BLOCK III:
ADVANCED CONCEPTS OF PARALLEL ARCHITECTURES

Unit 1 : Basic Parallel Architecture and Instruction Pipeline
Unit 2 : Vector Processing
Unit 3 : Advanced Concepts of Computer Architecture
Implicit Parallelism
Unit 4 : Advanced Concepts of Pipeline schedule
Unit 5 : Advanced CPU Architecture

UNIT 1: BASIC PARALLEL ARCHITECTURE AND INSTRUCTION PIPELINE

Unit Structure:

Space for learners:

1.1 INTRODUCTION

The chapter reviews architectural evolution of computers starting from uniprocessor systems to multiprocessor system through Flynn's classification of computer architecture. The chapter also compares various processors types like scalar processor, superscalar processor, pipelined processor and vector processor. The basic concept of pipelining and the working of instruction pipeline is

discussed in detail. Finally, the chapter ends with discussion on the types of dependencies that exists in pipelined processors which if not taken care of will affect the overall performance of the system. The three dependencies discussed are resource conflict, branch hazard and data hazard.

Space for learners:

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Classify computer architecture based on the notion of instruction and data stream.
- Compare different types of processors and their characteristics.
- Explain the basic concept of pipelining and the types of pipelining.
- Explain how pipelining improves the performance of a system.
- Explain how multiple instructions are executed in an overlapped fashion in instruction pipelining.
- Identify the types of dependencies in pipelined processors and ways to resolve the dependencies.

1.3 FLYNN'S CLASSIFICATION OF COMPUTER ARCHITECTURE

With the increase in the number of processing units and segmentation of a job/program into multiple segments where in each of the segment is placed on a different processing unit for concurrent execution has resulted in classification of systems. Flynn's classification or Flynn's taxonomy of computer architectures is proposed by Michael J. Flynn in the year 1972. The classification is based on the notion of instruction stream and data stream. A stream refers to sequence of instruction or data operated by the computer system. The flow of instruction from memory to

processor is called instruction stream and the flow of data between processor and memory is called data stream.

Space for learners:

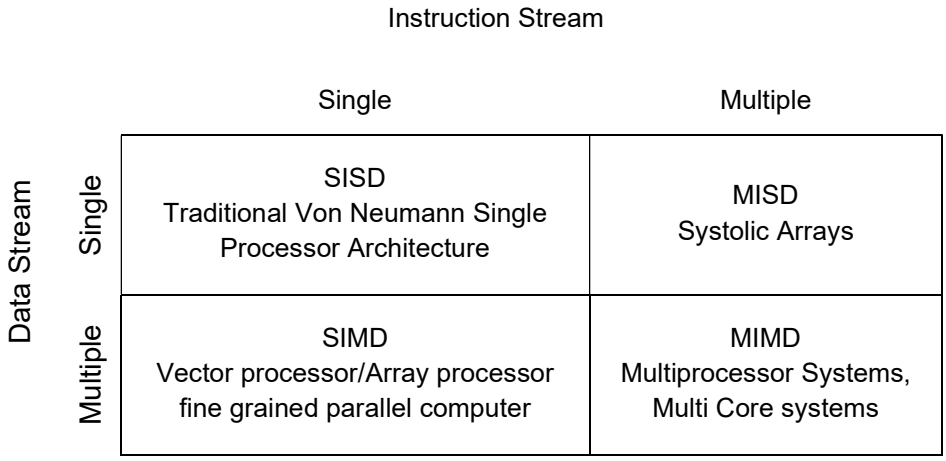


Figure 1.1: Flynn's classification of Computer Architecture

The Figure 1.1 shows four categories in which Flynn has classified computer architecture based on instruction and data stream. A conventional uniprocessor system is called SISD (Single instruction stream Single data stream) computers. A vector /array of processors is called SIMD (Single instruction stream Multiple data stream) computers. In MISD (Multiple instruction stream Single data stream) computers, different instructions work on the same data. Finally, in MIMD (Multiple instruction stream Multiple data stream) computers, multiple processors each working on different data increases the overall performance of the system.

1.3.1 SISD

A Single instruction stream single data stream (SISD) system as shown in Figure 1.2 is a uniprocessor system. Such systems work on

a single instruction stream and single data stream at a time. Instructions in SISD systems are processed in a sequential order and are therefore known as sequential computers. Conventional systems were of SISD architecture. Processing in SISD systems involve storing both instructions and data in primary memory. Processing speed of such systems depends on internal data transfer rate. However, performance of such systems can be improved with the help of multiple functional units or pipelining. Example of SISD systems includes CDC 6600, IBM PC.

Space for learners:

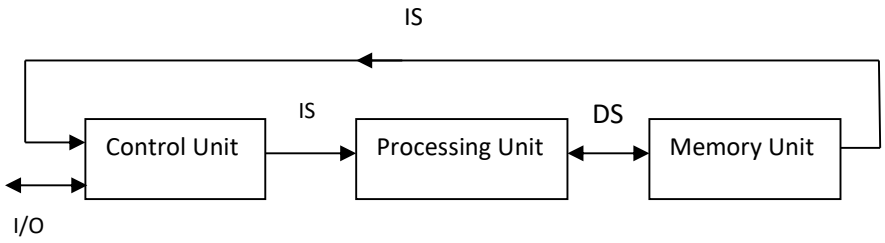


Figure 1.2: SISD Architecture

1.3.2 SIMD

A single instruction stream multiple data stream (SIMD) system as shown in Figure 1.3 are multiprocessor systems capable of working on different data streams through a single instruction stream. SIMD systems are used in scientific computation involving vector operations. They are also known as vector processors or array processors. There are *n* number of processing units each having its own memory and stream of data. All the *n* processing units receive the same instruction from the control unit. Example of SISD systems includes CRAY vector computers.

Space for learners:

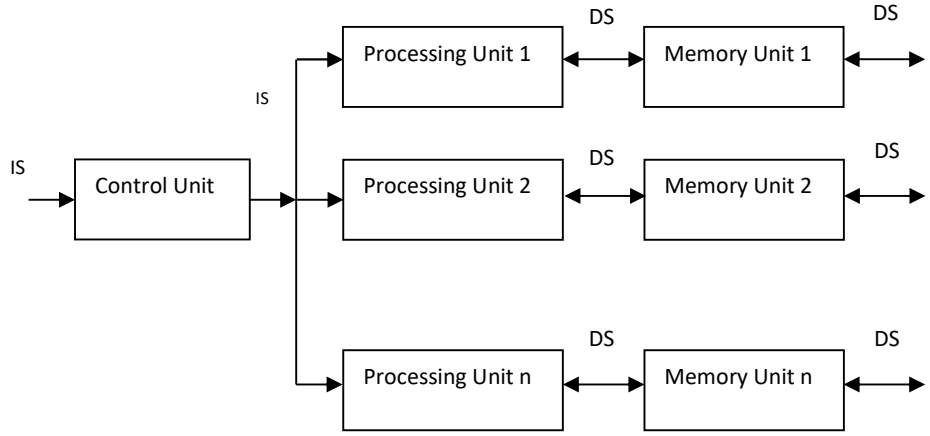


Figure 1.3: SIMD Architecture

1.3.3 MISD

A multiple instruction stream single data stream (MISD) system as shown in Figure 1.4 is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions. MISD systems are not practical in the majority of the application and therefore are not available commercially. One such example of MISD system is systolic array.

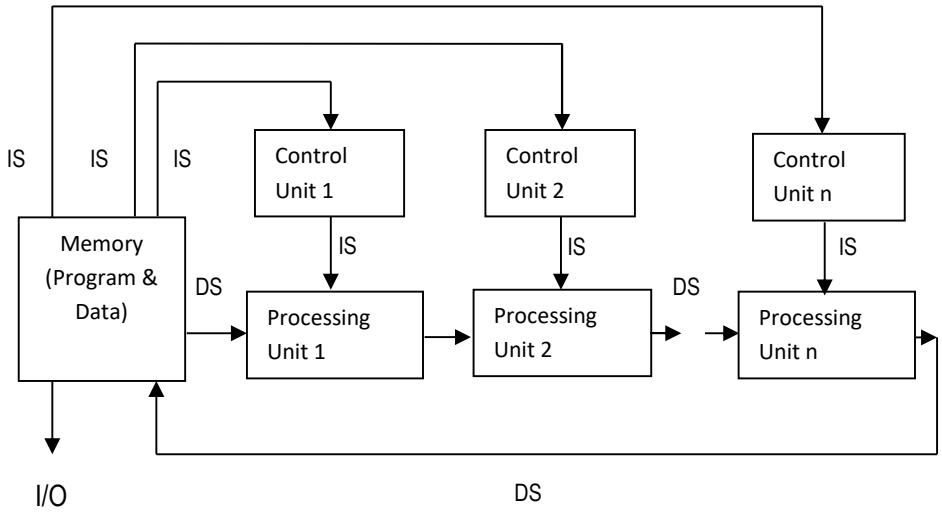


Figure 1.4: MISD Architecture

1.3.4 MIMD

Space for learners:

A multiple instruction stream multiple data stream (MIMD) system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously. Figure 1.5 shows MIMD architecture. Multiple SIMD systems connected together can be viewed as a MIMD system. MIMD systems can be classified into shared-memory MIMD and distributed-memory MIMD based on processing unit-main memory connections.

The shared memory MIMD system also known as tightly coupled multiprocessor system, are the one where all the processing units are connected to a single shared memory. Any form of communication between processing unit takes place with the help of shared memory. Changes done to data in shared memory by one processing unit is visible to all other processing units. In distributed memory, MIMD systems or loosely coupled multiprocessor systems, all processing units have their own local memory. The communication between processing units takes place through the interconnection.

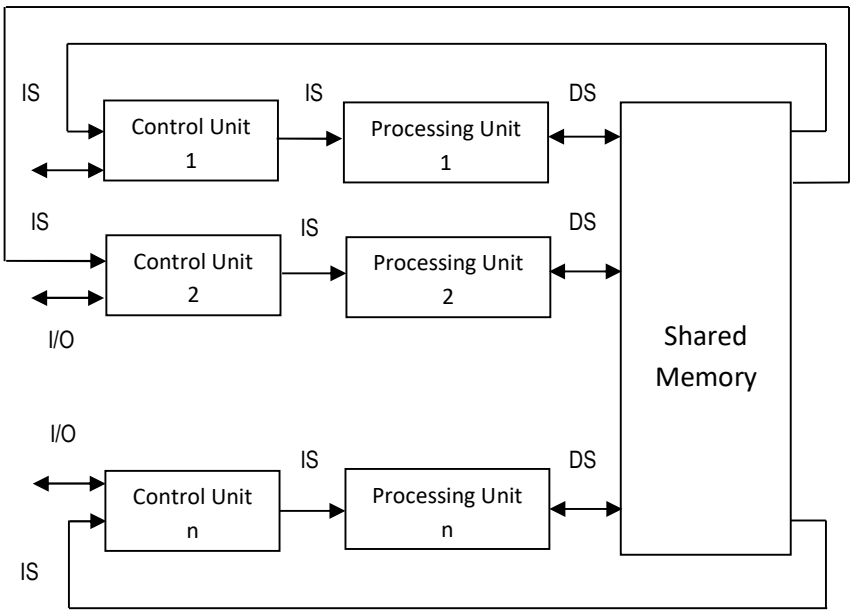


Figure 1.5: MIMD Architecture

1.4 TYPE OF PROCESSORS

Space for learners:

There are several types of processors. A brief description of each of these are provided below.

1.4.1 Scalar processor

A scalar processor also known as Single Instruction Stream Single Data Stream (SISD) can process a single data item at a time. Scalar processor can process either integer or floating point operands. The simplest scalar processor uses floating point unit to process integer operands. However scalar processor may have separate integer and floating point units for handling integer and floating point operands. AMD 2900, Motorola 68040, Intel 386, Intel 486, M88100 are some examples of scalar processor.

1.4.2 Superscalar Processor

Superscalar processors are found in parallel computing architecture to improve the performance of the system by executing multiple instructions in parallel. A superscalar processor manages multiple instruction pipelines to execute multiple instructions concurrently in a clock cycle. The performance of superscalar processor is highly dependent on the instruction dependency quotient. If the instructions to be executed are independent, then high performance is achieved.

Figure 1.6 shows a superscalar pipeline of degree 2 (i.e. Two instructions can be executed in parallel). There are five stages in the pipeline namely fetch, decode, operand fetch, execute and write. It can be observed from the Figure 1.6 that the superscalar pipeline has two units each of fetch, decode, operand fetch, execute and write, therefore two instructions can be simultaneously executed. In the first clock cycle instruction (1, 2) are fetched, in the second clock cycle next two instructions i.e. (3, 4) are fetched and the process

continues. Pentium, DEC Alpha, PowerPC are some of the example of superscalar processor computers.

Space for learners:

	1	2	3	4	5	6	7	8
Instruction 1	F	D	OF	E	W			
Instruction 2	F	D	OF	E	W			
Instruction 3		F	D	OF	E	W		
Instruction 4		F	D	OF	E	W		
Instruction 5			F	D	OF	E	W	
Instruction 6			F	D	OF	E	W	
Instruction 7				F	D	OF	E	W
Instruction 8				F	D	OF	E	W

Figure 1.6: A superscalar pipeline of degree two.

1.4.3 Pipelined Processor

There are four types of pipelined processors namely Scalar Pipeline, Superscalar Pipeline, Superpipeline, Superpipeline Superscalar as shown in Figure 1.7 depending upon the following criterions. It is assumed that all the pipelined processors are of k stages.

- Machine Pipeline Cycle (MPC): Time taken by each stage to process an instruction.
- Instruction Issue Rate (ISR): Number of instructions that can be issued simultaneously.
- Instruction Issue Latency (ISL): Time interval between issue of two instructions.
- Instruction Level Parallelism (ILP): Number of instructions that can be executed simultaneously in the pipeline.

Machine Type	Scalar Pipeline	Superscalar Pipeline	Super pipeline	Super pipeline Superscalar
MPC	1	1	1/n	1/n
ISR	1	m	1	m
ISL	1	1	1/n	1/n
ILP	1	m	n	mn

Figure 1.7: Parameters of Pipelined Processor.

Space for learners:

1.4.4 Vector processor

Vector processors are found mainly in supercomputers combining pipelining and interleaved memory unit. It is used mainly in scientific and multimedia applications involving processing of huge volume of data. It is capable of processing entire vector in single instruction. The operands in the instructions are vectors instead of a single element. One of the advantages of vector processors is less number of fetch and decode instructions.

Vector processor uses many optimization schemes to improve performance of the system such as use of memory banks to reduce load/store latency, use of strip mining technique to adjust the size mismatch between vector operands and vector registers, vector chaining to resolve data dependency between vector instructions etc.

Advantages of Vector processing:

- Programs are smaller in size as the number of instructions is quite less.
- As each data in registers is actually used by the vector processor therefore wastage in memory access is significantly less compared to cache memory.
- Requirement of power is limited to only functional unit and register buses during vector operation.

Based on how operands are fetched in vector processors is categorized into two types:

- Vector register Processor
- Memory-Memory Vector Processor

Vector-Register Processor

It requires that all the operations performed in the vector processor use the source operands and destination operands as vector registers. However, there is a small disadvantage initially that is vector data in memory must be divided into fixed length segments so that can be placed in vector register. But once the pipelining starts this disadvantage is nullified.

Memory-Memory Vector Processor

Such processors allow source operand and destination operand to be routed directly to the arithmetic logic unit (ALU). Once the processing is completed in the ALU, the result is routed back to memory. However due to memory latency the time between initializing the first instruction and the getting the first output from the pipeline is quite large.

Space for learners:

1.5 PIPELINING

A pipeline is similar to an assembly line in a production factory. A product has to go through multiple stages in the assembly line before the final product is manufactured. At a time, all the stages work simultaneously but on different phases of the product. This process is referred to as pipelining. Pipelining is also referred to as execution of multiple jobs/instructions parallelly in an overlapped fashion.

Let us look at a real life example that works on the concept of pipelining. Consider a packaged drinking water plant having the following 3 stages and each stage takes 1 minute to complete its operation.

- Filling (F)--- Stage 1
- Sealing (S) --- Stage 2
- Labeling (L) --- Stage 3

In a non-pipelined operation if we have to do the packaging of 4 bottles, it will take 12 min to complete the operation as shown in Figure 1.8. Each bottle spending 1 min in each of the filling, sealing and labeling stage respectively.

The bottle reaches stage-1 where it is filled and after 1 minute it moves to the stage-2 where it is sealed. At this point stage-1 is in idle state. Now after staying in stage-2 for 1 minute the bottle is moved to stage-3 where it is labeled. At this point stage-1 and stage-2 is in idle state as shown in the figure 1.8. This process of packaging does not utilize the time as the stages remain in idle state during the operation. To overcome the issue and to utilize the stages to its maximum limit, pipelining is used.

	Time in minute→											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Bottle 1	F	S	L									
Bottle 2				F	S	L						
Bottle 3							F	S	L			
Bottle 4										F	S	L

Figure 1.8: Non Pipelined Operation

Space for learners:

Now, in a pipelined operation if we have to do the packaging of 4 bottles, it will take 6 min to complete the operation as shown in Figure 1.9. Compared to 12 minutes taken in non-pipelined operation.

As it can be observed in Figure 1.9, when the first bottle is in stage-2 (Sealing), the second bottle is placed in stage-1(Filling). Similarly, when the first bottle is in stage-3(Labeling), second bottle is placed in stage-2(Sealing) and third bottle is placed in stage-1(Filling). Thus, none of the stages are idle at any moment. All the stages are working on a different bottle at a time. This process of working in an overlapped fashion to utilize the stages of a pipeline to its fullest is called pipelining.

	Time in minute→											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Bottle 1	F	S	L									
Bottle 2		F	S	L								
Bottle 3			F	S	L							
Bottle 4				F	S	L						

Figure 1.9: Pipelined Operation

1.6 INSTRUCTION PIPELINING

In a computer system the technique of executing multiple instructions in an overlapped fashion is known pipelining. A pipeline consists of many stages and these stages are connected to one another in a pipe like structure. An instruction enters one end of the pipeline, goes through several stages before exiting from another end. Pipelining improves the overall throughput of the system.

Space for learners:

In a pipeline system, each stage uses register to hold the output of that stage. Output of one stage is applied as input to the next stage.

Space for learners:

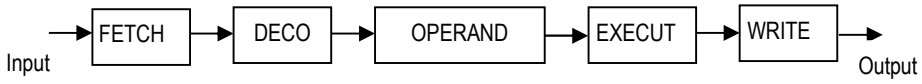


Figure 1.10: Five stage Instruction Pipeline

Figure 1.10 shows an example of five stage instruction pipeline consisting of fetch, decode, operand fetch, execute and write stages. Here streams of instructions are executed in overlapped fashion thereby increasing the throughput of the computer system.

Figure 1.11 shows the timing diagram of an instruction pipeline. While the instruction pipeline reads one instruction from the memory, previous instruction is executed in other stage of the pipeline. Thus, multiple instructions are executed simultaneously. From the Figure 1.11, it can be observed that while the first instruction started at time period one, the second instruction started at time period two and so on. Up to time period four, not all stages were working simultaneously but from time period five onwards all the five stages are working simultaneously. Therefore, from instruction number five onwards each stage is working on a different instruction as:

Instruction 1: Write
Instruction 2: Execute
Instruction 3: Operand Fetch
Instruction 4: Decode
Instruction 5: Fetch

Time →

	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	F	D	OF	E	W						
Instruction 2		F	D	OF	E	W					

Instruction 3			F	D	OF	E	W				
Instruction 4				F	D	OF	E	W			
Instruction 5					F	D	OF	E	W		
Instruction 6						F	D	OF	E	W	
Instruction 7							F	D	OF	E	W

Figure 1.11: Timing diagram for Instruction Pipeline Operation

If there are ***k*** number of stages and ***n*** number of instructions, then total time ***T*** taken to execute n instructions can be given as T = k + (n -1).

Space for learners:

1.7 DEPENDENCY IN PIPELINED PROCESSOR

A pipelined processor may be affected due to the following dependencies, which may also result in the stalls in the pipeline. A stall is a pipeline cycle with no operation or no new input.

- Structural Dependency or Resource Conflict
- Control Dependency or Branch Difficulty
- Data Dependency or Data Hazard

1.7.1 Structural Dependency or Resource Conflict

Structural dependency is the result of resource conflict in the pipeline. When several instructions in the same cycle try to access the same resource, a resource conflict arises. A resource can be a register, memory, or ALU.

Time →

	1	2	3	4	5	6	7	8	9	10	11
Instruction 1	F	D	OF	W							
Instruction 2		F	D	OF	W						
Instruction 3			F	D	OF	W					
Instruction 4				F	D	OF	W				

Figure 1.12: Timing diagram of a 4-Stage Instruction Pipeline

In cycle 4 of the Figure 1.12, instruction 1 is trying to do the write operation on memory and instruction 4 is trying to fetch from memory. As both the instructions are trying to access same resource i.e. memory at the same time, it introduces a resource conflict between the two instructions. Such situation can be avoided by keeping the instruction 2 in wait state until the required resource becomes available.

Space for learners:

1.7.2 Control Dependency or Branch Hazard

A pipeline achieves its maximum utilization if all the stages of the pipeline take equal amount of time to process and there is no branch instruction in the program. However, if the program contains branch instruction, the pipeline suffers from branch penalty.

The timing diagram of a 4 stage instruction pipeline containing branch instruction is shown in Figure 1.13 where instruction 1,2,3 and 4 are non-branch instruction and instruction 5 is a branch instruction.

Time → |←Branch Penalty→|

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	F	D	OF	E										
Instruction 2		F	D	OF	E									
Instruction 3			F	D	OF	E								
Instruction 4				F	D	OF	E							
Instruction 5 (branch to instruction 25)					F	D	OF	E						

Instruction 6						F	D	OF						
Instruction 7							F	D						
Instruction 25								F	D	OF	E			
Instruction 26									F	D	OF	E		

Figure 1.13: Timing diagram for Instruction Pipeline Operation

The pipeline executes instruction 1, 2, 3 and 4 sequentially, followed by instruction 5 (branch instruction). By the time instruction 5 is decoded by the pipeline decode stage, instruction 6 and instruction 7 enters the pipeline. At this point, the pipeline realized that it should have placed instruction 25 after the branch instruction 5 instead of instruction 6.

So the pipeline discards the instructions 6 and 7, that is the pipeline cycle at time period 6 and 7 are wasted. This is known as branch penalty as the processor could not anticipate the branch. So instruction 25 is assumed to be the instruction to be executed on the branch and starts at time period eight.

1.7.3 Data Dependency or Data Hazard

In a pipeline, there can be a situation where output of first instruction acts as an input to the second instruction. Such situation exhibits data dependency where the second instruction must wait in the pipeline for the first instruction to complete its execution. Otherwise the second instruction may be working on an invalid data. This dependency between the instructions is known as data dependency or data hazard. So the order of execution of the instructions does matter.

Space for learners:

There are mainly three types of data hazards:

- Read after Write (RAW) Hazard or Flow dependency
- Write after Read(WAR)Hazard or Anti-Data dependency
- Write after Write (WAW) Hazard or Output dependency

Read after Write (RAW) Hazard:
Instruction 1: R3 ← R4 + R5

Instruction 2: R6 ← R3 + R4

Here, the instruction 2 is reading a value in register R3 that is being produced by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Write after Read(WAR) Hazard:
Instruction 1: R3← R4 + R5

Instruction 2: R4← R6 + R7

Here the instruction 2 is writing a value in register R4 that is being read before by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Write after Write (WAW) Hazard:

Instruction 1: R1← R2 + R3

Instruction 2: R1← R4 + R5

Here the instruction 2 is overwriting the value in register R1 that is being produced by instruction 1. So instruction 2 should execute after instruction 1 completes its execution.

Space for learners:

1.7.4 Pipeline bubbles

A bubble or a pipeline bubble represents a stage in the pipeline that cannot perform any useful operation due to the lack of data from previous stage of the pipeline. It is a method to prevent structural, data and branch hazards. Pipeline control logic analyzes if a hazard

could arise while instructions are fetched. If this is the case, no operations (NOPs) are added to the pipeline by the control logic. As a result, before the next instruction runs, the previous one would have enough time to complete and avert the hazard.

Space for learners:

CHECK YOUR PROGRESS

i. CRAY systems are an example of_____.
   a) SISD
   b) SIMD
   c) MISD
   d) MIMD

ii. Pentium, DEC Alpha, PowerPC are some of the example of _____computers.
   a) superscalar processor
   b) Super pipeline
   c) Scalar
   d) Superscalar Super pipeline

iii. To _____data in between the pipeline stages, registers are used.
   a) Write
   b) Process
   c) Read
   d) Hold

iv. Motorola 68040 is an example of _____
   a) Scalar processor
   b) Superscalar processor
   c) Super pipeline processor
   d) Pipelined processor

v. A superscalar pipeline (5 stages) of degree 3 will need _____ cycles to complete 9 instructions.
   a) 6
   b) 7
   c) 8
   d) 9

vi. Instruction Issue Latency (ISL) in a pipelined processor means___
   a) Time interval between issuing of first and last instruction.
   b) Time interval between completion of first and second instruction.
   c) Time taken to complete execution of first instruction.
   d) Time interval between issuing of two instructions.

vii. Instruction Level Parallelism in a pipelined processor means_____
   a) Number of instructions in the pipeline.
   b) Number of instructions that can be completed simultaneously in the pipeline.
   c) Number of instructions that can be executed simultaneously in the pipeline.
   d) None of the above

viii. Vector processors or Array processors are also known as_____ systems
   a) SISD
   b) MISD
   c) SIMD
   d) MIMD

ix. The time period when the pipeline unit remains idle is called as _____
   a) Hazards
   b) Bubbles
   c) Stalls
   d) Both b) and c)

x. In pipelining, memory access speedup is achieved through _____
   a) Cache
   b) Buffers
   c) Memory Registers
   d) Special Registers

xi. In a pipeline branch instructions are handled by _____

Space for learners:

a) Pipeline flush operation
b) Pipeline Freeze operation
c) Pipeline Depth operation
d) Both a) and b)

xii. If second instruction tries to do a write operation before the first instruction can write on the same data, it is called as _____ dependency.
a) Data
b) Anti
c) Flow
d) Output

xiii. If second instruction tries to do a read operation after the first instruction does a write on the same data, it is called as _____ hazard.
a) RAW
b) WAR
c) Data
d) Control

xiv. Time taken by a 7 stage instruction pipeline to complete execution of 10 instructions is_____.
a) 70
b) 32
c) 16
d) 17

xv. Time taken by a 3 stage superscalar pipeline of degree 2 to execute 10 instructions is _____.
a) 10
b) 9
c) 8
d) 7

Space for learners:

1.8 SUMMING UP

- Flynn has classified computer architecture based on instruction and data stream and are SISD, SIMD, MISD, and MIMD.

