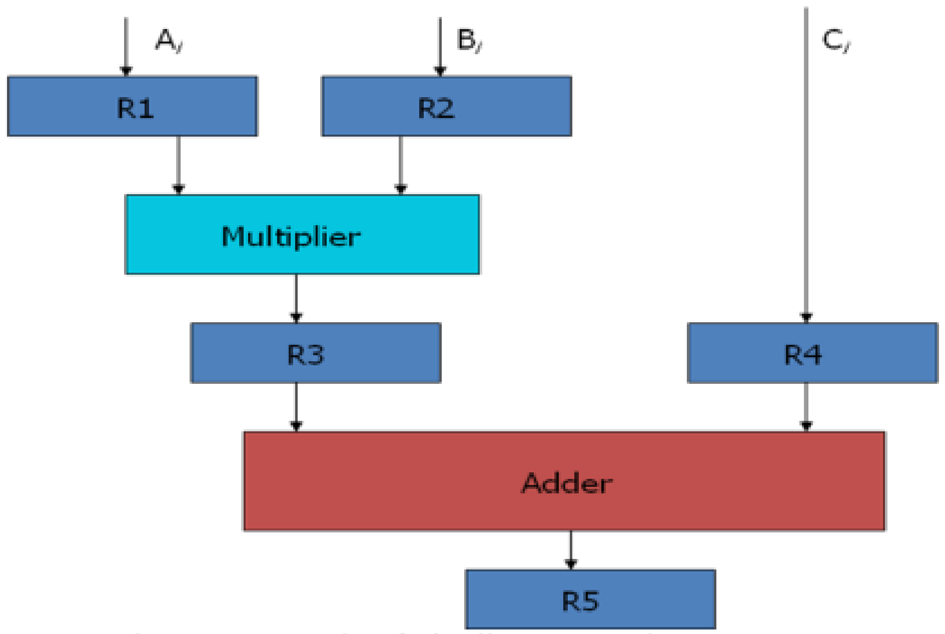
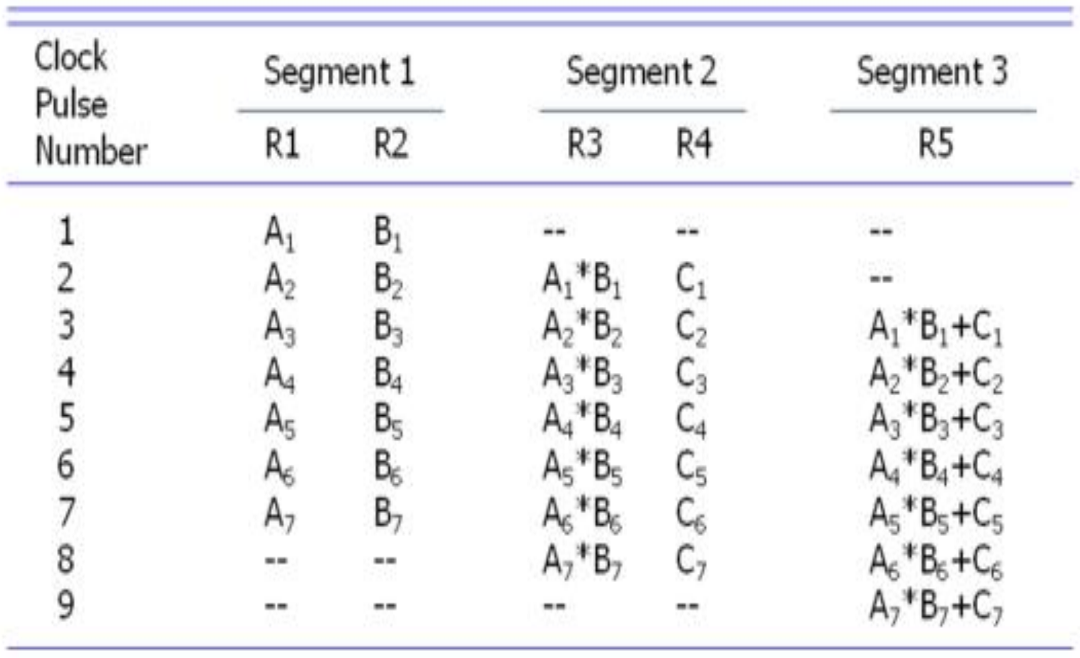


