Pipelining and Vector Processing

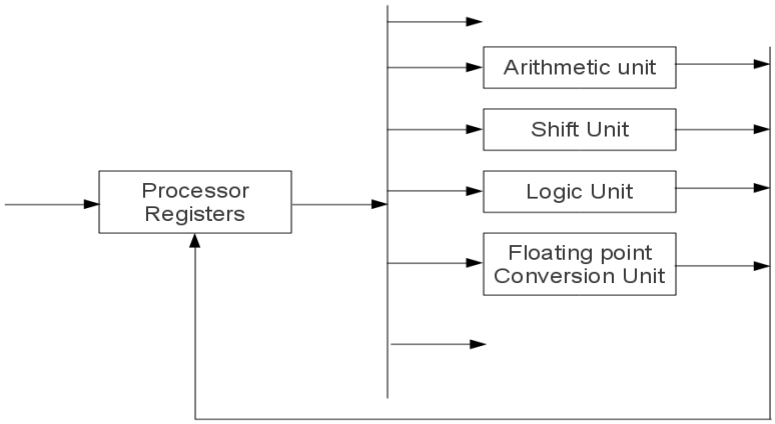
Parallel Processing:

The term parallel processing indicates that the system is able to perform several operations in a single time. Now we will elaborate the scenario, in a CPU we will be having only one Accumulator which will be storing the results obtained from the current operation. Now if we are giving only one command such that “a+b” then the CPU performs the operation and stores the result in the accumulator. Now we are talking about parallel processing, therefore we will be issuing two instructions “a+b” and “c-d” in the same time, now if the result of “a+b” operation is stored in the accumulator, then “c-d” result cannot be stored in the accumulator in the same time. Therefore the term parallel processing in not only based on the Arithmetic, logic or shift operations. The above problem can be solved in the following manner. Consider the registers R1 and R2 which will be storing the operands before operation and R3 is the register which will be storing the results after the operations. Now the above two instructions “a+b” and “c-d” will be done in parallel as follows.

- Values of “a” and “b” are fetched in to the registers R1 and R2
- The values of R1 and R2 will be sent into the ALU unit to perform the addition
- The result will be stored in the Accumulator
- When the ALU unit is performing the calculation, the next data “c” and “d” are brought into R1 and R2.
- Finally the value of Accumulator obtained from “a+b” will be transferred into the R3
- Next the values of C and D from R1 and R2 will be brought into the ALU to perform the “c-d” operation.
- Since the accumulator value of the previous operation is present in R3, the result of “c-d” can be safely stored in the Accumulator.

This is the process of parallel processing of only one CPU. Consider several such CPU performing the calculations separately. This is the concept of parallel processing.

Concept of Parallel Processing



In the above figure we can see that the data stored in the processor registers is being sent to separate devices basing on the operation needed on the data. If the data inside the processor registers is requesting for an arithmetic operation, then the data will be sent to the arithmetic unit and if in the same time another data is requested in the logic unit, then the data will be sent to logic unit for logical operations. Now in the same time both arithmetic operations and logical operations are executing in parallel. This is called as parallel processing.

Instruction Stream: The sequence of instructions read from the memory is called as an Instruction Stream

Data Stream: The operations performed on the data in the processor is called as a Data Stream.

The computers are classified into 4 types based on the Instruction Stream and Data Stream. They are called as the Flynn's Classification of computers.

Flynn's Classification of Computers:

- Single Instruction Stream and Single Data Stream (SISD)
- Single Instruction Stream and Multiple Data Stream (SIMD)
- Multiple Instruction Stream and Single Data Stream (MISD)
- Multiple Instruction Stream and Multiple Data Stream (MIMD)