- SISD systems are processed in a sequential order and are therefore known as sequential computers.
- SIMD systems are multiprocessor systems capable of working on different data streams through a single instruction stream.
- MISD system is a multiprocessor system which executes different instructions on different processing unit but data set is same for all the instructions.
- MIMD system is a multiprocessor system capable of executing different sets of instructions each working on a different set of data simultaneously.
- Superscalar processors are found in parallel computing architecture to improve the performance of the system by executing multiple instructions concurrently in a clock cycle.
- Pipelined processors namely Scalar Pipeline, Superscalar Pipeline, Super pipeline, Super pipeline Superscalar.
- Vector processors are used mainly in scientific and multimedia applications involving processing of huge volume of data. It is capable of processing entire vector in single instruction.
- Pipelining is also referred to as execution of multiple instructions parallelly in an overlapped fashion.
- Structural dependency is the result of resource conflict in the pipeline. When several instructions in the same cycle try to access the same resource, a resource conflict arises.
- If the program contains branch instruction, the pipeline suffers from branch penalty.
- Dependency between the instructions is known as data dependency or data hazard. Order of execution of the instructions does matter.

Space for learners:

- There are mainly three types of data hazards Read after Write (RAW), Write after Read (WAR), and Write after Write (WAW).
- A bubble or a pipeline bubble represents a stage in the pipeline that cannot perform any useful operation due to the lack of data from previous stage of the pipeline.

Space for learners:

1.9 ANSWERS TO CHECK YOUR PROGRESS

i. b	ii. a	iii. d	iv. a	v. b
vi. d	vii. c	viii. c	ix. d	x. a
xi. d	xii. d	xiii. a	xiv.c	xv.d

1.10 POSSIBLE QUESTIONS

1. Discuss Flynn's classification of computer architecture.
2. According to Flynn's classification, the architecture which is of theoretical interest but no real-world system has been developed on it?
3. Differentiate between shared memory and distributed memory MIMD systems.
4. Explain how pipelining can increase the performance of a system compared to a single processor system.
5. Differentiate between superscalar processor and Super pipeline processor.
6. Briefly describe the parameters on which different pipelined processors are measured in terms of their performance.
7. Discuss the types of processors that is helpful in parallel processing.
8. Discuss the factors that affect the performance of a pipeline.
9. Define instruction pipeline with the help of an example.
10. Discuss resource conflict in pipelining.

11. Discuss Data hazard in pipelining.
12. What is a pipeline bubble? In what situation a pipeline bubble is used?

Space for learners:

1.11 REFERENCES AND SUGGESTED READINGS

- Advanced Computer Architecture, 3e, Kai Hwang, Naresh Jotwani; McGraw-Hill Education, 2016
- Computer Organization and Architecture: Designing for Performance 10 Edition, by William Stallings, Pearson.
- Computer System Architecture Third Edition, M. Morris Mano, Rajib Mall, Pearson
- Computer Organization Fifth Edition, Carl Hamacher, McGraw Hill

UNIT 2: VECTOR PROCESSING

Space for learners:

Unit Structure:

2.1 INTRODUCTION

A normal processor sometimes called scalar processor, which works on simple instruction at a time, which operates on single data items. Standard von Neumann machine is based on the instruction and data are stored in memory, that has one operation at a time, maximum speed of the system is limited by the memory bandwidth(bits/sec or bytes/sec). It means normal processing having limitation on memory bandwidth; which the memory is shared by CPU and I/O. But in today's world, this technique is proved to be highly inefficient, as the overall processing of instructions will be very slow.

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer (with scalar processor) days or even weeks to complete.

Such complex instruction, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by vector computing technique using vector processors. Scalar CPUs can manipulate one or two data items at a time, which is not very efficient.

Space for learners:

2.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic concepts of vector computing and working principle of vector processor.
- Know about the pipelining techniques applied in vector computing.

- Understand how arithmetic pipelining works.
- Give the basic concept of array processor and its different categories.
- Know about the vector and scalar registers used in vector processing.
- Define what is a chaining and scatter-gather operation.
- Understand about register-register and memory-memory vector processors.

Space for learners:

2.3 VECTOR COMPUTING

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer(with scalar processor) days or even weeks to complete.

Such complex instructions, which operate on multiple data at the same time, requires a better way of instruction execution, which has been achieved by the vector computing technique that done by vector processors.

Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of Instruction Pipeline comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the third sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle; it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector computing technique, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be *ADD A, B*, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers (from 0 to n memory location) to another group of numbers (let's say, *n* to *k* memory location) then a scalar processor is unable to add these set values. This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Space for learners:

2.4VECTOR PROCESSOR

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.

Vector processors are co-processor to general-purpose microprocessor. Vector processors are generally register-register or memory-memory. A vector instruction is fetched and decoded and then a certain operation is performed for each element of the operand vectors, whereas in a normal processor a vector operation needs a loop structure in the code. To make it more efficient, vector processors chain several vector operations together, i.e., the result from one vector operation are forwarded to another as operand.

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. So, 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 (SIMD) or vector instructions**.** The CPU used in recent time makes use of vector processing as it is advantageous than scalar processing.

A vector processor is a processor that can operate on an entire vector in one instruction. The operand to the instructions are complete vectors instead of one element. Vector processors reduce the fetch and decode bandwidth as the number of instructions fetched are less. They also exploit data parallelism in large scientific and multimedia applications. Based on how the operands are fetched, vector processors can be divided into two categories - in memory-memory architecture operands are directly streamed to the

Space for learners:

functional units from the memory and results are written back to memory as the vector operation proceeds. In vector-register architecture, operands are read into vector registers from which they are fed to the functional units and results of operations are written to vector registers. Many performance optimization schemes are used in vector processors. Memory banks are used to reduce load/store latency. Strip mining is used to generate code so that vector operation is possible for vector operands whose size is less than or greater than the size of vector registers. Vector chaining - the equivalent of forwarding in vector processors - is used in case of data dependency among vector instructions. Special scatter and gather instructions are provided to efficiently operate on sparse matrices.

Instruction set has been designed with the property that all vector arithmetic instructions only allow element *N* of one vector register to take part in operations with element *N* from other vector registers. This dramatically simplifies the construction of a highly parallel vector unit, which can be structured as multiple parallel lanes. As with a traffic highway, we can increase the peak throughput of a vector unit by adding more lanes.

Space for learners:

2.4.1 Some Important Facts on a Vector Processor

- A vector processor is an ensemble of hardware resources, including vector registers, functional pipelines, processing elements and register counters for performing register operations.
- Vector processing occurs when arithmetic or logical operations are applied to vectors. It is distinguished from scalar processing which operates on one or one pair of data.

The conversion from scalar code to vector code is called vectorization.

- Both pipelined processors and SIMD computers can perform vector operations.
- Vector processing reduces software overhead incurred in the maintenance of looping control, reduces memory access conflicts and above all matches nicely with pipelining and segmentation concept to generate one result per each clock cycle continuously.

Space for learners:

2.4.2 Advantages of Vector Processor

Some advantages of vector processors are given below:

- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector memory access has no wastage like cache access. Every data item requested by the processor is actually used.
- Once a vector instruction starts operating, only the functional unit (FU) and the register buses feeding it need to be powered. Fetch unit, decode unit, ROB etc can be powered off. This reduces the power usage.

2.4.3 Applications of Vector Processors

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.

3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Space for learners:

2.4.4 Cost of Vector Processor

The reason behind the declining popularity of vector processors are their cost as compared to multiprocessors and superscalar processors. The reasons behind high cost of vector processors are

• Vector processors do not use commodity parts. Since they sell very few copies, design cost dominates overall cost.

• Vector processors need high speed on-chip memory which are expensive.

• It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.

• There have been few architectural innovations compared to superscalar processors to improve performance keeping the cost low.

2.4.5 Classification of Vector Processor

The classification of vector processor relies on the ability of vector formation as well as the presence of vector instruction for processing. So, depending on these criteria, vector processing is classified as follows:

(i) Register to Register Architecture (Vector register processors)and

(ii) Memory to Memory Architecture(Memory-memory vector processors)

According to the position from where the operands are retrieved in a vector processor, pipe lined vector computers are classified into two architectural configurations:

Space for learners:

2.4.5.1 Memory to memory architecture

In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from the main memory. For memory to memory vector instructions, the information of the base address, the offset, the increment, and the vector length must be specified in order to enable streams of data transfers between the main memory and pipelines. The processors like *TI-ASC, CDC STAR-100, and Cyber-205* have vector instructions in memory to memory formats. The main points about memory to memory architecture are:

- There is no limitation of size
- Speed is comparatively slow in this architecture

2.4.5.2 Register to Register Architecture

This architecture is highly used in vector computers. As in this architecture, the fetching of the operand or previous results indirectly takes place through the main memory by the use of registers. The several vector pipelines present in the vector computer help in retrieving the data from the registers and also storing the results in the desired register. These vector registers are user instruction programmable. In a vector-register processor, all vector operations—except load and store—are among the vector

registers. These architectures are the vector counterpart of a load-store architecture.

All major vector computers shipped since the late 1980s use a vector-register architecture, including the Cray Research processors (Cray-1, Cray-2, X-MP,YMP, C90, T90, SV1, and X1), the Japanese supercomputers (NEC SX/2 through SX/8, Fujitsu VP200 through VPP5000, and the Hitachi S820 and S-8300), and the mini-supercomputers (Convex C-1 through C-4).

In register to register architecture, operands and results are retrieved indirectly from the main memory through the use of large number of vector registers or scalar registers. The processors like *Cray-1 and the Fujitsu VP-200* use vector instructions in register to register formats. The main points about register to register architecture are:

(i) Register to register architecture has limited size.

(ii) Speed is very high as compared to the memory to memory architecture.

(iii) The hardware cost is high in this architecture.

Space for learners:

2.5 SUPERSCALAR PROCESSOR

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors.

A scalar processor works on one or two data items, itis a normal processor, which works on simple instruction at a time, which operates on single data items, while the vector processor works with multiple data items. A superscalar processor is a combination of

both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.

There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions.

It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.

While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are given below:

- Instructions are issued from a sequential instruction stream.
- CPU must dynamically check for data dependencies.
- Should accept multiple instructions per clock cycle.

Space for learners:

2.6 VECTOR COMPUTER

Space for learners:

The functional units of a vector computer are as follows:

(i) IPU or instruction processing unit
(ii) Vector register
(iii)Scalar register
(iv)Scalar processor
(v) Vector instruction controller
(vi)Vector access controller
(vii) Vector processor

Let us now understand the overall operation performed by the vector computer.

As 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.

While, 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 sub-vectors.

These sub-vectors 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.

A block diagram of a modern multiple pipeline vector computer is shown below:

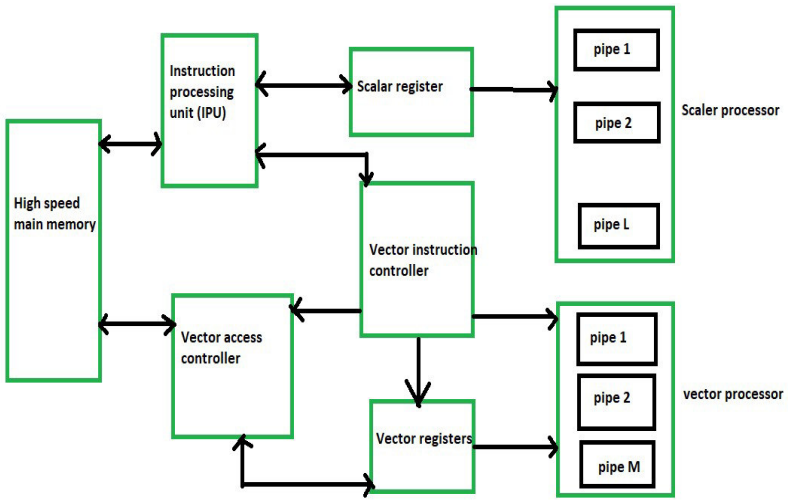


Fig.2.1 A block diagram of a modern multiple pipeline vector computer

2.6.1 Vector registers

Vector registers are the storage areas in a CPU core that contain the operands for vector computations, as well as the results. The size of

Space for learners:

the vector registers determines the level of SIMD instructions that can be supported by a given processor's CPUs.

Each vector register is a fixed-length bank holding a single vector. VMIPS has eight vector registers, and each vector register holds 64 elements. Each vector register must have at least two read ports and one write port in VMIPS. This will allow a high degree of overlap among vector operations to different vector registers. The read and write ports, which total at least 16 read ports and 8write ports, are connected to the functional unit inputs or outputs by a pair of crossbars. Real machines make use of the regular access pattern within a vector instruction to reduce the costs of the vector-register file circuitry. For example, the Cray-1 manages to implement the register file with only a single port per register.

Space for learners:

2.6.2 Scalar registers

Scalar processors represent a class of computer processors. A scalar processor processes only one data item at a time, with typical data items being integers or floating point numbers. A scalar processor is classified as a single instruction, single data (SISD) processor in Flynn's taxonomy.

Scalar registers can also provide data as input to the vector functional units, as well as compute addresses to pass to the vector load-store unit. These are the normal 32 general-purpose registers and 32 floating-point registers of MIPS. Scalar values are read out of the scalar register file, then latched at one input of the vector functional units.

2.7 ARRAY PROCESSORS

Array processors are also known as multiprocessors or vector processors. An array processor is a processor that performs computations on large arrays of data. Thus, they are used to improve the performance of the computer.

In other words, an array processor is a CPU which implements an instruction set that are designed to operate efficiently and effectively on large one-dimensional arrays of data called vectors.

Vector and array processing are essentially the same because, with slight and rare differences, a vector processor and an array processor are the same type of processor. A vector processor is in contrast of the simpler scalar processor, which handles only one piece of information at a time.

2.7.1 Types of Array Processors

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

2.7.1.1 Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units. 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.

Space for learners:

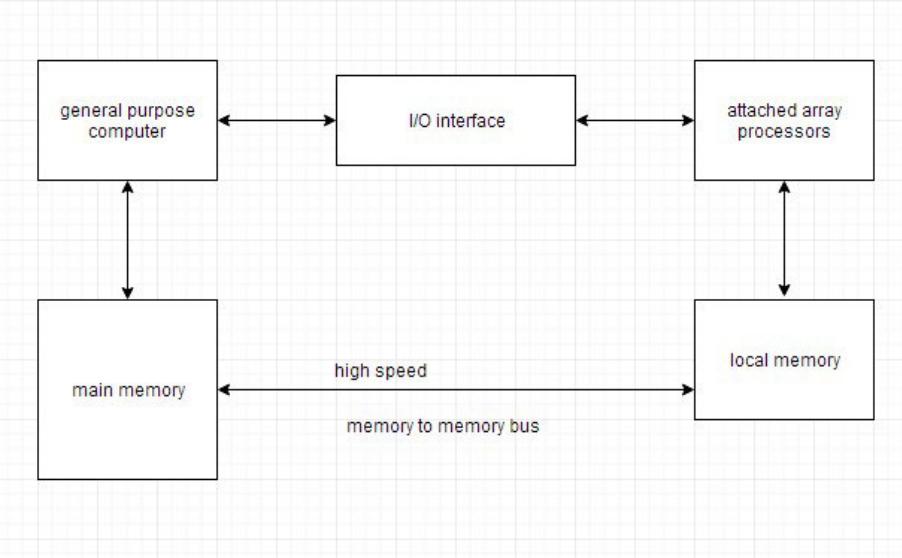


Fig.2.2 Block diagram of Attached Array Processors

Space for learners:

2.7.1.2 SIMD Array Processors

Single-instruction, multiple data(SIMD) is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU and registers. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are sent to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the ILLIAC IV computer developed by the Burroughs corps. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.

Space for learners:

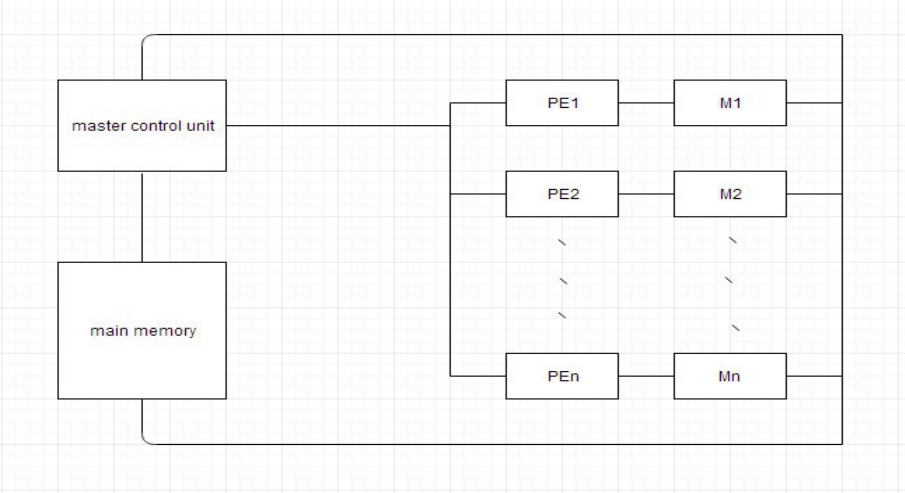


Fig.2.3A general block diagram of an array processor

2.7.2 Advantages of Array Processor

- An array processor increases the overall instruction processing speed.
- As most of the Array processors operate asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array processors have its own local memory, hence providing extra memory for systems with low memory.

2.8 PIPELINING

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

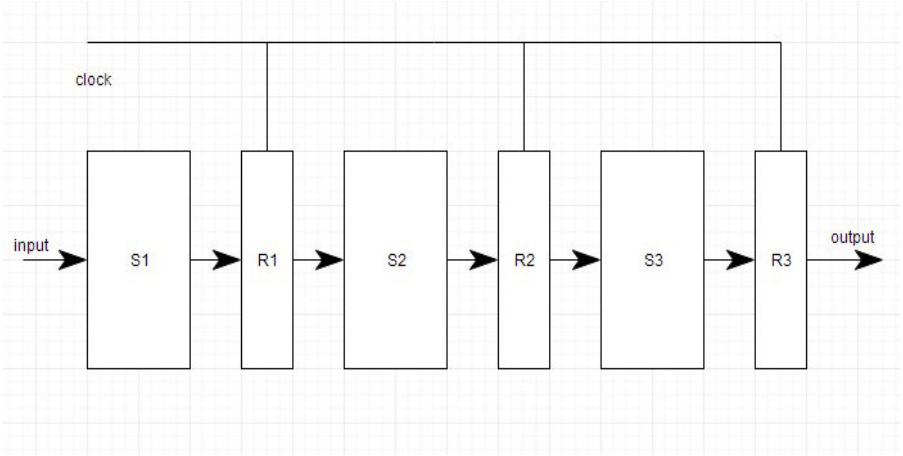


Fig.2.4 A general block diagram of a pipeline system

Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

In summary, we can say Pipelining is a technique of

- Decomposing a sequential process into sub-operations (segments).

Space for learners:

- Divide the processor into segment processors each one is dedicated to a particular segment.
- Each segment is executed in a dedicated segment processor operates concurrently with all other segments.
- Information flows through these multiple hardware 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.

Space for learners:

2.8.1 Types of Pipeline

It is divided into two categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

2.8.1.1 Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

Suppose X=A*2^a and Y=B*2^b

Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents.

The floating point addition and subtraction is done in 4 parts:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract mantissas
4. Produce the result.

Registers are used for storing the intermediate results between the above operations.

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the below diagram.

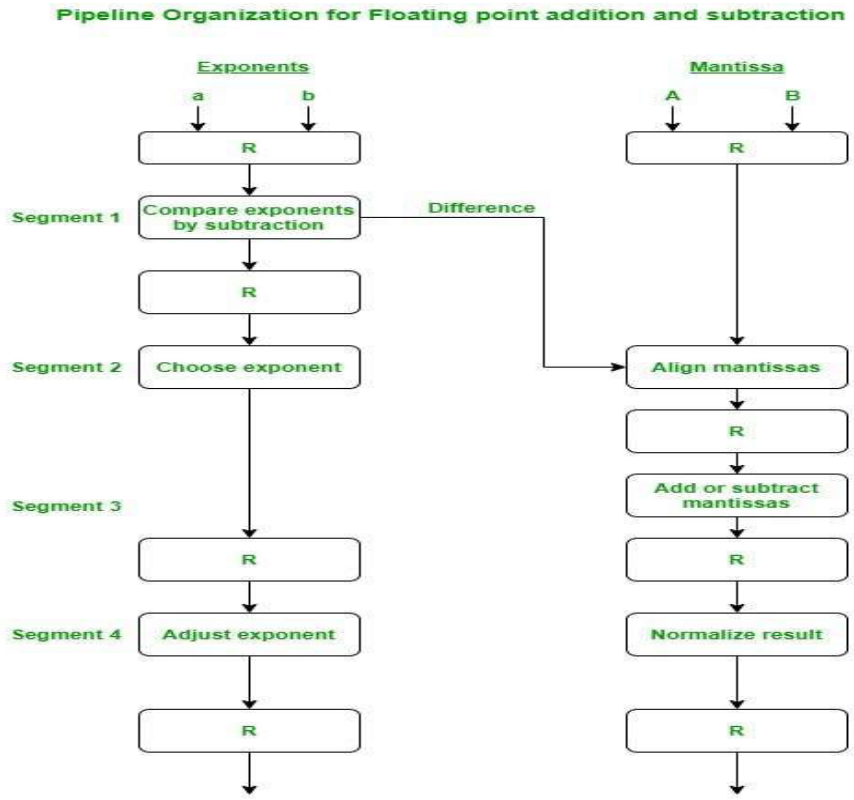


Fig 2.5 Pipelining for floating point addition and subtraction.

Floating point addition using arithmetic pipeline

The following sub operations are performed in this case:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalise the result

Space for learners:

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:
Let us consider two numbers,
X=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3

Finally the two numbers are added to produce

Z=0.3664*10^3

As the result is already normalized the result remains the same.

Space for learners:

2.8.1.2 Instruction Pipeline

In this 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 this 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 general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)
6. Store the result in the proper place

The flowchart for instruction pipeline is shown below.

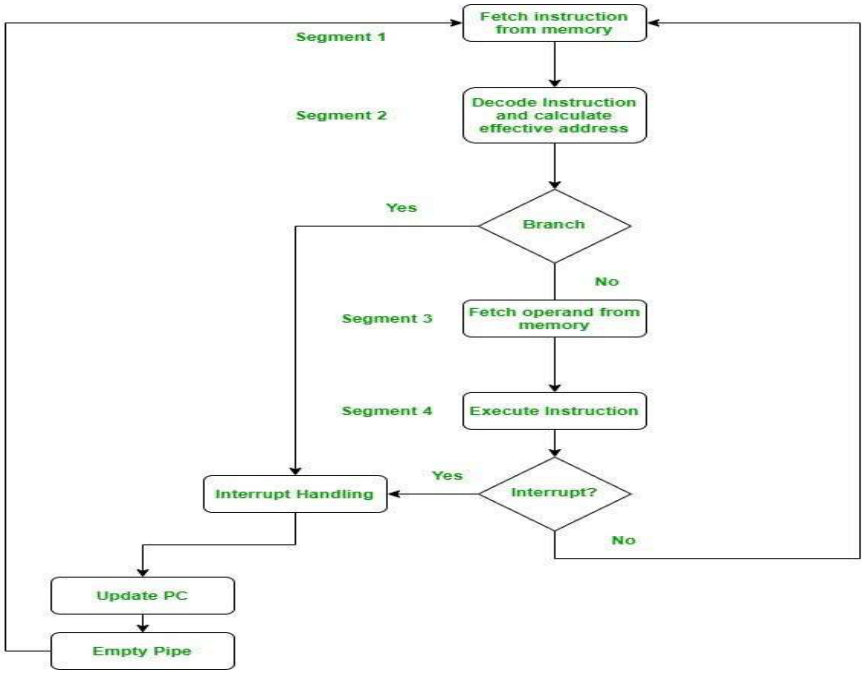


Fig 2.6 Flowchart for instruction Pipelining.

Space for learners:

Space for learners:

Let us see an example of instruction pipeline.

Example:

	Stage	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

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.

2.8.2 Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

(i) Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

(ii) Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

(iii) Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

(iv) Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

(v) Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

2.8.3 Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Space for learners:

2.8.4 Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

2.9 CHAINING TECHNIQUE

In computing, chaining is a technique used in computer architecture in which scalar and vector registers generate interim results which can be used immediately, without additional memory references which reduce computational speed.

Chaining allows the results of one vector operation to be directly used as input to another vector operation. A convoy is a set of vector instructions that can potentially execute together. Only structural hazards cause separate convoys as true dependences are handled via chaining in the same convoy.

2.10 GATHER-SCATTER OPERATION

Gather and scatter are two fundamental data-parallel operations, where a large number of data items are read (gathered) from or are written (scattered) to given locations.

Gather-scatter is also a type of memory addressing operation that often arises when addressing vectors in sparse linear algebra operations. It is the vector-equivalent of register indirect addressing, with gather involving indexed reads and scatter indexed writes. Vector processors (and some SIMD units in CPUs) have hardware support for gather-scatter operations, providing instructions such as *Load Vector Indexed* for gather and *Store Vector Indexed* for scatter.

Space for learners:

2.10.1 The basic concepts of Gather-scatter

Space for learners:

We are generally used to organizing our memories by row. Caches are built from rows so if we want one piece of data, we get the whole row. If we want to manage our performance tightly, then we try to have as many related variables as possible on the same row so that we get more bangs for our caching buck and reduce our cache misses.

