UNIT 13: INTRODUCTION TO PARALLEL PROCESSING

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 Pipelining

- 9.3.1 Hazards of Pipelining
- 9.3.2 Structural Hazard
- 9.3.3 Data Hazard
- 9.3.4 Control Hazard

9.4 Parallel Processing

9.5 Let Us Sum Up

9.6 Further Readings

9.7 Answers to Check Your Progress

9.8 Model Questions

13.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- define pipelining
- describe the different hazards associated with pipelining
- describe concepts related to superscalar and parallel processing

13.2 INTRODUCTION

In the previous units, we have already discussed the basic organization of a computer and how a processor executes a program. To execute a sequence of instructions, the processor uses a cycle of fetch – decode – execute operations: an instruction is fetched from memory, it is then decoded and executed; the next instruction in the sequence is fetched, decoded

and executed and the cycle is repeated until the end of the program (or sequence of instruction) is reached.

In this unit, we shall discuss how the simple fetch – decode – execute cycle can be optimized so that the performance of the system is improved. Pipelined execution of instructions is based on the fact that the different phases of the fetch – decode – execute cycle require completely different hardware units. Pipelining involves reordering instructions so that all hardware units are used optimally to achieve better performance. Parallel processing further involves executing several independent instructions in the program at the same time to achieve better performance. In the next unit, we will explore the concepts related to CISC and RISC architecture.

13.3 PIPELINING

The basic idea of pipelining is similar to the assembly line operation in manufacturing plants. For example, in a car manufacturing plant, there is a sequence of stations each of which performs different stages of the assembling the car. The first station prepares the chassis, the next station adds the body to the chassis, the next station installs the engine and so on. While one station is preparing the chassis of one car, the second station is adding the body to the chassis of another car, the third station is installing the engine in yet another car and so on. In this way, all stations are fully utilized at all times so that an optimal number of cars can be manufactured. Similarly, in pipelining, the execution of a single instruction has at least three different stages (fetch, decode and execute) that are performed by different hardware units. While the system bus is used to fetch an instruction from the memory, the decoder unit of CPU can be used to decode a different instruction and the ALU can be used to execute yet another instruction. Compared to sequential execution of instruction, pipelined execution leads to considerably better performance.

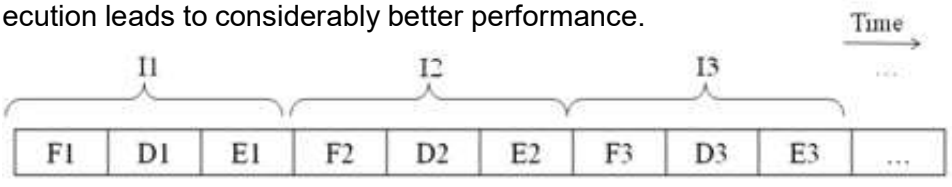


Figure 9.1: (a) Sequential Execution of Instructions I1, I2, I3, where F1, D1, E1 means fetch, decode and execution phase of instruction 1.

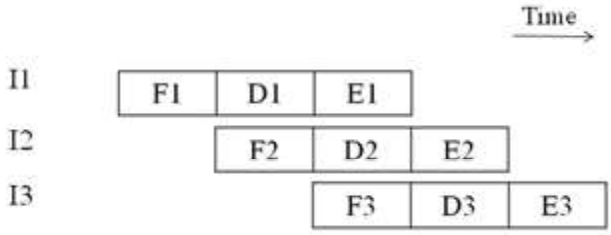


Figure 9.1: (b) Pipelined Execution of Instructions I1, I2, I3

Pipelined execution of sequence of instructions is shown in Fig 9.1 (b). The time taken to complete any of the fetch, decode or execute phases of an instruction is one clock cycle. While sequential execution (Fig 9.1(a)) completes execution of instructions I1, I2 and I3 in nine clock cycles, pipelined execution completes the same in just five clock cycles. An even greater difference in the number of clock cycles is seen if the total number of instructions to be executed is more. The number of pipeline stages may vary depending on the number of separate hardware units available in the system. The increase in performance, as a result of pipelining, is proportional to the total number of pipeline stages.