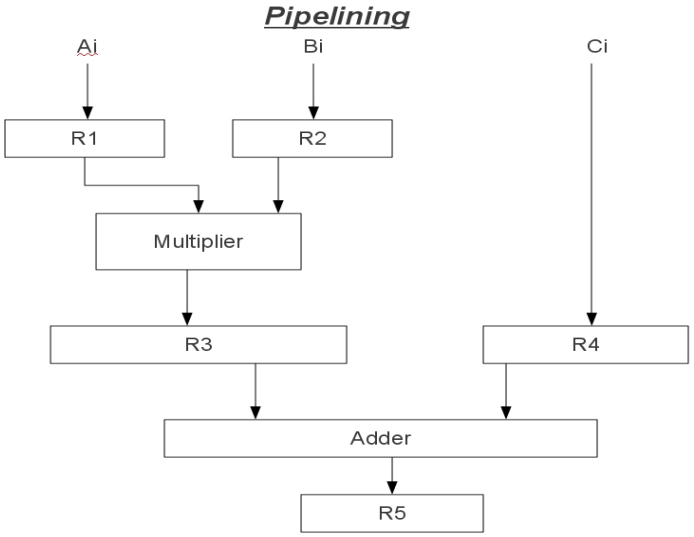
SISD represents the organization of a single computer containing a control unit, a processor unit and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

SIMD represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.
MISD structure is only of theoretical interest since no practical system has been constructed using this organization because Multiple instruction streams means more no of instructions, therefore we have to perform multiple instructions on same data at a time. This is practically impossible.
MIMD structure refers to a computer system capable of processing several programs at the same time operating on different data.

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. We can consider the pipelining concept as a collection of several segments of data processing programs which will be processing the data and sending the results to the next segment until the end of the processing is reached. We can visualize the concept of pipelining in the example below.
Consider the following operation: Result=(A+B)*C

- First the A and B values are Fetched which is nothing but a “Fetch Operation”.
- The result of the Fetch operations is given as input to the Addition operation, which is an Arithmetic operation.
- The result of the Arithmetic operation is again given to the Data operand C which is fetched from the memory and using another arithmetic operation which is Multiplication in this scenario is executed.
- Finally the Result is again stored in the “Result” variable.

In this process we are using up-to 5 pipelines which are the
→ Fetch Operation (A)| Fetch Operation(B) | Addition of (A & B) | Fetch Operation(C) | Multiplication of ((A+B), C) | Load ((A+B)*C), Result);



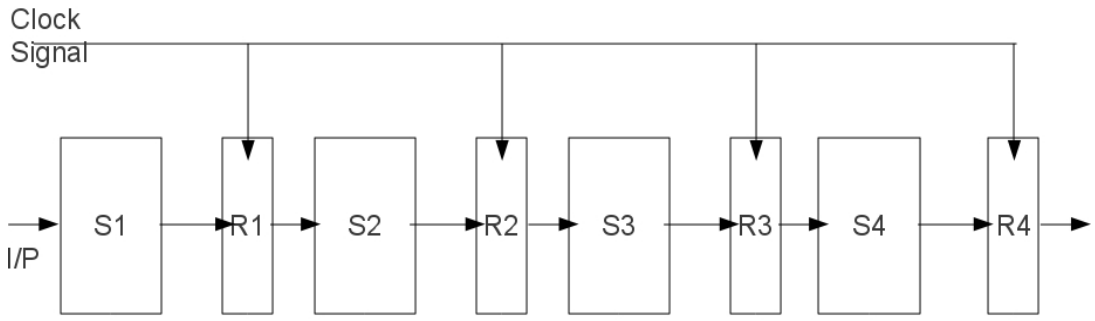
The contents of the Registers in the above pipeline concept are given below. We are considering the implementation of A[7] array with B[7] array.

Clock Pulse Number	Segment1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	-	-	-
2	A2	B2	A1*B1	C1	-
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	B6	A5*B5	C5	A4*B4+C4
7	A7	B7	A6*B6	C6	A5*B5+C5
8			A7*B7	C7	A6*B6+C6
9					A7*B7+C7

If the above concept is executed with out the pipelining, then each data operation will be taking 5 cycles, totally they are 35 cycles of CPU are needed to perform the operation. But if are using the concept of pipeline, we will be cutting off many cycles. Like given in the table below when the values of A1 and B1 are coming into the registers R1 and R2, the registers R3, R4 and R5 are empty. Now in the second cycle the multiplication of A1 and B1 is transferred to register R3, now in this point the contents of the register R1 and R2 are empty. Therefore the next two values A2 and B2 can be brought into the registers. Again in the third cycle after fetching the C1 value the operation (A1*B1)+C1 will be performed. So in this way we can achieve the total concept in only 9 cycles. Here we are assuming that the clock cycle timing is fixed. This is the concept of pipelining.

Below is the diagram of 4 segment pipeline.