The nice thing about a row of memory is that, especially with vector structures like SIMD (single-instruction, multiple data), we can operate on multiple pieces of data at the same time, in parallel. At the very least, if we can't do it in parallel, then we can loop along the row for the operation without further fetching hassles.

But there are several contexts where the world doesn't cooperate with this row-by-row structure. What if we want to be able to do is exactly that same thing, but without the requirement that addresses be contiguous?

This isn't so easy to do, since we need lots of fetches to populate a vector; we can't just copy over a chunk of memory and get busy operating on it. The idea is to find a way to "gather" data from far-flung locations, work with them as a single vector, and then, if we desired, take the results and "scatter" them back out into their original far-flung locations.

2.10.2 Different gather-scatter applications

Some application of gather-scatter operations are given below:

- A single block of in-memory data may represent data from a file that has been fractured into various sectors across the storage medium.

- A single in-memory buffer, if too large, may cause problems due to memory fragmentation. It can be more easily managed if it is stored in smaller fragments, but this requires management to make them look contiguous.
- Network traffic streams may be split up as they arrive, with various buckets in memory. This is referred to as "Scatter/gather I/O." In a way, this is the reverse of other applications. In other applications, scattered data is brought together in the processor. With this streaming version, it's a unified stream that then gets scattered about as it arrives at the processor.
- Embedded systems may require low-level access to data that's scattered throughout DRAM, treating it as contiguous. The illustrations above reflect this application. As we'll see, vision is a major driver of this usage.

Space for learners:

CHECK YOUR PROGRESS

Multiple Choice Questions:

1. A processor, which works on simple instruction at a time, which operates on single data items is known as

(A) Scalar (B) Vector
(C) Array (D) Superscalar

2.A processor that has the ability to execute the complete vector input in a single instruction is called

(A) Scalar (B) Vector
(C) Normal (D) Superscalar

3.In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from

(A) Main memory (B) Register
(C)Cache (D) Secondary memory

4. SIMD means

(A)Single Instruction Many Data
(B)Simple Instruction Multiple Data
(C)Single-Instruction, Multiple Data

(D) None of above

5. A technique where multiple instructions are overlapped during execution is known as

(A)Gathering (B) Scattering
(C) Chaining (D) Pipelining

Space for learners:

2.11 SUMMING UP

- Vector processor is basically a central processing unit that has the ability to execute the complete vector input in a single instruction. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.
- Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like *ADD A to B*, and store into *C* are not practically efficient.
- Vector computing technique, not only use instruction pipeline, but it also pipelines the data, working on multiple data at the same time.
- 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.
- Vector processing occurs when arithmetic or logical operations are applied to vectors. It is distinguished from scalar processing which operates on one or one pair of data. The conversion from scalar code to vector code is called vectorization.

- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector processors need high speed on-chip memory which are expensive.
- It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.
- The classification of vector processor relies on the ability of vector formation as well as the presence of vector instruction for processing. So, depending on these criteria, vector processing is classified as follows:
  (iii) Register to Register Architecture (Vector register processors) and
  (iv) Memory to Memory Architecture (Memory-memory vector processors)
- In memory to memory architecture, source operands, intermediate and final results are retrieved (read) directly from the main memory.
- Register to register architecture is highly used in vector computers. As in this architecture, the fetching of the operand or previous results indirectly takes place through the main memory by the use of registers.
- In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.
- Vector registers are the storage areas in a CPU core that contain the operands for vector computations, as well as the results. The size of the vector registers determines the level of SIMD instructions that can be supported by a given processor's CPUs.

Space for learners:

- Scalar registers can also provide data as input to the vector functional units, as well as compute addresses to pass to the vector load-store unit.
- An array processor is a CPU which implements an instruction set that are designed to operate efficiently and effectively on large one-dimensional arrays of data called vectors.
- An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks.
- 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.
- Single-instruction, multiple data(SIMD) is the organization of a single computer containing multiple processors operating in parallel.
- Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.
- Pipelining is a technique where multiple instructions are overlapped during execution.
- Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc.
- An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline.
- In computing, chaining is a technique used in computer architecture in which scalar and vector registers generate

Space for learners:

interim results which can be used immediately, without additional memory references which reduce computational speed.

- Gather and scatter are two fundamental data-parallel operations, where a large number of data items are read (gathered) from or are written (scattered) to given locations.

Space for learners:

2.12 ANSWER TO CHECK YOUR PROGRESS

1 (A),2 (B), 3 (A), 4 (C), 5 (D).

2.13 POSSIBLE QUESTIONS

Short Type Questions:

1. What is vector computing? How it differ from scalar computing?
2. What do you mean vector processor?
3. What are the advantages of vector processor?
4. What is array processor? What are its different categories?
5. What do you mean by pipelining in vector processing?

Long Answer Type Questions:

1. Explain about the arithmetic pipeline and instruction pipeline techniques.
2. Explain how vector-register processor differs from memory vector processor.
3. Explain about the working principle of array processor.
4. What do you mean by chaining? Explain about the scatter-gather techniques.

2.14 REFERENCES AND SUGGESTED READINGS

- M. Morris Mano, “Computer System Architecture”, 3nd Edition, Pearson,2006.
- William Stalling, ”Computer Organization and Architecture”, 8th Edition, Pearson, 2010.
- John P. Hayes, “Computer Architecture and Organization”, 2nd Edition, McGraw-Hill International Edition, 1988.

---×---

Space for learners:

UNIT 3: ADVANCED CONCEPTS OF COMPUTER ARCHITECTURE IMPLICIT PARALLELISM

Space for learners:

Unit Structure:

3.1 INTRODUCTION

Implicit parallelismallows programmers to write down their programs without any worry about parallelism exploitation. The exploitation of parallelism is instead automatically performed by the compiler and the runtime system. Thus, the parallelism is transparent to the programmer, maintaining the complication of software development at the same level as standard sequential programming. Implicit parallelism generally facilitates the design of parallel programs and therefore substantially improves programmer efficiency and productivity. Different applications utilize different aspects of parallelism - e.g., data-intensive applications utilize high aggregate throughput, server applications utilize high aggregate network bandwidth, and scientific applications typically utilize high processing and memory system performance. It is important to realize each of these performance bottlenecks and their interacting effect.

In this unit, you will learn about the pipelining technique, and the comparison/ discussion of super pipeline and super scalar pipeline will also be described in this unit. Various classes of superscalar architecture will be discussed in this unit. You will learn the measurement of the performance of pipeline architecture. Some of the benefits and drawbacks of the superscalar pipeline will be pointed out in this unit. You will learn the need for Branch prediction in the pipeline. Different branch prediction techniques (static and dynamic prediction) will be discussed with the proper example and diagram in this unit. You will learn various hazards (structural hazards, control hazards, and data hazards)that occur in the pipelining. Some of the delay slots will be discussed in this unit. You also learn the out-of-order execution and register renaming concept with an example at the last of the unit.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- understand the needs of implicit parallelism techniques.

Space for learners:

- describe the basic structure of the pipeline
- know different stages of instruction in the pipeline
- understand the design concept of super pipeline and superscalar pipeline
- understand the branch prediction logic
- understand the various Hazards in the pipeline
- know the idea of delay slots
- Describe out-of-order execution and Register Renaming

Space for learners:

3.3 INTRODUCTION OF PIPELINING

Pipelining is the practice of accumulating instruction from the processor through a pipeline. In pipelining, storing and executing of the instructions allows being in an orderly process. It is also known as pipeline processing. Multiple instructions are overlapped during execution in pipelining that's why process microprocessor begins executing a second instruction before the first instruction has been completed. A pipeline is separated into stages, and these stages are attached to one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining improved the overall instruction throughput.

In the pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and a combinational circuit performs operations on it. The output of the combinational circuit is applied to the input register of the next segment.

A processing circuit of a given stage is connected to the input latch of the next stage **(Figure 3.1)**. A clock signal is connected to each input latch. Every stage transfers its intermediate result to the input latch of the next stage on every clock pulse. This way, the final outcome is produced after the input data have passed through the whole Pipeline, finishing one stage for every clock pulse. The clock

pulse period should be large enough to grant sufficient time for a signal to go across through the slowest stage where the most extended amount of time to need complete (bottleneck stage). Also, there should be sufficient time for a latch to store its input signals.

If the clock's period, P, is expressed as $\mathbf{P = t_b + t_l}$, then $\mathbf{t_b}$ should be bigger than the utmost delay of the bottleneck stage, and t_l should be sufficient for storing data into a latch.

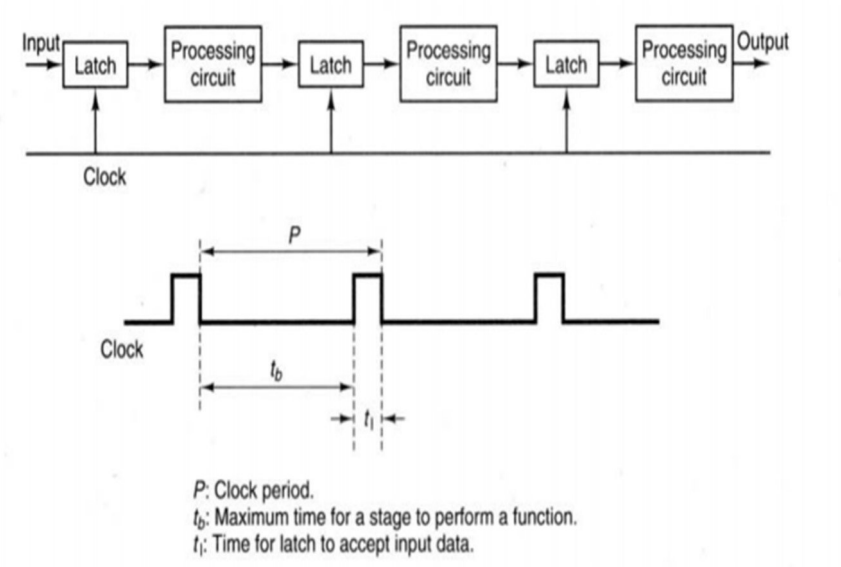


Figure 3.1: Basic structure of a pipeline. *adapted from [1]*

The instruction in pipelining is divided into five subtasks likely

1. Instruction Fetch (IF): In this subtask, the instruction is fetched.

2. Instruction Decode(ID): Here, the fetched instruction is decoded.

3. Operand Fetch (OF): In this stage, the operand is fetched of the instruction.

4. Instruction Execute (IE): In this stage,arithmetic and logical operations are performed on the operands to execute the instruction.

5.Operand Store (OS): The result of the earlier stage is stored in the memory.

Space for learners:

Let us visualize how pipelining is done for *N* numbers of instructions. In the **Figure 3.2** given below four instructions are pipelined. The instruction-1 gets completed in 5 clock cycles. After the first instruction is completed in every new clock, the proceeding instruction(i.e., Instruction 2, 3 & 4) completes its execution.

Pipeline system is like some assembly line set up in different factories. For example, in an automobile manufacturing industry, huge assembly lines are arranged and at each point, there are robotic arms to perform a particular task, and then the product moves on ahead to the next arm. Pipeline techniques are categories into 2 types. One is arithmetic pipeline, and the other is instruction pipeline.

- ✓ Arithmetic pipeline is designed to act upon high-speed floating-point addition, multiplication and division. Multiple arithmetic logic units (ALUs) are built to perform the parallel arithmetic computation in various data formats in this Pipeline. Examples of arithmetic pipelined processors are Cray-1, Cyber-205, Star-100, TI-ASC.Thefloating point addition and subtraction is done in 4 parts: Compare the exponents, align the mantissas, add or subtract mantissas and produce the result.
- ✓ In instruction Pipeline, the number of instructions are pipelined, and the subsequent instruction execution overlaps the execution of current instruction. It is also known as instruction lookahead.

Space for learners:

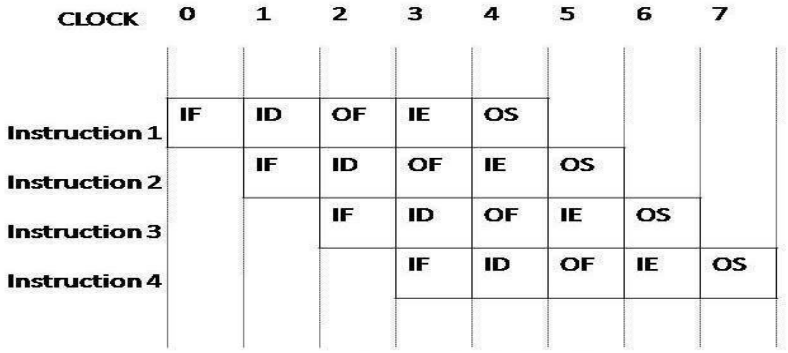


Figure 3.2: Pipelining of four Instructions

3.3.1 Register File

The register file is a hardware device with two read ports and one write port (corresponding to the two inputs and one output of the ALU). The register file and the ALU together comprise the two elements required to compute MIPS R-format ALU instructions. The register file is included of a set of registers that can be read or written by supplying a register number to be accessed, as well (in the case of write operations) as a write authorization bit. A block diagram of the register file is shown in Figure 3.3

Since reading of a register-stored value does not change the register state, no "safety mechanism" is needed to prevent inadvertent overwriting of stored data, and we need only supply the register number to obtain the data stored in that register. However, when writing to a register, we need:

1. A register number.
2. An authorization bit for safety (because the write operation overwrites the previous contents of the register selected for writing).
3. A clock pulse that controls the writing of data into the register.

Space for learners:

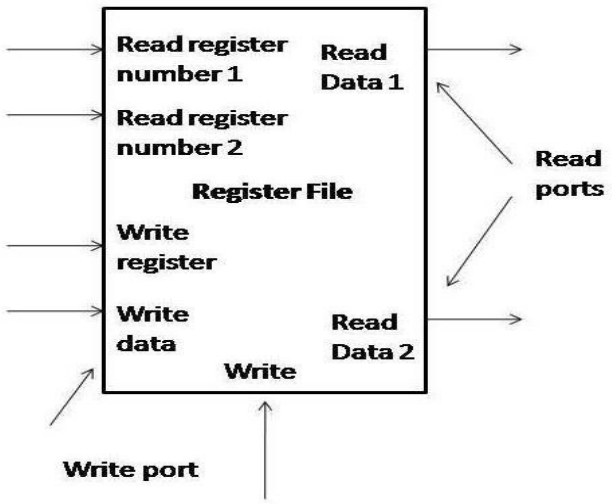


Figure 3.3. Register with two read ports and one write port,*adapted from [2].*

Space for learners:

3.3.2 Datapath Design

The datapath is the "brawn" of a processor, since it implements the fetch-decode-execute cycle. The general discipline for datapath design is to

a. Determine the instruction classes and formats in the ISA,
b. Design datapath components and interconnections for each instruction class or format, and
c. Compose the datapath segments designed in (Step 2) to yield a composite datapath.

Simple datapath components include *memory* (stores the current instruction), *PC* or program counter (stores the address of current instruction), and *ALU* (executes current instruction). Theinterconnection of these simple components to form a basic datapath is illustrated in Figure 3.4.

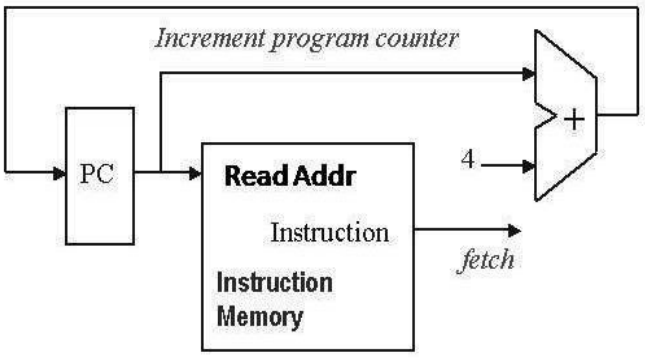


Figure 3.4: Schematic high-level diagram of MIPS datapath from an implementation perspective, *adapted from [2].*

3.4. SUPER PIPELINING

Super pipelining is another approach to reach better (faster) performance. Super-pipelining is the breaking of stages of a given pipeline into more miniature stages(thus making the pipeline deeper) to shorten the clock period and thus to enhance the instruction throughput by keeping more and more instruction in flight at a time.

For example, if we divide each stage into two, the clock cycle period t will be reduced to half, t/2; hence, at the maximum capacity, the pipeline produces a result every t/2 s.

For a given architecture and the subsequent instruction set, there is an optimal number of pipeline stages; increasing the number of stages over this boundary decrease the overall performance. Superscalar architecture is a solution to further improve speed. Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

Space for learners:

1. Further divide each of the pipeline stages into n sub stages.

This approach requires faster logic and the capability to subdivide the stages into segments with consistent latency. Here also Complex inter-stage interlocking and stall-restart logic are required.

2. Make available n pipelines that are overlapped.

This approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data could be fetched with a slight offset in time.

Unavoidably, super pipelining is controlled by the speed of logic, and the frequency of unpredictable branches. The Stage time cannot effectively grow shorter than the interstage latch time, and accordingly this is a limit for the number of stages. The MIPS R4000 is sometimes called a super pipelined machine, although its 8 stages really only split the I-fetch and D-fetch stages of the pipe and add a tag check stage. Nevertheless, the further stages enable it to operate with higher throughput. The UltraSPARC's 9-stage pipe definitely qualifies it as a super pipelined machine, and in fact it is a Super-Super design because of its superscalar issue. The Pentium 4 splits the pipeline into 20 stages to enable increased clock rate. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics. On more irregular applications, there is little performance advantage.

Space for learners:

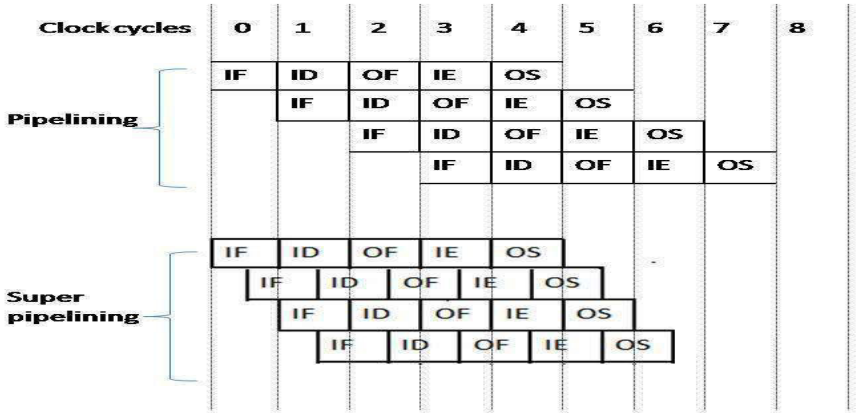


Figure 3.5: Comparison of normal pipeline and Super pipeline.

Space for learners:

3.5 PERFORMANCE OF A PIPELINED PROCESSOR

Consider a K segment pipeline with clock cycle time as T_p and N tasks to be completed in the pipelined processor.

Here, the first instruction is about to take K cycles to come out of the Pipeline but the other $N–1$instructions will take only one cycle each, i.e., a total of $N-1$ cycles. So, time is taken to N instructions in a pipelined processor:

$ET_P = K + N – 1$cycles
$= (K + N – 1)\ T_P$
In the same case, the Execution time of N instructions in a non-pipelined processor, will be:
$ET_{NP} = N * K * T_P$.

Here $\mathbf{ET_P}$ stands for estimate time taken in pipeline processor and $\mathbf{ET_{NP}}$ stands for estimate time taken in non- pipeline processor. Therefore, *speedup(S)* of the pipelined processor over the non-pipelined processor, when N tasks are executed on the same processor is:

$$S = \frac{\text{Performance of pipelined processor}}{\text{Performance of Non-pipelined processor}}$$

Since the performance of a processor is inversely proportional to the execution time, we have,

$S = ET_{NP} / ET_P$

$$\Rightarrow S = \frac{N * K * T_P}{(K + N - 1)\, T_P}$$

$$S = \frac{N * K}{(K + N - 1)}$$

We can ignore *(K-1)* When the number of tasks N are considerably larger than K, that is, $N >> K$

$$S = \frac{N * K}{N}$$

$S = K$, where K is the number of stages in the Pipeline.

Theoretically, maximum speedup ratio will be k where k are the total number of segments in which process is divided.

Again,

Efficiency = $\text{Given speed up} / \text{Max speed up} = S / S_{Max}$

We already know that $S_{Max} = K$

as a result, Efficiency = $\frac{S}{K}$

$$\textbf{Throughput} = \frac{\text{Number of instructions}}{\text{Total time to complete the instructions}}$$

hence, Throughput = $N / (K + N - 1) * T_P = N/T_P$

In ideal case as N -> 1 the throughout is equal to $1/ T_P$ that is equal to frequency. Thus

maximum throughput is obtained is there is one output per clock pulse.

Problem 1: A non-pipeline system takes 60 ns to process a task. The same task can be processed in six segment pipeline with a clock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speed up that can be achieved?

Space for learners:

Solution:
Total time taken by for non pipeline to complete 100 task is
= 100 * 60 = 6000 ns
Total time taken by pipeline configuration to complete 100 task is
= (100 + 6 –1) *10 = 1050 ns
Thus speed up ratio will be = 6000 / 1050 = 4.76
The maximum speedup that can be achieved for this process is = 60 / 10 = 6
Thus, if total speed of non pipeline process is same as that of total time taken to complete a
process with pipeline than maximum speed up ratio is equal to number of segments.

Space for learners:

3.6 SUPERSCALAR ARCHITECTURE

Superscalar architecture is a system of parallel computing used in many processors together. In a superscalar computer, the central processing unit manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle. It is achieved by feeding the different pipelines through several execution units within the processor. To successfully implement a superscalar architecture, the CPU's instruction fetching mechanism must intelligently retrieve and allot instructions. Otherwise, pipeline stalls may occur, resulting in execution units that are often inactive.

With each instruction that a superscalar processor issues, it must check the instruction's operands get in the way with the operands of any other instruction in flight. Once an instruction is independent of all other ones in flight, the hardware must also decide precisely when and on which available functional unit to execute the instruction. To fully utilize a superscalar processor of degree N must issue N instructions per cycle to execute in parallel at all times. This situation may not be accurate in every clock cycle. In that case, some of the pipelines may be stalling in a wait state. The simple operation latency should require only one cycle in a

superscalarprocessor, as in the base scalar processor. The superscalar processor depends more on an optimizing compiler to exploit parallelism to achieve a higher degree of instruction-level parallelism in program.

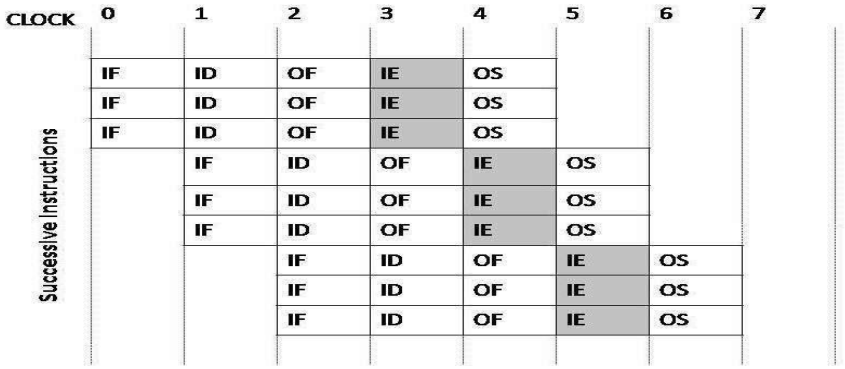


Figure 3.6: Pipeline structure of superscalar processor of degree-3

3.6.1 Structure Superscalar Architecture

Consider a machine organization capable of issuing more than one instruction per cycle depicted in Figure 3.7. Assume that the instruction set executed by the processor is $\mathbf{I = (I_1, I_2,.....I_N)}$ and that at most k instructions can be issued per cycle described by the k-tuple $\mathbf{P = (i_1, i_2.... i_k)}$, with $\mathbf{i_j \epsilon\ I,\ j = 1,2,....k}$. Furthermore, assume that at least k instructions are fetched into an instruction buffer and that a decision is reached on whether or not a k-instruction tuple can be issued and executed in parallel. This decision-making process is performed by the "Decode & Issue" logic. It is usually based on: the opcodes of the instructions, on availability of resources, and the structural and data dependencies.

Space for learners:

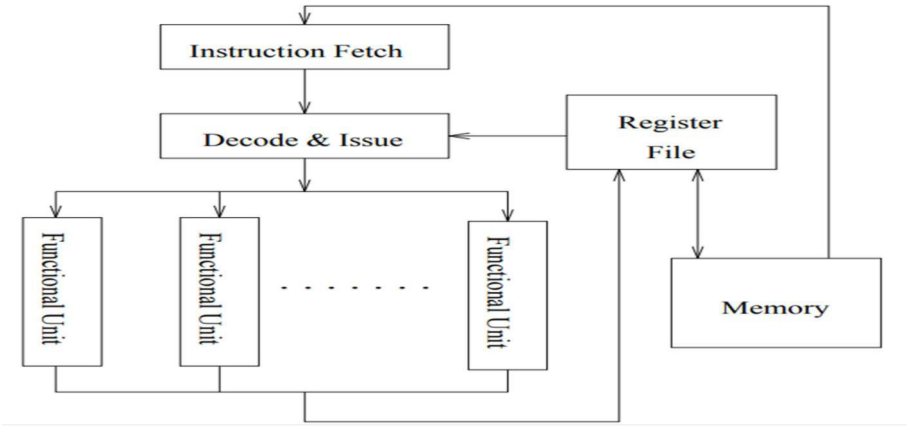


Figure 3.7: Basic Superscalar Architecture

We can classify superscalar processors into some classes of varying complexity.

1. Static Superscalar — This processor issue and execute program instructions in order. So, for example, in a degree 3 machine, it is possible to issue and execute three instructions simultaneously: given instructions i1,i2 and i3, we may choose to issue all, or only i1 (depending on the presence of hazards). We may *not* just issue i2 or i3. The instruction issues look dynamic because the hardware has a choice about issuing instructions. However, as the actual execution of instructions is in order, we state that scheduling is *static*.

2. Dynamic Superscalar — These types of machines permit out-of-order program execution, but they usually still *issue* instructions in program order. Since we can potentially re-order the execution, so we now state scheduling is dynamic.

