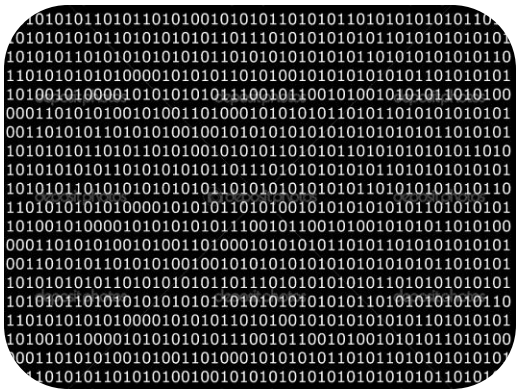
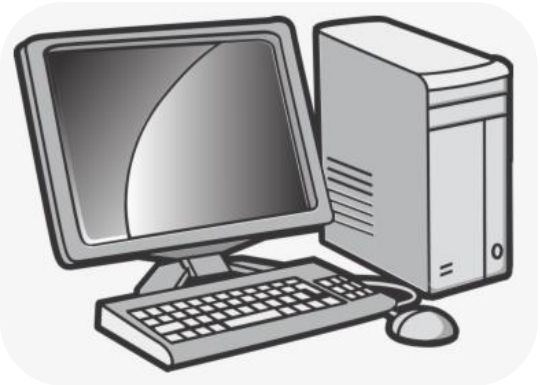
AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

Unit-6

Pipeline And Vector Processing





Subject:- COA **Code:-3140707**	**Prepared by:** **Asst. Prof. S.Y.JOSHI** **(CSE Department,ACET)**	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Topics to be covered

- Flynn's taxonomy,
- Parallel Processing,
- Pipelining,
- Arithmetic Pipeline,
- Instruction Pipeline,
- RISC Pipeline,
- Vector Processing,
- Array Processors

Parallel Processing

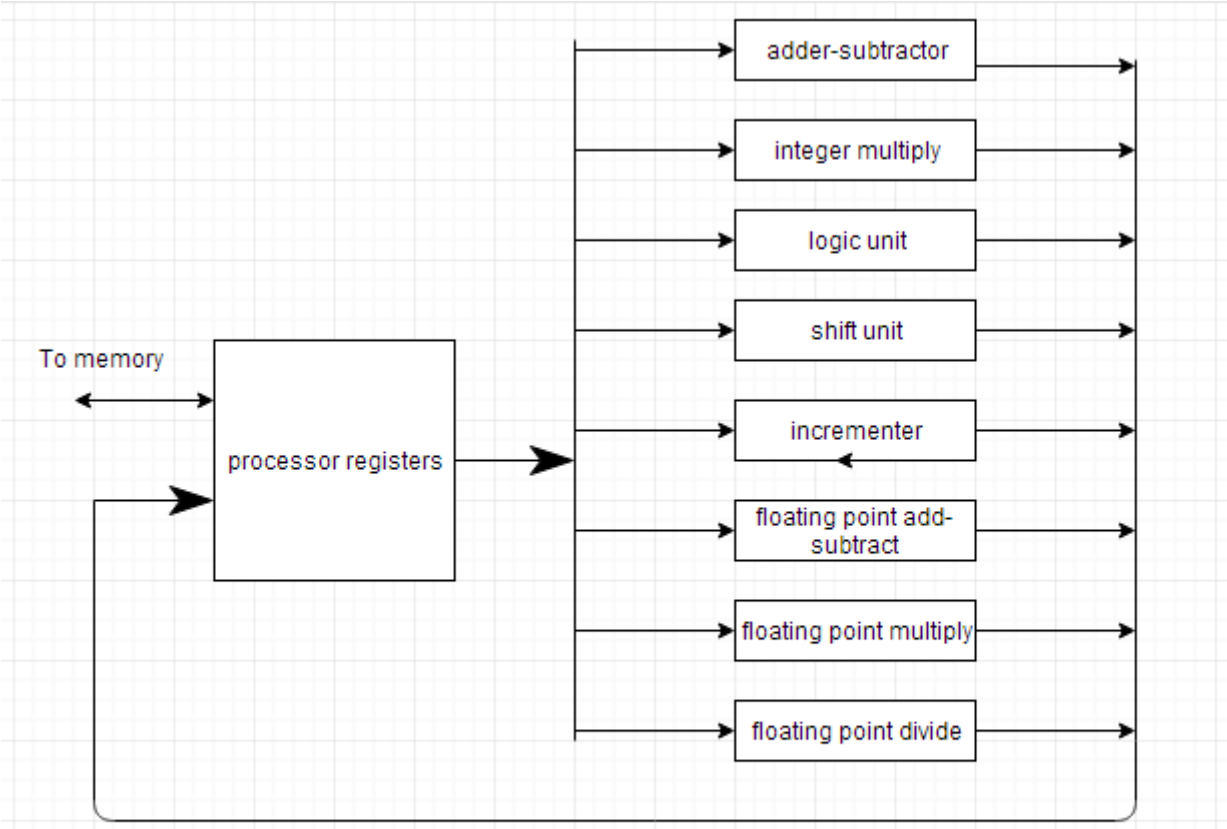
A parallel processing system is able to perform concurrent data processing to achieve faster execution time

