MODULE 5:

Basic Processing Unit and Pipelining

Basic Processing Unit: Some Fundamental Concepts: Register Transfers, Performing ALU operations, fetching a word from Memory, Storing a word in memory. Execution of a Complete Instruction.
Pipelining: Basic concepts, Role of Cache memory, Pipeline Performance.

SOME FUNDAMENTAL CONCEPTS

The **processing unit** which executes machine instructions and coordinates the activities of other units of computer is called the **Instruction Set Processor (ISP)** or **processor** or **Central Processing Unit (CPU).**

The **primary function of a processor** is to execute the instructions stored in memory. Instructions are fetched from successive memory locations and executed in processor, until a branch instruction occurs.

• To **execute an instruction**, **processo**r has to **perform** following **3 steps**:
1. **Fetch** contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as:
   IR ← [[PC]]
2. Increment PC by
   4. PC ←[PC] +4
3. Carry out the actions specified by instruction (in the IR).

The steps 1 and 2 are referred to as **Fetch Phase**.
Step 3 is referred to as **Execution Phase.**

SINGLE BUS ORGANIZATION

• Here the processor contain only a single bus for the movement of data, address and instructions.
• ALU and all the registers are interconnected via a **Single Common Bus** (Figure 7.1).
• Data & address lines of the external **memory-bus** is connected to the internal processor-bus via MDR & MAR respectively.
(MDR -> Memory Data Register, MAR -> Memory Address Register).
• **MDR** has 2 inputs and 2 outputs. Data may be loaded
→ into MDR either from memory-bus (external) or
→ from processor-bus (internal).
• **MAR**"s input is connected to internal-bus; MAR"s output is connected to external- bus. (address sent from processor to memory only)

- **Instruction Decoder & Control Unit** is responsible for

→ Decoding the instruction and issuing the control-signals to all the units inside the processor.

→ implementing the actions specified by the instruction (loaded in the IR).

- **Processor Registers** - Register R0 through R(n-1) are also called as General Purpose Register.

The programmer can access these registers for general-purpose use.

- **Temporary Registers** – There are 3 temporary registers in the processor. Registers

- **Y**, **Z** & **Temp** are used for temporary storage during program-execution. The programmer cannot access these 3 registers.

- In **ALU**, 1) "A" input gets the operand from the output of the multiplexer(MUX). 2) "B" input gets the operand directly from the processor-bus.
- There are 2 options provided for "A" input of the ALU.
- MUX is used to select one of the 2 inputs.
- **MUX** selects either

→ output of Y or

→ constant-value 4(which is used to increment PC content).

- An instruction is executed by performing one or more of the following operations:

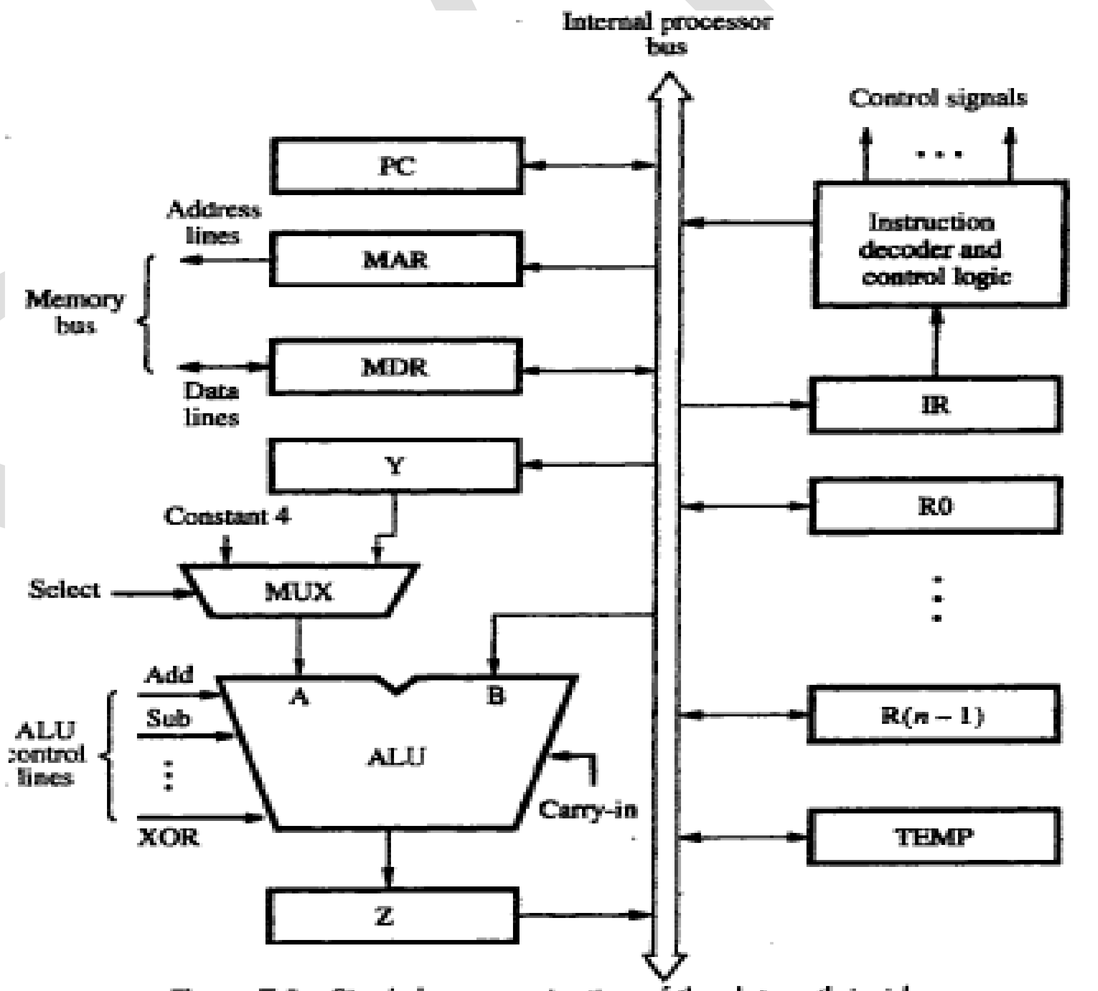


Figure 7.1 Single-bus organization of the datapath inside a processor.

1) Transfer a word of data from one register to another or to the ALU.
2) Perform arithmetic or a logic operation and store the result in a register.
3) Fetch the contents of a given memory-location and load them into a register.
4) Store a word of data from a register into a given memory-location.

- **Disadvantage:** Only one data-word can be transferred over the bus in a clock cycle. **Solution:** Provide multiple internal-paths. Multiple paths allow several data- transfers to take place in parallel.

REGISTER TRANSFERS

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register, two control-signals are used: Riin & Riout. These are called **Gating Signals**
- Riin=1,the data on the bus are loaded into Ri,
- Riout=1,the contents of register are placed on the bus,
- Riout=0,the bus can be used for transferring data from other registers.

Suppose we wish to transfer the contents of register R1 to register R2. This can be accomplished as follows:

1. Enable the output of registers R1 by setting R1out to 1 (Figure 7.2). This places the contents of R1 on processor-bus.
2. Enable the input of register R4 by setting R4in to 1. This loads data from processor-bus into register R4.

- All operations and data transfers within the processor take place within time-periods defined by the **processor-clock.**

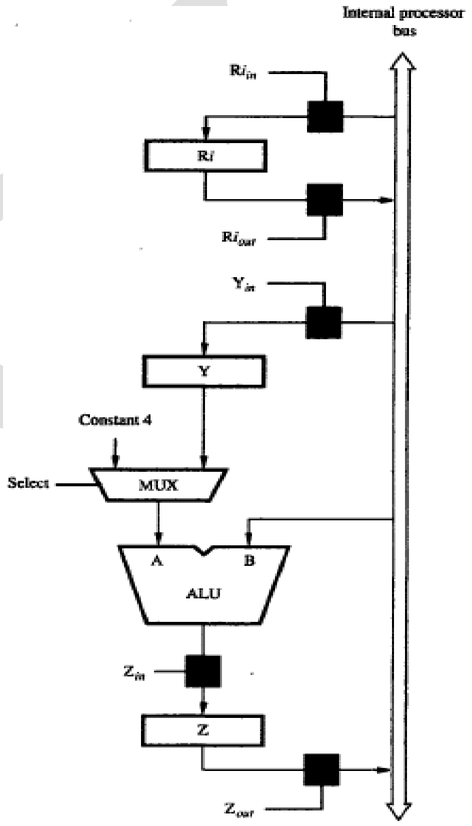


Figure 7.2 Input and output gating for the registers in Figure 7.1.