3. Dynamic with Speculation — These machines add the capability to speculate beyond branches, using different techniques.

Space for learners:

3.6.2 Advantages of Superscalar Architecture

- The compiler can keep away from many hazards through well-judged choice and order of instructions.
- The compiler should make every effort to interleave floating-point instructions and integer instructions. This would facilitate the dispatch unit to maintain both the integer units and floating-point units active most of the time.
- On the whole, high performance is achieved if the compiler can arrange program instructions to take maximum assistance of the available hardware units.

3.6.3 Disadvantages of Superscalar Architecture

- In a Superscalar processor, the unfavorable effect on the performance of various hazards becomes even more pronounced.
- The problem in scheduling can occur because of this complex architecture.

CHECK YOUR PROGRESS-I

1. What is pipelining?
2. What are the 5 pipeline stages?
3. What is meant by ILP?
4. Define Superscalar processor.

3.7 BRANCH PREDICTION

The existence of program transfer instructions, e.g., JMP, RET, CALL, etc., can reduce the gain produced by pipelining. These instructions change the sequence causing all the other instructions that entered the pipelining after program transfer instructions are

Space for learners:

worthless. Thus no effort is done as the pipeline stages are reloaded.

To keep away from this trouble, Pentium uses a scheme called Dynamic Branch Prediction. In this process, a prediction is prepared for the branch instruction currently within the pipeline. The prediction will either be taken or not taken. If the prediction became true, then the pipeline will not be flushed, and no clock cycles will be gone astray. If the prediction is false, then the pipeline is flushed and starts over with the present instruction.

Mainly Branch Prediction predicts two problems one is Direction predicting and other one is calculating the target address.

Space for learners:

3.7.1 Types of Branch Prediction

Basically there are two types of Branch prediction schemes :

1. Static branch schemes and
2. Dynamic branch schemes.

3.8. STATIC BRANCH SCHEME

A static branch scheme is a software-based technique which is very simple and easy. This scheme assembles the more significant part of the data/information earlier to the program's execution or during the compile time and it does not need any hardware. In the Static branch prediction technique, underlying hardware assumes that either the branch is not always taken or the branch is always taken.

Let us understand branch prediction with an example code:

```
//Code
int  number=0;
while(number <10)
 {
   //branch instruction, condition either true or false
   if(number%3= = 0)
        {
    Some statement ;
    }
   number=number++;
 }
```

Output:

Let us consider that underlying hardware has assumed that branch is taken constantly. The output predicted through underlying hardware, and actual output is shown in **figure3.8.**

Number	0	1	2	3	4	5	6	7	8	9
Actual Prediction	T	NT	NT	T	NT	NT	T	NT	NT	T
Static Prediction	T	T	T	T	T	T	T	T	T	T
	✔	×	×	✔	×	×	✔	×	×	✔

Figure 3.8: Prediction result of static branch prediction

3.9 DYNAMIC BRANCH SCHEME

A dynamic branch scheme is hardware-based technique based on the hardware and assembles the information during the program's run-time. Dynamic schemes are more assorted as they keep track during run-time of the program execution. In Dynamic branch prediction technique, prediction by underlying hardware is not rigid, rather it

Space for learners:

changes dynamically. The accuracy of this technique is higher than the static technique.

Some of the dynamic branch prediction techniques are listed below:

a. 1-bit branch prediction technique
b. 2-bit branch prediction technique and
c. Correlating branch prediction technique

Space for learners:

3.9.1 1-Bitbranch prediction technique

In this technique, hardware changes its assumption immediately after one false assumption. For instance if hardware assumes branch to be taken but in reality, branch is not taken, then after that step, hardware assumes branch to be not taken. Similarly, if hardware assumes branch not to be taken but in reality, branch is taken, then after that step, hardware assumes branch to be taken.

1-bit branch prediction Technique is shown in the Fig 3.8 below:

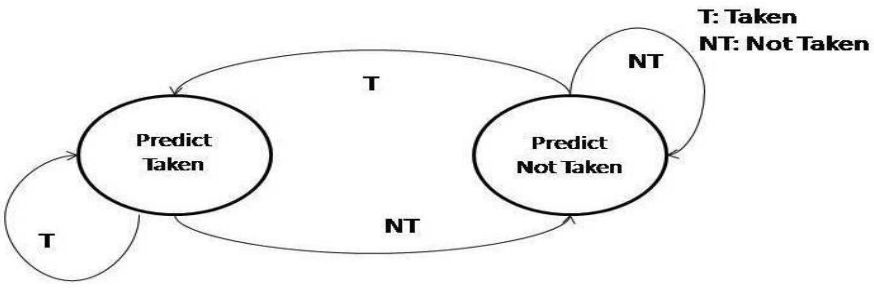


Figure3.9: Transition diagram of 1-bit prediction technique

Explanation –

In the beginning, let us declare hardware assume branch to be taken and so at number=0, branch is taken.
At number=1, hardware assumes branch to be taken but branch is not taken.

So now at number=2 hardware assumes branch not to be taken and also branch is not taken.

At number =3 hardware assumes branch not to be taken but branch is taken.

At number=4 hardware assumes branch to be taken but branch is not taken.

At number=5 hardware assumes branch not to be taken and branch is not taken.

In this way, the prediction is made till number=9.

The output predicted through underlying hardware, and actual output is shown in **Figure 3.9**:

Space for learners:

3.9.2 2-bit branch prediction technique

This predictor changes its earlier prediction only on two successive mispredictions occur and vice-versa. Two bits called as history Bit are maintained in the prediction buffer and there are 4-different states where Two states related to a taken state and two related to not taken state.

2-bit branch prediction technique is shown in the figure:

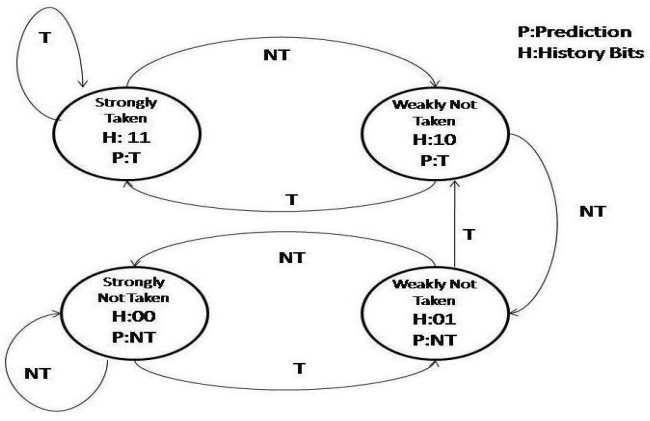


Figure 3.10: Transition diagram of 1-bit prediction technique

Explanation –

1. Let's assume that when number=0, everything is reset(00), so hardware assumes branch strongly not to be taken and the real branch is taken. As a result, the current state is (01).
2. When number =1, hardware assumes branch weakly not taken and in the real branch is not taken. Therefore the current state is (00).
3. When number =2, hardware assumes branch strongly not to be taken and branch is not taken in real. As a result, the current state remains (00).
4. When number =3, hardware assumes branch strongly not to be taken and in the real, branch is taken. As a result, the current state is (01).
5. When number =4, hardware assumes branch weakly not taken and in the real branch is not taken. So the current state is (00)
6. When number =5, hardware assumes branch strongly not to be taken and in the real branch is not taken. So the current state is (00). In this way, the prediction is done till number=9.

The comparison of the performance of Branch prediction schemes are given in the fig below:

Number	0	1	2	3	4	5	6	7	8	9
Actual Prediction	T	NT	NT	T	NT	NT	T	NT	NT	T
Static Prediction	T	T	T	T	T	T	T	T	T	T
	✔	×	×	✔	×	×	✔	×	×	✔
Output Predicted in 1-Bit Prediction	T	T	NT	NT	T	NT	NT	T	NT	NT
	✔	×	✔	×	×	✔	×	×	✔	×
Output Predicted in 2-Bit Prediction	NT 00	NT 01	NT 00	NT 00	NT 01	NT 00	NT 00	NT 01	NT 00	NT 00
	×	✔	✔	×	✔	✔	×	✔	✔	×

Figure 3.11: Comparison of performance of static,1-bit and 2-bit prediction scheme.

Space for learners:

3.9.3 Correlating Branch Prediction Technique

Due to interference with other branches, it is impossible to get significant accuracy from the 2-bit branch predictor. So correlating branch prediction comes into the picture whose prediction accuracy is enhanced as it considers the recent activities of other branches. It uses *k* least significant bits of branch target address which is fetched before. Also, it uses local history table (LHT), a table of shift registers where shift register refers to the last effect of m branches having similar k least significant bits. It also uses local prediction table(LPT) to predict the result depending on its present state.

CHECK YOUR PROGRESS-II

5. Define Branch prediction.
6. Define register file.
7. Define static branch prediction.
8. Define Dynamic branch prediction.

3.10 HAZARDS IN PIPELINING AND ITS TYPES

The situation that prevents the next instruction in the instruction stream from executing during its selected clock cycle is Hazard. Hazards decrease the performance from the ideal speedup gained by pipelining.

- **Structural Hazards**: Structural Hazards arises from resource conflicts when the hardware cannot support all possible combinations of instructions in the pipeline requiring the same resource due to some functional unit not being fully pipelined. Then the sequence of instructions using that un-pipelined unit cannot proceed at one per clock cycle rate. This generally isn't a problem with simple pipelines, but if some instructions take longer than

Space for learners:

others, this can become a problem. Another common way that it may appear is when some resources are not duplicated enough to allow all combinations of instructions in the Pipeline to execute. So by fully pipelining the stages and duplicating resources will avoid a structural Pipeline.

- **Data hazards**: Data hazards occur when instructions that show signs of data dependence modify data in different stages of a pipeline. This hazard cause delays in the pipeline., Data hazards occur when the Pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an un-pipelined processor. It can be minimized by a simple hardware technique called forwarding or by adding stalls.

  There are generally three types of data hazards:

  1) Read after Write (RAW)
  2) Write after Read (WAR)
  3) Write after Write (WAW)

  Let , there be two instructions I_1 and I_2, such that I_2 follow I_1.

  Then,

  - RAW Hazard occurs when instruction I_2 tries to read data before instruction I_1 writes it.
    E.g.,:

    I_1: R2 ← R1 + R3
    I_2: R4 ← R2 + R3

  - WAR hazard occurs when instruction I_2 tries to write data before instruction I_1 reads it.
    E.g.:
    I_1: R2 ← R1 + R3
    I_2: R3 ← R4 + R5

  - WAW hazard occurs when instruction I_2 tries to write output before instruction I_1 writes it.
    E.g.:

Space for learners:

I_1: R2 ← R1 + R3
I_2: R2 ← R4 + R5

WAR and WAW hazards occur during the out-of-order execution of the instructions.

Control hazards: It is caused by a delay between the fetching of instructions and decisions about changes in control flow (branches and jumps). Here instruction depends on the results of previous instruction in a way exposed by the overlapping of instructions in the pipeline. Control Hazards are also known as Branch Hazards. The simplest method to handle branches hazard is to freeze or flush the pipeline, holding or deleting any instructions after the branch until the branch destination is identified. In this case branch penalty is fixed and cannot be reduced by software. The other scheme is the predicted-not-taken or predicted-untaken and *delayed branch*. The number of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

3.11 DELAY SLOT

An instruction slot being executed devoid of the effects of a preceding instruction is known as a delay slot. The most familiar form is a particular arbitrary instruction located without delay after a branch instruction on a DSP or RISC architecture; this instruction will execute even if the prior branch is taken. In that way, by design, the instructions appear to execute in an incorrect or illogical order. It is usual for assemblers to automatically rearrange instructions by default, hiding the unease from assembly developers and compilers.[3]

Space for learners:

Load Delay Slot

In pipelined architecture, the load word instruction loads a word from memory to the specified register. The next instruction executes concurrently with the current instruction; if the following instruction uses the LW destination register as one of its source registers, it cannot continue before the LW data is fetched from memory and written back to the destination register; otherwise, it will read invalid data.

Branch Delay Slot

The branch delay slot is also a consequence of the branch hazard. A simple design would insert stalls into the pipeline after a branch instruction until the new branch target address is computed and loaded into the program counter. Each cycle where a stall is inserted is considered one branch delay slot. A more sophisticated design would execute program instructions that are not dependent on the branch instruction. This optimization can be performed in software at compile time by moving instructions into branch delay slots in the in-memory instruction stream if the hardware supports this. Another side effect is that special handling is needed when managing breakpoints on instructions and stepping while debugging within branch delay slot[3].

Space for learners:

3.12 OUT-OF-ORDER EXECUTION

A processor that executes the instructions one after the other may use the resources inefficiently, leading to poor performance of the processor. To improve the performance of the processor, this can be done in two ways. One is by executing different sub-steps of sequential instructions simultaneously, and others execute the instructions entirely simultaneously. Additionally, improvement in the processor can be achieved through out-of-order execution by

executing the instruction in a diverse from the original order they appear. Out-of-order execution can be achieved. The approach of an out-of-order execution,used in high-performance microprocessors.

This approach efficiently uses instruction cycles and minimized costly delay. As an alternative of the original order of the instructions, a processor will execute the instructions in an order of accessibility of data or operands. The processor will avoid being idle while data is retrieved for the next instruction in a program. In other words, a processor that uses multiple execution units completes the processing of instructions in the wrong order. For example, I_1 and I_2 are the two instructions where I_1 comes first, then I_2. In the out-of-order execution, a processor can execute I_2 instruction before I_1 instruction has been executed. This flexibility of allowing execution with less waiting time will improve the performance of the processor. The main benefit of the out-of-order processor is it avoids instruction waits when the data needs to perform an operation is unavailable.

STOP TO CONSIDER

Differentiate in-Order Execution from Out-of-Order execution.

Out-of-order execution is a situation in pipelined execution when an instruction is blocked from executing does not cause the following instructions to wait. It preserves the data flow order of the program.

In-order execution requires the instruction fetch and decode unit to issue instructions in order, which allows dependences to be tracked, and requires the commit unit to write results to registers and memory in program fetch order. This conservative mode is called in-order commit.

Space for learners:

CHECK YOUR PROGRESS-III

9. Define hazard and its types.

10. Define data hazard.

11. What is meant by delayed branch?

12. What is pipeline stall?

Space for learners:

3.13 REGISTER RENAMING

To deal with data dependences in pipelining between instructions by renaming their register operands is known as register renaming. *Renaming* replaces architectural register names by, in effect, value names with a new value name for each instruction destination operand. This process eliminates the name dependences (anti-dependences and output dependences) between instructions and automatically recognizes true dependences. An assembly language programmer or a compiler specifies these operands using *architectural registers* - the registers are explicit in the instruction set architecture.

The identification of true data dependences between instructions allows a more flexible life cycle for instructions. Maintaining a status bit for each value indicating whether or not it has been computed yet allows the execution phase of two instruction operations to be performed out of order when there are no true data dependences between them.

Being more explicit about the action precise by an instruction data dependences can be realized. The action specified by an instruction is more apparent if we illustrate instructions in terms of values rather than registers. We have to name the values in a manner that captures changes in register contents over time.

By replacing register names in all operands with value names, we can confine the intent of a sequence of instructions. For this, we use a table that records the value names assigned to each register name. Then we apply the following algorithm.

i. Replace each source operand with the most recent value name in the designated register column.
ii. Replace the destination operand with a new name and place the new name in the designated register column.

It is essential that **step i** is done first. Otherwise, when the same register is used both as a source operand and a destination operand, we indicate that the instruction execution phase cannot begin until its result is ready. This makes it impossible for the execution phase to begin.[4]

For an Example:

We will start with the following instructions.

MUL R6, R0, R2
DIV R4, R2, R0
ADD R0, R6, R2

Instruction	R0	R2	R4	R6	Renamed Instruction
Initial values	P0	P1	P2	P3	-----
MUL R6, R0, R2				P4	MUL P4, P0, P1
DIV R4, R2, R0			P5		DIV P5, P1, P0
ADD R0, R6, R2	P6				ADD P6,P4,P1

Space for learners:

3.13.1 Advantages of register renaming

The instructions with value names capture the intent of a sequence of instructions by specifying relationships between register values rather than just registers. This simplifies determining when the execution of an instruction can begin. We only need to check if its source operand values have been computed. The name dependencies no longer complicate the picture.

To determine when source operands have been computed, the value registers contain a status bit in addition to a data value. The status bit is initialized to 0 (not ready) when the value register is allocated for an instruction. It is set to 1 when a functional unit writes a result.[4]

Space for learners:

3.14 SUMMING UP

In this unit, efforts have been made to acquaint you with the advanced concept of computer architecture implicit parallelism. Here in this unit, you learn the basic pipeline structure. Pipeline techniques are categories into two types. One is arithmetic pipeline, and the other is instruction pipeline. Super-pipelining is the breaking of stages of a given pipeline into more miniature stages to shorten the clock period and thus to enhance the instruction throughput by keeping more and more instruction in flight at a time. Superscalar architecture is a system of parallel computing used in many processors together. Here the central processing unit manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle. Here in this unit you learned the different branch prediction techniques. In the static branch prediction technique, underlying hardware assumes that either the branch is not always taken or the branch is always taken. A dynamic

branch scheme is hardware-based technique based on the hardware and assembles the information during the program's run-time. The comparison of static prediction,1- bit branch prediction and 2-bit branch prediction also elaborate with an example. Hazard is the situation that prevents the next instruction in the instruction stream from executing during its selected clock cycle. An instruction slot being executed devoid of the effects of a preceding instruction is known as a delay slot. Here we discuss the out-of-order execution that avoids instruction waits when the data needs to perform an unavailable operation. At the last of the unit you learned the register renaming process, which deals with data dependences.

Space for learners:

3.15 ANSWERS TO CHECK YOUR PROGRESS

1. "Pipelining, also known as "pipeline processing", is the process of collecting instruction from the processor through a pipeline. It stores and executes instructions in an orderly process."

2. The 5 stages of instruction execution in a pipelined processor are:
   a. Instruction Fetch (IF)
   b. Instruction Decode(ID)
   c. Operand Fetch (OF)
   d. Instruction Execute (IE)
   e. Operand Store (OS)

3. Pipelining exploits the potential parallelism among instructions. This parallelism is called instruction-level parallelism (ILP). There are two primary methods for increasing the potential amount of instruction-level parallelism. a. Increasing the depth of the pipeline to overlap more instructions. b. Multiple issue.

4. There are processors which are capable of achieving an instruction executing throughput of more than one instruction per cycle. They are known superscalar processor.

5. It is a technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken.

6. The processor‘s 32 general-purpose registers are stored in a structure called a register file. A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer.

7. The branch prediction decision is always the same every time a given instruction is executed. Any approach that has this characteristic is called static branch prediction.

8. The branch prediction where decision may change depending on execution history is called Dynamic branch prediction.

9. Any condition that causes the pipeline to stall is called a hazard. Its types are:

a. Data hazard
b. Instruction hazard
c. Structural hazard

10. A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.

11. Delayed branch is a type of branch where the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.

12. Pipeline stall, also called bubble, is a stall initiated in order to resolve a hazard. They can be seen elsewhere in the pipeline.

Space for learners:

3.16 POSSIBLE QUESTIONS

Space for learners:

Multiple Choice Questions:

1. Arithmetic Pipeline is used for
   a. floating point operations
   b. integer operations
   c. character operations
   d. None of the above

2. Which of the following is not a Pipeline Conflicts
   a. Timing Variations
   b. Branching
   c. Load Balancing
   d. Data Dependency

3. How many types of Pipelining exist
   a. 2
   b. 3
   c. 4
   d. 5

4. Which of the following is disadvantage of Pipelining
   a. cycle time of the processor is reduced.
   b. The design of pipelined processor is complex and costly to manufacture.
   c. The instruction latency is more.
   d. Both b and c

5. Which of the following is an advantage of pipelining
   a. Instruction throughput increases.
   b. Faster ALU can be designed when pipelining is used.
   c. Pipelining increases the overall performance of the CPU.
   d. All of the above

6. In arithmetic pipeline, the floating point addition and subtraction is done in _____ parts.
   a. 2
   b. 3
   c. 4
   d. 5

7. _____ have been developed specifically for pipelined systems.
   a. Utility software
   b. Speed up utilities
   c. Optimizing compilers
   d. None of the above

8. The pipelining process is also called as _____
   a. Assembly line operation
   b. Von Neumann cycle
   c. Superscalar operation
   d. None of the above
9. Each stage in pipelining should be completed within _____ cycle.
   a. 1
   b. 2
   c. 3
   d. 4
10. The periods of time when the unit is idle is called as _____
   a) Stalls
   b) Bubbles
   c) Hazards
   d) Both Stalls and Bubbles

Fill in the blanks:

1. The situation wherein the data of operands are not available is called _____.
2. The contention for the usage of a hardware device is called _____
3. Each stage in pipelining should be completed within _____ .
4. The fetch and execution cycles are interleaved with the help of _____.
5. The pipelining process is also called as _____ .
6. A pipeline _____ is a delay in execution of an instruction in order to resolve a hazard.

Space for learners:

7. The number of stalls introduced during the branch operations in the pipelinedprocessor is known as _____ .

Short answer type questions:

1. What do you mean by implicit parallelism?
2. Write a brief note on pipelining.
3. Explain Basic structure of pipelining technique.
4. Write short notes on arithmetic pipeline and instruction pipeline.
5. How do you calculate performance of a pipeline.
6. Short note on super pipelining techniques.
7. Differentiate between normal pipeline and super pipeline.
8. Compare super pipeline with superscalar architecture.
9. What are the advantages and disadvantages of superscalar architecture?
10. write down different type of superscalar processor.
11. Why do we require branch prediction?
12. What are the types of branch prediction scheme?
13. What is static branch prediction?
14. What is dynamic branch prediction?
15. What are the different types of dynamic branch prediction?
16. Write a short note on correlating branch prediction.
17. What do you mean by hazards in pipeline ?
18. What are the different types of hazard in the pipeline?
19. What do you mean by delay slot.
20. What is the need of register renaming?

Long answer type questions:

1. Explain pipeline structure with diagram.
2. How many sub-tasks of instruction are there in pipeline. Explain.

Space for learners:

3. Explain super pipeline technique. What are the benefit over normal pipeline.
4. Explain basic structure of superscalar architecture.
5. Explain 1-bit branch and 2-bit branch prediction technique with example.
6. Explain different type of hazards occurs in pipeline.
7. Explain data hazard with their types.
8. Explain different delay slots in pipeline.
9. Describe out-of-order execution.
10. What is register renaming? Explain how register renaming is done.

Space for learners:

3.17 REFERENCES AND SUGGESTED READINGS

[1] Pipelining. cs.siu.edu/~cs401/Textbook/ch3.pdf

[2] *Course Notes,* Mafla, E. *CDA3101*, at cise.ufl.edu/~emafla/

[3] Delay slot - WikiMili, The Best Wikipedia Reader. wikimili.com/en/Delay_slot

[4] Register Renaming ,University of Minnesota,d. umn.edu/ ~gshute/arch/register-renaming.html

[5] "Advanced Computer Architecture" Hwang ,Publisher Tata McGraw-Hill Education, 2003 ISBN:007053070X, 9780070530706

[6] "Computer Organization and Design – The Hardware / Software Interface", David A. Patterson and John L. Hennessy, 4th.Edition, Morgan Kaufmann, Elsevier, 2009.

[7] "Computer system Architecture", Mano, M. Morish, 3rd Edition, Pearson Education,1993

UNIT 4:ADVANCED CONCEPTS OF PIPELINE SCHEDULE

Space for learners:

Unit Structure:

Space for learners:

4.1. INTRODUCTION

In this unit, you will get to learn in detail about pipelining scheduling which is a very important concept of parallelism in computer organization and architecture (COA). As you already know that in pipelining, the instructions are accumulated through a pipeline from the processor. Many instructions are overlapped with each other. Performances of the CPU are improved due to the use of pipelines. So we will discuss the main concepts of pipelining through the dynamic scheduling approaches.

After going through the chapter, you will get to learn some of the important concepts of pipelining scheduling such as

- **DATA DEPENDENCY** – Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged.
- **SCOREBOARD** – When the data dependencies are not present and when sufficient resources are present in the system, the score-boarding technique allows the execution of out-of-order performances.
- **SLACK TIME** – When the time of a process gets delayed without other processes getting delayed, is termed slack time.
- **RATE MONOTONIC** – Rate monotonic (RM)is a type of static scheduling algorithm in which the instructions that have

the smallest job or rate are given more priority than the bigger jobs.

Space for learners:

4.2. UNIT OBJECTIVES

Studying this unit, you will be able to:

- Understand the concept of pipelining and pipeline scheduling.
- Discuss the different data structures related to pipelining processes.
- Know the different dynamic and static scheduling algorithms.

4.3. PIPELINING

Pipelining is a technique where instruction overlapping occurs at the time of execution. Instructions are accumulated from a processor into the input registers through a pipeline, and therefore this process is known as pipelining. The order in which the instructions are stored and executed is defined as the pipelining processing [1]. Different stages are linked together to form a single-stage pipeline and the instructions enter through one end of the pipeline and come out through the other end. Each stage of the pipeline consists of some input registers which hold the instructions at every stage and are then operated by some combinational circuit. When a combinational circuit works on a register, the output of it is shifted to the next registers present in the lined-up segments. All the instruction inside the pipeline works concerning some clock time[1].

4.3.1. Types of Pipeline

Since instructions encountered in the pipeline are of different types, so to cope up with this situation, the pipeline is split into two types. They are Arithmetic Pipelining and Instruction Pipelining, which are explained as follows [1]:

4.3.1.1. Arithmetic Pipelining

When arithmetic operations come as instruction into the pipeline they are then stored in the Arithmetic Pipeline. Arithmetic operations may include addition, subtraction, operations on floating-point numbers, etc. [1].

4.3.1.2. Instruction Pipelining

Instruction pipelining helps in increasing the throughput of the system. Fetch, execute and decode instructions are overlapped in the instruction cycle. When a new instruction is present in the memory then it is read by an instruction pipeline, and the instructions that were already existing are executed in the segments present in the pipeline. The efficiency of the pipeline will increase if the instruction cycle is split into the equal time clock. By doing this we execute multiple instructions simultaneously. That is, we can say that parallel processing occurs in the pipeline thus increasing the efficiency of the system along with an increasing throughput [1].