13.3.1 Hazards of Pipelining

The increase in performance, due to pipelining, is proportional to the number of pipelining stages only if the pipelined executions (as shown in figure 9.1 (b)) can be continued throughout the entire program. However, during pipelined execution of a program, there are many situations when one or more stages of executions may be halted or ***stalled*** temporarily. As a result, it is often not possible to achieve an uninterrupted pipelined execution of instructions.

The pipelined execution of instructions may be interrupted due to various reasons – for example, the execution phase of some instruction requires more than one clock cycle. Figure 9.2 shows a situation where the execution phase of I1 (E1) requires two clock cycles instead of one. The execution phase of I2 (E2) has to be ***stalled*** for one clock cycle – instead of being completed in clock cycle 4, it is completed in clock cycle 5. This is because the execution unit had not completed E1 and can handle execution of only instruction at a time. Any condition that causes the pipeline to stall is called a ***hazard***.

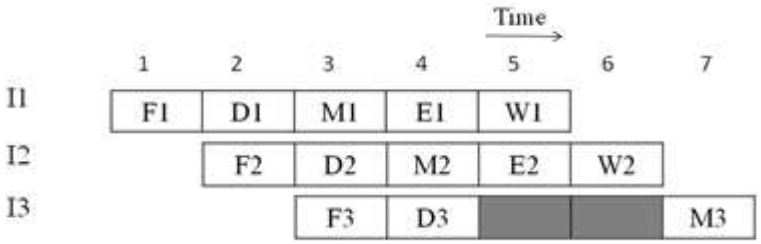


Figure 9.3: Structural hazard caused by I1 and I3 requiring memory access at the same time

To handle structural hazards, in general the number of hardware resources may be increased.

13.3.3 Data Hazard

Data hazards occur when an instruction operates on a data that is to be produced by some other instruction in the pipeline. Say, there is sequence of instructions as follows:

I1: Add A, 5

I2: Load R1, A

I3: Add R2, R1

The pipelined execution of these instructions is shown in Fig 9.4. In this sequence, immediately after adding 5 to the memory location A, the value of A has to be loaded to register R1. However, in the pipelined execution, even before the result of the addition operation is written into memory location A, instruction I2 attempts to access the value in A. If the initial value of A is 4, then after execution of I1 the value changes to 9 and the value loaded into register R1 should be 9. However, due to pipelining, even before the value 9 is stored in memory location A, I2 is executed and the value loaded into register R1 is 4. Such an incorrect execution will affect the execution of instruction I3 and the end result.

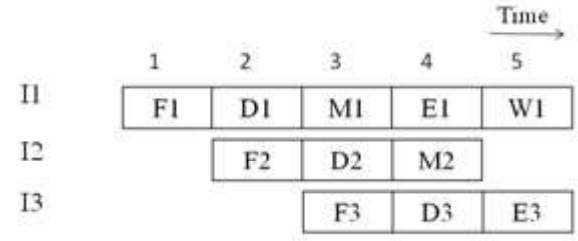


Figure 9.4 : Effect of data hazard if not handled properly

```
ADD      A, AH
SUB      AH, 30h
ADD      CH, B
CMP      CH, 10
JG       label1
ADD      AH, B
JMP      label2
label1:  SUB  AH, B
label2:  MOV  BH, B
```

13.4 PARALLEL PROCESSING

Traditionally, the execution of instructions in a computer is assumed to be strictly sequential in nature. However, with pipelined execution of instructions, the sequential view of program execution is somewhat inaccurate. During pipelined execution, at any given clock cycle, there are several instructions in the pipeline that are in different stages of their execution. However, only a single instruction is in its execution stage at any given clock cycle. This is taken further with ***superscalar processors*** that allow multiple instructions to be executed at the same time.

Superscalar Processors

Superscalar organizations have several execution units within a single processor. This allows multiple instructions from the same program to be executed in parallel. As a result, the number instructions executed per cycle is more than one. This is also known as ***instruction-level parallelism***. Figure 9.6 shows an example of instruction level parallelism in a superscalar processor with two execution units.