- The system may have two or more ALUs and be able to execute two or more instructions at the same time

- Goal is to increase the throughput – the amount of processing that can be accomplished during a given interval of time

- Parallel processing is established by distributing data among multiple functional units. Fig 1 shows the separation of execution unit into eight functional units operating in parallel.

Parallel Processing



PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

Flynn's Taxonomy

There are variety of ways parallel processing can be classified. Parallel processing occuring in instruction stream/data stream or both.

Flynn's classification divides computer into four major groups as follows:

Single instruction stream, single data stream – SISD

Single instruction stream, multiple data stream – SIMD

Multiple instruction stream, single data stream – MISD

Multiple instruction stream, multiple data stream – MIMD

Pipelining

- Pipelining is a technique of decomposing a sequential process into suboperations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- Pipeline can be visualized as collection processing segments. And binary info flows through each of them
- Pipeline implies flow of info similar to industry assembly line.

Pipelining example

for example we want to perform combined multiply and add operations with stream of numbers

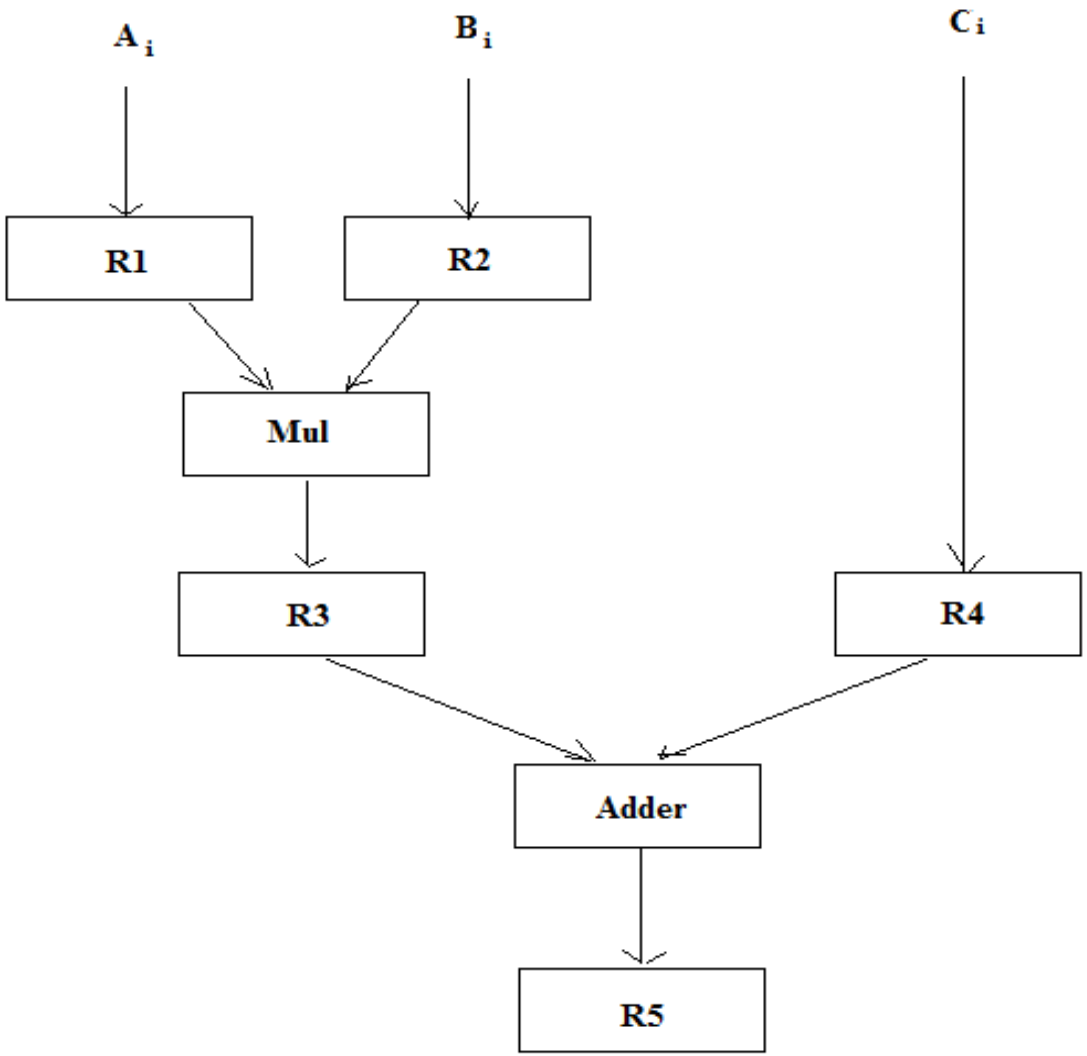
$A_i*B_i + C_i$ *for* $i=1,2,3...7$

The sub operations performed in each segment of the pipeline are as follows:

R1 ← Ai, R2 ← Bi

R3 ← R1 * R2 R4 ← Ci

R5 ← R3 + R4



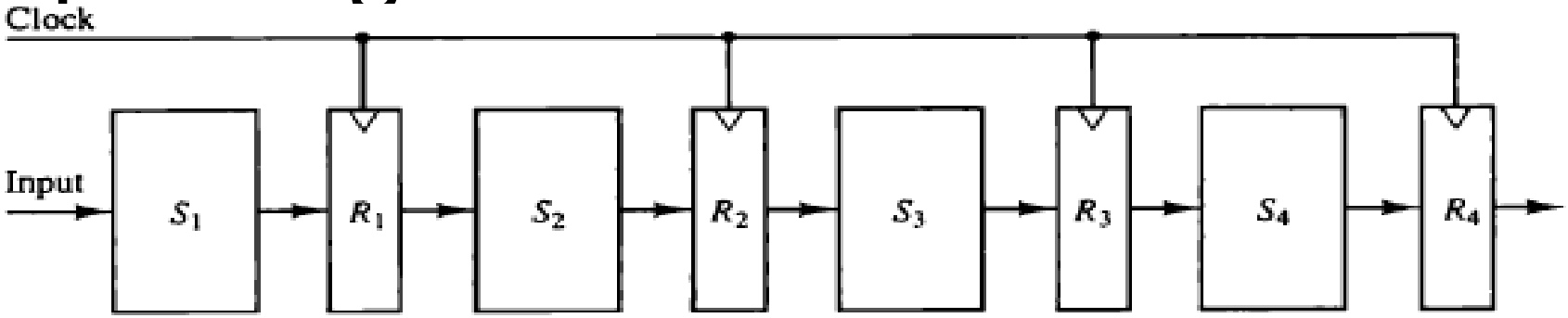
Pipelining example

Content of registers of pipelining example

- The five registers are loaded with new data every clock pulse.
- The effect of each clock is shown in table as shown
- The clock must continue until last output emerges out of the pipeline.

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

Pipelining



- The figure above shows the general structure of four segment pipeline.
- Each segment consists of combinational ckt S_i that performs a sub operation over data stream through pipe.
- The behavior of pipeline can be understood by the space time diagram below.