Segment Representation



4 Segment pipeline

The below table is the space time diagram for the execution of 6 tasks in the 4 segment pipeline.

Space and Time Diagram

Seg/ clock	C1	C2	C3	C4	C5	C6	C7	C8	C9
S1	T1	T2	T3	T4	T5	T6			
S2		T1	T2	T3	T4	T5	T6		
S3			T1	T2	T3	T4	T5	T6	
S4				T1	T2	T3	T4	T5	T6

$$S= nTn/(K+n-1)*tp$$

In the above diagram the horizontal axis displays the time in clock cycles and the vertical axis give the segment number,this diagram show six tasks T1 through T6 executed in four segment .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 pipline is full,it takes only one clock period to obtain an output

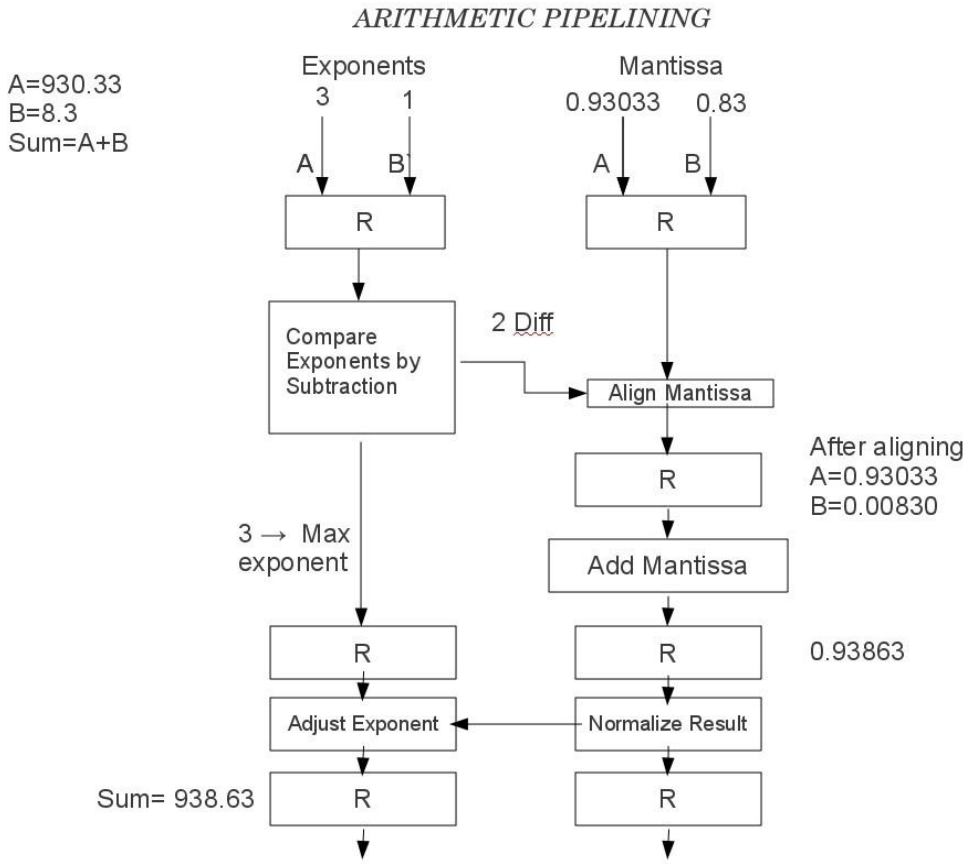
Speedup

Consider a nonlinepipeline unit that performs the same operation and takes a time equal to Tn to complete each task.The totl 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

To clarify the meaning of the speedup ratio, consider the following numerical example.
Let the time it takes to process a sub operation in each segment be equal to tp=20ns.Assume that the pipeline has k=4 segment and execute n=100 task in sequence. The pipeline system will take (k+n-1) tp=(4+99)*20=2060ns to complete Assuming that tn = ktp = 4*20 = 80ns, a nonpipeline system require nktp = 100*80=8000ns to complete the 100 task . The speedup ratio is equal to 8000/2060 =3.88

Arithmetic pipeline:



The above diagram represents the implementation of arithmetic pipeline in the area of floating point arithmetic operations. In the diagram, we can see that two numbers A and B are added together. Now the values of A and B are not normalized, therefore we must normalize them before start to do any operations. The first thing is we have to fetch the values of A and B into the registers. Here R denote a set of registers. After that the values of A and B are normalized, therefore the values of the exponents will be compared in the comparator. After that the alignment of mantissa will be taking place. Finally, we will be performing addition, since an addition is happening in the adder circuit. The source registers will be free and the second set of values can be brought. Like wise when the normalizing of the result is taking place, addition of the new values will be added in the adder

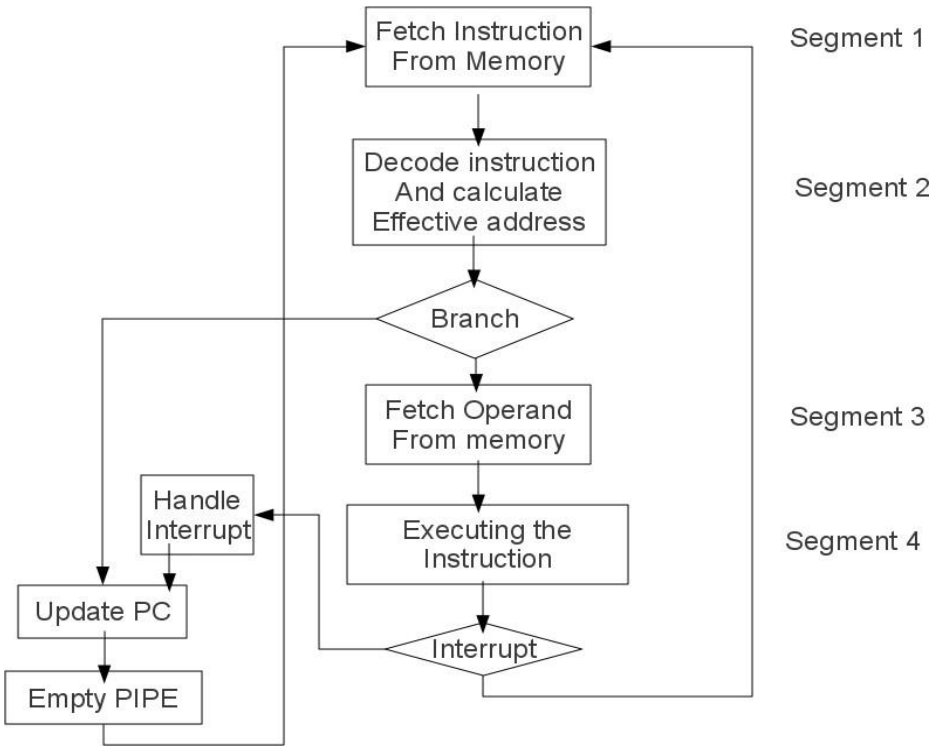
circuit and when addition is going on, the new data values will be brought into the registers in the start of the implementation. We can see how the addition is being performed in the diagram.

Instruction Pipeline: Pipelining concept is not only limited to the data stream, but can also be applied on the instruction stream. The instruction pipeline execution will be like the queue execution. In the queue the data that is entered first, will be the data first retrieved. Therefore when an instruction is first coming, the instruction will be placed in the queue and will be executed in the system. Finally the results will be passing on to the next instruction in the queue. This scenario is called as Instruction pipelining. The instruction cycle is given below

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

In a computer system each and every instruction need not necessary to execute all the above phases. In a Register addressing mode, there is no need of the effective address calculation. Below is the example of the four segment instruction pipeline.

Instruction pipelining



In the above diagram we can see that the instruction which is first executing has to be fetched from the memory, there after we are decoding the instruction and we are calculating the effective address. Now we have two ways to execute the instruction. Suppose we are using a normal instruction like ADD, then the operands for that instruction will be fetched and the instruction will be executed. Suppose we are executing an instruction such as Fetch command. The fetch command itself has internally three more commands which are like ACTDR, ARTDR etc.., therefore we have to jump to that particular location to execute the command, so we are using the branch operation. So in a branch operation, again other instructions will be executed. That means we will be updating the PC value such that the instruction can be executed. Suppose we are fetching the operands to perform the original operation such as ADD, we need to fetch the data. The data can be fetched in two ways, either from the main memory or else from an input output devices. Therefore in order to use the input output devices, the devices must generate the interrupts which should be handled by the CPU. Therefore the handling of interrupts is also a kind of program execution. Therefore we again have to start from the starting of the program and execute the interrupt cycle.

