UNIT STRUCTURE

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to

- define and elaborate what is pipelining
- describe instruction pipelining
- illustrate the hazards in pipelining
- know the benefits of pipelining

6.2 INTRODUCTION

So far, you have came across with lots of the interesting topics of computer organization like addressing modes, instruction formats, ALU and CU organizations etc. Whatever you have learnt from the previous units - the main objective in the mind of the researchers is to design a computer system with highest performance by using those techniques. In this unit, we will discuss one of the most important technique called pipelining, which is used in the modern computers to achieve extreme performance.

6.3 PIPELINE - THE BASIC TECHNIQUE

The speed of computation of the modern processors increases day by day to a certain extent. This is happens by adopting new techniques for organizing the concurrent activities in a computer system. One of such technique is called pipelining. The idea behind the pipelining is very simple.

In everyday life, people do many tasks in stages. For instance, when we do the laundry, we place a load in the washing machine. When it is done, it is transferred to the dryer and another load is placed in the washing machine. When the first load is dry, we pull it out for folding or ironing, moving the second load to the dryer and start a third load in the washing machine. We proceed with folding or ironing of the first load while the second and third loads are being dried and washed, respectively. We may have never thought of it this way but we do laundry by **pipeline processing**.

A Pipeline is a series of stages, where some work is done at each stage. The work is not finished until it has passed through all stages.

Now let us see, how the idea of pipelining can be used in computers. We know that the processor executes a program by fetching and executing instructions, one after the other. In the following figure, shows two hardware unit of a processor, one for fetching the instructions and the other for executing instructions. The intermediate storage buffer B1 stores the instruction fetched by the fetch unit. This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction. Thus, with pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing execution of the first instruction, holding them in a buffer close to the processor until each instruction operation can be performed.

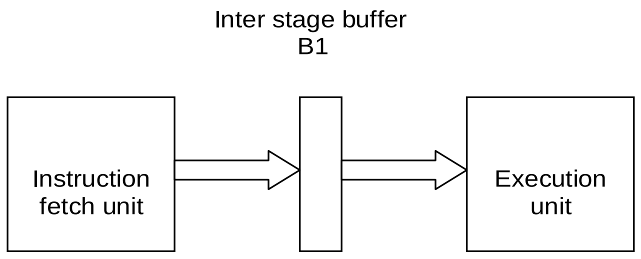


Fig 6.1 Fetch and Execution Unit

The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

So in computers, a pipeline is the continuous and somewhat overlapped movement of instruction to the processor to perform an instruction.

But what will happen without pipelining ? Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory, and so forth. While fetching (getting) a instruction, the executing part of the processor will remain idle. It must wait until it gets the next instruction. Thus, it results slower in execution because less number of instructions will be executed during a given time slot.

NOTE

The **IBM 7030**, also known as **Stretch**, was IBM's first transistorized supercomputer in 1961. Original priced was $13.5 million. Even though the 7030 was much slower than expected, it was the fastest computer in the world from 1961 until the first CDC 6600 became operational in 1964.

Thus, the Pipelining is used to obtain improvements in processing time that would be unobtainable with existing non-pipelined technology. The IBM 7030 (the Stretch Computer) had attained its over-all peprformance of 100 times by using the pipelining technique, whereas circuit improvements would only give a factor-of-10 improvement. This goal could only be met with overlapping instructions, i.e. pipelining.

John Hayes provides a definition of a pipeline as it applies to a computer processor.

"*A pipeline processor consists of a sequence of processing circuits, called segments or stages, through which a stream of operands can be passed. Partial processing of the operands takes place in each* *segment. A fully processed result is obtained only after an operand set has passed through the entire pipeline."*

6.4 TYPES OF PIPELINES

The concept of pipelining is divided into two categories :

- **Instructional pipeline**, where different stages of an instruction fetch and execution are handled in a pipeline.
- **Arithmetic pipeline**, where different stages of an arithmetic operation are handled along the stages of a pipeline.

NOTE

The processor operates at the speed of an internal clock whose clock rate is determined by the frequency of an oscillator crystal (quartz crystal) that, when subjected to an electrical currant, send pulses, called "peaks". The clock speed (also called cycle), corresponds to the number of pulses per second, written in Hertz (Hz). Thus, a 200 MHz computer has a clock that sends 200,000,000 pulses per second. Clock frequency is generally a multiple of the system frequency (FSB, Front-Side Bus), meaning a multiple of the mother-board frequency.