1	2	3	4	5	6	7	8	9
T1	T2	T3	T4	T5	T6			•
	T1	T2	T3	T4	T5	T6		
		T1	T2	T3	T4	T5	T6	
			T1	T2	T3	T4	T5	T6

Refer the space time diagram for 6-tasks and 4 segments

Pipelining

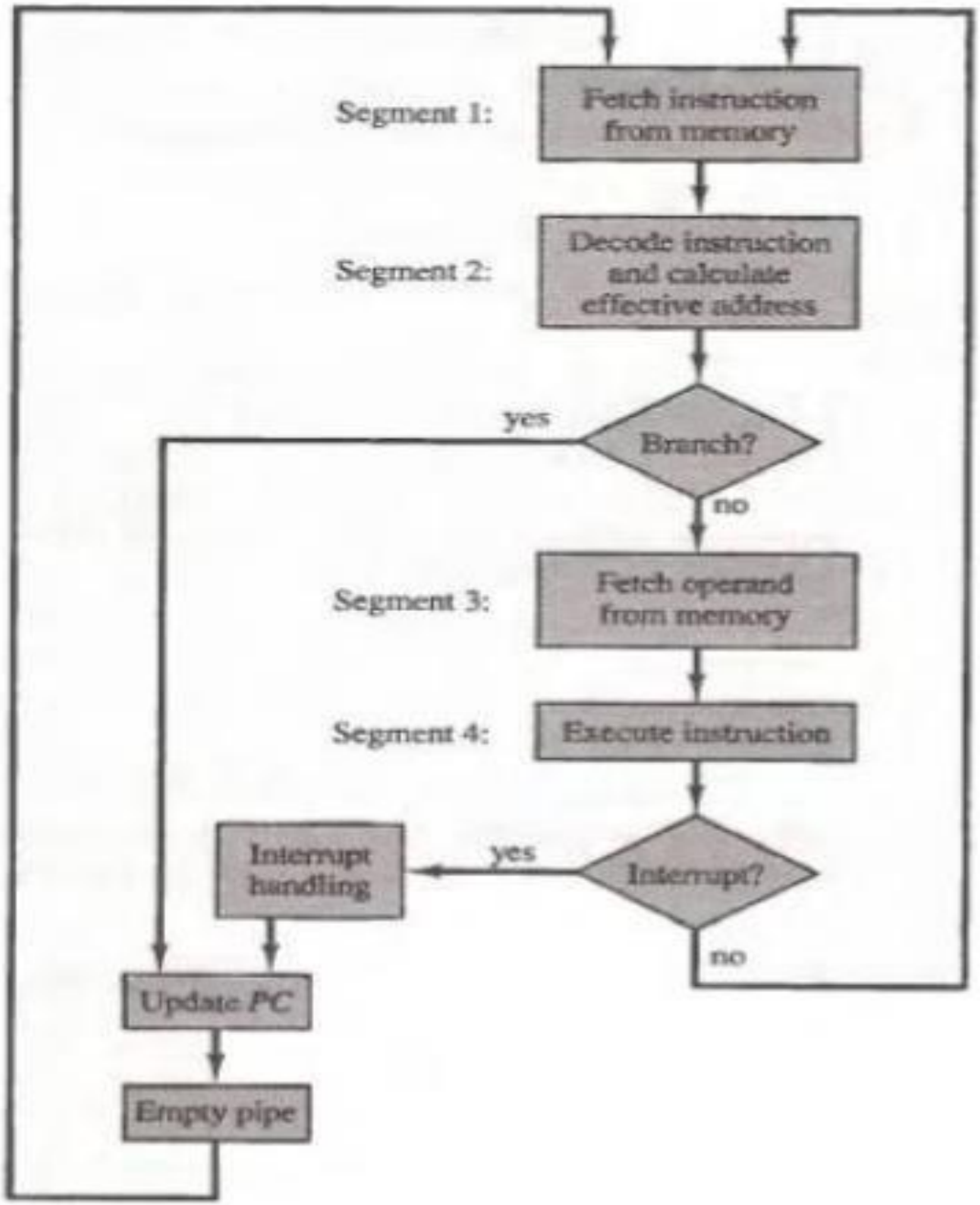
- Consider a case : k-segment pipeline with a clock time t_p to execute n tasks
- First task T1requires time equal to ***Ktp***
- After that the remaining (n - 1) results will come out at each clock cycle.
- It therefore takes k + (n - 1) clock cycles to complete n tasks using k-segment pipeline
- If a non pipeline unit performs the same operation and takes time equal to t_n to execute each task ; then total time required for n tasks is nt_n
- The speedup gained by using the pipeline is:

$$S = nt_n / (k + n - 1)t_p$$

Pipelining

Instruction Pipeline

- Fetch the instruction from memory
- Decode the instruction
- Calculate the effective address
- Fetch the operands from memory
- Execute the instruction
- Store the result in the proper place



Pipelining
Instruction Pipeline

Consider the timing of instruction pipeline as shown adjoining

FI : Instruction Fetch
DA : Decode Instruction &calculate EA
FO : Operand Fetch
EX : Execution

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

Pipeline Conflicts : 3 major difficulties
- Resource conflicts : memory access by two segments at the same time
- Data dependency :when an instruction depend on the result of a previous instruction, but this result is not yet available
- Branch difficulties :branch and other instruction (interrupt, ret, ..) that change the value of PC

