<u>CHAPTER : 4</u>
PIPELINE AND VECTOR PROCESSING

4.5 RISC Pipeline

- RISC – Reduced Instruction Set Computer
- Efficient instruction pipeline
- Implements an Instruction Pipeline using a small number of suboperations, each being executed in one clock cycle
- Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection
- Almost all operations are register-to-register operations

4.5 RISC Pipeline…

- Since all operands are in registers, no need of extra segment for calculating effective address and fetching operands from memory
- Hence 2 or 3 segments pipeline can be implemented
  - First segment fetches instructions from program memory
  - Second segment executes instruction in ALU
  - Third Segment can be used to store the result of the ALU in a destination register

4.5 RISC Pipeline…

- **The data transfer instructions in RISC are limited to load and store instructions.**
  - Use register indirect addressing.
  - Usually need three or four stages in the pipeline.
  - To prevent conflicts between memory access, two separate buses are used along with two memories.
  - Cache memory operates at the same speed as the CPU clock
- **Major advantages is the ability to execute instructions at the rate of one per clock cycle.**
  - May not be able to do both fetch and execution in a single cycle
  - Hence we start each instruction with each clock cycle and pipeline them to achieve the goal
  - RISC can achieve pipeline segments, requiring just one clock cycle.

4.5 RISC Pipeline…

- *Compiler supported Pipeline* that translates the high-level language program into machine language program
  - Use of Compiler replaces the use of unnecessary hardware
  - RISC processors rely on the efficiency of the compiler to detect and minimize the delays

4.5 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 address evaluation

4.5 RISC Pipeline - Hardware operation

- The *control section* fetches the instruction from program memory into an instruction register.
  - The instruction decoding and the registers selection for execution of the instruction are done at the same time
- The processor unit consists of a number of registers and an arithmetic logic unit (ALU).
- A data memory is used to load or store the data from a selected register in the register file.
- The instruction cycle can be divided into three suboperations and implemented in three segments:

4.5 RISC Pipeline - Hardware operation…

- I: Instruction fetch
  - Fetches the instruction from program memory
- A: ALU operation
  - The instruction is decoded and any one of the three ALU operations is performed:
    1. Operation for a data manipulation instruction.
    2. Evaluation of the effective address for a load or store instruction.
    3. Calculation of the branch address for a program control instruction.
- E: Execute instruction
  - Directs the output of the ALU to one of three destinations:
    1. It transfers the result of the ALU operation into a destination register in the register file.
    2. It transfers the effective address to a data memory for loading or storing.
    3. It transfers the branch address to the program counter.

4.5 RISC Pipeline - Delayed Load

- Consider the operation of the following four instructions:

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

- *Data Conflict* in instruction 3 - the operand in R2 is not yet available in the 'A' segment.
- This can be seen from the timing of the pipeline as shown below in fig (a):
  - The E from instruction 2 is not completed but instruction 3 is trying to execute A
- It is up to the *compiler* to make sure that the conflict does not occur

4.5 RISC Pipeline - Delayed Load…

- The Compiler either re-order the instructions or insert a no-op (no-operation) instruction (in this case no-op)
- This concept of wasting clock cycle without confliction and delaying the use of the data is referred to as *delayed load*.
- Fig. (b) shows the same program with a no-op instruction inserted after the load to R2 instruction.
- Thus the *no-op instruction* is used to advance one clock cycle in order to compensate for the *data conflict* in the pipeline.
- The advantage is due to the use of *compiler rather than the hardware.*

4.5 RISC PIPELINE – 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

(a) Pipeline timing with data conflict

	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

(b) Pipeline timing with delayed load

Fig: Three-segment pipeline timing

4.5 RISC Pipeline - Delayed Branch

- Relies on the *compiler to redefine the branches*
- Compiler makes sure that they take effect at the proper time in the pipeline
- This method of handling Branch Conflict is referred to as *delayed branch*.
- Compiler analyzes the instructions *before and after the branch*
- *Rearrange the program sequence* by inserting useful instructions in the delay steps.
- Useful instructions to put after the branch instruction are found
- If not found, then the compiler can insert *no-op* instructions.