6.5 INSTRUCTION PIPELINING

An instruction pipeline is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). The fundamental idea is to split the processing of a computer instruction into a series of independent steps, with storage at the end of each step.

Most of the modern CPUs are designed to operate on a synchronization signal which known as a **clock signal**. In each clock pulse, the fetch step and execution step should be completed. The above situation discussed in the previous section (Fig. 6.1) can be illustrated by introduction of clock pulse as shown in the figure below :

In the first clock cycle, the fetch unit fetches an instruction I_1(step F_1) and stores it in buffer B_1 at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2). At the same time, the execution unit performs the operation specified by instruction I_1, which is available to it in buffer B_1 (step E_1).

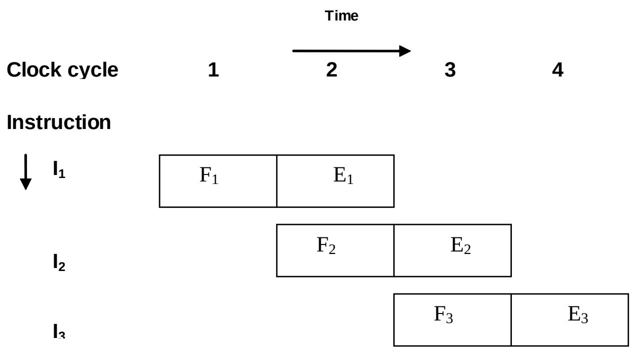


Fig. 6.2 Pipelined Execution

By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available. Instruction I_2 is stored in B_1, replacing I_1, which is no longer needed. Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetched unit.

Thus, the above fetch and execute unit constitute a two stage pipeline. Both the fetch and execute units are kept busy all the time after each clock pulse and also new information is loaded into the buffer after each clock pulse.

Actually, the steps of processing an instruction is not limited within two steps, instead four steps are used as shown in Fig. 5.4 to process an instruction. These steps are :

- **Fetch** : read the instruction from the memory
- **Decode** : decode the instruction and fetch the source operands
- **Execute** : perform the operation specified by the instruction
- **Write** : store the result in the destination location

So, it will need four different hardware units as shown in Fig 5.3 for performing each of these steps simultaneously and without interfering with one another (because for two steps we get two hardware unit as shown in the fig. 6.1).

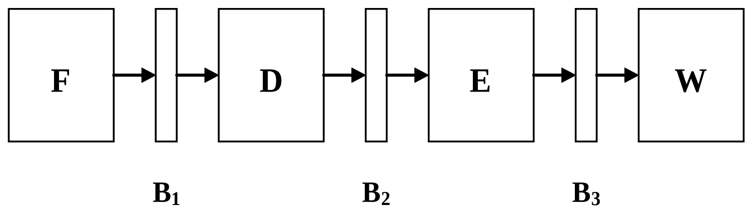


Fig. 8.3 Hardware units need for Instruction Processing

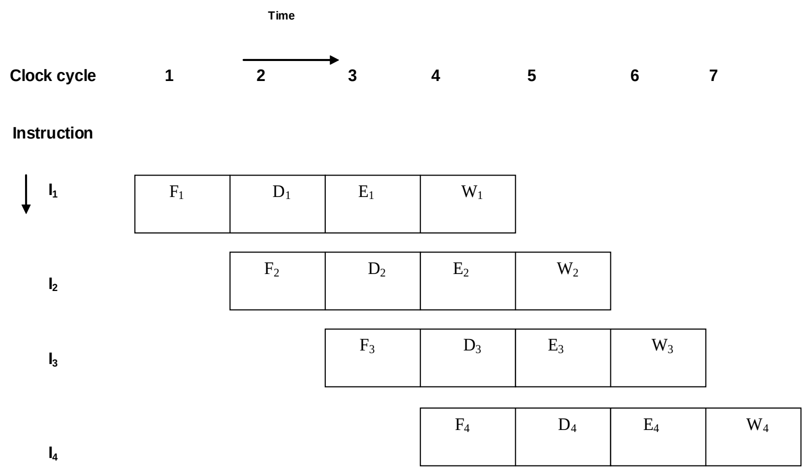


Fig. 6.4 Instruction processing in four steps

Information is passed from one unit to the next unit through a storage buffer. The information in the buffers at the *clock pulse 4* is given in the following table :