Pipelining

RISC Pipeline

RISC CPU has Instruction Pipeline ;Single-cycle instruction execution

Compiler support

Here for an example 3 Suboperations Instruction Cycle is considered (show in fig) where I : Instruction fetch A : Instruction decoded and ALU operation E : Transfer the output of ALU to a register, memory, or PC

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

Pipeline timing with data conflict

Clock cycles	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

Pipeline timing with delayed load

RISC Pipeline

Pipelining

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

- There are several techniques for reducing branch penalties one of the method is ofdelayed branch
- The above figshows using no operation instruction ;

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

- The below figshows rearranging the instructions

Vector Processing

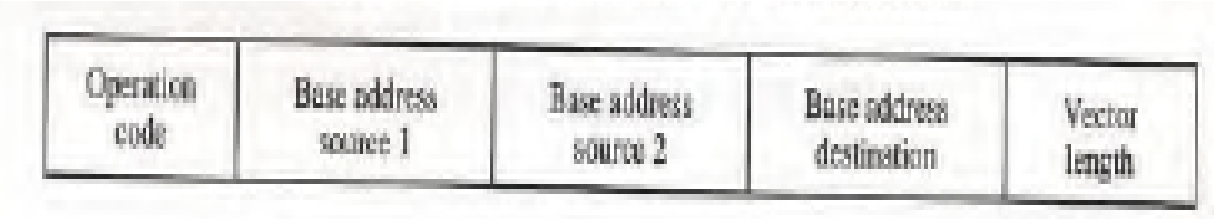
Computers with vector processing capabilities are in demand in specialized applications. The following are representative application areas where vector processing is of the utmost importance.

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

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 = [Vj V2 V3 •• Vn]

Vector Processing

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, aU in one composite instruction.
A possible instruction format for a vector instruction is shown in Fig

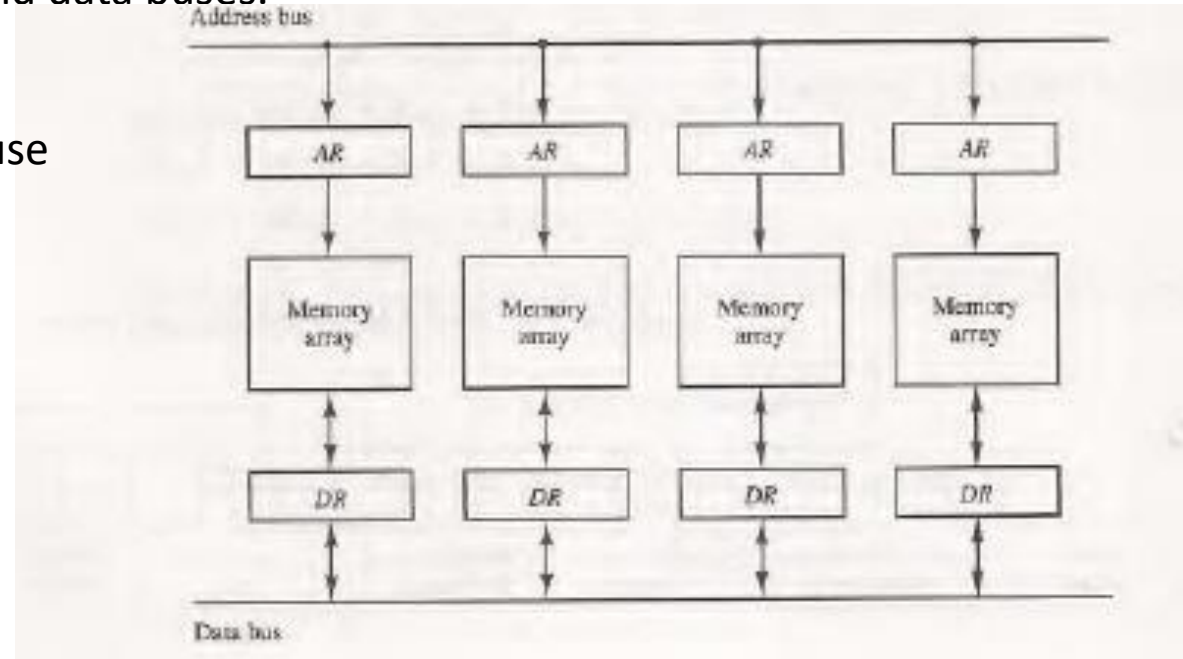


Vector Processing

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. Similarly, 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.

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.