4.5 RISC PIEPLINE - Delayed Branch…

- Consider an example with five instructions that causes branch conflict:

  Load from memory to R1
  Increment R2
  Add R3 to R4
  Subtract R5 from R6
  Branch to address X

- Fig (a) below shows the insertion of *two no-op instructions* after the branch.
- Fig. (b) shows the rearrangement by placing the add and subtract instructions *after the branch instruction*.

4.5 RISC PIPELINE - 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

(a) Using no-operation instructions

4.5 RISC PIPELINE - Delayed branch…

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

4.6 Vector Processing

- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices
- These calculation for conventional computers may take very long time or beyond their capacity
- Thus the solution for such problems lends themselves to vector processing.
- Some of the major application areas of Vector Processing are:
  - Long-range weather forecasting
  - Petroleum explorations
  - Seismic data analysis
  - Medical diagnosis
  - Artificial intelligence and expert systems
  - Image processing
  - Mapping the human genome

4.6 Vector Processing…

- Since the normal computers cannot solve the required computations within the time,
- It is necessary to utilize the *fastest and most reliable hardware* and
- apply innovative procedures from *vector and parallel processing techniques*

4.6 Vector Processing - Vector Operations

- Many scientific problems require arithmetic operations on large arrays of numbers.
- These numbers generally are Vectors or 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=[V1 V2 … Vn]
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

4.6 Vector Processing - Vector Operations…

- This is implemented in machine language as:

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

- Vector processing eliminates the overhead by specifying a single vector operation as:

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

4.6 Vector Processing - Vector Operations…

- A possible instruction format for a vector instruction is shown in Fig below:
  - This assumes that the vector operands reside in *memory*.
  - A processor with a large number of *registers can* also be designed where all operands are stored before the addition operation
  - In second case, base address and length in the vector instruction specify a group of CPU registers.

4.6 Vector Processing - Vector Operations...

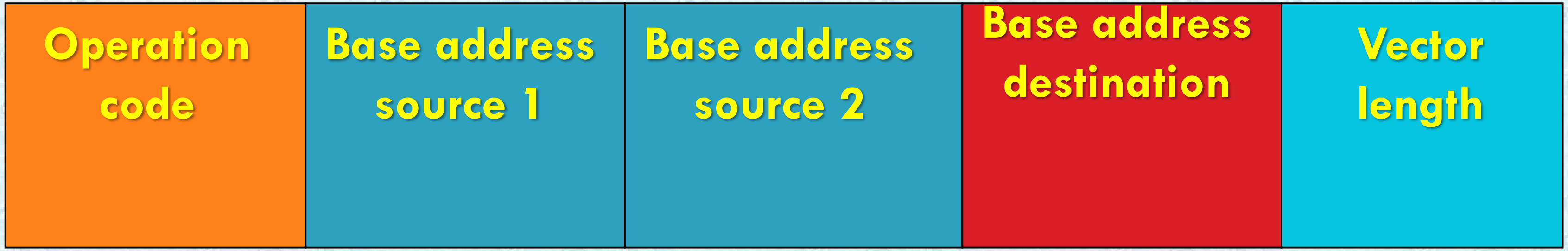


Fig: Instruction format for vector processor

4.6 Vector Processing - Matrix Multiplication

- A single matrix of order m×n can be considered as a set of m row vectors or n column vectors
- The multiplication of two n x n matrices consists of n^2 inner products or n^3 multiply-add operations.
  - Consider, for example, the multiplication of two 3 x 3 matrices A and B.
  - $c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$
  - This requires three multiplication and (after initializing c_{11} to 0) three additions.
- In general, the inner product consists of the sum of *k* product terms of the form $C = A_1B_1 + A_2B_2 + A_3B_3 + \ldots + A_kB_k$.
  - 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 below:

4.6 Vector Processing - Matrix Multiplication...

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

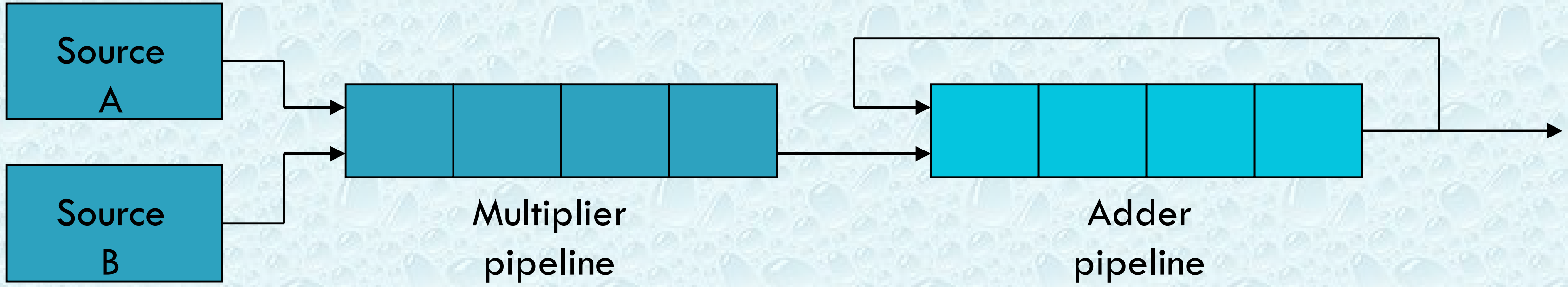
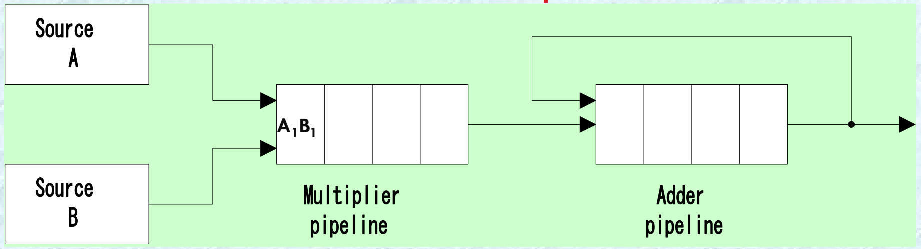


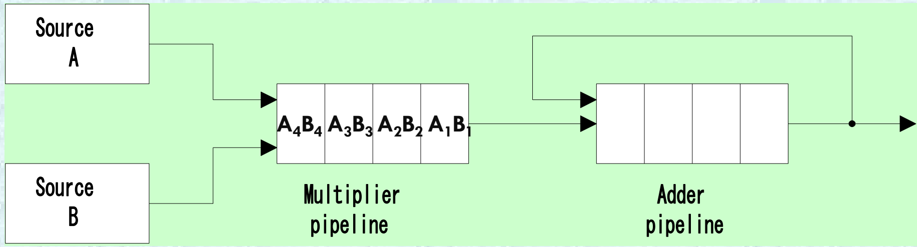
Fig: Pipeline for calculating an inner product