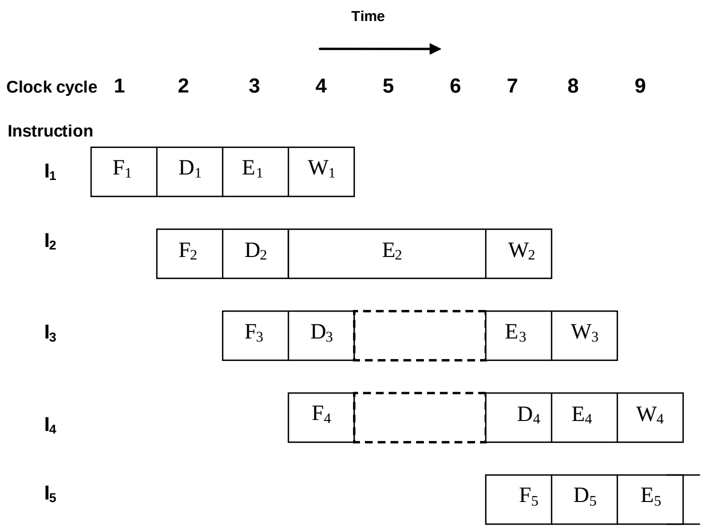
Buffers	**Contents**
B_1	Holds instruction I_3 that was fetched in clock pulse 3 and will be decoded bythe decoding unit.
B_2	Holds the information produced by the decoding unit in clock pulse 3. This buffer also holds the information needed for the write step of instruction I_2 (step W_2). This information must be passed on to stage W in the following clock cycle to enable that stage toperform the required Write operation.
B_3	Holds the result produced by the execution unit and the destination information for instruction I_1.

6.6 PIPELINE PERFORMANCE

Pipelining increases the CPU instruction throughput, its meaning is that pipelining increases the number of instructions completed per unit of time. But it does not reduce the execution time of an individual instruction. In fact, it usually slightly increases the execution time of each instruction due to overhead in the pipeline control. The increase in instruction throughput means that a program runs faster and has lower total execution time.

The potential increase in performance resulting from pipelining should be proportinal to the number of pipeline stages. But in practical, this is not absolutely correct because due to some other factors the performace varies.

A pipeline stage may not be able to complete its processing task during the alloted time slot. The execution stage actually resposible for arithmetic and logic operation and one clock cycle may not be sufficient for it. The following figure shows that in instruction I_2 the Execution stage takes three cycles (4,5,6) to complete. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with.



Time
Clock cycle 1 2 3 4 5 6 7 8 9
Instruction
I_1 F_1 D_1 E_1 W_1
I_2 F_2 D_2 E_2 W_2
I_3 F_3 D_3 E_3 W_3
I_4 F_4 D_4 E_4 W_4
I_5 F_5 D_5 E_5

Fig. 6.5 Execution stage taking more than one clock cycle

Meanwhile, the information in buffer B_2 must remain intact until the Execute stage has completed its operation. This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in B_1 cannot be overwritten. Thus, steps D_4 and F_5 must be postponded as shown.

Thus, the normal pipeline operation interrupted for two clock cycle, which is called **stalled**. In stalled situation, the normal pipeline operation halt for more than one clock cycle. In the figure, we see that, pipeline functioning resumes normally from the clock cycle 7.

6.7 PIPELINE HAZARDS

Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its assigned clock cycle. Then, the instruction is said to be stalled. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall. This condition is clearly illustrated in the Fig. 5.5. Thus, the hazards reduces the performance gained by pipelining.

There are three classes of hazards :

a) Structural hazards

b) Data hazards

c) Control hazards

a) Structural hazards :

Structural hazards arise from resource conflict. This is the situation when two instructions ***require the use of a hardware resource*** at the same time. This hazard may arise frequently when two instruction refers to the memory locations at the same time. Suppose one instruction try to access the memory as result of Execute or Write stage and another instruction tries to access the memory for fetching instruction. In such a condition, the structural hazards arises. If instruction and data reside inthe same cache unit, only one instruction can proceed and the other instruction is delayed. In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.

b) Data hazards :

Data hazard arises when the output of one instruction is fed to the input of the next instruction. 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. As a result, some operation has to be delayed, and the pipeline stalls. We have already discussed a situation (in Fig. 6.5) where the normal pipeline operations were halted for two clock cycle. That was happened due to the data hazards only.

c) Control hazards :

Due to the delay in availability of an instruction the pipeline may stalled. For example, a cache miss causes the instruction to be fetched from the main memory. Such hazards are often called *control hazards or instruction hazards*.

The following Fig. illustrates a cache miss in pipeline operation. Here, the instruction I_1 is fetched from the cache in cycle 1, and its execution proceeds normally. Next, the fetch operation for the instruction I_2, which is started in cycle 2, results in a cache miss, it means that the instruction I_2 is not available in the cache memory and that instruction will be fetched from the main memory. The instruction fetch unit must now suspend any further fetch requests and wait for I_2 to arrive. At the end of the clock cycle 5, the instruction I_2 is received and loaded into the buffer B_1. The pipeline resumes its normal operation from the clock cycle 5.

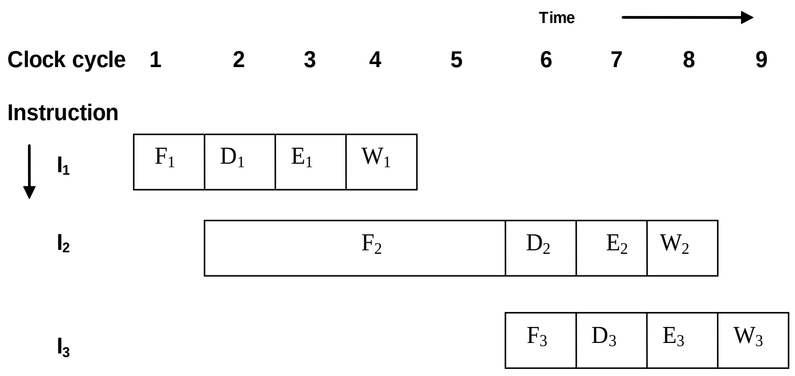


Fig. 6.6 Cache miss in pipeline operation

6.8 MERITS AND DEMERITS OF PIPELINING

Pipelining does not help in all cases. There are several possible disadvantages. An instruction pipeline is said to be *fully pipelined* if it can accept a new instruction at every clock cycle. A pipeline that is not fully pipelined has wait cycles that delay the progress of the pipeline.

Advantages of Pipelining:

1. The cycle time of the processor is reduced, thus increasing instruction issue-rate in most cases.
2. Some combinational circuits such as adders or multipliers can be made faster by adding more circuitry. If pipelining is used instead, it can save circuitry vs. a more complex combinational circuit.

Disadvantages of Pipelining:

1. A non-pipelined processor executes only a single instruction at a time. This prevents branch delays (in effect, every branch is delayed) and problems with serial instructions being executed concurrently. Consequently the design is simpler and cheaper to manufacture.
2. The instruction latency in a non-pipelined processor is slightly lower than in a pipelined equivalent. This is due to the fact that extra flip flops must be added to the data path of a pipelined processor.

3. A non-pipelined processor will have a stable instruction bandwidth. The performance of a pipelined processor is much harder to predict and may vary more widely between different programs.

LET US KNOW

We have come to know that pipeline operation will be effective if it completes the every stage in each clock cycle. So, each clock cycles should sufficently long to complete the task of each stage. Thus, the performance of a pipeline will increase if the task performed in each stage require the same amount of time.

Now, consider a fetch operation where the instructions are fetched from the main memory and for that fetched operation one clock cycle may not be sufficient. Because the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor. So, if each instruction fetch requires access to the main memory, then the pipelining technique will be a time consuming process and meaning less job.

We have already know that cache memory is the second fastest memory element in the memory hierarchy. When a cache is included on the same chip of the processor, called *processor cache*, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor. Thus, the use of the cache memory eliminates the above difficulty and accelerates the pipeline operations.