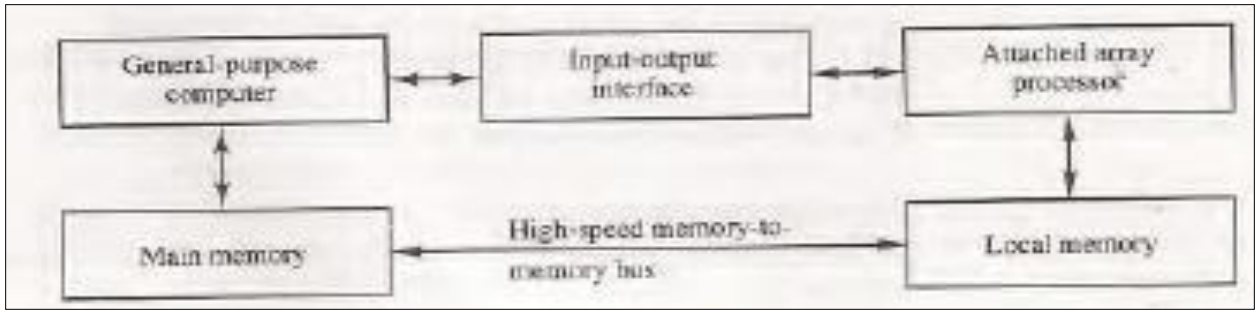


AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

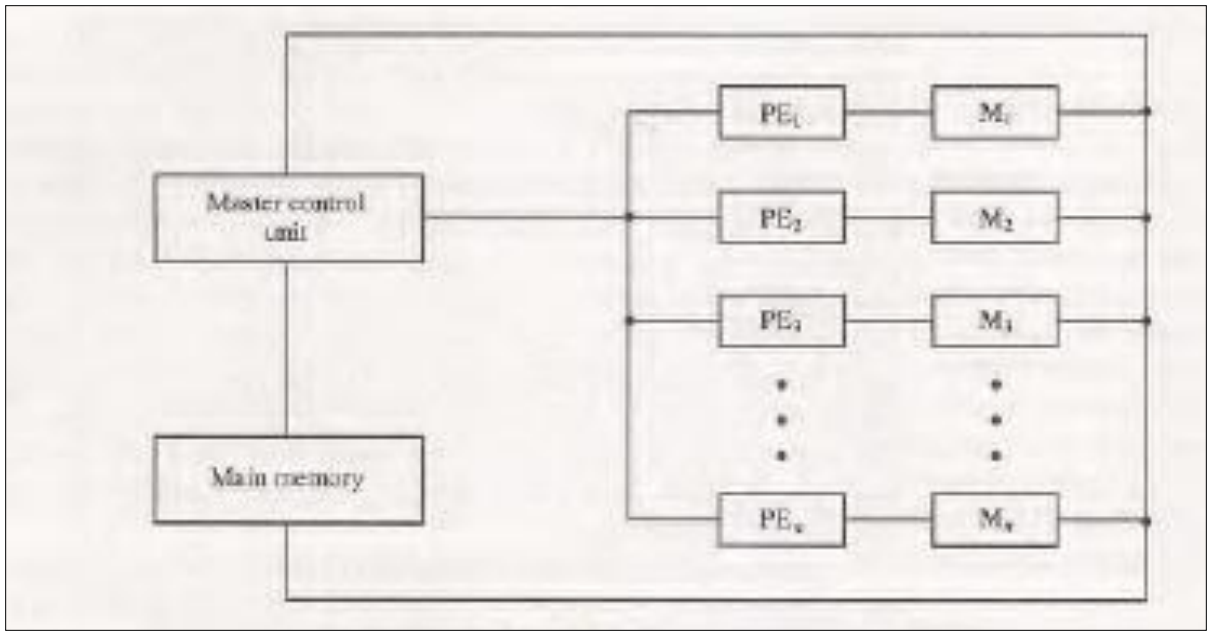
Array Processors

- An array processor is a processor that performs computations on large arrays of data..
- 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.
- Although both types of array processors manipulate vectors, their internal organization is different.

Array Processors



Attached array processor with host computer.



SIMD array processor organization.

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

References

- **Images , descriptive Tables , from Computer System Architecture, Morris Mano, 3rd edition Prentice Hall**

- **Note: These pdf/ppt notes are for purpose of teaching aids to classroom/online sessions study, and in no case imply for GTU syllabus or GTU exam. For GTU syllabus or exam related preparation, one may, however will need to attend college/online lectures and refer books given by GTU in their official syllabus.**