4.6 Vector Processing - Matrix Multiplication…

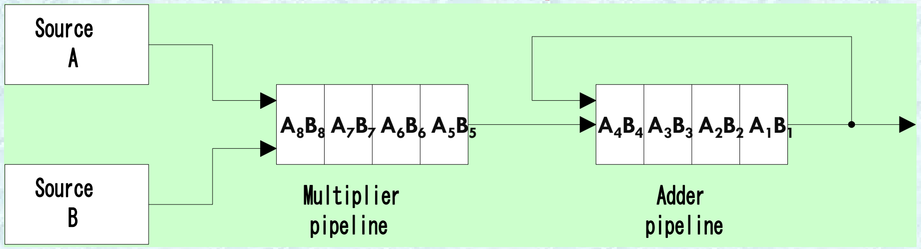
» after 1st clock input



» after 4th clock input

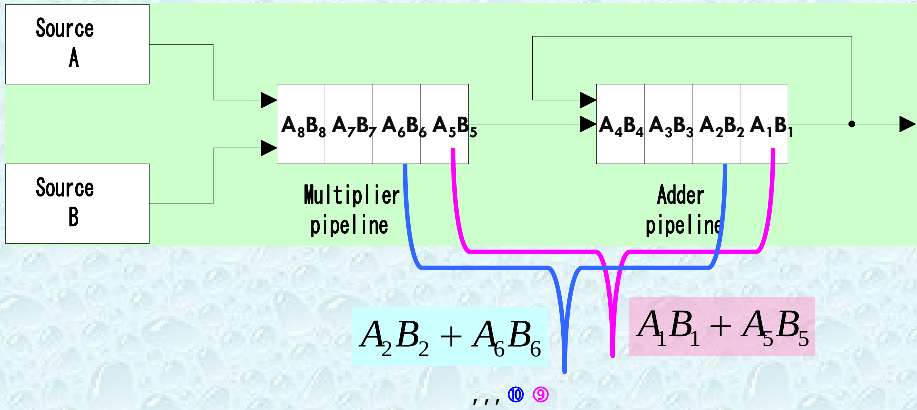


» after 8th clock input



Source A
Source B
A_8B_8 A_7B_7 A_6B_6 A_5B_5
A_4B_4 A_3B_3 A_2B_2 A_1B_1
Multiplier pipeline
Adder pipeline

» after 9th, 10th, 11th ,…



Compiled By: Shiva Raj Luitel

4.6 Vector Processing - 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 below shows a memory unit with four modules.
- The use of memory modules helps in permitting the access of one memory while other processes are busy on writing or reading a word from other memories

4.6 Vector Processing - Memory Interleaving...

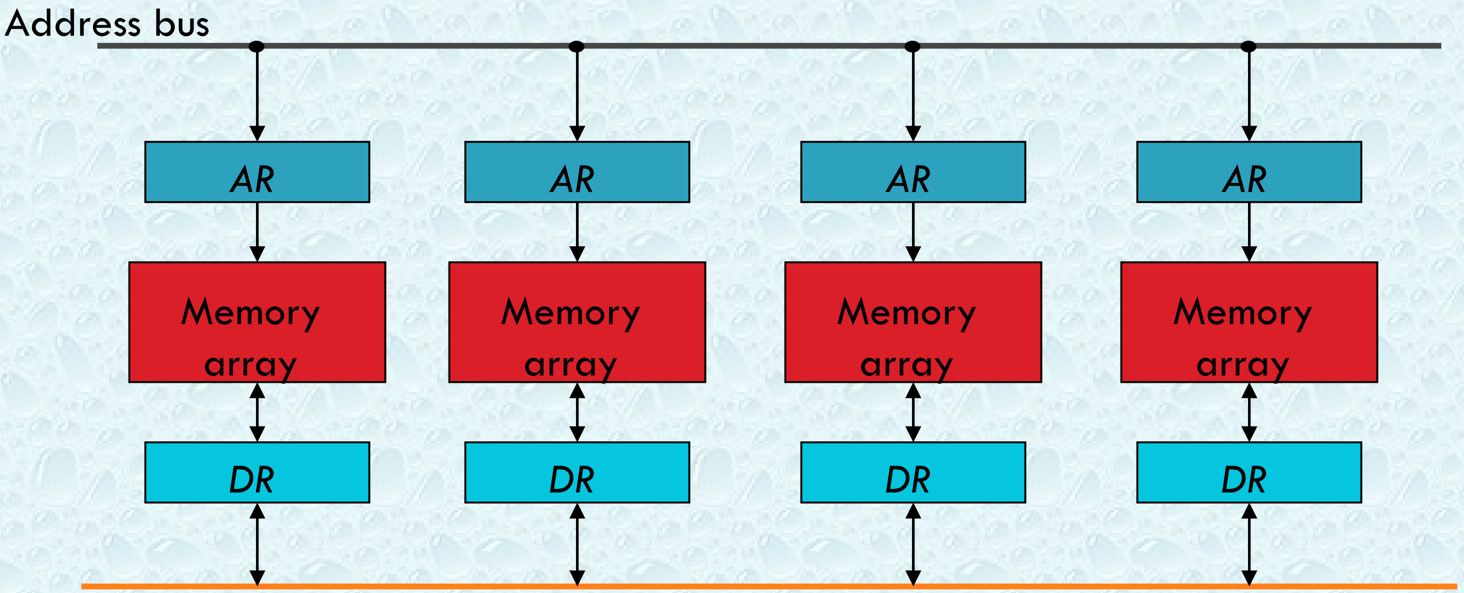


Fig: Multiple module memory organization

4.6 Vector Processing - Memory Interleaving…

- 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*
- The LSBs of address info are used to select the memory modules and the remaining bits selects the specific location within the module

4.6 Vector Processing - Supercomputers

- A commercial computer with vector instructions and pipelined floating-point arithmetic operations
- To speed up the operation, the components are *packed tightly* together to minimize the distance that the electronic signals have to travel.
- Instructions are similar to conventional computer instructions but are also capable of processing vectors and combinations of scalars and vectors.
- high computational speed, fast and large memory systems, and the extensive use of parallel processing.
  - equipped with *multiple functional units*
  - each unit has its own *pipeline* configuration.