Fig 4-1: Example of pipeline processing

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



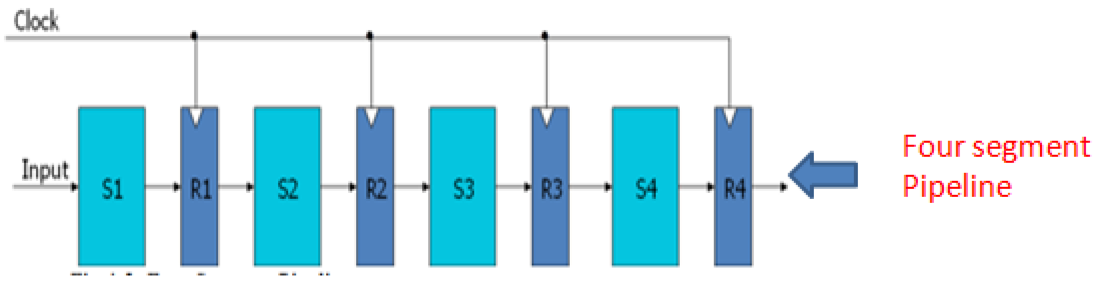
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	--	--	--
2	A_2	B_2	A_1*B_1	C_1	--
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	--	--	A_7*B_7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

Table 4-1: Content of Registers in Pipeline Example

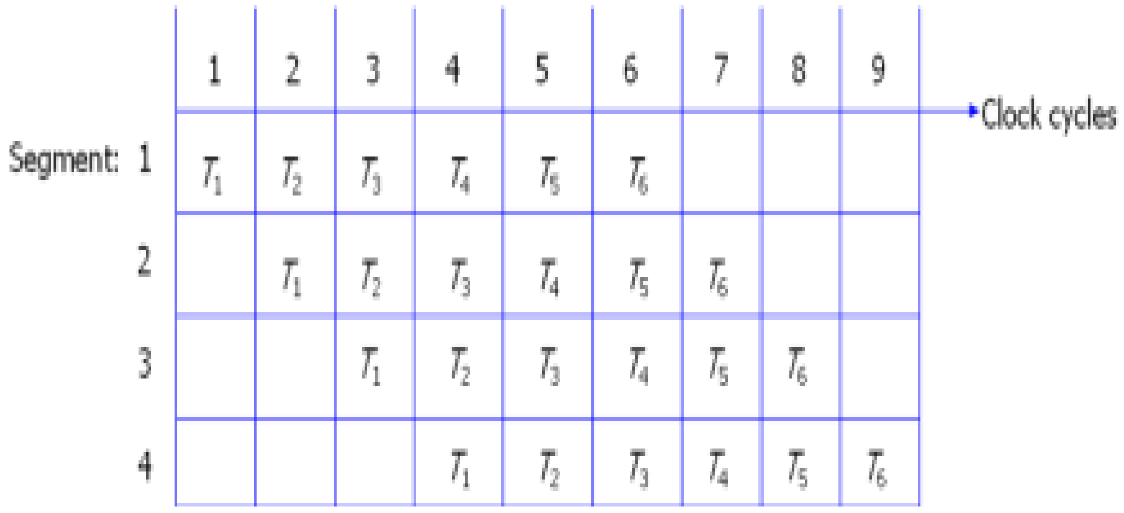
General Considerations:

- Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor.

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



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



	1	2	3	4	5	6	7	8	9
Segment: 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 cycles

Fig 4-3: Space-time diagram for pipeline

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

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

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

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

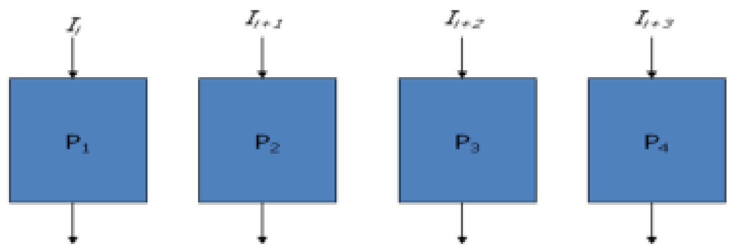


Fig 4-4: Multiple functional units in parallel

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

Arithmetic pipeline

Instruction pipeline

RISC pipeline

Arithmetic pipeline:

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

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

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

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

1.Compare the exponents

2. Align the mantissa

3. Add or subtract the mantissas

4. Normalize the result

Example: Consider two floating point numbers binary addition

$$\mathbf{X = 0.9504 * 10^{3}}$$

$$\mathbf{Y = = 0.8200 * 10^{2}}$$

1. Compare exponents by subtraction:

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

2. Align the mantissas:

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

$$\mathbf{X = 0.9504 * 10^{3}}$$

$$Y = 0.08200 * 10^3$$

3. Add mantissas:

- The two mantissas are added in segment three.

