# Unit # 5

**Pipeline and Vector Processing**

#### Dr. Rajesh Tiwari Professor ( CSE – AIML) CMREC, Hyderabad, Telangana

**Reduced Instruction Set Computer(RISC)**

* The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, store command will store the data.
* **RISC:** Reduce the cycles per instruction at the cost of the

number of instructions per program.

* **Characteristic of RISC –**
  + Simpler instruction, hence simple instruction decoding.
  + Instruction comes undersize of one word.
  + Instruction takes a single clock cycle to get executed.
  + More general-purpose registers.
  + Simple Addressing Modes.
  + Less Data types.
  + Pipeline can be achieved.

**Complex Instruction Set Computer (CISC)**

* The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it’s complex.
* The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.
* **Characteristic of CISC –**
  + Complex instruction, hence complex instruction decoding.
  + Instructions are larger than one-word size.
  + Instruction may take more than a single clock cycle to get executed.
  + Less number of general-purpose registers as operation get performed in memory itself.
  + Complex Addressing Modes.
  + More Data types.
* **Example –** Suppose we have to add two 8-bit number:
  + **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
  + **RISC approach:** Here programmer will write the first load command to load data in registers then it will use a suitable operator and then it will store the result in the desired location.

**Difference b/w RISC & CISC**

|  |  |
| --- | --- |
| **RISC** | **CISC** |
| Focus on software | Focus on hardware |
| Uses only Hardwired control unit | Uses both hardwired and micro programmed control unit |
| Transistors are used for more registers | Transistors are used for storing complex  Instructions |
| Fixed sized instructions | Variable sized instructions |

|  |  |
| --- | --- |
| **RISC** | **CISC** |
| Can perform only Register to Register Arithmetic operations | Can perform REG to REG or REG to MEM or MEM to MEM |
| Requires more number of registers | Requires less number of registers |
| Code size is large | Code size is small |
| An instruction execute in a single clock cycle | Instruction takes more than one clock cycle |
| An instruction fit in one word | Instructions are larger than the  size of one word |

### Parallel processing

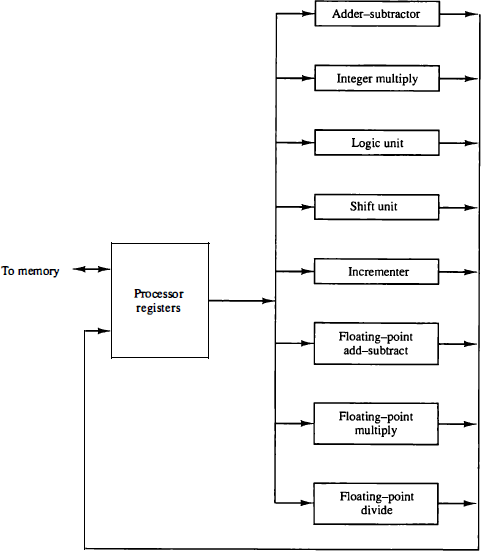
* Parallel processing is used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of incensing the computational speed of a computer system.
* Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system i.e. able to perform concurrent data processing to achieve faster execution time.
* For example, while an instruction is being executed in the ALU, the next instruction can be read from memory.
* The system may have two or more ALUs and be able to execute

two or more instructions at the same time.

* The system may have two or more processors operating concurrently.
* The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, i.e, the amount of processing that can be accomplished during a given interval of time.
* The amount of hardware increases with parallel processing, and with it, the cost of the system increases.
* Technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.
* Figure 5.1 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.