4.4. PIPELINING PROCESSOR

Depending upon the work it follows, the pipelining processors are divided into two categories, one is the scalar processor and the other is the vector(array) processor [1].

4.4.1. Scalar Processor

The simple processor which executes one instruction at a time and that too simple instructions are known as the scalar processor. But as it

Space for learners:

works on single instruction at a time therefore it proves to be an inefficient processor. The speed of the processor is also very slow.

For example, we need to add two numbers and store the answer in the third location which requires only simple calculation[1].

ADD B, D and store it in E.

Space for learners:

4.4.2. Vector (Array) Processor

When complex instructions are executed on numerous data synchronously, then a vector processor is used. This processor executes the instructions very fast as compared to the scalar processor and has much efficiency.

In the Instruction pipelining, at a particular time, different works are performed by the processor on the different data. Vector processor uses the instruction pipeline for data processing. Here the CPU remains busy all the time[1].

4.5. ADVANTAGES OF PIPELINING [1]

- Using pipelining, the total time of the processor's instruction cycle gets reduced thereby increasing the throughput of the instruction. In an actual case, multiple instructions are executed simultaneously and it looks like that the total time gets reduced.
- The time delay in between two instructions is greatly reduced hence increasing the throughput.
- Nowadays, for a faster and more complex Arithmetic and design Logic Unit the pipeline is developed into several stages.
- Performance of the pipeline increases, meaning the clock cycle also increases.
- The speed at which the clock cycle of the RAM works is much lower than the clock cycle of the pipelining processor hence increasing the performance.

4.6. DISADVANTAGES OF PIPELINING [1]

- Branching delay can occur in a pipelined processor. For reducing this branching delay, address of the target branch need to be pre-fetched at the stage of decoding. Doing this the delay occurred may be reduced until 1 clock cycle.
- The flip-flops that are inserted between the data modules increase the latency in the instructions.
- In pipelined processing, you may get some unexpected performances.
- When there are many branches in the stages of the pipeline, then the throughput gets reduced.
- Memory delay can occur in the pipelined processor. Cache miss occurs when searched data or instructions are not present in the cache memory and therefore searched in the main memory which then consumes more number of the instruction cycle. This is known as the Memory delay which becomes the reason for the delay for the other data or instructions that are lined up.
- When the pipeline does not validate the assumptions of the instructions, then incorrect behavior of the program might occur, which leads to hazards.

Space for learners:

4.7. PIPELINING SCHEDULING

Pipeline scheduling is a type of mechanism where executions are overlapped for different inputs and the computations are performed at different stages. It improves the performances of the machine that have parallel instructions usually termed as instruction pipelines. Let us explain this pipelining scheduling with the help of the following example. Suppose you have to manufacture a washing machine by developing two models[2].

a) **For model 1**, suppose you have designed the washing machine in such a way it washes (W), dries (D), and iron (I) one cloth at a time (T). That is, for performing the mentioned operations on(n) number of clothes, the time required would be **(n.T)**.

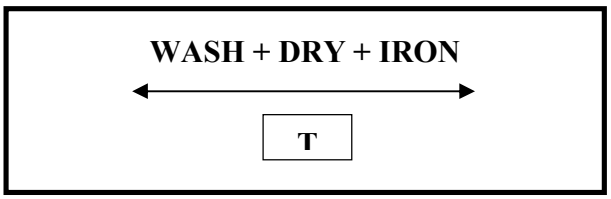


Figure 4.1: Model 1 for pipelining example

b) **For model 2**, suppose you have split the work of one washing machine into different machines that can wash (W), dry (D), and iron (I) the clothes separately. For each separate machine, the mentioned work is performed on more than one number of clothes in time (T). Now the time, that is required by **each** machine to perform the above task (for 1 cloth at a time) will be (T/3)[2]. And the time required for performing the operations on (n) number of clothes will be

$$\{T_3 = (2 + n).\ T/3\}.$$

For a larger number of clothes, (2 + n) will become 'n'. Then the time required will be

$$\{T_3 = n.\ T/3\}$$

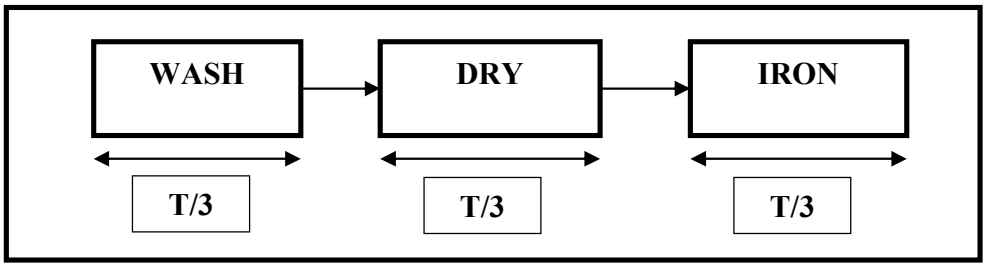


Figure 4.2: Model 2 for pipelining example

Here, model 2 explains the pipelining process. Let us explain this as follows[2].

- For time T/3, the cloth is washed in the machine.
- After it finishes the process of washing, it enters the second stage that is the dry stage, and works there for time T/3. When

Space for learners:

the second cloth was in the dry stage another cloth has entered the washing stage and took the same time T/3. It is being pipelined.

- When the first cloth entered the iron stage for time T/3, the second cloth is in the dry stage for time T/3 (it is being pipelined) and the third cloth is in the washing stage for time T/3 (it is being pipelined).
- Then cloth one finishes its task, cloth two is pipelined and enters into the iron stage. Meanwhile, cloth 3 enters into the dry stage and new cloth 4 enters into the washing stage.
- Simultaneously all the machines are working having each time T/3.
- In this way, all the machines are working keeping themselves busy without remaining idle.
- By keeping in mind the formula (**T_3 = (2 + n). T/3), where n is several clothes.**
- We can say that Cloth one took (for all the three stages) time

  T = (2 + 1) T/3 = **3T/3**

- Cloth two took time; T = (2 + 2) T/3 = **4T/3**
- Cloth three took time; T = (2 + 3) T/3 = **5T/3**and so on.

This explains the pipelining process.

4.7.1. Data Dependency

Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged. As said, there are three types of data dependencies[4].

- **Read After Write (RAW)** – At first, suppose Instruction 1 writes an instruction. That same instruction is read by Instruction 2 later. After Instruction 1 writes the value, then only Instruction 2 will read, therefore Instruction 1 must be written first otherwise instead of reading the new value, Instruction 2 will read the old value.

Space for learners:

- **Write After Read (WAR)** – At first, the location of a value is read by Instruction 1. After that Instruction 2 again rewrites the value. Instruction 1 must be written first, otherwise, instead of reading the new value, Instruction 2 will read the old value.
- **Write After Write (WAW)** – Both Instruction 1 and 2 when write a value in the same location, then this dependency is termed as write after write and it must be in the same order as the original order.

Space for learners:

4.8. DYNAMIC SCHEDULING

At the time of compilation, sometimes some dependencies occur in the system and we are unable to recognize these dependencies. In this case, handling of the dependencies is performed by the dynamic scheduler and hence the process is termed dynamic scheduling. For instructions having simple pipelining techniques, the major drawback is that all instructions are scheduled in some order, and once the instructions are pipelined then no new instructions or instructions after the scheduled instructions can be executed earlier than the pipelined instructions. If two or more instructions are spaced closely and have the same dependencies, then it might so happen that the instructions might come to a halt or become idle[3]. When hardware is taken into account, dynamic scheduling comes into force.

4.8.1. Out of Order Completion

The WAR and WAW hazards create the possibility of out-of-order execution. For handling the exceptions, major complications are created by the out-of-order completion. There are two possible cases where non-precise exceptions might occur[3].

- Suppose there are many instructions in a pipeline and it may so happen that one instruction present in the pipeline can cause exceptions. The possibility of a non-precise exception might occur when instructions that are present after the 'exception instruction' have been executed first[3].

- Another possibility might occur where there are many instructions present in a pipeline and it may so happen that one instruction present in the pipeline can cause exceptions. The possibility of a non-precise exception might occur when some instructions present in the pipeline before the 'exception instruction' are not executed at all[3].

Execution of out-of-order is allowed if the five stages pipeline is transformed into two stages in the following ways[3].

- Issue – Instruction decoding and to check whether any structural hazards are present in the pipeline or not.
- Read operands – The pipeline will wait till no data hazards are encountered and then the operands will be read.

For the dynamic scheduling, the instruction in the pipeline should pass in an ordered way through the Issue stage and then into the read operands, which is the second stage.

4.8.2.Dynamic Scheduling Algorithms

As the name suggests, a dynamic scheduler helps in making efficient decisions during the runtime of the system. Therefore, the system that works on dynamic scheduling is more flexible but at the same time calculation overhead also occurs. It checks which instruction has the most priority than the other and simultaneously works on that instruction at first. As it takes instruction during the runtime therefore the priority of executing the instruction might also change accordingly[5]. There are many dynamic scheduling algorithms based on different approaches, some of them are discussed below.

4.8.2.1. Earliest Deadline First (EDF)

EDF is a type of dynamic scheduling algorithm in which the instructions that have the nearest deadline to complete are given the task of highest priority and are executed first. When the current process gets completed and new processes are scheduled then this algorithm is worked upon. It is applicable for real-time systems. The CPU is utilized fully making sure that all the tasks are completed. An optimal feasible schedule is processed where all the tasks are executed within the

Space for learners:

stipulated deadline. The task must mention its deadline once it is made ready for execution and given a fixed CPU burst timing. Preemption can occur in EDF, and any instances that are scheduled for later but are engaged with an earlier deadline get ready for execution and becomes active[5].

But there are some limitations of the Earliest Deadline First Algorithm such as

- Overloading problems for the transient might occur.
- There might be some problems when resources are shared.
- Sometimes implementations are not done efficiently.

Let us explain the EDF algorithm with the help of an example by taking a flowchart[5].

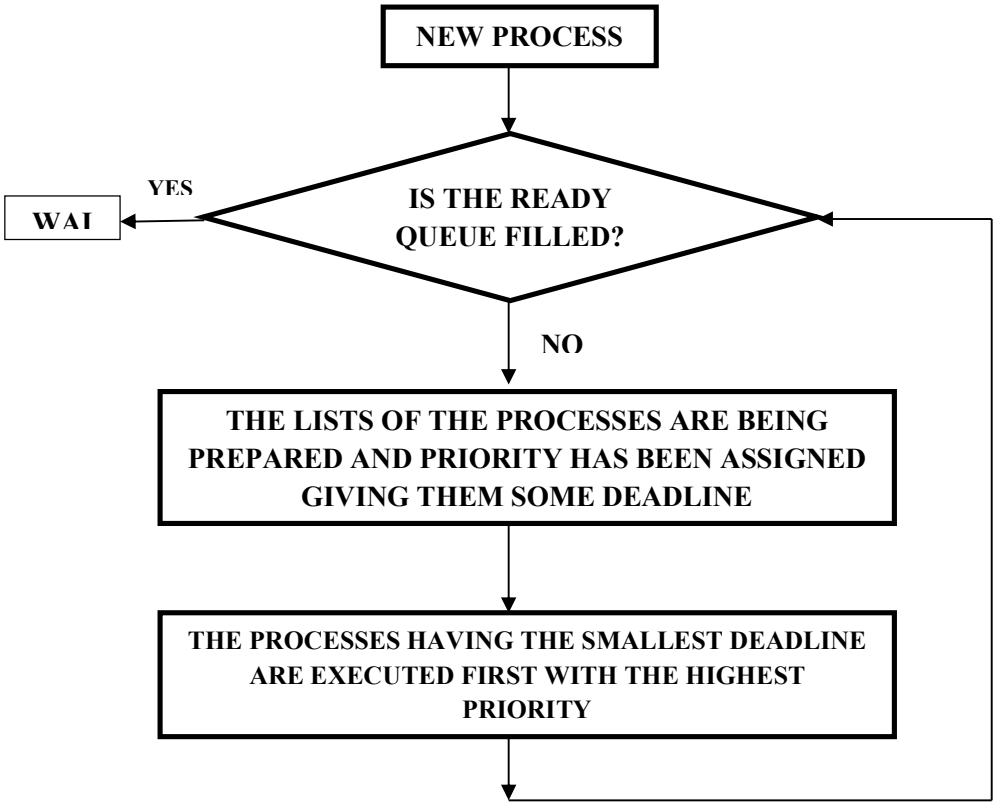


Figure 4.3: Flowchart of the Earliest Deadline First Algorithm

Space for learners:

4.8.2.2. Least Slack Time First (LST)

LST is a type of dynamic scheduling algorithm in which the instructions that have the smallest slack time are given the task of highest priority and are executed first. When the time of a process gets delayed without other processes getting delayed, is termed as the slack time. Like that of the EDF, when the current process that has the lowest slack time gets completed and new processes are scheduled then this algorithm is worked upon. For the slack time to be given as *l,* starting time is given as *t,* deadline interval is given to be *d,* and the remaining execution time is given to be *c*, the formula is given as ($l = d - c - t$) [5]. The algorithm is somewhat complex therefore requires extra information like the deadline and the execution timing. In real-time systems, it is sometimes difficult to predict the burst time. If the processes have the same slack time, then first cum first serve (FCFS) algorithm is applied together with LST.

Let us explain the LST algorithm with the help of an example by taking a flowchart [5]

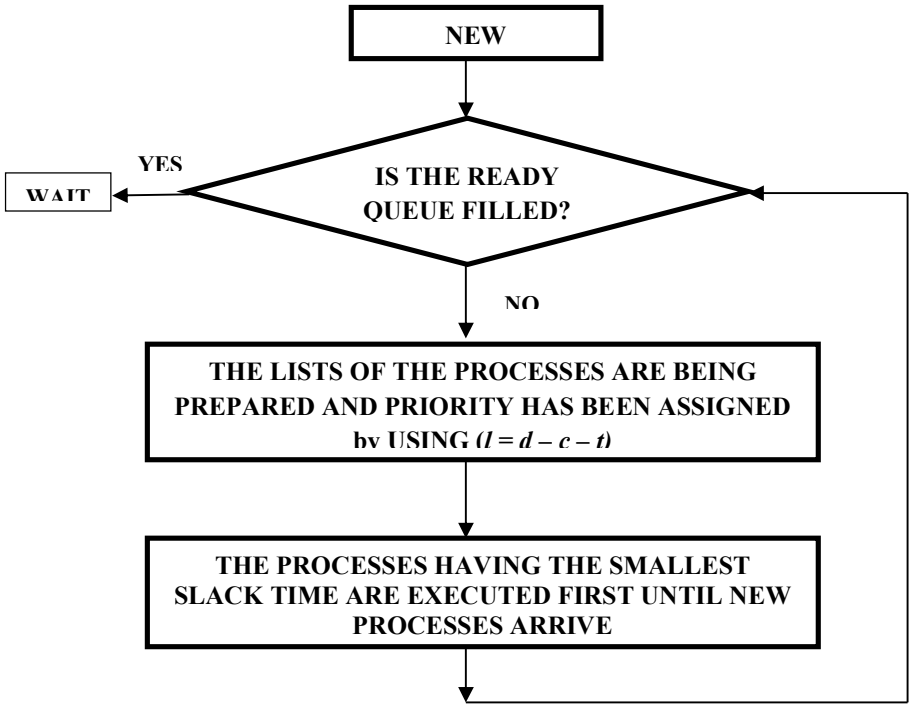


Figure 4.4: Flowchart of the Least Slack Time Algorithm.

Space for learners:

Space for learners:

4.8.3. Advantages of Dynamic Scheduling[6]

- Unknown dependencies during compile time are handled by dynamic scheduling because memory references are included.
- It simplifies the performance of the compiler.
- Codes on a pipeline are compiled so efficiently that they can run on different pipelines.
- Hardware speculations are often built on dynamic scheduling.

4.8.4. Disadvantages of Dynamic Scheduling[7]

- The complexities of the hardware increase substantially.
- Dynamic scheduling surely complicates exception handling.
- WAW and the WAR dependencies are created for out-of-order execution as well as out-of-order completion.

4.9. STATIC SCHEDULING

In static scheduling, all the processes are fixed for a particular stage of the pipeline. Before the execution takes place, the processes are given the tasks. They are usually processor non-preemptive. The overall time of the execution is minimized by the static algorithm. It tries to indicate the behavior of the execution of the program such as the execution time, process, and communication delays during the compile time. The smaller tasks are partitioned for reducing the communication costs. Processes are allocated to the processors. Static scheduling has a more efficient execution time environment as compared to the dynamic scheduling algorithm [5].

4.9.1. Static Scheduling Algorithms

Just like the dynamic scheduler, the priority scheduler works on the tasks that have more priority than the other but the value of the priority does not change. The static scheduler can make an efficient decision before runtime as well. There are many static scheduling algorithms, some of them are discussed below[5].

4.9.1.1. The Rate Monotonic (RM)

RM is a type of static scheduling algorithm in which the instructions that have the smallest job or rate are given more priority than the bigger jobs. The size or rate of the job is already scheduled in the RTOS. When the current process that has the smallest job gets completed and new processes are scheduled then this algorithm is worked upon[5]. The priorities are assigned just before the execution and remain the same throughout its execution period. Rate monotonic works based on the preemption, that is, during the execution time, if a shorter job is encountered by the system, then that job is given more priority for the execution. A job that has more time period has less priority and a job that has a lesser time period have more priority. The implementation of it is very much easy.

Let us explain the rate monotonic algorithm with the help of an example by taking a flowchart [5]

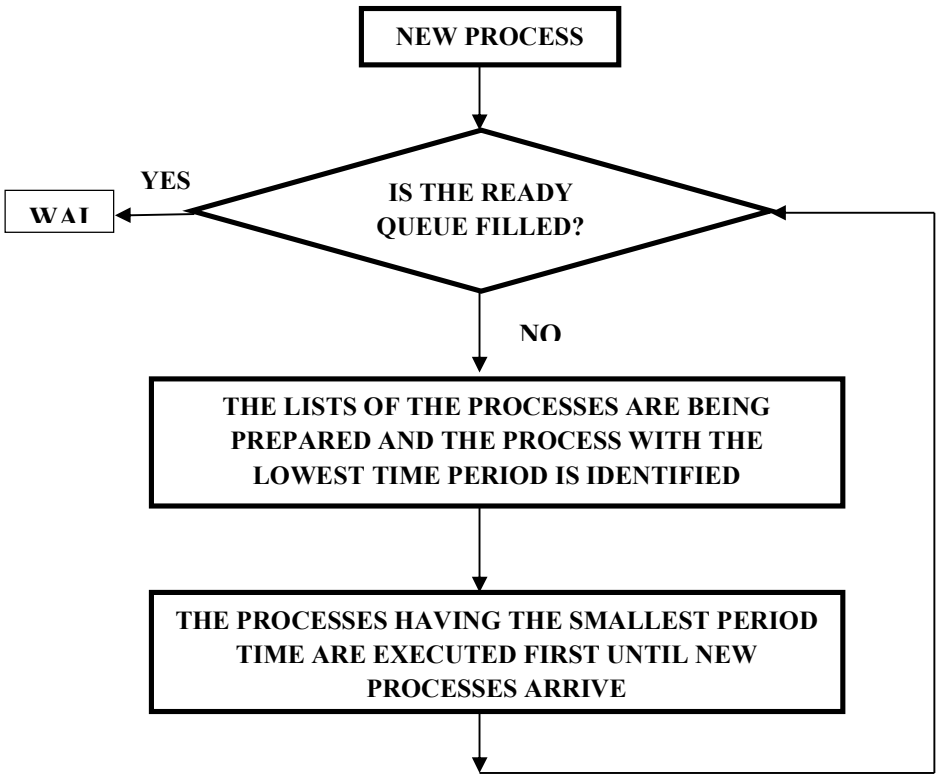


Figure 4.5: Flowchart of the Rate Monotonic algorithm

Space for learners:

Space for learners:

4.9.1.2. The Shortest Job First (SJF)

SJF is a type of static algorithm in which the instructions that have the smallest execution time are executed first. The time of the job is already scheduled in the RTOS. It is kept as the CPU time. When the current process that has the smallest job time gets completed and new processes are scheduled then this algorithm is worked upon[5]. This algorithm is suitable for a processor having batch-type processing and the waiting time for the jobs is not critical. SJF can be applied in both preemptive and non-preemptive scheduling algorithms. Starvation of the processes might occur if the processes have a larger burst time.

Let us explain the SJF algorithm with the help of an example by taking a flowchart [5].

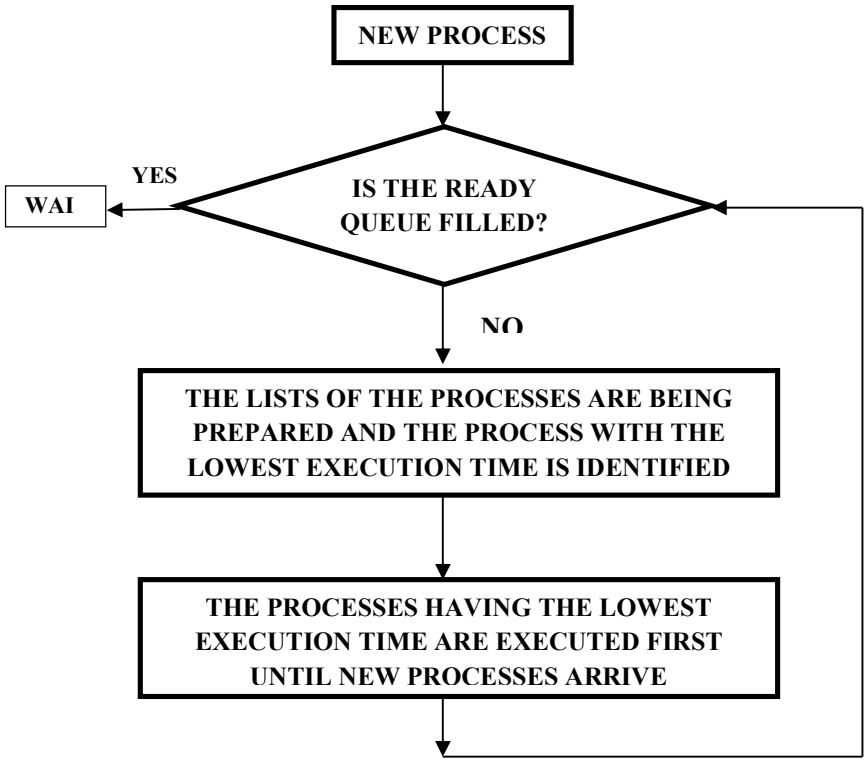


Figure 4.6: Flowchart of the Shortest Job First algorithm

4.10. TOMASULO'S ALGORITHM

Space for learners:

A scientist named Robert Tomasulo invented the Tomasulo Algorithm to be used in IBM 360/91.Tomasulo's algorithm is a type of hardware algorithm in computer architecture that is used for implementing dynamic scheduling allowing out-of-order execution and enabling multiple execution units. The hardware includes the reservation stations, the register renaming, and a common data bus (CDB) for carrying the values towards the reservation stations. Because of the presence of this hardware architecture, parallel processing is possible. WAR and WAW hazards are removed using register renaming. And this register renaming is done through reservation stations. Register Renaming is implemented through reservation stations. Tomasulo's algorithm work in some stages which are discussed below [8]. Here reservation station provides the register renaming.

1) **The Issue stage** – Instructions are present in a queue (FIFO queue) in which all the instructions are given some space with some deadline. When one instruction completes its job, the next instruction remains at the head of the queue. The work of the Issue stage is to call the instruction from the queue that is present at the head of the queue. If the reservation station matches the called instruction, then the instruction is issued some operand values in the renaming register [3]. One condition is that; the reservation station must remain free when the instruction is called. If the reservation station is not free, then the instruction stalls, and subsequently structural hazards occur. If there is a problem in issuing one instruction, then the instructions that are lined up in the queue will not get executed. Another case may occur; the instruction waits in the reservation stations if the values of the operands are not found from the common data bus [9].

2) **The Execution stage** – If all the values of the operands are available by the CDB into the reservation stations, then execution of the processes takes place. Until and unless operand values are not available, execution does not occur [3].With the help of effective addressing, load and store are maintained.

Executions are not initiated by the instructions until and unless the previous instructions are completed that were in order [9].

3) **The Write Result stage**– Once the results are obtained through execution, the result is written on the CDB and then transferred into the registers and then into the reservation stations that contain the store buffers. At this write result step, the data is written into the memory. As soon as the data values and addresses are present the data is transferred to the memory and in this stage, the storage gets completed[3].

Space for learners:

4.10.1. Out-of-Order Execution Implementation

For the complexities of the pipeline to be enhanced, the out-of-order processor needs to be implemented. Now the WAR and WAW hazards can be tackled because the system can reorder the instructions. The following are some of the issues that have been tackled in the pipelining structure and which are very important for Tomasulo's algorithm.

4.10.1.1. Reservation Stations

Reservation stations are one of the features of the CPU which permits the register renaming and Tomasulo's algorithm uses these reservation stations for use in dynamic scheduling. Reservation stations work as the data buffer that fetches and stores the instruction operand values as soon as they are made available and it does not allow the data to get stored in the register. One instruction specifies a single reservation station and the operands once available are sent for its execution and the completed instructions are stored in the buffer of the reservation stations. When there are many instructions and when all of their needs to write in the same register than by the terms, logically only the last instruction can be written in that same register[10].

Sequential instructions are issued to the reservation stations in Tomasulo's algorithm that helps in buffering the instructions. Reservation stations checks the common data bus for the availability of the data operand and if it is available in the buffer then only the instructions get activated.