4.6 Vector Processing – Supercomputers..

- It is specifically optimized for the type of numerical calculations involving vectors and matrices of floating-point numbers.
- Specific Purpose and used in
  - scientific applications, such *as numerical weather forecasting, seismic wave analysis, and space research.*
- Speed is measured in number of floating-point operations per second referred to as *flops*.
- A typical supercomputer has a basic cycle time of 4 to 20 ns.
- If one floating-point operation through one clock cycle time, then the speed can range between 50 to 250 megaflops
- The examples of supercomputer:

4.6 Vector Processing – Supercomputers..

- Some Examples of Supercomputers are:
  - Cray-1:
    - it uses vector processing with 12 distinct functional units in parallel
    - a large number of registers (over 150)
    - Speed of up to 80 megaflops
    - multiprocessor configuration (Cray X-MP and Cray Y-MP)
    - Cray – 2 is 12 times faster than Cray - 1
  - Fujitsu VP-200:
    - 83 vector instructions
    - 195 scalar instructions
    - Speed of up to 300 megaflops
    - The new VP-2600 has the speed of up to 5 gigaflops

4.7 Array Processors

- a processor that performs computations on large arrays of data.
- The term is used to refer to two different types of processors.
  - Attached array processor:
    - Is an auxiliary processor
    - Attached to general-purpose computers
    - 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.

4.7 Array Processors - Attached Array Processor

- Its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
  - Parallel processing is achieved by implementing multiple functional units
- Fig below shows the interconnection of an attached array processor to a host computer.
- The attached array processor is connected to the general computer through an input-output controller
- The local memory of the processor is connected to main memory of general computer through high speed bus
- The objective is to provide *vector manipulation capabilities* to a conventional computer at very cheap price than that of supercomputer.