**Figure 5.1: Processor with multiple functional units.**

# Parallel processing

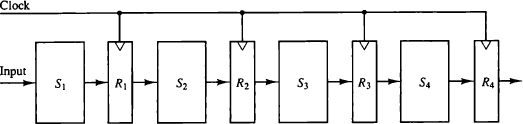
* There are a variety of ways that parallel processing can be classified.
* One classification introduced by M. J. Flynn considers the
* organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
* The normal operation of a computer is to fetch instructions from memory and execute them in the processor.
* The sequence of instructions read from memory constitutes an instruction stream .
* The operations performed on the data in the processor constitutes a data stream .
* Parallel processing may occur in the instruction stream, in the data stream, or in both.
* **Flynn's classification** divides computers into four major groups as follows:
  + Single instruction stream, single data stream (SISD)
  + Single instruction stream, multiple data stream (SIMD)
  + Multiple instruction stream, single data stream (MISD)
  + Multiple instruction stream, multiple data stream (MIMD)

### 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.

* Each segment performs partial processing dictated by the way the task is partitioned.
* The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
* The final result is obtained after the data have passed through all segments.
* The overlapping of computation is made possible by associating a register with each segment in the pipeline.
* The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

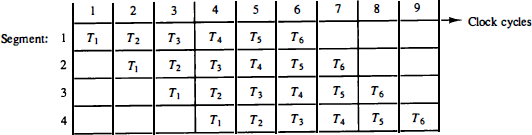


**Figure 5.2: Four-segment pipeline**

* The general structure of a four-segment pipeline is shown in Fig. 5.2.
* The operands pass through all four segments in a fixed

sequence.

* Each segment consists of a combinational circuit S; that performs a sub-operation over the data stream flowing through the pipe.
* The segments are separated by registers R; that hold the intermediate results between the stages.
* Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
* We define a task as the total operation performed going through all the segments in the pipeline.



**Figure 5.3: Space-time diagram for pipeline**

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

Fig. 5.3.

* The horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.
* The diagram shows six tasks T1 through T6 executed in four segments.
* Initially, task T1 is handled by segment 1.
* After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2.
* Continuing in this manner, the first task T1 is completed after the fourth clock cycle.
* From then on, the pipe completes a task every clock cycle.
* No matter how many segments there are in the system, once the pipeline is full, it takes only one clock period to obtain an output.
  + Now consider the case where a k-segment pipeline with a clock cycle time tp is used to execute n tasks.
  + The first task T1 requires a time equal to ktp to complete its operation since there are k segments in the pipe.
  + The remaining n - 1 tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n - 1) tp .
  + To complete n tasks using a k-segment pipeline requires k + (n - 1) clock cycles.
  + For example, the diagram of Fig. 5.3 shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.
  + Consider a non-pipeline 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 non-

pipeline processing is defined by the ratio



**Pipelining**

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



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



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

#### Arithmetic Pipeline

* + Arithmetic pipeline 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.
  + A pipeline multiplier is essentially an array multiplier with special adders designed to minimize the carry propagation time through the partial products.
  + Floating-point operations are easily decomposed into sub- operations.
  + Now take an example of a pipeline unit for floating-point

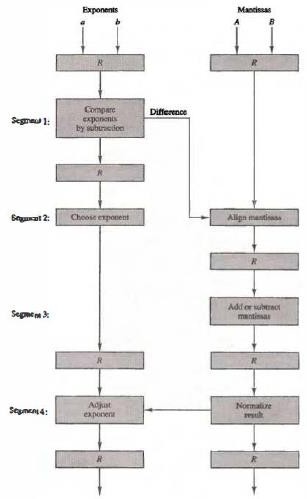
addition and subtraction.

## Arithmetic Pipeline

* + The floating-point addition and subtraction can be performed in four segments, as shown in Fig. 5.4.
  + The registers labeled R are placed between the segments to store

intermediate results.

* + The sub-operations that are performed in the four segments are:
    - Compare the exponents.
    - Align the mantissas.
    - Add or subtract the mantissas.
    - Normalize the result.



**Fig. 5.4: Pipeline for floating point addition and subtraction**

# Arithmetic Pipeline

* + The comparator, shifter, adder-subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits.
  + Suppose that the time delays of the four segments are t1 = 60 ns, t2 = 70 ns, t3 = 100 ns, t4 = 80 ns, and the interface registers have a delay of tr, = 10 ns.
  + The clock cycle is chosen to be tp = t3 + tr = 110 ns.
  + An equivalent non-pipeline floating point adder-subtractor will have a delay time tn = t1 + t2 + t3 + t4 + tr = 320 ns.
  + In this case the pipelined adder has a speedup of 320/1 10 = 2. 9 over the non pipelined adder.