There are some fields of register present in the reservation stations which are explained as follows[3]

- Op – Op is the operations that need to be performed on the operand data. It is the functional unit of the associated reservation station. The functional unit can be the arithmetic as well as the logical interpretations such as {AND, OR, NOT, ADD, SUBTRACT, etc.).
- Qj, Qk–In these fields of the reservation station, the source operands are produced and the value of zero indicates that the reservation station has delivered the value to its corresponding source. It produces the source registers.
- Vj, Vk – The source operands have some actual values depicted as Vj and Vk. The actual values will be only valid if the Qj and Qk have the value zero which indicates that the value of the source has arrived in the reservation station.
- A – The 'A' field holds the address of the memory information for a load or a store. Until and unless effective address computation occurs, instructions containing in the immediate field only hold.
- Busy – It works in two Boolean conditions that are True or False. If the condition is True that means the reservation station is occupied or busy. And if the condition is False that means the reservation station is not occupied. The value 1 indicates that the station is busy and o indicates that the station is not busy.
- Qi –all the results of the reservation station are stored in this register. If the value is 0 that means the value present in the register is the actual value of that register. At this point, the register is not renamed.

4.10.1.2. Register Renaming

Instruction results that are stored in the registers are particularly renamed. There may be more than one type of name in the registers that might be used in the system hardware. The reservation stations and the registers are mapped after which the renaming is performed. For correctly performing the out-of-order executions, the register renaming is usually applied by Tomasulo's algorithm[11].

Space for learners:

It is a pipelining technique that renames the register operands by dealing with the dependences of the data. The operands are specified with the help of a compiler using the architectural registers that are explicit instructions. The renaming register restores the name of these architectural registers by a new value name for the operands of each instruction. It recognizes the true dependencies automatically. It removes the WAR and WAW hazards by dynamically assigning values to the registers [11].

Space for learners:

4.10.1.3. Common Data Bus

The functional units and the reservation stations are connected directly with the help of the common data bus (CDB). Tomasulo's algorithm depicts that the CDB "preserves precedence while encouraging concurrency". It can be in two different structures [11].

- Operation results can be accessed by the functional unit without demanding any register with a floating-point and allow multiple functional units to access the register file [11].
- In CDB, control execution and hazard detection are distributed while controlling of the execution of the instruction are done by the reservation stations rather than by a hazard unit [11].

4.10.1.4. Scoreboarding

The scoreboard also follows the dynamic scheduling technique or we can say helps in implementing it so that the execution of the out-of-order can be performed with the condition that no conflicts occur and there is the availability of the hardware. During scoreboard, if data dependencies occur then it is tracked, logged, and observed very often. The scoreboard monitors the system every time to check whether any instruction got stalled or not and tries to resolve the dependencies before any instruction gets stalled[12]. It monitors every instruction that waits for it to get dispatched.

The scoreboard keeps all the latest information into its registers and also determines the time period when the instruction will begin and end. The scoreboard contains some stages in which the instruction must pass through it. The stages are given as follows[13].

a) Issue – In the issue stage, the scoreboard checks whether any hazards such as the WAW hazards are available or not. If it is present, then the instruction gets stalled[13].

b) Read operands – The scoreboard reads or finds whether any source operands are available or not. If it is present, the functional units are instructed by the scoreboard to check the register file and read the operands so that it can start its execution. The RAW hazards are corrected in this stage. If instructions do not write or use any operand, then that operand is said to be free or available and is present in the register file. If multiple instructions come to the register file, then ambiguity might occur as to which instruction will get the preference to write the operand[13].

c) Execution – The scoreboard gets notified here by the functional unit as to when the execution gets over[13].

d) Write result – As soon as the scoreboard gets notified from the execution stage that the execution has finished, the scoreboard investigates whether any WAR hazards are present or not. If WAR hazards are present, then the functional unit is instructed to get stall by the scoreboard until the hazards are being cleared[13].

The main difference between Tomasulo's algorithm and scoreboarding is that there is no distribution system in scoreboarding. Scoreboarding keeps the track of all instructions and information within itself and is the sole control unit. Whereas Tomasulo's algorithm is a distributed system. All the functional control is distributed among different registers.

Space for learners:

4.11. REORDER BUFFER (ROB)

The reorder buffer creates an apparition to the Users that their instructions are working in order. When a system encounters an instruction, the instruction gets renamed and decoded and then gets transferred to the ROB as well as the issue queue and simultaneously marked as busy. ROB receives the information once the instruction gets

executed and the ROB is marked as not busy. Not busy means that it is now 'committed' and the architectural state gets visible. But if an exceptional instruction remains at the head of the ROB then the architectural changes are not visible[14].

The structure of the ROB is normally a circular buffer that keeps the track of all instructions in order, while the commit head points to the oldest instruction and simultaneously new instructions will be managed within the ROB.

Like the other forms, reorder buffer has also some stages that help in the smooth working of it. They are explained as follows[14]:

a) Exception State –The oldest instruction in the pipeline when gets encountered by the ROB and is pointed to the head pointer then an exception is thrown by the system. A single bit is used to depict the instruction that has entered the ROB or not but the oldest exception instruction is only tracked by the additional exception state. By doing this saves space[14].

b) PC Storage – Branch and Jump instruction are used to access the information into the ROB's PC file at the time of register read[14].

c) Commit Stage – When the head of the ROB does not contain any instruction then it can be committed which means that any changes that occur in the system are made available. ROB releases single instruction in the pipeline but does not check for multiple instructions to get committed. The instruction gets stored into the memory only when the commit is performed. After commit, the instruction physically releases the register[14].

d) Exception and flushes – When ROB contains the instruction at the commit head then only exceptions are handled. The ROB gets emptied by flushing the pipeline. Reset of the rename map table must be done. Control status register (CSR) receives the accepting instruction if the instruction is an architectural exception and if it is a micro-architectural exception re-fetching is done of the failing instructions and execution can begin afresh[14].

Space for learners:

e) Point of no return – For marking the instruction for which exception might be generated, another pointer head runs just in front of the ROB commit head which is known as the point-of-no-return. It includes memory operations that are untranslated and branches that are unresolved. RoCC instructions that do not tolerate miss peculation are nowadays used by the PNR which means that instruction that has passed the PNR head only gets issued by the ROB[14].

Space for learners:

CHECK YOUR PROGRESS

Fill in the following blanks.:

1. Pipelining is a technique where instruction overlapping occurs at the time of its _____.

2. The efficiency of the pipeline will _____ if the instruction cycle is split into the equal time clock.

3. _____ delay can occur in pipelined processor.

4. The flip-flops that are inserted between the data modules increases the _____ in the instructions.

5. _____ algorithm is a type of hardware algorithm in computer architecture that is used for implementing dynamic scheduling allowing out-of-order execution and enabling multiple execution units.

6. If the reservation station matches the called instruction, then the instruction is issued some operand values in the _____.

7. Reservation Stations checks the _____ for the availability of the data operand.

8. The reservation stations and the registers are mapped after which the _____ is performed.

9. The structure of the _____ is normally a circular buffer that keeps the track of all instructions in order.

10. _____ receives the excepting instruction if the instruction is architectural exception.

4.12. SUMMING UP

- The order in which the instructions are stored and executed is defined as pipelining processing.
- All the instruction inside the pipeline works concerning some clock time.
- When arithmetic operations come as instruction into the pipeline they are then stored in the arithmetic pipeline.
- When complex instructions are executed on numerous data synchronously, then a vector processor is used. This processor executes the instructions very fast as compared to the scalar processor and has much efficiency.
- The time delay in between two instructions in a pipeline is greatly reduced hence increasing the throughput.
- Branching delay can occur in a pipelined processor.
- Data dependency is a concept that is applied to check whether a block works properly even if the instructions present in that block are rearranged.
- If two or more instructions are spaced closely and have the same dependencies, then it might so happen that the instructions might come to a halt or become idle.
- The WAR and WAW hazards create the possibility of out-of-order execution.
- If the processes have the same slack time, then first cum first serve (FCFS) algorithm is applied together with LST.

Space for learners:

4.13 ANSWERS TO CHECK YOUR PROGRESS

1. Execution, 2. Increase, 3. Branching, 4. Latency, 5. Tomasulo, 6. Renaming Register, 7. Common Data Bus, 8. Renaming, 9. Reorder Buffer, 10. Control Status Register.

4.14 POSSIBLEQUESTIONS

Short Type Questions:

1) What do you mean by pipelining?
2) What is pipelining processing?
3) What are the two types of a pipeline?
4) Explain in brief the data dependency.
5) Explain in brief the dynamic scheduling.
6) What are the different types of dynamic scheduling and static scheduling algorithms?
7) Write the function of the Issue stage in Tomasulo's algorithm.
8) What is a reservation station?
9) How does the register renaming work in the pipelining schedule?
10) What do you mean by reorder buffer?
11) What is the function of a common data bus?
12) What do you mean by scoreboarding? Explain in brief.
13) What are the different dependency hazards?
14) What do you understand by point of no return? Explain in brief.

Long Type Questions:

1) What do you mean by pipelining? Explain the different types of pipelines.

2) Discuss the advantages and disadvantages of pipelining.

3) Explain the pipelining scheduling with a relevant example.

Space for learners:

4) What are the different types of data dependencies?

5) What are the advantages and disadvantages of dynamic scheduling?

6) Explain the earliest deadline first algorithm.

7) Explain the least slack time first algorithm.

8) Explain the rate monotonic algorithm.

9) Explain the shortest job first algorithm.

10) Explain Tomasulo's algorithm.

11) What do you mean by reservation station? Explain all its stages.

12) What do you mean by register renaming?

13) What do you understand by reorder buffer?

Space for learners:

4.15 REFERENCES AND SUGGESTED READING

[1] https://www.lkouniv.ac.in/site/writereaddata/siteContent/20200422161333844 5rohit_engg_pipelining_and_hazzard.pdf

[2] https://www.quora.com/What-is-pipelining-scheduling-in-computer-architecture

[3] https://www.brainkart.com/article/Dynamic-Scheduling_8832/

[4] https://en.wikipedia.org/wiki/Instruction_scheduling

[5] Teraiya, J., & Shah, A. (2020). Analysis of dynamic and static scheduling algorithms in soft real-time system with its implementation. In *Soft Computing: Theories and Applications* (pp. 757-768). Springer, Singapore.

[6] https://www.cs.umd.edu/~meesh/411/CA-online/chapter/advanced-concepts-of-ilp-dynamic-scheduling/index.html

[7] COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE EIGHTH EDITION, BY

WILLIAM STALLINGS, published by Prentice Hall (an imprint of Pearson)

[8] https://www.cse.iitk.ac.in/users/biswap/CS422/L12-Tomasulo.pdf

[9] http://www.cs.umd.edu/~meesh/411/CA-online/chapter/dynamic-scheduling example/index.html

[10] https://www.cs.umd.edu/~meesh/cmsc411/website/projects/dynamic/tomasulo.html

[11] https://en.wikipedia.org/wiki/Tomasulo_algorithm

[12] https://en.wikipedia.org/wiki/Scoreboarding

[13] https://www.cs.umd.edu/~meesh/cmsc411/website/projects/dynamic/scoreboard.html

[14] https://docs.boom-core.org/en/latest/sections/reorder-buffer.html

Space for learners:

UNIT 5 – ADVANCED CPU ARCHITECTURE

Space for learners:

Unit Structure:

5.1 INTRODUCTION

In last three decades, the architectures of CPU design have been implemented on an unprecedented scale on a single chip due to the advancement of Integrated Circuit fabrication technology. So, this becomes very much relevant for you to learn about different instruction set architectures. Moreover, it is also very important to know about the interconnection between multiple processors & cache memory and the cache coherence problem which may arise with such interconnections.

This unit is divided into two parts. In the first part, we will take a close look at CPU architectures. Our primary focus is Very Large Instruction Word (VLIW)architecture. You will get a brief introduction to different instruction set architectures like CISC and RISC, which are implemented in superscalar processors. The detailed architecture of VLIW processors will be discussed in this unit along with basic working, instruction format, advantages and disadvantages. Additionally, the implementation details of Explicitly Parallel Instruction Computing (EPIC) architecture will also be discussed, which is a VLIW inspired architecture, developed by HP and Intel. You will learn how EPIC differs from VLIW and how EPIC overcomes certain limitations of VLIW.

The second part of the unit focuses on the concept of Multiprocessor Systems, which is based on Multiple Instruction stream, Multiple Data stream (MIMD)design scheme. We will discuss about the different classification of multiprocessors – namely tightly coupled and loosely coupled. You will also learn about different interconnection structures between multiprocessors along with pros and cons of each of the structure. We will also discuss how cache memory is used in uniprocessor and multiprocessor systems to increase the performance along with the cache coherence problem. We will also look into different hardware and software-based solutions to the cache coherence problem.

Space for learners:

5.2 UNIT OBJECTIVES

The objective of this unit is to give an introduction to advanced CPU architectures and multiprocessor systems. After completing this unit, you will be able to

- Learn about the VLIW architecture and how it differs from the superscalars
- Know the EPIC architectures and how it differs from VLIW
- Learn the concept of multiprocessor systems and its types
- Understand the different interconnection structures in multiprocessor systems with pros and cons.
- Define cache coherence problem and its solutions.

5.3 INTRODUCTION TO ADVANCED CPU ARCHITECTURES

In the field of Computer Science, an abstract model of a computer system is defined by Instruction Set Architecture (ISA),also known as Computer Architecture. Implementation of ISA corresponds to the realization of ISA, such as CPU, registers, main memory, data types to be supported, etc. An ISA is like a contract between the set of microprocessor implementation of an architecture and the class of programs that are written for that architecture. ISA defines a basic set of operations that must be performed by the system and serves as boundary between hardware and software. The set of operations may include arithmetic, logical, branching and memory operations. The ISA provides details about how a machine code over the implementation of a particular instruction set architecture doesn't depend on the prime characteristics of that implementation. Thus, it allows multiple implementations of ISA, which may differ in physical size, overall performance and prices, but can run the same machine code or software.

Space for learners:

5.3.1 Classification of Instruction Set Architectures:

1. **Complex Instruction Set Computer (CISC):**In CISC architecture, there are hundreds of instructions or commands of variable lengths, that instructs the system to perform addition of numbers, storing and displaying results. This approach is carried out in order to save the memory since all instructions of same length will contribute to wastage of memory. Here, simple commands may require 8-bits and complex commands may require 120 bits. An implementation of CISC architecture is *Intel x86*, which was introduced in 1978. CISC provides convenient addressing modes and enables copying the block of instructions through support for functions using CALL instructions. Thus, in CISC, it is easy to expand the ISA.

2. **Reduced Instruction Set Computer (RISC):** In RISC architecture, the computer system uses sets of instructions which are highly optimized and CPU design focuses on raw performance. In contrast to CISC, RISC uses relatively simple, fixed length instructions of 32-bits. Although fixed

length instructions may mean more space wastage, however the instructions are faster & easier to execute. Moreover, in terms of CPU design perspective, RISC integrated chips requires a smaller number of transistors as compared to CISC, since RISC implementation deals with only handful types of instructions and delivers high performance. However, due to short instruction size, a greater number of instructions are executed compared to CISC, in order to accomplish a given function. Example of RISC architectures includes Sun Microsystem's SPARC, IBM/Motorola's PowerPC, Hewlett-Packard's PA-RISC,SGI's MIPS, ARM architecture, etc. In recent times, almost all low-end portable devices are based on ARM architecture, which includes most Android-based systems, Apple's iPhone and iPads, Nintendo's video gaming console Switch, Raspberry Pi and many more.

Please note that the simplicity of RISC allows to easily design superscalar processor that can execute more than one command or instruction at a time. This concept is known as *Instruction-level parallelism (ILP)*. In modern times, almost all CISC & RISC processors are superscalar in nature, however, this has introduced new levels of design complexity for CPU architects.

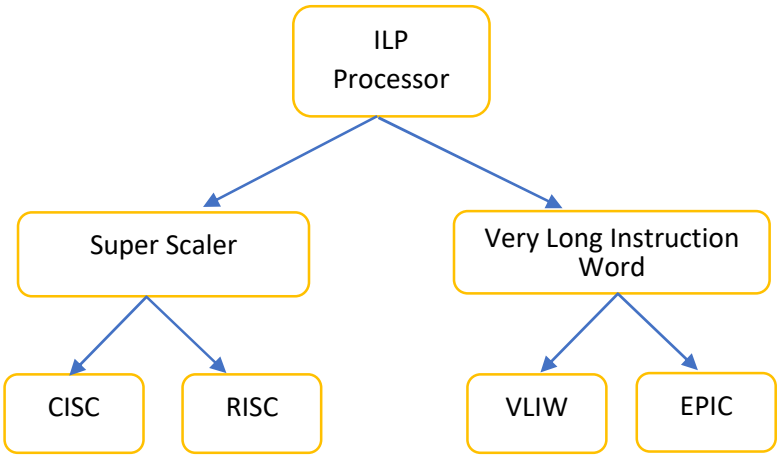


Fig. 5.1. Classification of ILP Processor Architecture

Now, there are two most significant types of ILP processors, namely *Superscalar* processors and *Very Long Instruction Word (VLIW)* processors. We have already come across superscalar processors, an implementation for ILP processor architectures in which programs

Space for learners:

doesn't have any explicit information about parallel execution of instruction and it is the responsibility of the system hardware to detect and construct action plans for any ILPs to be exploited for parallelism. On the other hand, VLIW processors are built on an architecture in which programs contain explicit information about parallelism and it is the responsibility of the software, called compiler to identify and communicate it to the hardware by specifying all the independent operations. Thus, the hardware doesn't have to check further on the operations which can execute in the same cycle, since the information is already provided by the compiler. Let's explore VLIW architecture in details in next section.

Space for learners:

CHECK YOUR PROGRESS-I

1. An abstract model of a computer system is defined by _____

2. What type of operations are defined by an instruction set architecture?

3. What are the different class of Superscalars based on Instruction Set Architecture?

4. RISC stands for _____

5. Give one example each for implementation of CISC and RISC architecture.

State TRUE or FALSE:

6. CISC uses simple, fixed length instructions.

7. In RISC, CPU design focuses on raw performance and the instructions are highly optimized.

8. Instruction-level parallelism is the ability of a processor to execute more than one instruction at a time.

5.4 VLIW ARCHITECTURE

In the early 1980s, John Fisher, a faculty from Yale University, invented the architectural concept and coined the term VLIW among his research group. He later joined HP Labs. VLIW refers to a processor architecture designed to take advantage of instruction-level parallelism (ILP). It is less complex approach to allow higher

performance i.e., multiple operations are performed simultaneously or level of parallelism increases.

In VLIW Processor,

- Instruction consists of multiple independent operations grouped together.
- There are multiple independent functional units.
- Each operation in the instruction is assigned to different functional units.
- All functional units share the use of a common large register file.

For example –

ADD R1, R2; SUB R5, R6; LD R7, data; STR R8, data;

In this example, there are four operations. ADD (Addition) and SUB (Subtraction) are arithmetic operations, which corresponds to Arithmetic & Logic Unit (ALU). Similarly, LD (Load) and STR (Store) are memory operations, which corresponds to Memory Unit (MU). Here, we can see that independent operations are grouped together in a single instruction word. Now, the CPU will assign each of these operations to different independent functional units to execute the operations parallelly, thus to achieve instruction level parallelism (ILP) and higher performance.

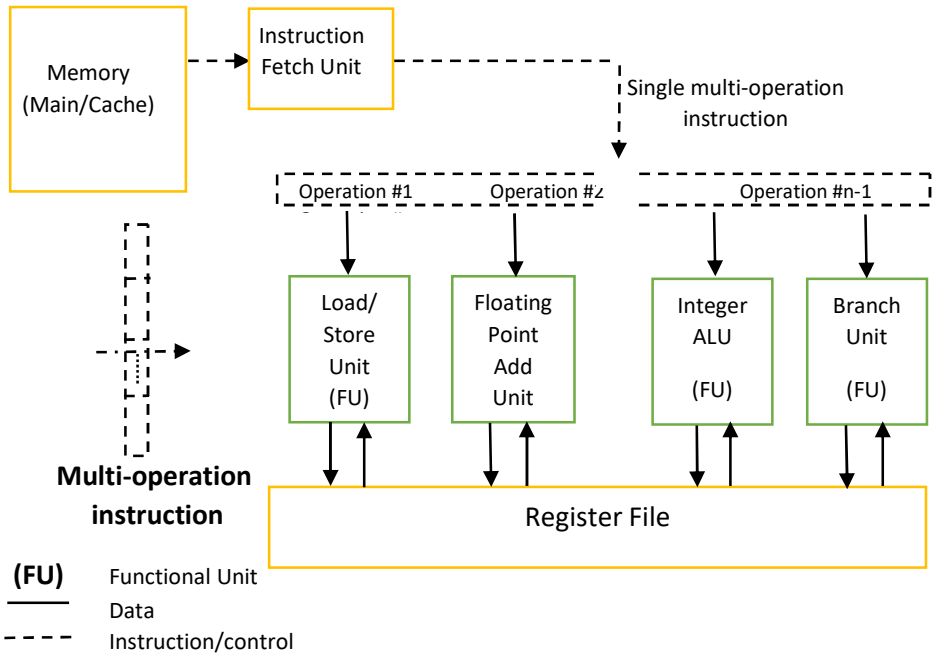


Fig. 5.2 (a). A typical VLIW Processor

Space for learners:

Load / Store	Floating-Point Addition	Floating-Point Multiplication	Branch		Integer ALU

Fig. 5.2 (b). A VLIW Instruction Format

In VLIW Processor, compiler is responsible for static scheduling of instructions i.e., compiler finds out which operations can be executed in parallel in the program. Compiler groups together these independent operations in a single instruction (VLIW) which is the VLIW. It also makes sure that before the operands are ready, an operation is not issued.

VLIW instruction contains operands & operations to be performed by the various functional units. One VLIW instruction encodes at least one operation for each functional (or execution) unit on each cycle. So, length of the instruction increases with the number of functional (or execution) units. For example, as we have seen earlier if we have two ALU and two Load/Store functional units in our VLIW architecture, then VLIW instructions length will be four. These operations are assigned to functional units by the position in the given fields within the long instruction word. This is known as *slotting*.

Stop to Consider

- ✓ The instructions within a VLIW instruction are issued and executed in parallel.
- ✓ Since in VLIW processor, one VLIW instruction word encodes multiple operations, which allows them to be initiated in a single clock cycle.
- ✓ The start of execution of the operations is bound by the VLIW instruction in which it appears, and all the operations in a VLIW start executing together in parallel
- ✓ VLIW instructions are at least around 64 bits wide and 1024 wide in some architecture.

Space for learners:

5.4.1 Example of VLIW code:

RISC Code	VLIW Code	
MUL R1, R3, 3	MUL R1, R3, 3	SUB R3, R3, 1
LD R4, 0(R1)	LD R4, 0(R1)	NOP
ADD R2, R2, R4	NOP	NOP
SUB R3, R3, 1	ADD R2, R2, R4	BNEZ R3, -4
BNEZ R3, -4		

In the above example for RISC Code, content of Register R3 is multiplied by 3 and is stored in R1. The R4 is loaded with the data stored in the address that R1 contains. Then the content of registers R2 and R4 are added and stored in R2. The content is R3 acts as counter and is decremented by 1. BNEZ instruction is a conditional branch which checks if content of R3 is not equal zero and if the condition satisfies, the control is passed back to -4 instructions from the top i.e., to the MUL instruction at the beginning. In VLIW code, this sequence is divided by the compiler in such a way that similar task can be carried out parallelly on different execution or functional units to achieve high performance.

5.4.2 Examples of VLIW Processors:

- VLIW Mini Supercomputers – Multiflow TRACE 7/300, 14/300 and 28/300
- Single Chip VLIW Processors – Philip's LIFE Chips
- DSP Processors - Analog Devices' SHARC DSP, Texas Instruments' C6000 DSP family
- Intel's Itanium IA-64 EPIC (embedded & nonembedded)
- TileraTILEPro

5.4.3 Advantages of VLIW

1. Compiler determines data dependency checks and other instruction issues; it becomes a lot simpler.
2. Reduces hardware complexity
3. Compiler is used to schedule according to functional units.
4. Compiler issues instructions corresponding to the position of functional units.
5. Ensures low power consumption due to reduction of hardware complexity.
6. Increases potential clock rate.

Space for learners:

5.4.4 Disadvantages of VLIW

1. Higher complexity of the compiler, which are hard to design.'
2. VLIW processors cannot react on dynamic or unscheduled events. It can work only on static instructions. Unscheduled events, for example a cache miss could lead to a stall which will stall the entire processor.
3. Large memory bandwidth & more registers for software pipelining, etc.
4. Increased program code size.
5. The number of instructions in a VLIW instruction word is usually fixed.
6. If issued bandwidth is not met, padding of VLIW instruction word is needed, which results in increase in code size.
7. In case of un-filled opcodes in a VLIW, padding of VLIW instructions with No-Ops (No Operations) is required, for which there is waste of memory space and instruction bandwidth.

5.4.5 Applications of VLIW Processors

- It is suitable for Digital Signal Processing Applications.
- It is used for tasks, which involves processing of media data, like compression /decompression of image and speech data.

5.5 EPIC Architecture:

Explicitly Parallel Instruction Computing (EPIC)is a term proposed by Hewlett Packard & Intel, which formed an alliance in early 90s for the research and implementation of Intel Itanium architecture (IA-64). In 2001, IA-64 was launched as a collection of 64-bit Intel Itanium microprocessors. Though the original ISA specifications were by HP, but it was later evolved and implement as a new processor micro architecture by Intel.

5.5.1 EPIC vs VLIW

EPIC is inspired by VLIW architecture at roots, so it permits execution of instructions in parallel using a compiler instead of complex circuits, which were earlier used to control instruction level parallelism (ILP). In contrast to VLIW, apart from identifying and grouping the independent operation in a single instruction, the

Space for learners:

compiler communicates this via explicit information in the instruction set. That's why EPIC is also known as "independence architecture" *(Fisher & Rau)*. Unlike VLIW, EPIC retains backward compatibility across different implementations like superscalars, but doesn't need any hardware for *dependency checks* like superscalars. EPIC is a mix of software & hardware, incorporating the advantages of both superscalars and VLIW architectures, while fixing several shortcomings of VLIW.

1. VLIW instructions had a backward compatibility issue between wider and narrower implementations. Wider implementation uses greater number of execution units (EU), which also increases the size of an instruction since the number of operations to run in parallel also increases. Such a wider instruction set doesn't work well with narrower implementations with lesser number of execution units.
2. The static scheduling by the compiler for load instructions became quite difficult since memory operations need to work with several devices from memory hierarchy, like CPU cache memory and DRAM, which doesn't have any deterministic delay for load responses. In other words, the compiler couldn't predict the delay in response time efficiently for the load instructions using different memory technologies.