The different instruction cycles are given below:

- FI → FI is a segment that fetches an instruction
- DA → DA is a segment that decodes the instruction and identifies the effective address.
- FO → FO is a segment that fetches the operand.
- EX → EX is a segment that executes the instruction with the operand.

Timing of Instruction Pipeline

FI → Fetch Instruction
FO → Fetch Operand
DA → Decode instruction and Fetch Effective Address
EX → Execute the Instruction

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

Pipelining Conflicts: There are different conflicts that are caused by using the pipeline concept. They are

- Resource Conflicts: These are caused by access to memory by two or more segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories
- Data Dependency: These conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- Branch difficulties: These difficulties arise from branch and other instructions that change the value of PC.

Data Dependency Conflict: The data dependency conflict can be solved by using the following methods.

- Hardware Interlocks: The most straight forward method is to insert hardware interlocks. An interlock is a circuit that detects instructions whose source operands are destination of instructions farther up in the pipeline. Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict. This approach maintains the program sequence by using hardware to insert the required delay.
- Operand Forwarding: Another technique called operand forwarding uses special hardware to detect a conflict and avoid the conflict path by using a special path to forward the values between the pipeline segments.
- Delayed Load: The delayed load operation is nothing but when executing an instruction in the pipeline, simply delay the execution starting of the instruction such that all the data that is needed for the instruction can be successfully updated before execution.

Branch Conflicts:

The following are the solutions for solving the branch conflicts that are obtained in the pipelining concept.

- Pre-fetch Target Instruction: In this the branch instructions which are to be executed are pre-fetched to detect if any errors are present in the branch before execution.
- Branch Target Buffer: BTB is the associative memory implementation of the branch conditions.
- Loop buffer: The loop buffer is a very high speed memory device. Whenever a loop is to be executed in the computer. The complete loop will be transferred in to the loop buffer memory and will be executed as in the cache memory.

- Branch Prediction: The use of branch prediction is such that, before a branch is to be executed, the instructions along with the error checking conditions are checked. Therefore we will not be going into any unnecessary branch loops.
- Delayed Branch: The delayed branch concept is same as the delayed load process in which we are delaying the execution of a branch process, before all the data is fetched by the system for beginning the CPU.

RISC Pipeline:

The ability to use the instruction pipelining concept in the RISC architecture is very efficient. 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. Due to fixed length instruction format, the decoding of the operation can occur at the same time as the register selection. Since the arithmetic, logic and shift operations are done on register basis, there is no need for extra fetching or effective address decoding steps to perform the operation. So pipelining concept can be effectively used in this scenario. Therefore the total operations can be categorized as one segment will be fetching the instruction from program memory, the other segment executes the instruction in the ALU and the third segment may be used to store the result of the ALU operation in a destination register. The data transfer instructions in RISC are limited to only Load and Store instructions. To prevent conflicts in data transfer, we will be using two separate buses one for storing the instructions and other for storing the data.

Example of three segment instruction pipeline:
We want to perform a operation in which there is some arithmetic, logic or shift operations. Therefore as per the instruction cycle, we will be having the following steps:

- I: Instruction Fetch
- A: ALU Operation
- E: Execute Instruction.

The I segment will be fetching the instruction from program memory. The instruction is decoded and an ALU operation is performed in the A segment. In the A segment the ALU operation instruction will be fetched and the effective address will be retrieved and finally in the E segment the instruction will be executed.

Delayed Load:

Consider the following instructions:

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

The below tables will be showing the pipelining concept with the data conflict and without data conflict.

Pipeline timing with data conflict

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 delayed load

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

Vector Processing:

Normal computational systems are not enough in some special processing requirements. Such as, in special processing systems like artificial intelligence systems and some weather forecasting systems, terrain analysis, the normal systems are not sufficient. In such systems the data processing will be involving on very high amount of data, we can classify the large data as a very big arrays. Now if we want to process this data, naturally we will need new methods of data processing. The vectors are considered as the large one dimensional array of data. The term vector processing involves the data processing on the vectors of such large data.

The vector processing system can be understand by the example below.

Consider a program which is adding two arrays A and B of length 100;

Machine level program

```
        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
```

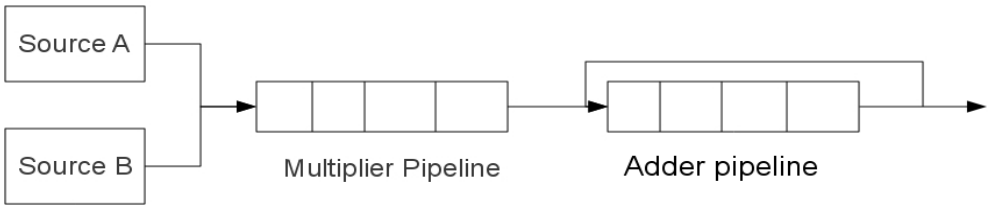
so in this above program we can see that the two arrays are being added in a loop format. First we are starting from the value of 0 and then we are continuing the loop with the addition operation until the I value has reached to 100. In the above program there are 5 loop statements which will be executing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

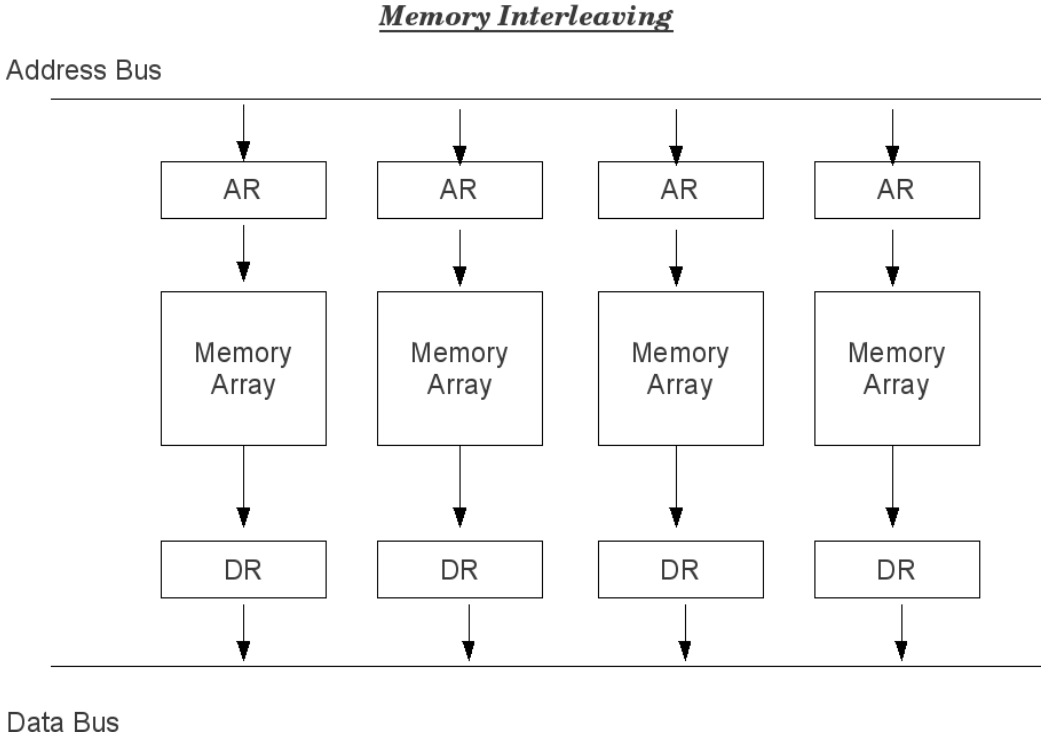
Instruction format of Vector Instruction

Below we can see the implementation of the vector processing concept on the following matrix multiplication. In the matrix multiplication, we will be multiplying the row of A matrix with the column of the B matrix elements individually finally we will be adding the results.