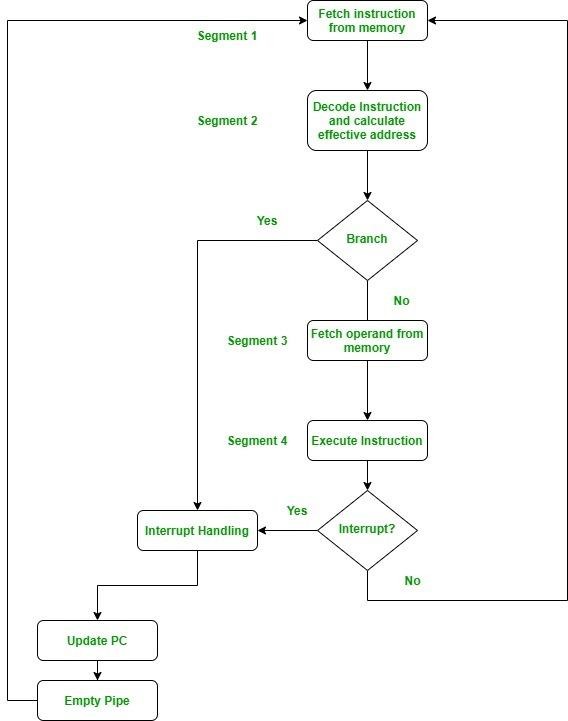
#### Instruction Pipeline

* An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
* This causes the instruction fetch and execute 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.
  + In general case, the computer needs to process each instruction

with the following sequence of steps.

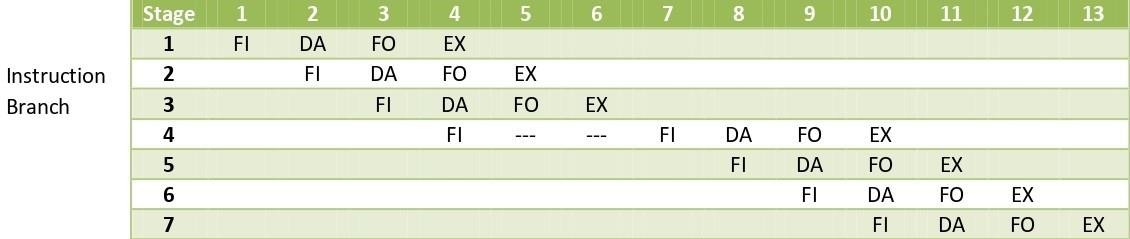
* + - Fetch the instruction from memory.
    - Decode the instruction.
    - Calculate the effective address.
    - Fetch the operands from memory.
    - Execute the instruction.
    - Store the result in the proper place.
* In instruction pipeline a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle.
* This type of technique is used to increase the throughput of the computer system.
* An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline.
* Thus we can execute multiple instructions simultaneously.
* The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.
* In the most case computer needs to process each instruction in following sequence of steps:
  + Fetch the instruction from memory (FI)
  + Decode the instruction (DA)
  + Calculate the effective address
  + Fetch the operands from memory (FO)
  + Execute the instruction (EX)
  + Store the result in the proper place

The flowchart for instruction pipeline is shown below.



**Instruction Pipeline**

* Here the instruction is fetched on first clock cycle in segment 1.
* Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed.
* We can see that here the fetch and decode phase overlap due to pipelining.
* By the time the first instruction is being decoded, next instruction is fetched by the pipeline.



* In case of third instruction we see that it is a branched instruction.
* Here when it is being decoded 4th instruction is fetched simultaneously.
* But as it is a branched instruction it may point to some other instruction when it is decoded.
* Thus fourth instruction is kept on hold until the branched instruction is executed.
* When it gets executed then the fourth instruction is copied

back and the other phases continue as usual.