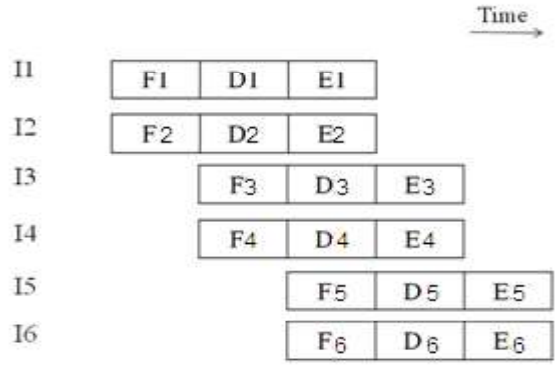


Figure 9.6 : Instruction level parallelisms in a superscalar processor with two execution units

However, such an aggressive approach to increase performance of the system also has several issues. The various hazards encountered in pipelining are encountered in superscalar processors as well and are seen to be even more prominent. Further, most programs are written assuming a sequential execution of the instructions, with several dependencies amongst the instructions. In superscalar processors, multiple instructions are executed at the same time. If the dependencies amongst the instructions being executed parallel are not handled, then it may result in incorrect executions. Thus, in such systems, in order to achieve high performance and correct executions, it is important for the compiler to be able to deal with the hazards and the dependencies effectively by proper selection and reordering of the instructions.



13.5 LET US SUM UP

- Pipelining is based on dividing the work done in the instruction execution cycle into stages such that each stage can be executed by completely different hardware units.
- Pipelining involves concurrent execution of multiple instructions even though the instructions in the pipeline are in different stages of their execution.
- Pipelining does not follow a strictly sequential execution of instruction.
- Pipelining hazards occur because most programs are written for a sequential execution.
- Instructions in a program have several dependencies amongst themselves which are properly executed only if executed sequentially.
- There are three most common types of pipelining hazards: data hazards, control hazards, structural hazards.
- Pipelining aims to achieve a performance of one instruction per cycle.

- Superscalar processors have more than one execution units within the same processor. When combined with pipelining the performance of superscalar processors is expected to be more than one instruction per cycle.
- Superscalar processors encounter more aggressive forms of pipelining hazards.

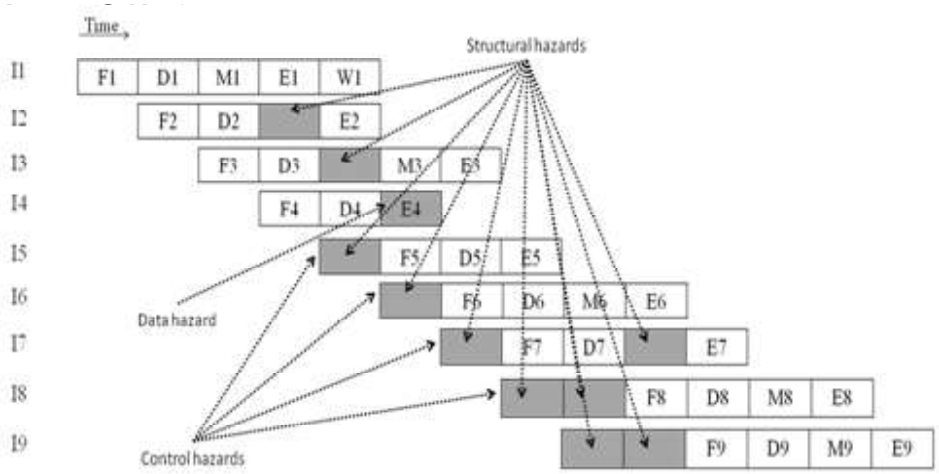
13.6 FURTHER READINGS

1) Hamacher, V. C., Vranesic, Z. G., Zaky, S. G., Vransic, Z., & Zakay, S. (1984). Computer organization (Vol. 3). New York et al.: McGraw-Hill.
2) Stallings, W. (2000). Computer organization and architecture: designing for performance. Pearson Education India.

13.7 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: The different categories of pipelining hazards are: Structural hazards, Data Hazards and Control hazards.

Ans to Q No 2: Data hazards occur when an instruction operates on a data that is also being operated on some by some other instruction in the pipeline.





13.8 MODEL QUESTIONS

Q1. What is pipelining?

Q2. What are superscalar processors?

Q3. Differentiate between pipelining and superscalar processors.

Q4. Define instruction level parallelism?

Q5. Explain the pipelining hazards with example.

Q6. Differentiate between data hazard and control hazard.

Q7. Define parallel processing. Give the advantages of using parallel processing?

****