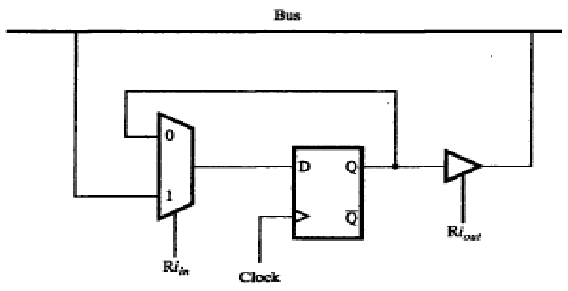


Figure 7.3 Input and output gating for one register bit.

- The control-signals that govern a particular transfer are asserted at the start of the clock cycle.

Input & Output Gating for one Register Bit

Implementation for one bit of register Ri(as shown in fig 7.3)

- All operations and data transfers are controlled by the **processor clock.**
  - A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.
  - Riin=1,Multiplexer selects data on the bus. This data will be loaded into flip-flop at rising-edge of clock.
  - Riin=0,Multiplexer feeds back the value currently stored in the flipflop
  - Q output of flip-flop is connected to bus via a tri-state gate.
  - When Riout=0, gates output in the high-impedance state.
  - When Riout=1,gate drives the bus to 0 or 1,depending on the value of Q.

PERFORMING AN ARITHMETIC OR LOGIC OPERATION(refer fig:7.2)

- **The ALU is a combinational circuit that has no internal storage.**
- The ALU performs arithmetic and logic operations on the 2 operands applied to its A and B inputs.
- ALU gets the two operands, one is from MUX and another from bus. The result is temporarily stored in register Z.
- Therefore, a sequence of operations [R3]=[R1]+[R2].

1) R1out, Yin
2) R2out, Select Y, Add, Zin
3) Zout, R3in

Instruction execution proceeds as follows:

Step 1 --> Contents from register R1 are loaded into register Y.

Step2 --> Contents from Y and from register R2 are applied to the A and B inputs of ALU; Addition is performed & Result is stored in the Z register.

Step 3 --> The contents of Z register is stored in the R3 register.

- The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

FETCHING A WORD FROM MEMORY

- To fetch instruction/data from memory, the processor has to specify the address of the memory location where this information is stored and request a Read operation.
- processor transfers required address to MAR. At the same time, processor issues Read signal on control-lines of memory-bus.
- When requested-data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers in the processor.

The Connections for register MDR has shown in fig 7.4

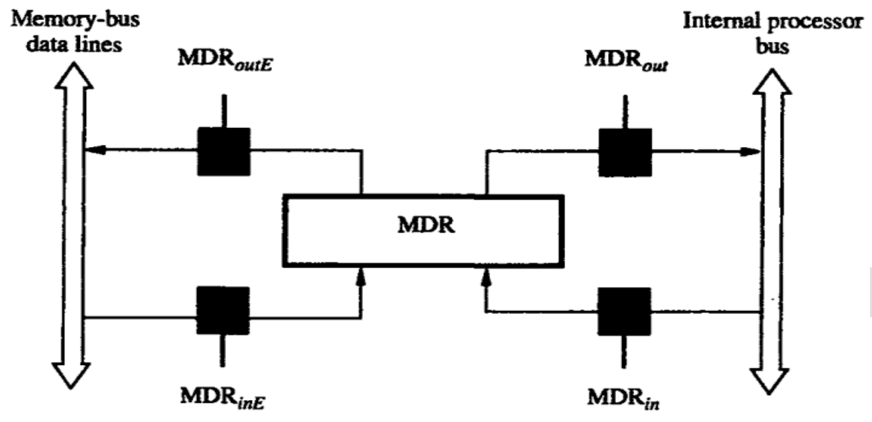


Figure 7.4 Connection and control signals for register MDR.

CONTROL-SIGNALS OF MDR

- The **MDR register has 4 control-signals** (Figure 7.4):

1) **MDRin & MDRout** control the connection to the **internal** processor data bus &
2) **MDRinE & MDRoutE** control the connection to the **external** memory Data bus.

- **Similarly, MAR register has 2 control-signals.**

1) **MARin:** controls the connection to the internal processor address bus &
2) **MARout:** controls the connection to the memory address bus.

The response time of each memory access varies. To accommodate this MFC is used(MFC= Memory Function Completed)

MFC=1 indicate that contents of specified location have been read and are available on the data lines of the memory bus.

- Consider **the instruction Move (R1),R2**. The action needed to execute this instruction are

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

The sequence of steps is (Figure 7.5):

1) R1$_{out}$,MAR$_{in}$,Read ;desired address is loaded into MAR & Read command is issued.
2) MDR$_{inE}$,WMFC; load MDR from memory-bus & Wait for MFC response from memory.
3) MDR$_{out}$, R2$_{in}$; load R2 from MDR.

where WMFC=control-signal that causes processor's control. circuitry to wait for arrival of MFC signal.

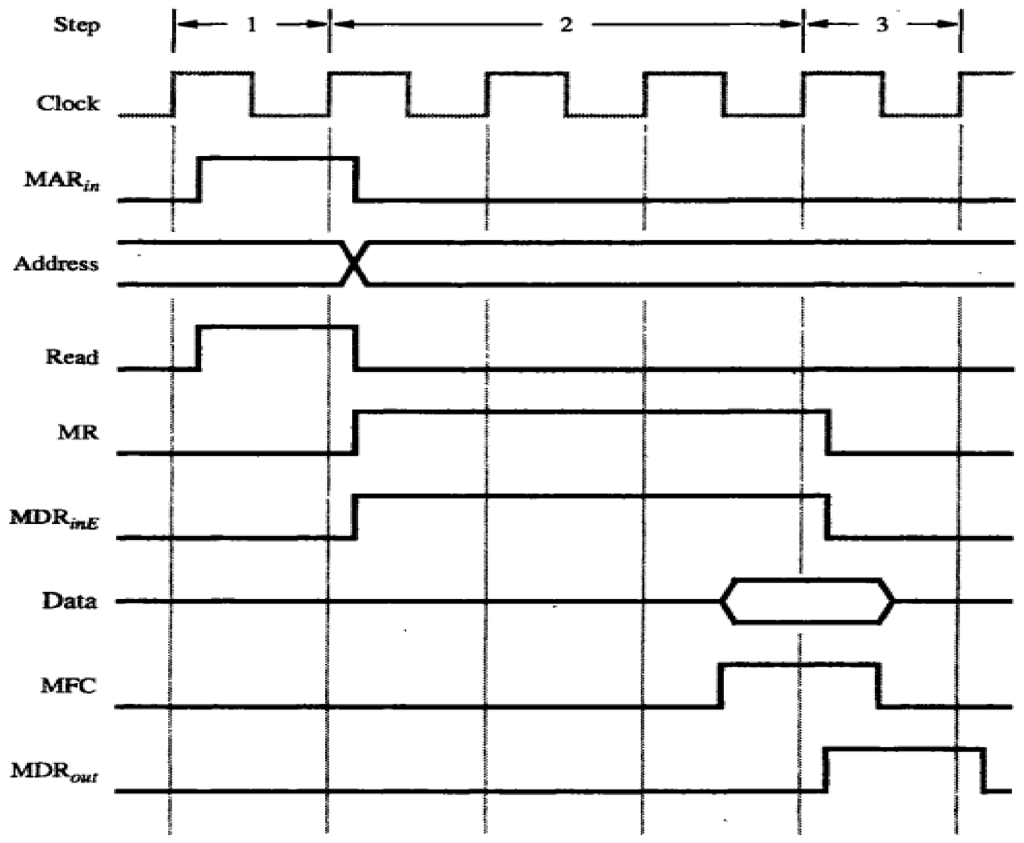


Figure 7.5 Timing of a memory Read operation.

Storing a Word in Memory

• Consider the instruction ***Move R2,(R1).*** This requires the following sequence:

1) R1$_{out}$, MAR$_{in}$;desired address is loaded into MAR.
2) R2$_{out}$,MDR$_{in}$,Write ;data to be written are loaded into MDR & Write command is issued.
3) MDR$_{outE}$, WMFC ;load data into memory-location pointed by R1 from MDR.

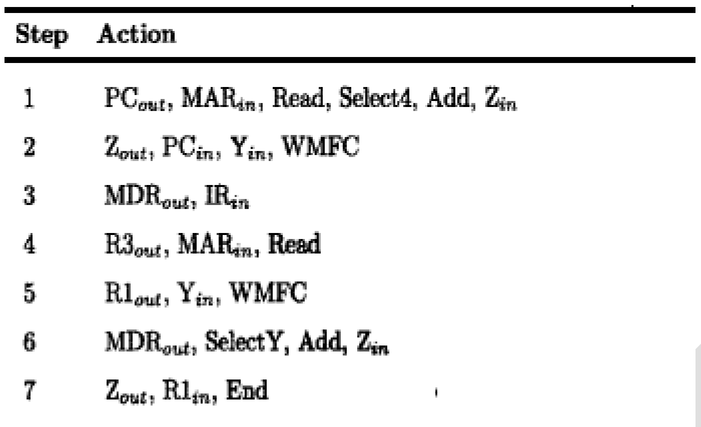
EXECUTION OF A COMPLETE INSTRUCTION

• Consider the instruction ***Add (R3),R1*** which adds the contents of a memory-location pointed by R3 to register R1.

• Executing this instruction requires the following actions:

1) **Fetch the instruction.**
2) **Fetch the first operand.**
3) **Perform the addition**
4) **Load the result into R1.**

Fig:7.6 gives the sequence of control steps required to perform these operations for the single -bus architecture .



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Figure 7.6 Control sequence for execution of the instruction Add (R3),R1

- **Step1**--> The instruction-fetch operation is initiated by loading contents of PC into MAR & sending a Read request to memory. The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC"s content), and the result is stored in Z.
- **Step2**--> Updated value in Z is moved to PC. This completes the PC increment operation and PC will now point to next instruction.
- **Step3**--> Fetched instruction is moved into MDR and then to IR. The step 1 through 3 constitutes the **Fetch Phase**.
- At the beginning of step 4, the instruction decoder interprets the contents of the IR. This enables the control circuitry to activate the control-signals for steps 4 through 7.

  The step 4 through 7 constitutes the **Execution Phase**.
- **Step4**--> Contents of R3 are loaded into MAR & a memory read signal is issued.
- **Step5**--> Contents of R1 are transferred to Y to prepare for addition.
- **Step6**--> When Read operation is completed, memory-operand is available in MDR,
- **Step7**--> Sum is stored in Z, then transferred to R1.The End signal causes a new instruction fetch cycle to begin by returning to step1.

Pipelining:

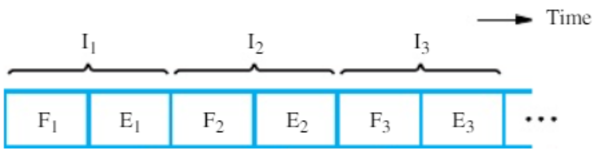
Basic Concepts:

The **speed of execution of programs is influenced by many factors**.

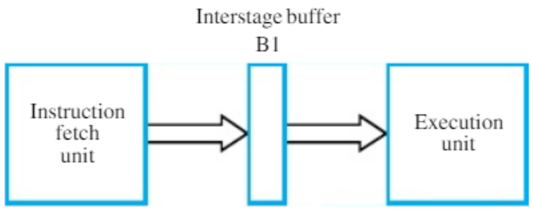
- **One way to improve** performance is to use **faster circuit technology to build the processor and the main memory.** Another possibility is to **arrange the hardware so that more than one operation can be performed at the same time.** In this way, **the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.**
- **Pipelining is a particularly effective way of organizing concurrent activity in a computer system.**
- The technique of **decomposing** a sequential process into sub-operations, with each sub-operation being executed in a dedicated segment .
- **pipelining** is commonly known as an **assembly-line** operation.

Consider **how the idea of pipelining can be used in a computer**. The **processor executes a program by fetching and executing instructions, one after the other**.

Let **Fi and Ei refer to the fetch and execute steps for instruction Ii** . Execution of a program consists of a sequence of fetch and execute steps, as shown in Figure a.



(a) Sequential execution



(b) Hardware organization

Now consider a computer that has **two separate hardware units**, **one** for **fetching instructions** and **another for executing** them, as shown in Figure b. The instruction fetched by the fetch unit is deposited in an **intermediate storage buffer, B1**. This **buffer is needed to enable the execution unit** to execute the instruction while the fetch unit is fetching the next instruction. The results of execution are deposited in the destination location specified by the instruction.

The computer is controlled by a clock.

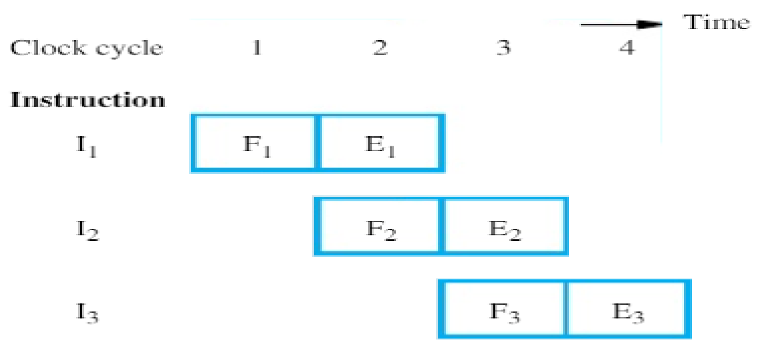
any instruction **fetch and execute** steps completed in **one clock cycle.**

Operation of the computer proceeds as in Figure 8.1c.

In the **first clock cycle**, the fetch unit fetches an instruction I1 (step F1) and stores it in buffer B1 at the end of the clock cycle.

In the **second clock cycle**, the instruction fetch unit proceeds with the fetch operation for instruction I2 (step F2). Meanwhile, the execution unit performs the operation specified by instruction I1, which is available to it in buffer B1 (step E1). By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available. Instruction I2 is stored in B1, replacing I1, which is no longer needed.

Step E2 is performed by the execution unit during the **third clock cycle,** while instruction I3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time. If the pattern in Figure 8.1c can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation depicted in Figure a.

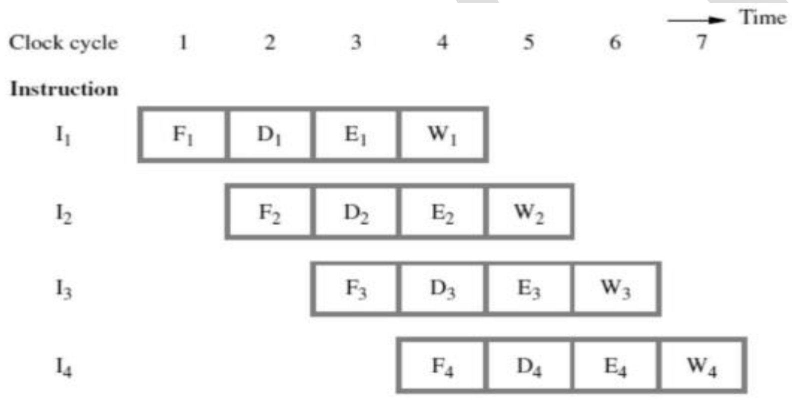


(c) Pipelined execution

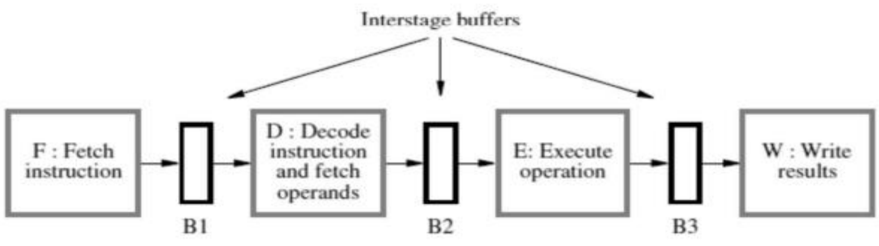
Figure 8.1 Basic idea of instruction pipelining.

Idea of Pipelining in a computer

a pipelined processor may process each instruction in four steps, as follows:
F (**Fetch**): **read the instruction from the memory.**
D (Decode): decode the instruction and fetch the source operand(s).
E (Execute): perform the operation specified by the instruction.
W (Write): store the result in the destination location.



(a) Instruction execution divided into four steps



(b) Hardware organization

The sequence of events for this case is shown in Figure a. Four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in Figure b. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer. As an instruction progresses through the pipeline, all the information needed by the stages downstream must be passed along. For example, during clock cycle 4, the information in the buffers is as follows:

- Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction I2 and the specification of the operation to be performed. This is the information produced by the decoding hardware in cycle 3. The buffer also holds the information needed for the write step of instruction I2 (stepW2). Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to enable that stage to perform the required Write operation.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I1.

Role of Cache Memory

Each stage in a pipeline is expected to complete its operation in **one clock cycle**. Hence, the **clock period should be sufficiently long to complete the task being performed in any stage.** If different units require different amounts of time, the clock period must allow the longest task to be completed. A unit that completes its task early is idle for the remainder of the clock period. Hence**, pipelining** is most effective in **improving performance** if the tasks being performed in different stages require about the same amount of time. This consideration is particularly important for the instruction fetch step, which is assigned one clock period in Figure a. The clock cycle has to be equal to or greater than the time needed to complete a fetch operation. However, 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, such as adding two numbers. Thus, **if each instruction fetch required access to the main memory, pipelining would be of little value.**

The **use of cache memories solves the memory access problem**. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor. This makes it possible to divide instruction fetching and processing into steps that are more or less equal in duration. Each of these steps is performed by a different pipeline stage, and the clock period is chosen to correspond to the longest one.

Pipeline Performance:

- The potential increase in performance resulting from pipelining is proportional to **the number of pipeline stages**.
- However, this increase would be achieved only if pipelined operation as depicted in Figure a could be sustained without interruption throughout program execution.
- Unfortunately, this is not the True.
- Floating point may involve many clock cycle.
- For a variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted. For example, stage E in the four stage pipeline of Figure b is responsible for arithmetic and logic operations, and one clock cycle is assigned for this task. Although this may be sufficient for most operations, some operations, such as divide, may require more time to complete. Figure shows an example in which the operation specified in instruction I2 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 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 B1 cannot be overwritten. Thus, steps D4 and F5 must be postponed as shown.

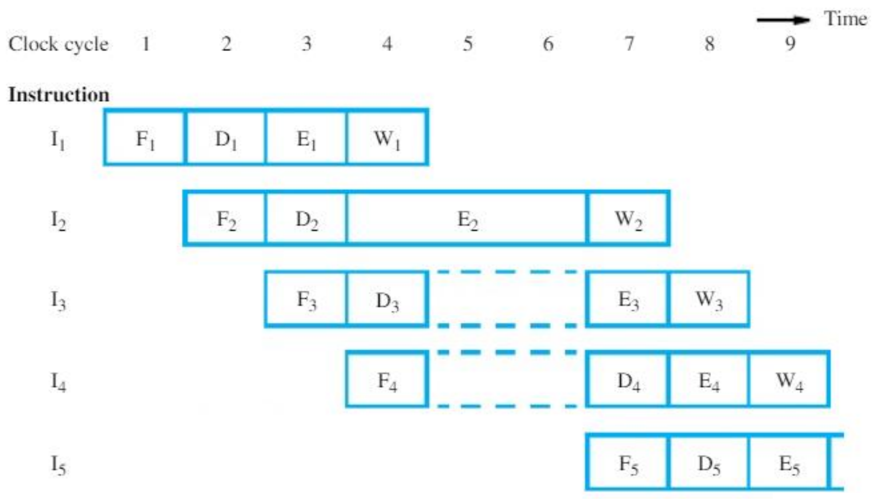


Figure 8.3 Effect of an execution operation taking more than one clock cycle. **Eg: for Data Hazard**

Pipelined operation in Figure 8.3 is said to have been **stalled for two clock cycles**. Normal pipelined operation resumes in cycle 7. Any condition that causes the pipeline to stall is called **a hazard**. We have just seen an example of a data hazard.

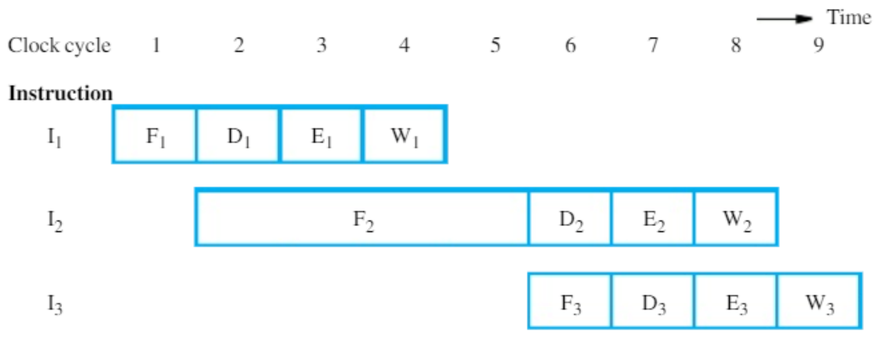
1) **A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed, and the pipeline stalls.**

2) **control hazards** or **instruction hazards**: The pipeline may also be stalled because of a **delay in the availability of an instruction**.