#### RISC Pipeline

* In the early days of computer hardware, Reduced Instruction Set Computer Central Processing Units (RISC CPUs) was designed to execute one instruction per cycle, five stages in total.
* Those stages are, Fetch, Decode, Execute, Memory, and Write.
* The simplicity of operations performed allows every instruction to

be completed in one processor cycle.

##### Fetch

* + In the Fetch stage, instruction is being fetched from the memory.

##### Decode

* + During the Decode stage, we decode the instruction and fetch the source

operands

##### Execute

* + During the execute stage, the computer performs the operation specified by the instruction

##### Memory

* + If there is any data that needs to be accessed, it is done in the memory stage

##### Write

* + If we need to store the result in the destination location, it is done during the writeback stage,

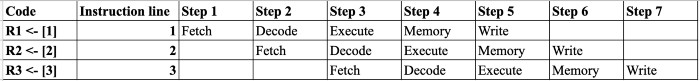
##### Example

* Suppose we have the following 3 lines of code:

R1 <- [1]

R2 <- [2]

R3 <- [3]

* In the code above, we are performing three load types.
* In line one, we are storing the address 1 to R1,
* line 2, we are storing address of 2 to R2 and
* finally in line 3, we are storing the address 3 to R3.
  + The RISC Pipeline will look something like this:
  + We know that Load Types execute all 5 stages of the RISC pipeline which again are, fetch, decode, execute, memory, and write.

##### The image above shows how the example three line code of all

**load types will execute.**

* + In step 1, the first line will execute the first step, which fetches.
  + Then in step 2, while line 1 is in the decode phase, line two will

start fetching, and so on.

* + The 3 lines of code will need to go through seven steps in order to complete all RISC pipeline for all three lines.
* Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction.
* More specifically we can say, it is a complete unit of hardware resources that executes a sequential set of similar data items in the memory using a single instruction.
* We know elements of the vector are ordered properly so as to have successive addressing format of the memory.
* This is the reason why we have mentioned that **it implements the**

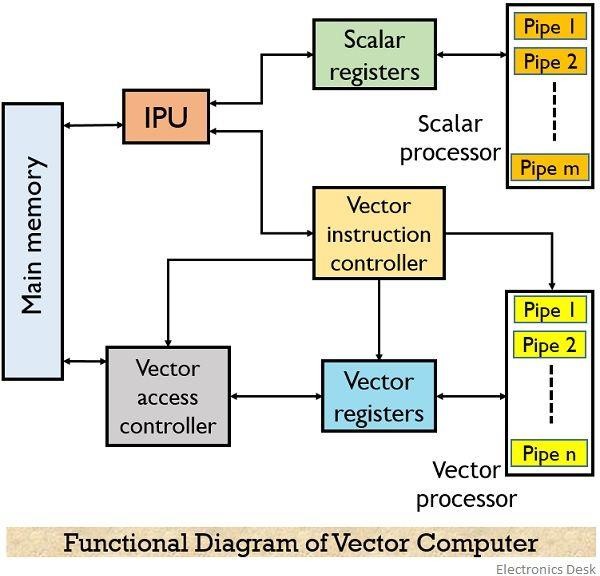
**data sequentially**.

* It holds a single control unit but has multiple execution units that perform the same operation on different data elements of the vector.
* Unlike scalar processors that operate on only a single pair of data, a vector processor operates on multiple pair of data.
* However, one can convert a scalar code into vector code. This conversion process is known as **vectorization**.
* We can say vector processing allows operation on multiple data elements by the help of single instruction.
* These instructions are said to be **single instruction multiple**

##### data or vector instructions.

* The CPU used in recent time makes use of vector processing as it is advantageous than scalar processing.

### VECTOR PROCESSING

* The figure below represents the typical diagram showing vector processing by a vector computer:

#### VECTOR PROCESSING

* + The functional units of a vector computer are as follows:
    - IPU or instruction processing unit
    - Vector register
    - Scalar register
    - Scalar processor
    - Vector instruction controller
    - Vector access controller
    - Vector processor

### VECTOR PROCESSING

* It has several functional pipes thus it can execute the instructions over the operands.
* We know that both data and instructions are present in the memory

at the desired memory location.