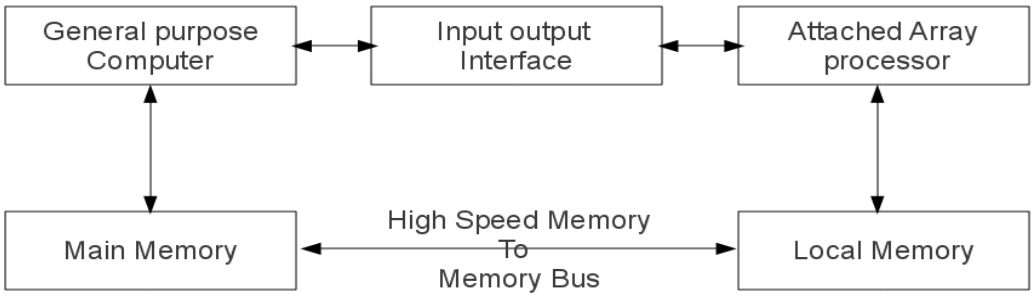
In the above diagram we can see that how the values of A vector and B Vector which represents the matrix are being multiplied. Here we will be considering a 4x4 matrix A and B. Now the from the source A vector we will be taking the first 4 values and will be sending to the multiplier pipeline along with the 4 values from the vector B. The resultant 1 value is stored in the adder pipeline. Like wise remaining values from a row and column multiplication will be brought into the adder pipeline, which will be performing the addition of all the things finally we will have the result of one row to column multiplication. 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:



Pipelining and vector processing naturally requires the several data elements for processing. So instead of using the same memory and selecting one at a time, we will be using several modules of the memory such that we can have separate data for each processing unit. As we can see in the above in the diagram each memory array is designed independently of the next memory array. Such that when the data needed for a operation is stored in the first memory array, another data for another operation can be safely stored in the next memory array, so that the operations can be performed concurrently. This process is called as memory interleaving.

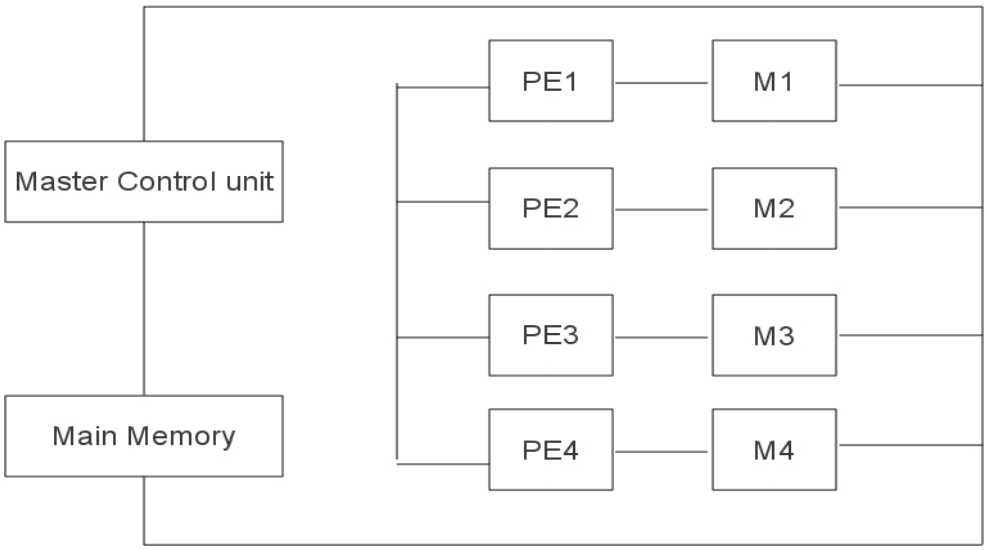
Array Processors: In a distributed computing we will be having several computers working on the same task such that their processing power will be shared among all the systems so that they can perform the task fast. But the disadvantage of the distributed computing is that we have to give separate resources for each system and every system need to be controlled by a task initiating system or can be called as a central control unit. The management of this kind of systems is very hard. In order to perform a specific operation involving a large processing there is no need of distributed computing. The alternate for this kind of scenarios is array processors or attached array processors. The simplest is the SIMD Attached array processor.



Attached Array processor

The above diagram shows that the system is attached a separate processor which will be used for operation specific purpose. If the array processor is designed for solving floating point arithmetic, then it will only perform that operations. The detailed figure of the attached array processor is given in the diagram below. This will be having the SIMD architecture. In this we will be having a master control unit which will be coordinating all the process in the array processor. Each processing unit in the array processor is having a local memory unit as in the memory interleaving concept on which it performs the operations. Finally we will be having a main memory in which the original source data and the results that are obtained from the array processor will be stored. This is

the working principle of the SIMD array processor technology.



SIMD Array Processor Technology