So, although EPIC evolved from VLIW architecture, it tries to retain some properties from superscalar architecture. There are several additions to features of EPIC architecture in contrast of VLIW as discussed in next section.

Space for learners:

5.5.2 EPIC architectural details

In EPIC architecture, we have a "*bundle*" of multiple software instructions. Each of these bundles includes a stop bit to indicate if there is some interdependencies between two subsequent bundles. The dependency information is calculated by the compiler. This information could help in issuance of multiple bundles in future implementations. Typically, a bundle is of 128 bits, with thee 41-bit instructions per bundle and only two bundles can be issued at once. For data prefetch, software prefetch instruction is used, which not only increases cache hit for load operation, but also indicates the

requirements of temporal locality in different cache levels. For these purpose, two types of load instructions, namely *speculative load instruction* and *check load instructions* are used in EPIC to bypass control and data dependencies.

Moreover, EPIC follows a fully predicated instruction set architecture, that enables *predicated execution,* which decreases the occurrence of branching and increase *speculative execution* of instructions.

Space for learners:

Stop to Consider

- ✓ **Predication:**
  "In computer science, Predication is an architectural feature that provides alternative to conditional transfer of control, implemented by machine instructions such as conditional jump, conditional call, conditional return and branch tables. It means if a register condition bit is set, the instruction is executed; if the bit is clear, it is not." – Predication *(on Wikipedia)*

- ✓ **Speculation:**
  "Speculative execution is an optimization technique where a computer system performs some tasks that may not be needed. Work is done before it is known whether it is actually needed, so as to prevent a delay that would have to be incurred by doing the work after it is known that it is needed. If it turns out the work was not needed after all, most changes made by the work are reverted and the results are ignored. The objective is to provide more concurrency if extra resources are available. This approach is employed in a variety of areas, including branch prediction in pipelined processors, value prediction for exploiting value locality, prefetching memory and files, etc." – Speculation *(on Wikipedia)*
- ✓ **Register renaming:**

  Register renaming is a technique of managing data dependencies between the instructions in the pipeline by renaming the register operands.

In this architecture, the register files are very large and there are wide range of registers at disposal to avoid *register renaming*. Registers include 128 integer and floating-point registers, 128 additional registers for loop unrolling & optimization, 8 indirect branch registers and other miscellaneous registers. Moreover, predication (or multi-way branch instruction) improves the prediction of branch instruction by combining branches as alternate instruction in one bundle.

Lastly, let us revise the difference between Superscalars, EPIC & VLIW.

	Grouping of instructions *(Checking dependencies between instructions to find group able instructions for parallel execution)*	**Assigning of functional unit** *(Assigning instructions to the functional or execution units of the hardware)*	**Initiation of execution** *(Determining when the execution starts or instructions are initiated)*
Superscalar	Hardware	Hardware	Hardware
EPIC	Compiler	Hardware	Hardware
VLIW	Compiler	Compiler	Compiler

CHECK YOUR PROGRESS-II

9. EPIC stands for_____

10. The first implementation of EPIC architecture is _____ family of processors.

11. In VLIW, _____issues instructions corresponding to the position of functional units.

12. EPIC is developed as a joint collaboration between _____

13.EPIC follows a fully _____ instruction set architecture

State TRUE or FALSE:

14. VLIW is inspired by EPIC architecture

15. In EPIC, the functional units are assigned by compiler.

Space for learners:

5.6 INTRODUCTION TO MULTIPROCESSOR SYSTEMS

A multiprocessor system is a computer system with more than one processor (typically two or more), where each processor is linked with one another. The connection between these processors is known interconnection network. The primary focus of a multiprocessor system is to achieve parallel processing, which enhances the overall performance. Apart from high performance, the multiprocessor system focusses on –

1. *Fault Tolerance and graceful degradation:* These systems have high fault tolerance since multiple processors are at play. In case of system failure, the system can continue to run in low power, until it stops completely.
2. *Scalability and modular growth:* The number of processors, memory units, etc. can be added or removed at any point of time. This modularity allows for scalable enhancements in future.

Multiprocessor system falls under MIMD architecture. It isone of the types of parallelism as per Flynn's classification of computer organization. The MIMD refers to multiple control units and multiple execution units or processors. There are multiple instruction and data steams as shown in figure below.

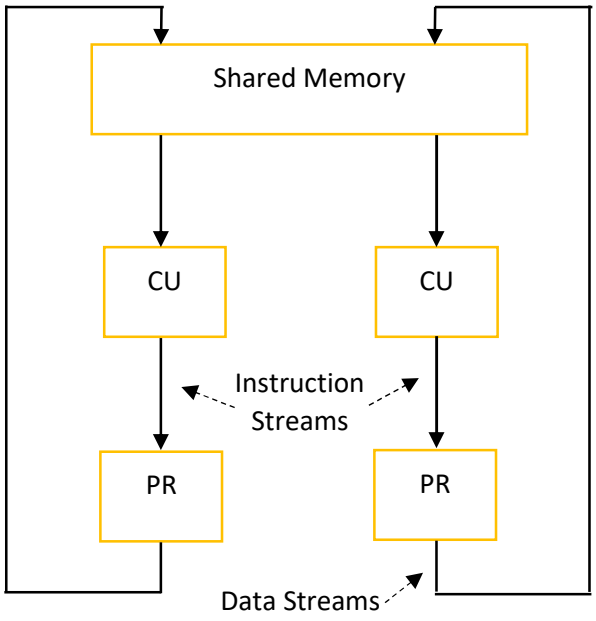


Fig. 5.3. MIMD with shared memory

Space for learners:

Stop to Consider

- ✓ Please note that *multiprocessor* and *multicomputer* systems may sound similar, but there exists an important difference.
- ✓ Both of them support concurrent operations, but a multicomputer system is a system with multiple computers and a multiprocessor system is a system with multiple processors.

- ✓ In multiprocessor systems, there is a single operating system, which provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- ✓ In multicomputer system, each computer has a separate operating system, however these computers work together as a single entity.

Space for learners:

5.6.1 Classification of Multiprocessors

The following figure shows different types of multiprocessors. They are primarily divided into two-types: *tightly coupled system* and *loosely coupled system*.

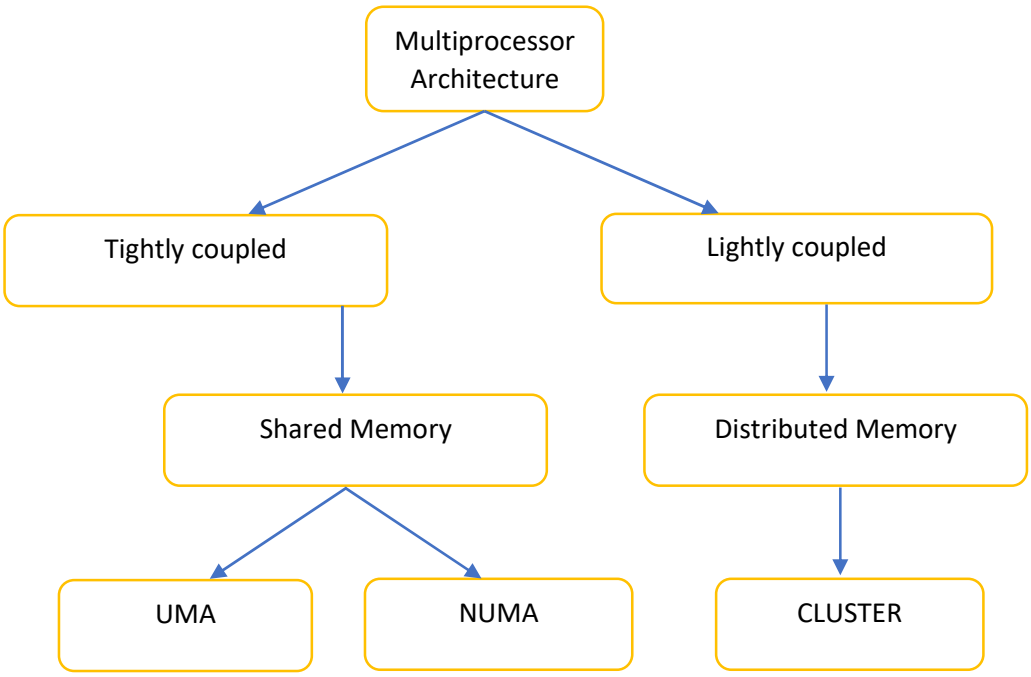


Fig. 5.4 Classification of Multiprocessors

A *tightly coupled* multiprocessor, also known as shared memory multiprocessor system, share information between multiple processors via a shared or global memory. Here, all processors share a single memory address space and communicate among themselves through shared variable in memory. Each of the processors can access any location in the shared memory. Apart from shared memory, each processor can also have a dedicated local memory which other processors cannot access. Please note that all the processors in the multiprocessors system communicate to perform tasks in a highly synchronized fashion.

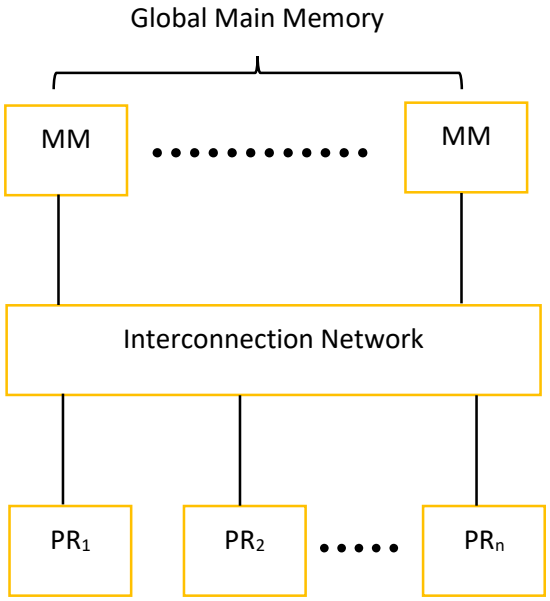


Fig. 5.5 Tightly coupled multiprocessor system

In tightly-coupled multiprocessors, we have Uniform Memory Access (UMA). In a UMA multi-processor system, the access time of memory is equal for all the processors irrespective of which processor accesses which portion of the common memory. Although the access time of memory is almost equal, the memory access in UMA is bit slow due to the use of a single memory controller. We also have Symmetric Multiprocessor (SMP) system, which is an UMA multi-processor system with identical, homogenous processors, which are capable of performing similar functions and utilizes a centralised shared main memory.

There is also another type of tightly-coupled multi-processor system known as Non-Uniform Memory Access (NUMA) system. In NUMA multi-processor systems, the memory area is virtually

Space for learners:

divided into faster access area and slower access area. The faster access areas are assigned to the processors and the slower common area is used for the exchange of data. Several memory controllers are used for this purpose for allowing local faster memories to be used as actual main memories. This enables NUMA to manage workloads to achieve higher performance than UMA multi-processor systems. These systems are also known as Distributed Shared Memory (DSM). In DSM multiprocessor system, the processors have a shared address space for all the memories.

A *loosely coupled* multiprocessor system, also known as the distributed memory multi-processor system, doesn't share information between multiple processors via a shared memory, since each processor has its local dedicated memory, which together forms a distributed memory. Please note that all the processors in the multiprocessors system do not communicate to perform tasks in a highly synchronized fashion. Processors communicate and share explicit information among each other using a common message passing protocol via interconnection network, for which the overhead of data exchange is high.

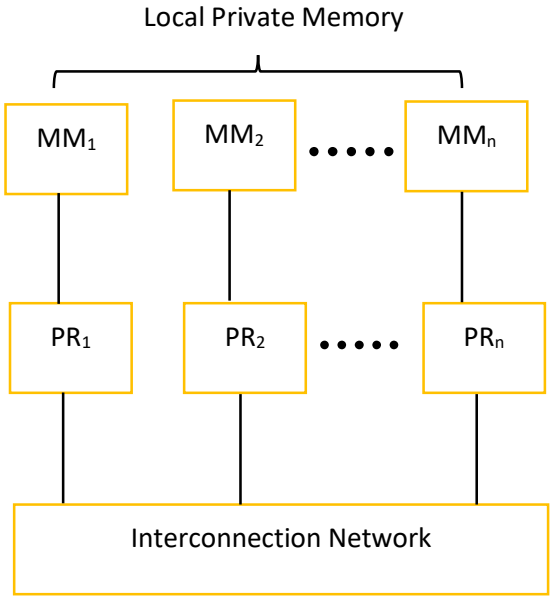


Fig. 5.6 Loosely coupled multiprocessor system

The loosely coupled multiprocessor system has physically distributed memories like in the case of cluster. A cluster consists of a set of computers connected over a local area network (LAN) which function as a single large multiprocessor. In the cluster

Space for learners:

system, there is no sharing of address space and each cluster node works together, although it can also work independently. Since a cluster act like a multiprocessor, it can provide the benefits of multiprocessor system along with additional benefits like load sharing and better fault tolerance.

Space for learners:

Stop to Consider

- ✓ Tightly-coupled multiprocessor systems use a shared memory (can be a virtually distributed shared memory) and Loosely-coupled multiprocessor systems use physically dedicated distributed memory.
- ✓ In literature, the terms UMA and SMP are used interchangeably, since access to shared memory is balanced in both the cases.
- ✓ NUMA can be considered as a tightly coupled form of cluster.
- ✓ Cluster is not same as a Computer Network. The primary objective of a computer network is *resource sharing* but for Cluster, it is *parallel computing*.

5.7 INTERCONNECTION TYPES

In multiprocessor systems, the components like CPU and I/O Ports are connected to I/O devices and a memory unit, which can be shared or distributed in nature. The interconnection between the components be of different physical configurations, described as follows:

a) Time-Shared Common Bus Structure:

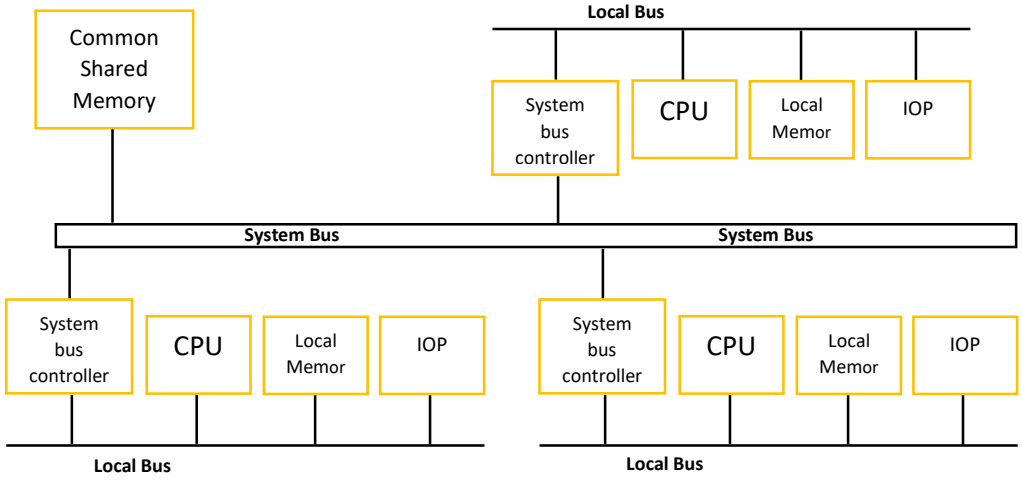


Fig. 5.7 Time-shared common bus structure

In this structure, all the processors in the microprocessor system are connected to shared memory and other common resources using a common interconnecting path, called as common system bus. In this structure, only one processor out of others can communicate with the shared memory or any other processor over the system bus at a given time, thus *time-shared*. Each processor can also have a local bus to communicate with its local memory and local I/O. The benefit of this is while one processor is working on system bus, other processors can communicate with local memory and local I/O through local bus. Please note that a part of local memory can be designed as cache and can be attached to CPU to reduce the average access time of the local memory.

Pros

1. The design is simple due to the use of single common system bus.
2. It is a cheap and affordable structure.

Cons

1. Since only one processor at a time can transfer or communicate over the system bus, the communication is quite slow. It means when one processor is accessing the shared memory using the bus, others can't perform any other operation using the bus.

b) Multiport Memory Structure:

In this structure, the system has separate buses between each memory module and the processors. For example, if we have 4 processors and 4 memory modules, then each memory module will have 4 ports each connecting to each of the processor bus. The processor bus consists of data, address and control lines. Each of the memory module has an internal control or priority logic to determine which processor request will be granted i.e., which port will have access to memory module at a given time, when there is a conflict of simultaneous requests from multiple processors in the system. Generally, a fixed priority is assigned to each memory ports to avoid memory access conflicts. Moreover, each processor is associated with a priority of the memory access, which is determined by the physical position of the port that the processor bus occupies in each module. So, processor P1 will have the highest priority and priority of the processor P4 will be the lowest.

Space for learners:

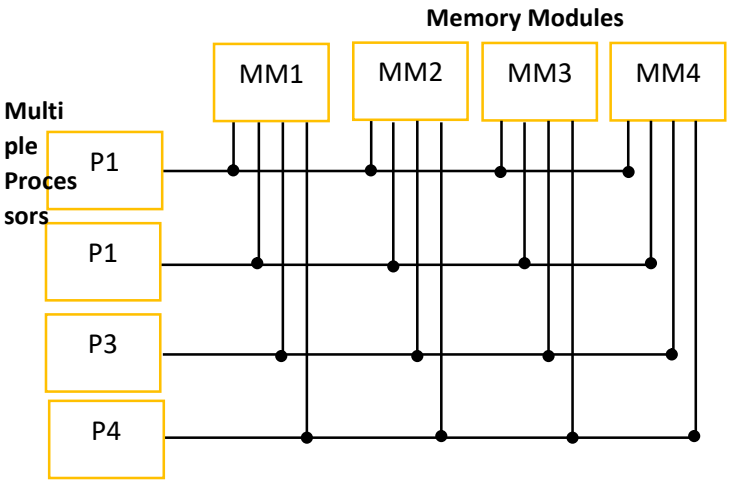


Fig. 5.8 Multiport memory structure

Pros

1. Due to multiple paths between the processors and memory modules, multiple processors can simultaneously access the memory, thus, high transfer rate can be achieved through this organization.

Cons

1. It requires a huge number of interconnecting cables to connect all the processors with memory modules. Thus, it is suitable for systems with small number of processors.
2. It also requires large hardware in memory modules in the form of memory control logic, which is very expensive in cost.

c) Crossbar Switch Structure:

In this structure, a number of crosspoints are placed at the intersection of memory paths and processor buses. At each crosspoint, there is a control logic to set the desired path between a memory module and a processor. This control logic is basically a switch, which is an electronic circuit. A switch can also resolve the conflict of simultaneous requests from multiple processors to access same memory module in the system based on a fixed priority basis. The following figure shows a crossbar switch interconnection for a system with 4 processors and 4 memory modules with 16 switches represented by small-squares marked by S_1 to S_{16}.

Space for learners:

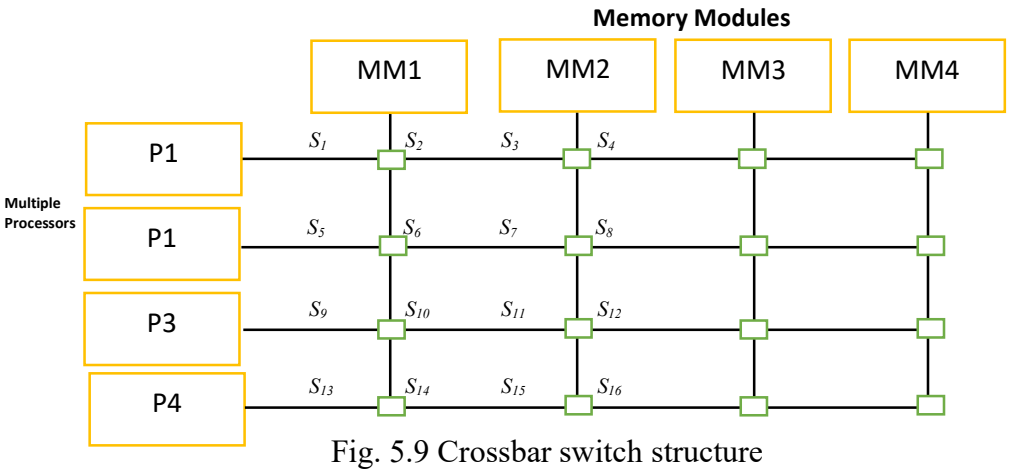


Fig. 5.9 Crossbar switch structure

Pros

1. Since there exists a separate path associated with each memory module, simultaneous transfer from all memory module is possible.

Cons

1. The entire connection here relies on switches. So, if large number of processors are present, then the design & implementation of switch requires large hardware and becomes complex.

d) Multistage Switching Network Structure

In this structure, we use a switch which can interchange *two-inputs, two-outputs*, in contrast to that of crossbar switches, which allows one stage of electronic switches – either input or output - to determine the path between multiple processors and multiple memory modules. Hence the name, multistage switching network since it allows to build different possible stages for different combination of inputs & outputs. Let us take the example of 2 x 2 interchange switch, which has 2 inputs – X & Y and 2 outputs – 0 and 1.

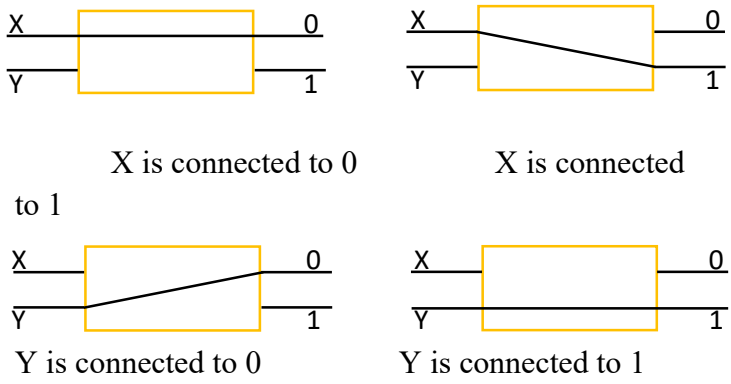


Fig. 5.10 Interchange switch states

Space for learners:

You can see that how four different states are possible in a single switch. Now, in place of X & Y, if we have two processors connected say P1 and P2, then we can have definite control to reach a particular memory module from a processor. These interchange switches allow to connect a source to a destination through multiple stages using a control logic.

A very popular topology is called omega switching network which allows exactly one path from each source to any particular destination. Simultaneous connections by two sources to two destinations connected to same switch is prohibited.

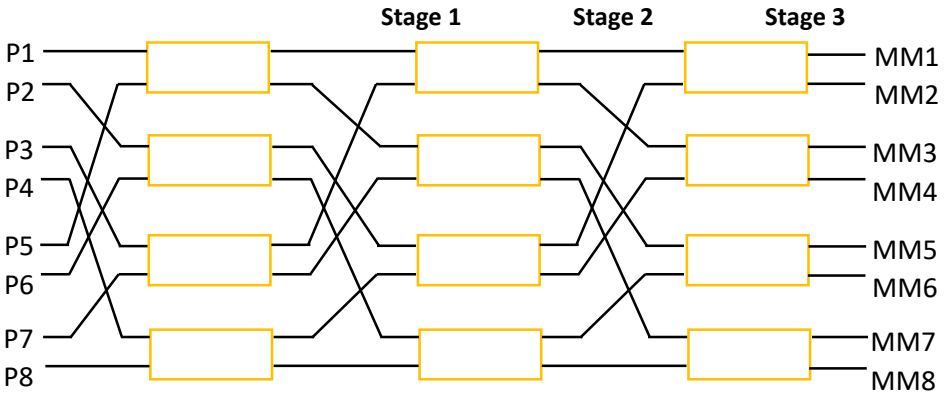


Fig. 5.11 An 8 x 8 Omega Multistage Switching Network

Pros:

1. The structure is cost effective since we can connect multiple sources to multiple destinations with less amount of wiring compared to crossbar switch structure.

Cons:

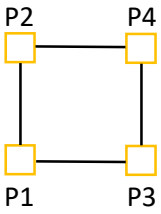
1. There is a restriction on the number of simultaneous connections, since simultaneous connections by two sources to two destinations connected to same switch is prohibited.

e) Hypercube Network Structure

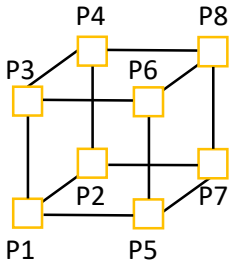
In this structure, a loosely coupled system is realized with the help of a concept called hypercube. A hypercube structure is comprised of $N = 2^n$ numbers of processors interconnected to each other in a N-dimensional cube. This structure is also known as a *binary N-node* multiprocessor structure. A node of the cube is represented by a processor and an edge of the cube is a communication path connection two nodes. Moreover, there exists dedicated paths or edges for a processor to communicate with the neighbouring nodes.

Space for learners:

n=1, N=2
Binary two-node cube
(One Dimensional)



n=2, N=4
Binary four-node cube
(Two Dimensional)



n=3, N=3
Binary 8-node cube
(Three Dimensional)

Fig.5.12 Hypercube structures

Pros:

1. It is easy to scale the current network to higher configurations simply by increasing the value of *n* which is the dimension of the cube.
2. Since it is a loosely coupled system, intelligent communication protocols could be easily implemented.

Cons:

1. The multiple paths between processors increases the routing complexity.

CHECK YOUR PROGRESS-III

16. MIMD stands for_____

17. Uniform Memory Access corresponds to _____ multiprocessors and Clusters corresponds to _____ multiprocessors.

18. A hypercube contains _____ numbers of processors.

State TRUE or FALSE:

19. In multistage switching network, an interchange switch has two-inputs, two-outputs.

20. In multiport memory structure with 3 processors, one memory module will have 3 ports connecting to each processor.

Space for learners:

5.8 CACHE MEMORY: UNIPROCESSOR VS MULTIPROCESSOR

A cache memory is a faster memory which sits in between a processor and main memory. Its primary role is to reduce the average access time for a particular data. In a system without a cache memory, the processor might have to higher access time given that the access to a particular data is needed on consecutive execution of instructions. The cache tends to hold those data in itself which has high probability of being asked by processor in next cycle. Also note that cache is also realised as a random-access memory, so the time taken to access any part of cache is almost same. Both this factor makes cache memory quite efficient solution to reduce the average access time.