* So, the instruction processing unit i.e., IPU fetches the instruction from the memory.
* Once the instruction is fetched then IPU determines either the fetched instruction is scalar or vector in nature. If it is scalar in nature, then the instruction is transferred to the scalar register and then further scalar processing is performed.
* When the **instruction is a vector** in nature then it is fed to the vector instruction controller.
* This vector instruction controller first decodes the vector instruction then accordingly determines the address of the vector operand present in the memory.
* Then it gives a signal to the **vector access controller** about the demand of the respective operand.
* This vector access controller then fetches the desired operand from the memory. Once the operand is fetched then it is provided to the instruction register so that it can be processed at the vector processor.
* At times when multiple vector instructions are present, then the vector instruction controller provides the multiple vector instructions to the task system.
* And in case the task system shows that the vector task is very long then the processor divides the task into **subvectors**.
* These subvectors are fed to the vector processor that makes use of several pipelines in order to execute the instruction over the operand fetched from the memory at the same time.
* The various vector instructions are scheduled by the vector instruction controller.

### VECTOR PROCESSING

* Vector Processing Applications
  + Problems that can be efficiently formulated in terms of vectors
    - Long-range weather forecasting
    - Petroleum explorations
    - Seismic data analysis
    - Medical diagnosis
    - Aerodynamics and space flight simulations
    - Artificial intelligence and expert systems
    - Mapping the human genome
    - Image processing
* Vector Processor (computer)
  + Ability to process vectors, and related data structures such as matrices and multi-dimensional arrays, much faster than conventional computers
  + Vector Processors may also be pipelined

### Array processors

* Array processors are also known as multiprocessors or vector processors.
* They perform computations on large arrays of data.
* They are used to improve the performance of the computer.

##### There are basically two types of array processors:

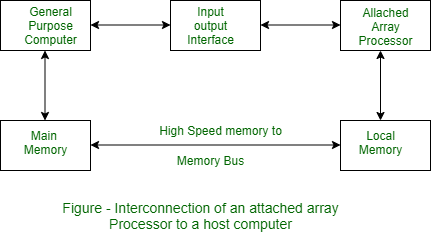
* + Attached Array Processors
  + SIMD Array Processors

**Attached Array Processor**

* To improve the performance of the host computer in numerical computational tasks auxiliary processor is attached to it.
* Attached array processor has two interfaces:
  + Input output interface to a common processor.
  + Interface with a local memory.
* Here local memory interconnects main memory.
* Host computer is general purpose computer.
* Attached processor is back end machine driven by the host computer.
* The array processor is connected through an I/O controller to

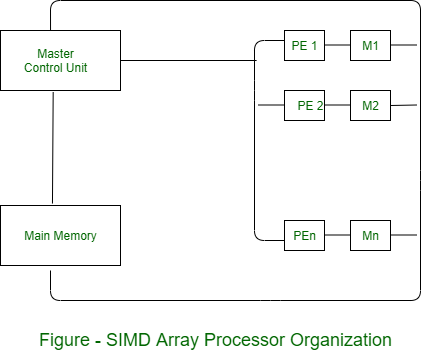
the computer & the computer treats it as an external interface.

### Attached Array Processor



#### SIMD array processor

* This is computer with multiple process unit operating in parallel Both types of array processors, manipulate vectors but their internal organization is different.



### SIMD array processor

* SIMD 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.
* As shown in figure, SIMD contains a set of identical processing elements (PES) each having a local memory M.

##### Each PE includes –

* + ALU
  + Floating point arithmetic unit
  + Working registers

**SIMD array processor**

* Master control unit controls the operation in the PEs.
* The function of master control unit is to decode the instruction and determine how the instruction to be executed.
* If the instruction is scalar or program control instruction then it is directly executed within the master control unit.
* Main memory is used for storage of the program while each PE uses operands stored in its local memory.

## Thank You