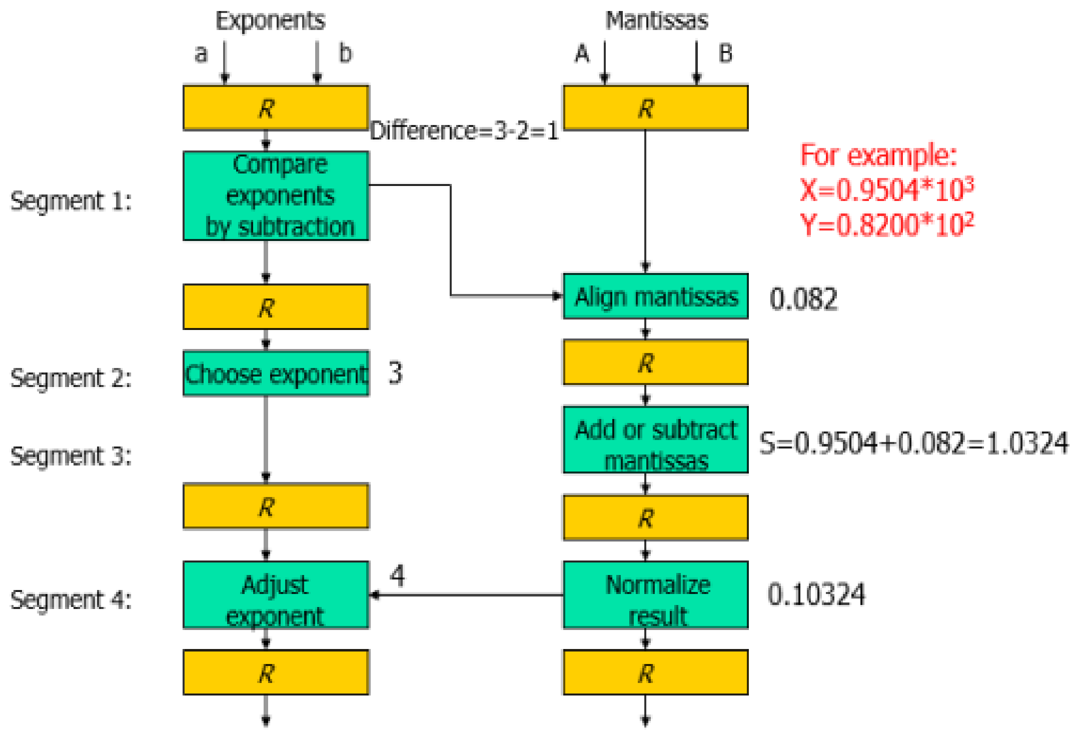
$$Z = X + Y = 1.0324 * 10^3$$

4. Normalize the result:

- After normalization, the result is written as:

$$Z = 0.1324 * 10^4$$

Flow chart for floating point addition and subtraction using Pipelining



Pipelining for Floating point Addition and Subtraction

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

with the smaller exponent must be shifted to the right.

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

Instruction Pipeline:

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

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Calculate the effective address.