4.7 Array Processors - Attached Array Processor..

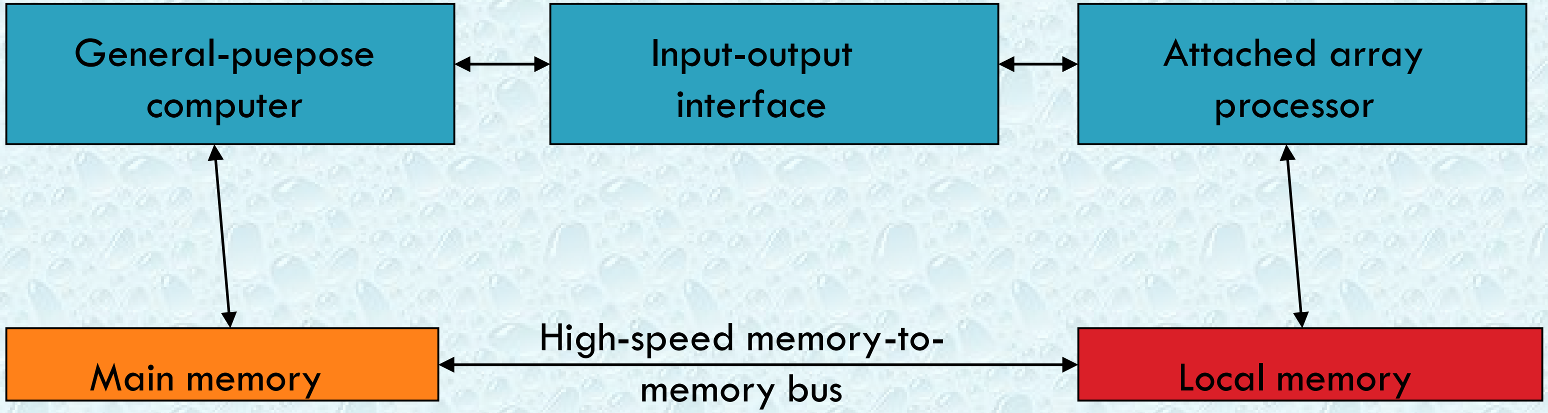


Fig: Attached array processor with host computer

4.7 Array Processors - SIMD Array Processor

- computer with multiple processing units operating in parallel.
- A general block diagram of an array processor is shown in Fig below
  - It contains a set of identical processing elements (PUs), each having a local memory LM.
  - Each PU includes an ALU, a floating-point arithmetic unit, and working registers.
  - Vector instructions are broadcast to all PUs simultaneously from Master CU.
  - All PUs are synchronized to perform same operation
  - Masking schemes are used to control the status of each PU during the execution of vector instructions.
  - Each PU has a flag that is set when the PU is active and reset when the PU is inactive.

4.7 Array Processors - SIMD Array Processor..

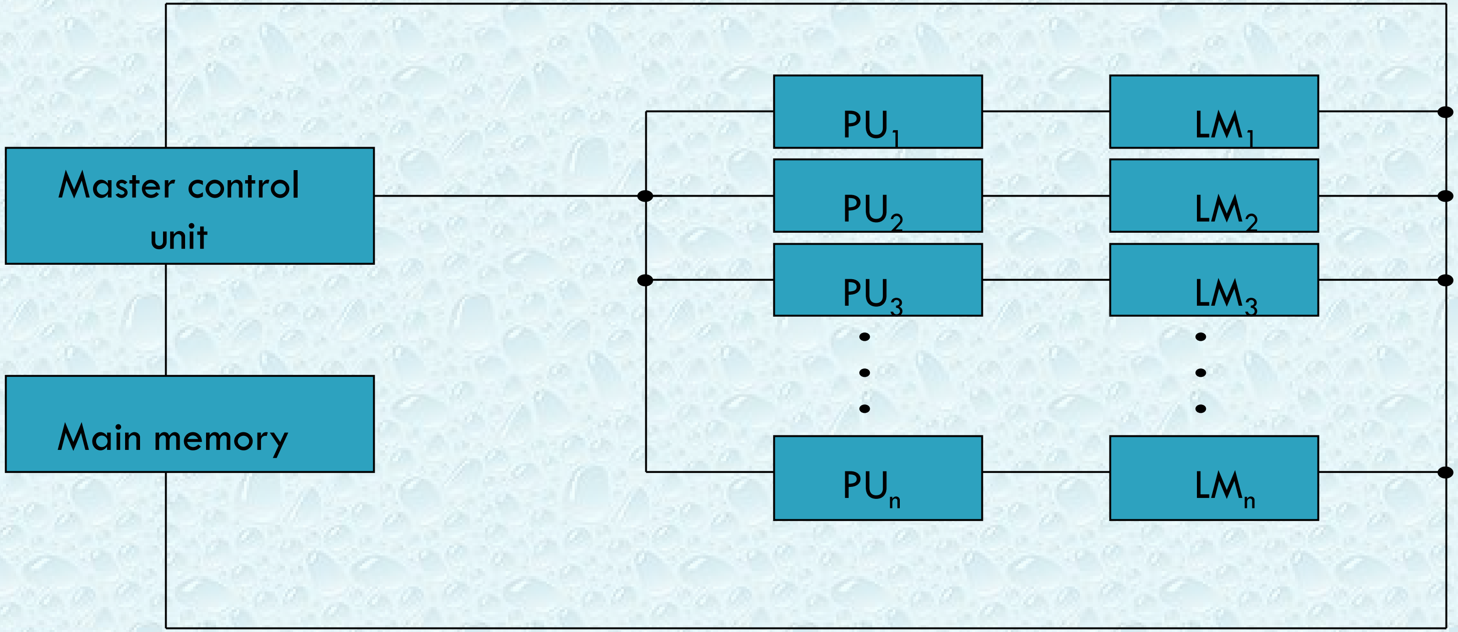


Fig: SIMD array processor organization

4.7 Array Processors - SIMD Array Processor..

- Example of SIMD Array Processor
  - ILLIAC IV computer
    - Developed at the University of Illinois
    - Manufactured by the Burroughs Corp.
    - Are highly specialized computers.
- SIMD Processor are suited primarily for numerical problems that can be expressed in vector or matrix form.