For example, this may be a **result of a miss in the cache** .

3) A **third type of hazard** known as a **structural hazard**: **This is the situation when two instructions require the use of a given hardware resource at the same time.**

The effect of a cache miss on pipelined operation is illustrated in Figure. Instruction I1 is fetched from the cache in cycle 1, and its execution proceeds normally. However, the fetch operation for instruction I2, which is started in cycle 2, results in a cache miss. The instruction fetch unit must now suspend any further fetch requests and wait for I2 to arrive. We assume that instruction I2 is received and loaded into buffer B1 at the end of cycle 5. The pipeline resumes its normal operation at that point.



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

(a) Instruction execution steps in successive clock cycles

Clock cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

(b) Function performed by each processor stage in successive clock cycles

Figure 8.4 Pipeline stall caused by a cache miss in F2. **Eg: for Instruction Hazard**

An alternative representation of the operation of a pipeline in the case of a cache miss is shown in Figure b. This figure gives the function performed by each pipeline stage in each clock cycle. Note that the **Decode unit is idle in cycles 3 through 5,** the Execute unit is idle in cycles 4 through 6, and the **Write unit is idle in cycles 5 through 7**. Such **idle periods are** called **stalls.** They are **also often referred** to as **bubbles in the pipeline**.

If instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed. Many processors use separate instruction and data caches to avoid this delay.

An **example of a structural hazard** is shown in Figure. This figure shows how the load instruction

Load X(R1),R2

- The memory address, X+[R1], is computed in stepE2 in cycle 4, then memory access takes place in cycle 5. The operand read from memory is written into register R2 in cycle 6. This means that the execution step of this instruction takes two clock cycles (cycles 4 and 5). It causes the pipeline to stall for one cycle, because both instructions I2 and I3 require access to the register file in cycle 6.
- Even though the instructions and their data are all available, the pipeline is stalled because **one hardware resource**, the register file, cannot handle two operations at once. If the register file had two input ports, that is, if it allowed two simultaneous write operations, the pipeline would not be stalled. In general, **structural hazards are avoided by providing sufficient hardware resources on the processor chip.**

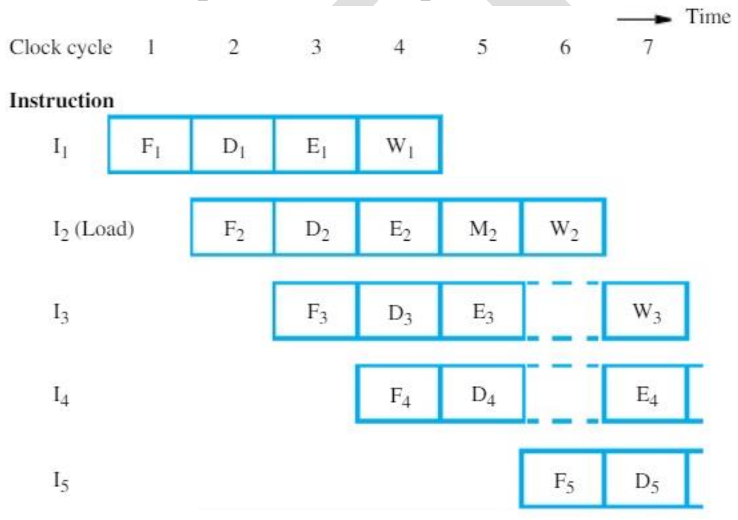


Figure 8.5 Effect of a Load instruction on pipeline timing.

It is important to understand that **pipelining does not result in individual instructions** being executed faster; rather, it is the throughput that increases, where **throughput is measured by the rate at which instruction execution is completed.**

The **pipeline stalls, causes degradation in pipeline performance.**

We need to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.