4. Fetch the operands from memory.

5. Execute the instruction.

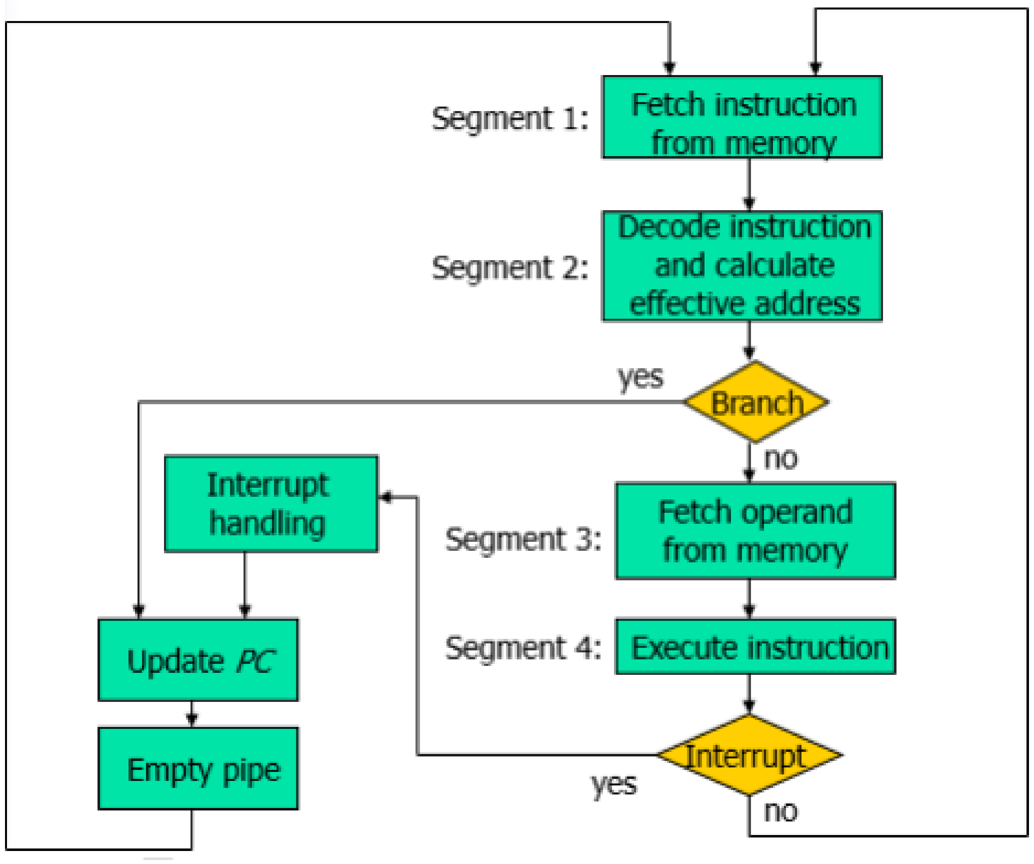
6. Store the result in the proper place.

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

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

Example: four-segment instruction pipeline:

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



- **Thus up to four suboperations in the instruction cycle can overlap** and up to four different instructions can be in progress of being processed at the same time.

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

    1. FI is the segment that fetches an instruction.

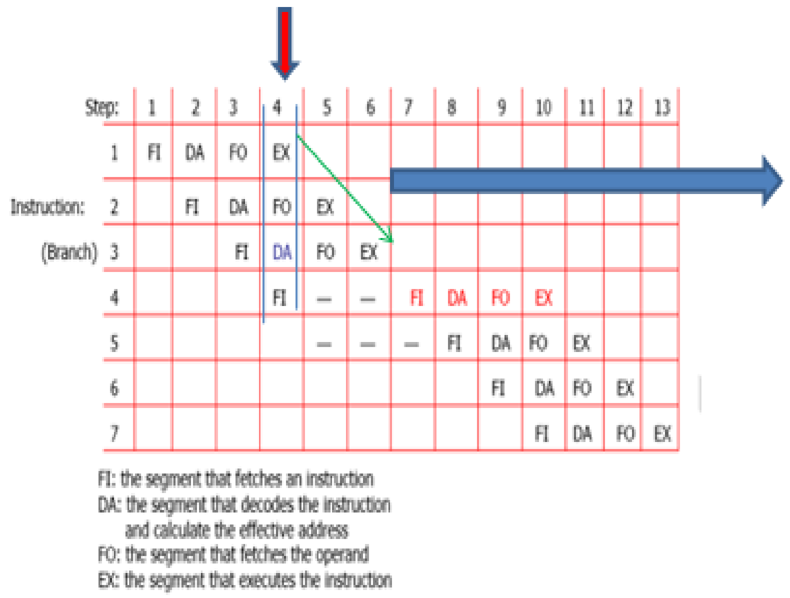
    2.DA is the segment that decodes the instruction and calculates the effective address.

    3. FO is the segment that fetches the operand.

    4. EX is the segment that executes the instruction

Timing of Instruction Pipeline

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



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

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

Instruction 1,2 are executed sequentially, and at instruction 3, there is a branching address

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

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

1. Structural Hazards

2. Data Hazard

3. Control hazard

1. Structural Hazards:

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

- Can be resolved by using separate instruction and data memories

2. **Data Hazard:**

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

3. Control Hazard:

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

RISC (Reduced Instruction Set Computer)Pipeline:

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

I: Instruction fetch

- Fetches the instruction from program memory

A: ALU operation

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

E: Execute instruction

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

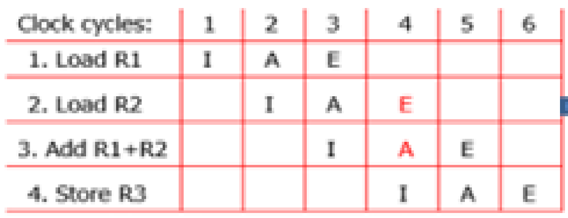
Delayed Load:

- Consider the operation of the following four instructions:

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$

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

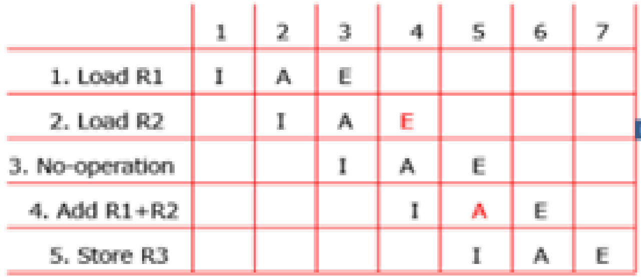
Pipelining Timing with Delayed load:



Clock cycles:	1	2	3	4	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

9 (a) Pipeline timing with data conflict

At 4th clock cycle, the data is not placed in R2 but the A segment in clock cycle 4 wants the data from R2 to perform addition operation.



	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No-operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

9 (b) Pipeline timing with delayed load

This conflict can be overcomed by inserting a no operation instruction in the insturction3 and thus delaying the addition operation by one clock cycle.

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

Delayed Branch

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

    1. Load from memory to R1

2. Increment R2

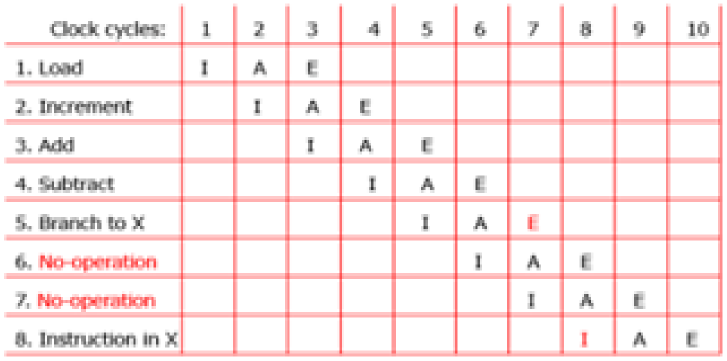
3. Add R3 to R4

4. Subtract R5 from R6

5. Branch to address X

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

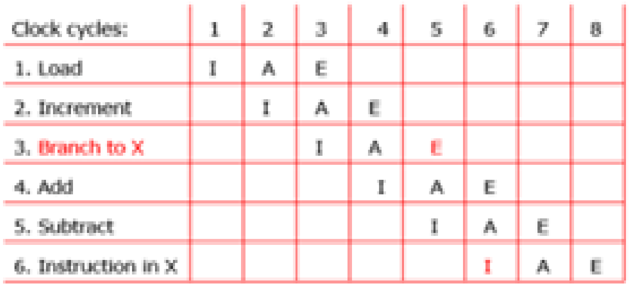
Pipelining Timing with Delayed Branch:



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

10 (a) Using no-operation instructions

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



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

10 (b) Rearranging the instructions

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

Vector processing:

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

Examples:

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

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

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

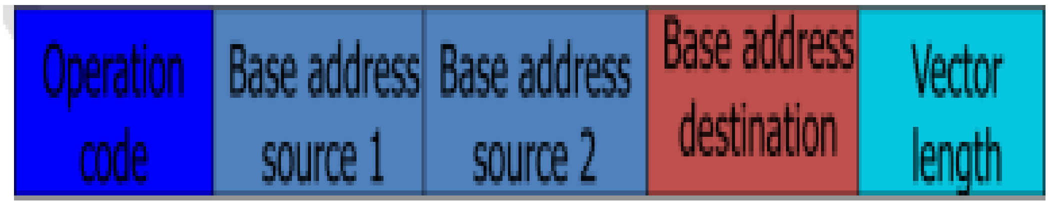
```
                    Initialize I=0
                    Read A(I)
```

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

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

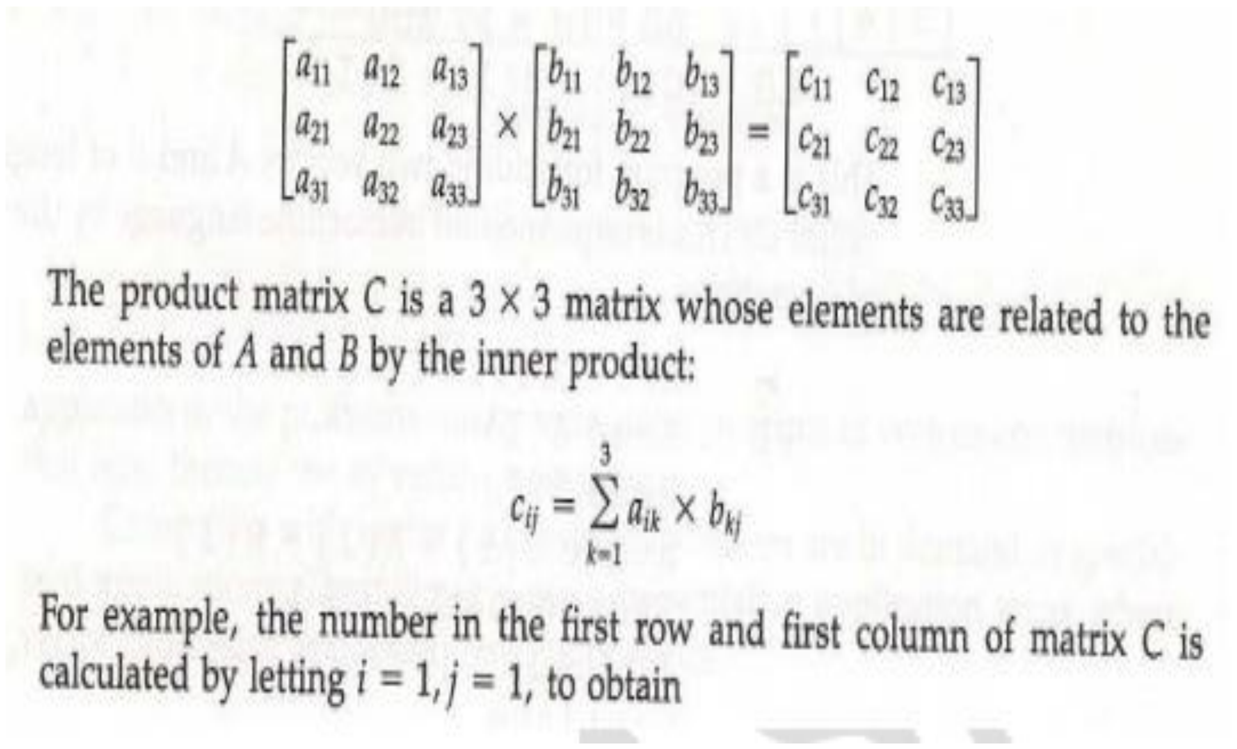
  C(1:100)=A(1:100)+B(1:100)

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



Matrix Multiplication

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



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

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

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

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

c11= a11b11+ a12b21+ a13b31

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

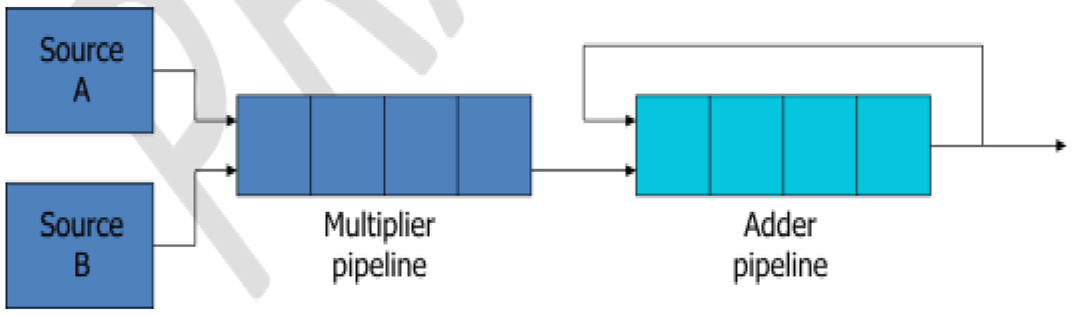
C = A1B1+A2B2+A3B3+…+AkBk.

- In a typical application k may be equal to 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown in Fig. 9-12.

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

Implementation of the Vector Processing

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

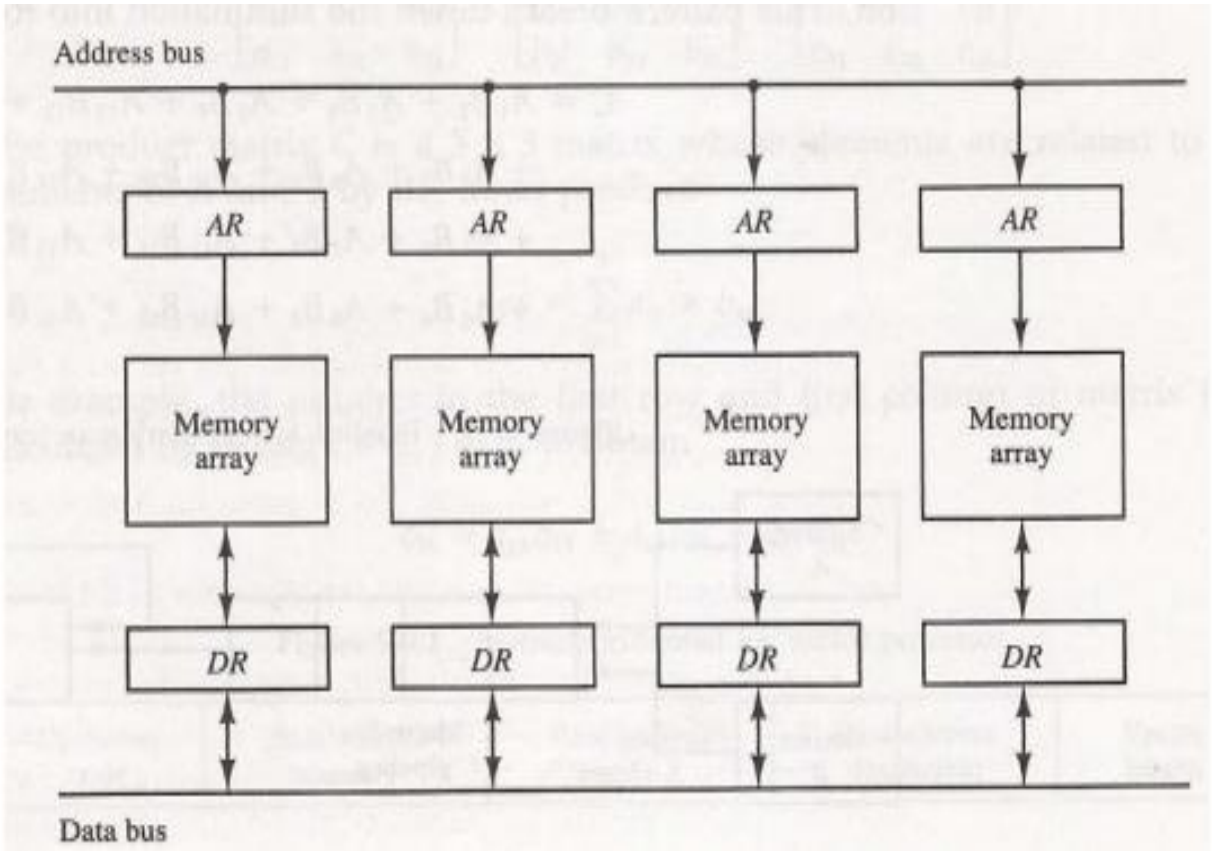


- In the above diagram we can see that how the values of A vector and B Vector which represents the matrix are being multiplied. Here we will be considering a 4x4 matrix A and B.
- When addition operation is taking place in the adder pipeline the next set of values will be brought into the multiplier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

Memory Interleaving:

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

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



Multiple module Memory Organization

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

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

Array Processors:

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

Attached array processor:

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

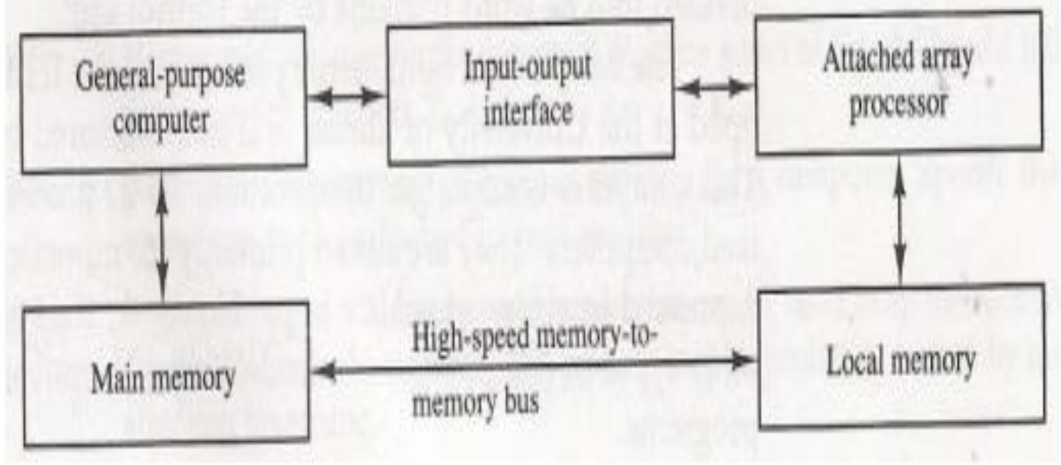
SIMD array processor:

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

Attached Array Processor

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

Figure 9-14 Attached array processor with host computer.



Attached Array Processor with host computer

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

SIMD Array Processor:

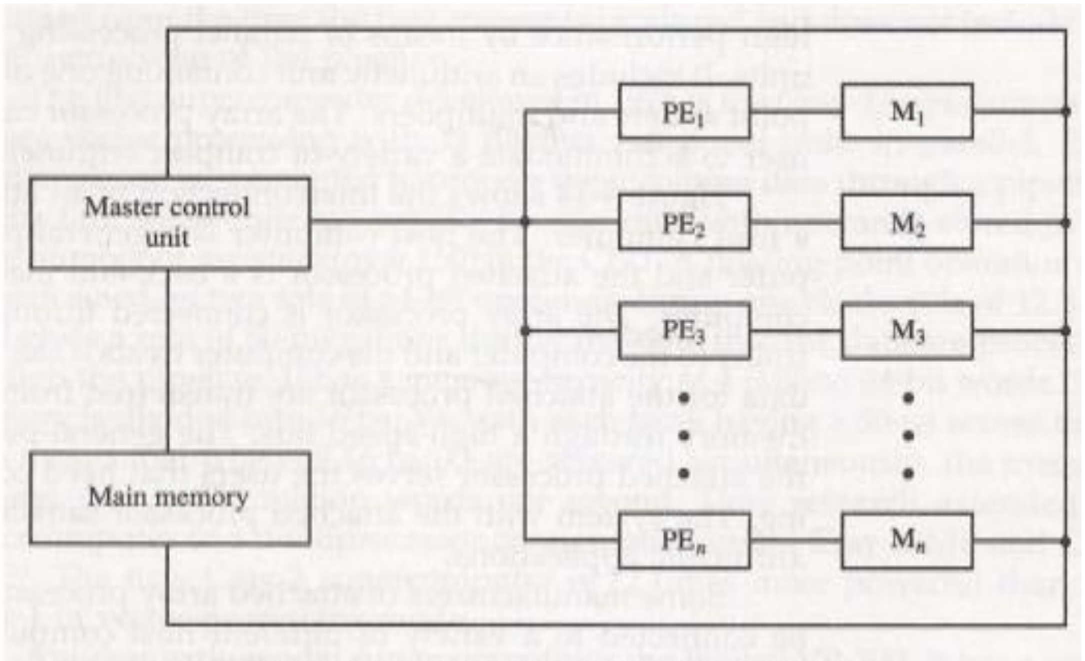
- An SIMD array processor is a computer with multiple processing units operating in parallel.
- A general block diagram of an array processor is shown in Fig. 9-15.
- 

Figure 9-15 SIMD array processor organization.

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

Illinois and manufactured by the Burroughs Corp.

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