In a system with single processor, the read and write operation on cache works as follows. During a read operation, a *word* from cache line is sent to the processor, the main memory is not involved in the transfer. During the write operation, two widely-used policies – *write back* and *write through* – are used. In *write back*, the cache memory is regularly updated after every write operation and all the changes made in cache are marked and is updated on main memory later. On the other hand, in *write through*, the cache and the main memory are updated simultaneously.

However, in a multiprocessor system, we can have a common shared main memory among all the processors. Each processor can also have local cache memory in order to reduce the average access time on an instruction cycle time. We already know for a processor writes its cache memory during a write operation. During the execution of an instruction, if any processor locally writes its cache, the new values must be made available to the all the other processors to maintain the consistency of the system. In case if the new value is not updated in common shared memory, then the other processors will receive and use the old values in their cache, which should not be allowed. Thus, when any of the processor makes modification in its cache,

a. All the other processors should either update their cache with the new modified value, or
b. Mark the old data in their cache as invalid.

Stop to Consider

- ✓ In a uniprocessor system, the main memory is for use by a single processor. In multiprocessor, the main memory is shared among all the processors.
- ✓ Each processor has its own cache memory and can incorporate either *write through* or *write back policy* to update its own cache.

Space for learners:

5.8.1 Cache Coherence Problem:

Cache coherence is a condition which states that all the cache lines with a particular shared main memory block must contain same information at any given point of time. This ensures that a multiprocessor system can perform memory operation correctly, by keeping identical multiple copies of information in the caches of the processors involved in execution of a particular instruction. Cache coherence problem occurs when cache coherence is not maintained, i.e., a processor updates its cache and other processors doesn't get an updated copy of newly modified data in their cache. This hampers the uniformity of data in all the caches of processors. Cache coherence problem happens in a multiprocessor system, since multiple processor access and works on non-identical multiple copies of data. Now as the cache coherence problem has been discussed, let's see the solutions for this problem.

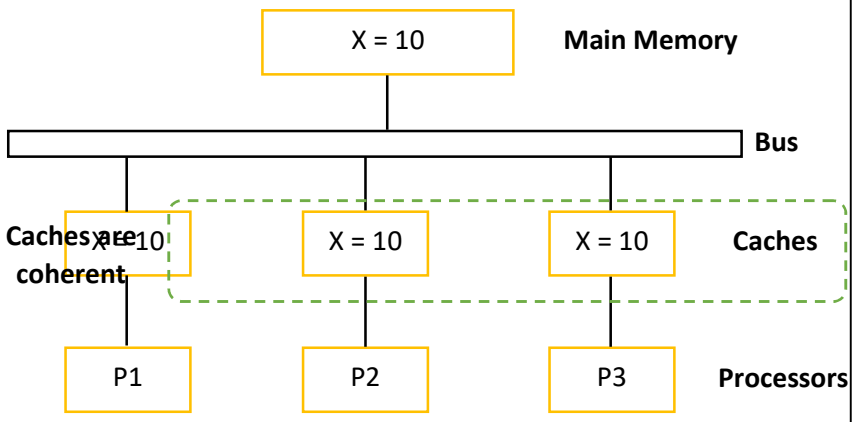


Fig. 5.13 Cache configuration after variable X = 10 is loaded from Main Memory

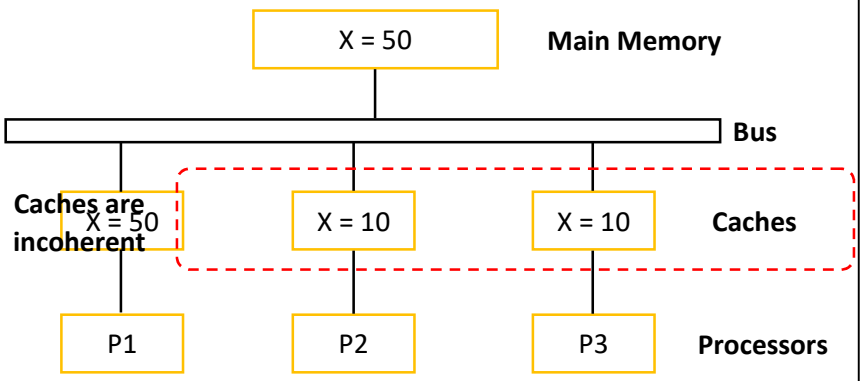


Fig. 5.14 Write-through policy. Modified value X=50 in Cache & Main Memory

Space for learners:

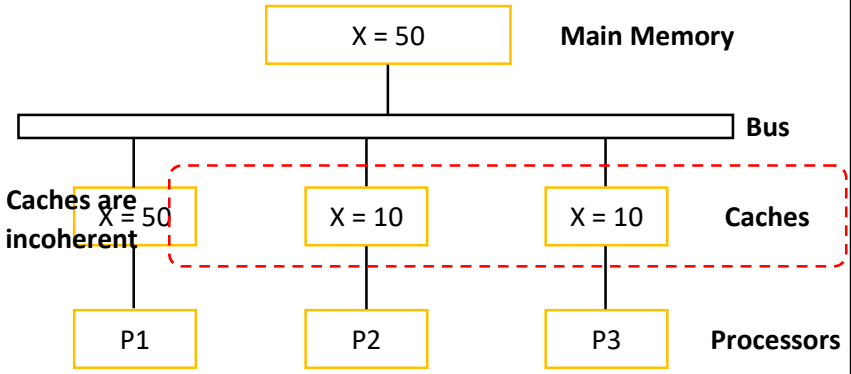


Fig. 5.15 Write-back policy. Modified value X=50 only in Cache, MM to be updated later.

5.8.2. The "All-is-well" Solution:

One of simple scheme can be to restrict the association of local caches for each processor and force them to use a shared cache memory instead. However, this simply overshadows the idea of having cache memory close to the processor, since a common cache will increase the average access time as compared to local cache. Thus, this scheme simply ignores cache coherence problem.

Considering the significance of performance, it is better to allow local caches in each processor and move towards more practical *software* & *hardware* solutions.

5.8.3. Software-based solutions:

The compiler is used to analyse the source code as the object code is generated in order to identify the parts of the program which uses shared items. These writable shared items are marked with a tag as *non-cacheable*, i.e., processors cannot write *non-cacheable* data into their local caches and have to access it directly from main memory for both read and write operations. A shared item can be identified by the processors using the tags associated with it. This is cheap to implement and can be achieved during the compilation process. However, it increases the average access time since during execution of an instruction, the processors have to access the main memory instead of their local caches. It is also an extra overhead on the software which also affects the system's performance. Please note that the program also uses non-sharable and read-only items,

Space for learners:

which are marked as *cacheable* i.e., these data are allowed to be stored in the local cache of the processor. Only non-cacheable items remain in main memory.

Space for learners:

Stop to Consider

- ✓ The *all-is-well* approach is not a viable solution for cache coherence solution, since a common cache memory in multiprocessor system decreases performance.
- ✓ The *software-based* solutions for cache coherence problem are cheap but slow.
- ✓ The *hardware-based* solutions for cache coherence problems are costly but fast.

5.8.4. Hardware Solutions:

1. Cache Snooping Protocol:

Here, a bus controller is assigned to each processor, which monitors the write operations on the bus by other processors. This bus controller is known as *snoopy cache controller*. The snoopy cache controller is responsible to identify if a shared item is being modified by any processor and ensures that all other cache controllers have the most recent updated copy of the shared item to avoid the usage of outdated information from their caches. There are two methods as discussed below, which can either be followed as a snooping cache protocol.

a) *Write-update protocol (or Write-broadcast protocol)*:
In this protocol, whenever a processor writes to a shared item in its cache, it broadcasts to all the other cache controllers about the updated value of the shared item through the system bus. All the cache controllers update their local cache accordingly. This scheme makes the update value readily available in caches of other processors, thus consumes more bandwidth in terms of memory. A solution to this over consumption of memory bandwidth is to keep tracks of the shared items to avoid unnecessary re-broadcasts. An example of write-update protocol is Firefly Protocol, which is used by SPARC center 2000.

b) Write-invalidate protocol:

In this protocol, whenever a processor (let's say P1) writes into a shared item (word) in its cache, it informs all the other cache controller about the location (let's say 3000) of the updated word in its cache. All the cache controllers' snoops on the bus for write operation. They will check if they have a copy of the word which has been overwritten by P1. If yes, then they mark the location of that word in their cache as *invalid* for future reference and *removes* the word from their caches. Afterwards, whenever another processor (let's say P2) tries to access the invalid word (which was a copy of the word from location 3000), it will result in a cache miss and any one of the following operations will

i. If the cache follows *write-through* policy, then the updated item will be transferred to processor P2 from the main memory. Here, the updated item will be available in both - cache memory of processor P1 and main memory, but main memory is the preferred choice in an event of cache miss.

ii. If the cache follows *write-back* policy, then the updated item will be transferred from the cache memory of Processor P1 to Processor P2 via main memory, since at any time, the latest value of the word will only be available in cache of P1 and will be updated in main memory later.

An example of write-invalidate protocol is MESI protocol (Modified Exclusive Shared Invalid), which is used by Intel Pentium 4 and PowerPC.

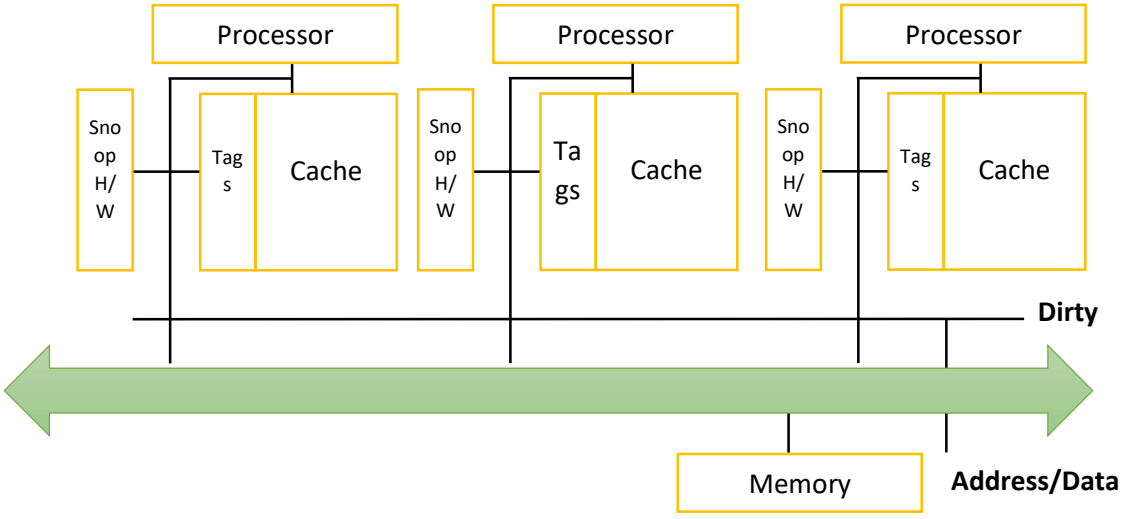


Fig.5.16 Cache Snooping Protocol

Space for learners:

Space for learners:

2. Directory Protocol:

Here, a centralized approach is considered by maintaining a *directory* in the main memory. We define one directory per cache to keep track of state (or information) of every block of main memory present in that cache. In other words, the information in a directory is about the cache memories of processors containing same block from main memory and the state of the block - either *valid* or *invalid*. In order to prevent bottleneck in a directory, the entries in the directory can be distributed.

Whenever an information in the cache is modified by a processor, it the responsibility of the directory controller to check the directory and identify the affected processors. Then the affected processor receives an explicit information from the directory controller about the appropriate action to be taken in order to avoid any incoherency in cache.

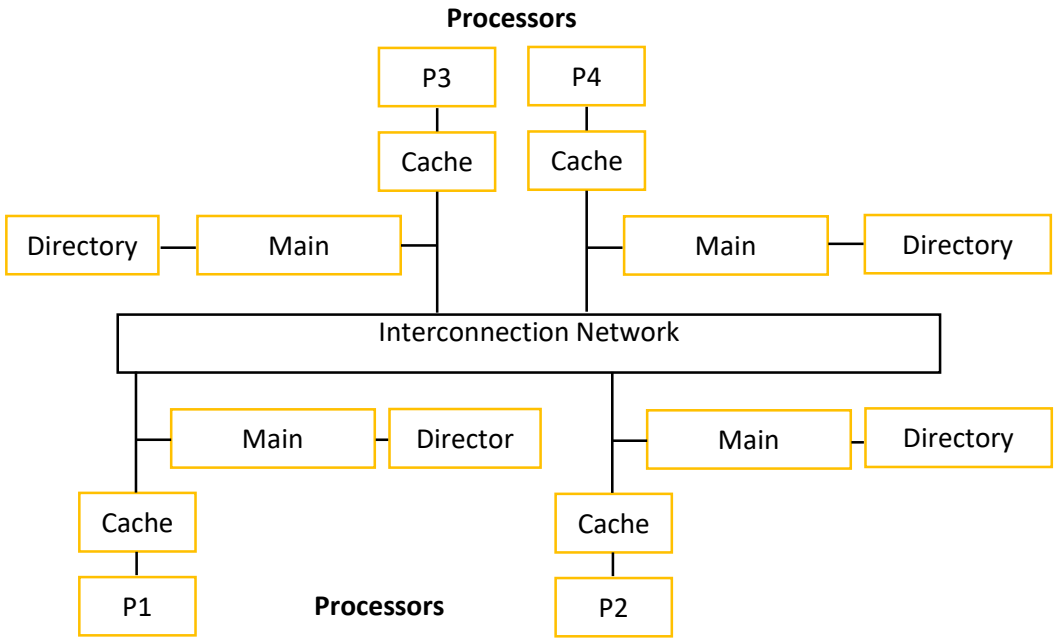


Fig 5.17 Distributed Directory Protocol

5.9 SUMMING UP

- Instruction Set Architecture (ISA) defines a basic set of operations - like arithmetic, logical, branching and memory operations, that must be performed by the system and also provides details about how a machine code doesn't depend on the prime characteristics of the implementation of a particular ISA. Based on architectural complexity, ISA can be classified into CISC and RISC.
- CISC stands for Complex Instruction Set Computer. This approach attempts to reduce the number of instructions per program. In order to do so, the number of cycles per instruction increases. CISC takes several clock cycles to execute instruction. In CISC architecture, the instructions are of variable lengths (from 8-bits to 120-bits)
- RISC stands for Reduced Instruction Set Computer. This approach is needed to minimize the cycles per instruction. In order to do so, the number of instructions per program increases. RISC takes single clock cycle to execute an instruction In RISC architecture, the instructions are of fixed lengths (32-bits)
- In modern times, almost all CISC & RISC processors are superscalar in nature. Superscalar is an implementation for ILP processor architectures in which programs doesn't have any explicit information about parallel execution of instruction and it is the responsibility of the system hardware to detect and construct action plans for any ILPs to be exploited for parallelism.
- VLIW processors are built on an architecture in which programs contain explicit information about parallelism and it is the responsibility of the software, called compiler to identify and communicate it to the hardware by specifying all the independent operations.
- In VLIW Processor, Instruction consists of multiple independent operations grouped together. There are multiple independent functional units. Each operation in the instruction is assigned to different functional units. All functional units share the use of a common large register file.

Space for learners:

- One VLIW instruction encodes at least one operation for each functional unit on each cycle. So, length of the instruction increases with the number of functional units. These operations are assigned to functional units by the position in the given fields within the long instruction word. This is known as slotting.
- EPIC stands for Explicitly Parallel Instruction Computing and is implemented by Hewlett Packard & Intel as Intel Itanium architecture (IA-64). EPIC is a mix of software & hardware, incorporating the advantages of both superscalars and VLIW architectures.
- Like VLIW, EPIC it permits execution of instructions in parallel using a compiler. However, in EPIC apart from identifying and grouping the independent operation in a single instruction, the compiler communicates this via explicit information in the instruction set. That's why EPIC is also known as "independence architecture".
- Unlike VLIW, EPIC retains backward compatibility across different implementations like superscalars, but doesn't need any hardware for dependency checks like superscalars.
- A multiprocessor system is a computer system with more than one processor (typically two or more), where each processor is linked with one another via interconnection network. The focus of a multiprocessor system is to achieve parallel processing, Fault Tolerance, graceful degradation, scalability and modular growth. Multiprocessor system falls under MIMD architecture (Multiple Instruction stream, Multiple Data stream). They are primarily divided into two-types: tightly coupled system and loosely coupled system.
- A tightly coupled multiprocessor, also known as shared memory multiprocessor system, share information between multiple processors via a shared or global memory. Example of tightly-coupled multiprocessor system - Unform Memory Access (UMA) and Non-Uniform Memory Access (NUMA). Symmetric Multiprocessor (SMP) system is an UMA multi-processor system with identical, homogenous processors, which are capable of performing similar functions and utilizes a centralised shared main memory.
- A loosely coupled multiprocessor system, also known as the distributed memory multi-processor system, doesn't share information between multiple processors via a shared

Space for learners:

memory, since each processor has its local dedicated memory, which together forms a distributed memory. Example of loosely-coupled multiprocessor system - Clusters. A cluster consists of a set of computers connected over a local area network (LAN) which function as a single large multiprocessor.

- In multiprocessor systems, the components like CPU and I/O Ports are connected to I/O devices and a memory unit, which can be shared or distributed in nature. The interconnection between the components be of different physical configurations - Time-Shared Common Bus, Multiport Memory, Crossbar Switch, Multistage Switching Network, Hypercube Network
- In time-shared common bus structure, all the processors in the microprocessor system are connected to shared memory and other common resources using a common interconnecting path, called as common system bus. Only one processor at a time can communicate over the bus. The design is simple due to the use of single common system bus. It is a cheap and affordable structure. Since only one processor at a time can transfer or communicate over the system bus, the communication is quite slow.
- In multiport memory structure, the system has separate buses between each memory module and the processors. Each of the memory module has an priority logic to resolve conflict of simultaneous requests from multiple processors. A fixed priority is assigned to each memory ports to avoid memory access conflicts. Due to multiple paths between the processors and memory modules, multiple processors can simultaneously access the memory with high transfer rate. But it is expensive in cost due to huge interconnecting cables requirements.
- In crossbar switch structure, a number of crosspoints are placed at the intersection of memory paths and processor buses. At each crosspoint, there is a control logic to set the desired path between a memory module and a processor. This control logic is basically a switch, which is an electronic circuit. A switch can also resolve the conflict of simultaneous requests from multiple processors to access same memory module in the system based on a fixed priority basis.

Space for learners:

- In multistage switching network structure, we use a switch which can interchange two-inputs, two-outputs to determine the path between multiple processors and multiple memory modules. Hence the name, multistage switching network since it allows to build different possible stages for different combination of inputs & outputs. A very popular topology is called omega switching network which allows exactly one path from each source to any particular destination. The structure is cost effective since we can connect multiple sources to multiple destinations with less amount of wiring. But there is a restriction on the number of simultaneous connections to two destinations connected to same switch is prohibited.
- In hypercube structure, a loosely coupled system comprised of N = 2n numbers of processors are interconnected to each other in a N-dimensional cube. A node of the cube is represented by a processor and an edge of the cube is a communication path connection two nodes. The advantage of hypercube is that is easy to scale the current network to higher configurations and intelligent communication protocols could be easily implemented. However, the multiple paths between processors increases the routing complexity.
- Cache coherence is a condition which states that all the cache lines with a particular shared main memory block must contain same information at any given point of time. This ensures that a multiprocessor system can perform memory operation correctly, by keeping identical multiple copies of information in the caches of the processors involved in execution of a particular instruction.
- Cache coherence problem occurs when cache coherence is not maintained, i.e., a processor updates its cache and other processors doesn't get an updated copy of newly modified data in their cache. This hampers the uniformity of data in all the caches of processors. Cache coherence problem happens in a multiprocessor system, since multiple processor access and works on non-identical multiple copies of data.
- All-is-well approach One of simple scheme can be to restrict the association of local caches for each processor and force

Space for learners:

them to use a shared cache memory. This scheme simply ignores cache coherence problem and moreover increases average access time.

- In software-based solution, the compiler is used to mark data as cacheable and non-cacheable. The cacheable items are allowed to be stored in the local cache of the processor but the non-cacheable items can't be stored in cache and remain in main memory. All the non-sharable & read-only items are tagged as cacheable and the writable shared items are tagged as non-cacheable.
- The cache snooping protocol is a hardware-based solution. Here a bus controller is assigned to each processor, which monitors the write operations on the bus by other processors. This bus controller is known as snoopy cache controller. The snoopy cache controller is responsible to identify if a shared item is being modified by any processor and ensures that all other cache controllers have the most recent updated copy of the shared item to avoid the usage of outdated information from their caches. There is two ways to implement this protocol - write-update protocol (or Write-broadcast protocol) and write-invalidate protocol.
- In write-update/write-broadcast protocol, whenever a processor writes to a shared item in its cache, it broadcasts all the other cache controllers about the updated value of the shared item through the system bus. All the cache controllers update their local cache accordingly.
- In write-invalidate protocol, whenever a processor writes into a shared word in its cache, it informs all the other cache controller about the location of the updated word in its cache. All the cache controllers' checks if they have a copy of that old word. If yes, then they mark the location of that word in their cache as invalid for future reference and removes the word from their caches. Afterwards, whenever another processor tries to access the invalid word, there will be cache miss and actions will be taken depending on whether write-back or write-through policy is followed.
- Directory Protocol is also a hardware solution for cache coherence problem. Here, a centralized approach is considered by maintaining a directory in the main memory.

Space for learners:

We define one directory per cache to keep track of state (either valid or invalid) of every block of main memory present in that cache. The entries in the directory can be distributed. A central memory controller checks the directory to find affected processors in case of any modification of shared data in its cache by a processor and sends explicit instruction to the affected processors to avoid cache incoherence.

Space for learners:

5.10 ANSWERS TO CHECK YOUR PROGRESS

1. Instruction Set Architecture
2. The set of operation defined by instruction set architecture may include arithmetic, logical, branching and memory operations.
3. RISC & CISC
4. Reduced Instruction Set Computer
5. Example of CISC: Intel x86, Example of RISC: ARM
6. False
7. True
8. True
9. Explicitly Parallel Instruction Computing
10. Intel Itanium
11. Compiler
12. Hewlett Packard (HP) & Intel
13. Predicated
14. False
15. False
16. Multiple Instruction stream, Multiple Data stream
17. Tightly-coupled, Loosely-coupled
18. 2^n
19. True
20. True

5.11 POSSIBLE QUESTIONS

1. What is instruction-set architecture? Why is it important?
2. Explain the different types of ISAs.
3. Define Instruction-level parallelism (ILP). How VLIW takes advantage of ILP?

4. Explain the VLIW architecture and the instruction format.
5. State the advantages, disadvantages and applications of VLIW architecture.
6. What is EPIC? How does it differ from VLIW?
7. Write a short note of EPIC architecture.
8. Explain in brief about multiprocessor system. How does it differ from multicomputer system?
9. Differentiate between tightly-coupled and loosely-coupled microprocessor
10. How does uniform memory access differ from non-uniform memory access?
11. Explain in brief about different interconnection structures in multiprocessor systems.
12. What is the difference between the cross-switch and multistage-switch?
13. What are the two widely-used policies of cache write operation?
14. What is the software-based approach to solve the cache coherence problem?
15. Write a short note of hardware-based solutions for cache coherence problem.

Space for learners:

5.12 REFERENCES AND SUGGESTED READINGS

1. Mano, M. Morris. *Computer System Architecture, 3E.* Pearson Education India, 2007.
2. Govindarajalu, B. *Comp Arch and Org, 2E*. Tata McGraw-Hill Education, 2010.
3. Hamacher, V. Carl, Zvonko G. Vranesic, and Safwat G. Zaky. *Computer organization and Embedded Systems, 6E*. McGraw-Hill, Inc., 2012.
4. Al-Hothali, Samaher. "Snoopy and directory-based cache coherence protocols: A critical analysis." *Journal of Information & Communication Technology (JICT)* 4.1 (2010): 11.
5. Semiconductors, Philips. "An introduction to very-long instruction word (VLIW) computer architecture." *Philips Semiconductors* (1997).
6. Smotherman, Mark. "Understanding EPIC architectures and implementations." *40th Annual Southeast ACM Conference*. 2002.

7. Halfhill, Tom R. “VLIW Microprocessors” *Computerworld India*, 14 Feb. 2000, https://www.computerworld.com/article/2593626/vliw-microprocessors.html.
8. “Instruction set architecture” *Wikipedia*, https://en.wikipedia.org/wiki/Instruction_set_architecture. Accessed 01 Aug. 2021.
9. “Very long instruction word” *Wikipedia*, https://en.wikipedia.org/wiki/Very_long_instruction_word. Accessed 01 Aug. 2021.
10. “Explicitly Parallel Instruction Computing" *Wikipedia*, https://en.wikipedia.org/wiki/Explicitly_parallel_instruction_computing. Accessed 01 Aug. 2021.
11. Zaccone, Giancarlo. *Python parallel programming cookbook*. Packt Publishing Ltd, 2015.
12. Beckmann, Nathan. “Static Scheduling & VLIW.” *Carnegie Mellon University*, https://www.cs.cmu.edu/afs/cs/academic/class/15740-s17/www/lectures/13-static-scheduling.pdf. Accessed 02 Aug. 2021
13. Shanthi, A. P. “Multiple Issue Processors II” Univeristy of Maryland,https://www.cs.umd.edu/~meesh/411/CA-online/chapter/multiple-issue-processors-ii/index.html. Accessed 01 Aug. 2021.
14. "VLIW Processors" Slideshare, https://www.slideshare.net/shudhanshu29/vliw-processors. Accessed 01 Aug. 2021.
15. Schlansker, Michael S., and B. Ramakrishna Rau. *EPIC: An architecture for instruction-level parallel processors*. Hewlett-Packard Laboratories, 2000.

Space for learners: