# MODULE 5: BASIC PROCESSING UNIT

## SOME FUNDAMENTAL CONCEPTS

• To execute an instruction, processor 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 can be written as

   IR ← [PC]

2) Increment PC by 4

   PC ← [PC] +4

3) Carry out the actions specified by instruction (in the IR).

• The first 2 steps are referred to as fetch phase;
Step 3 is referred to as execution phase.

## SINGLE BUS ORGANIZATION

• 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, and
  MAR's output is connected to external-bus.
• Instruction-decoder & control-unit is responsible for
  → issuing the signals that control the operation of all the units inside the processor (and for interacting with memory bus).
  → implementing the actions specified by the instruction (loaded in the IR)
• Registers R0 through R (n-1) are provided for general purpose use by programmer.
• Three registers Y, Z & TEMP are used by processor for temporary storage during execution of some instructions. These are transparent to the programmer i.e. programmer need not be concerned with them because they are never referenced explicitly by any instruction.
• MUX(Multiplexer) selects either
  → output of Y or
  → constant-value 4(is used to increment PC content). This is provided as input A of ALU.
• B input of ALU is obtained directly from processor-bus.
• As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform arithmetic or logic operation.
• An instruction can be executed by performing one or more of the following operations:
  1) Transfer a word of data from one processor-register to another or to the ALU.
  2) Perform arithmetic or a logic operation and store the result in a processor-register.
  3) Fetch the contents of a given memory-location and load them into a processor-register.
  4) Store a word of data from a processor-register into a given memory-location.

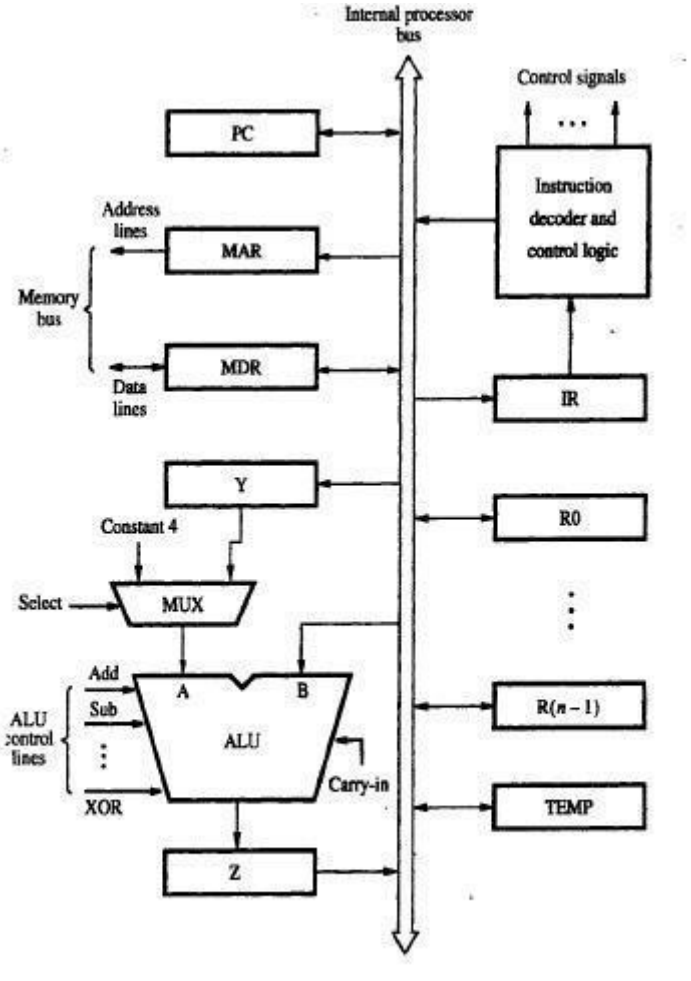



Figure 7.1 Single-bus organization of the datapath inside a processor.

## REGISTER TRANSFERS

• Instruction execution involves a sequence of steps in which data are transferred from one register to another.

• Input & output of register Ri is connected to bus via switches controlled by 2 control-signals: Riin & Riout. These are called *gating signals.*

• When Riin=1, data on bus is loaded into Ri.

• When Riout=0, bus can be used for transferring data from other registers.

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

• When edge-triggered flip-flops are not used, 2 or more clock-signals may be needed to guarantee proper transfer of data. This is known as *multiphase clocking*.

### Input & Output Gating for one Register Bit

• A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.

• When Riin=1, mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock.

• Q output of flip-flop is connected to bus via a tri-state gate.

When Riout=0, gate's output is in the high-impedance state. (This corresponds to the open-circuit state of a switch).

When Riout=1, the gate drives the bus to 0 or 1, depending on the value of Q.

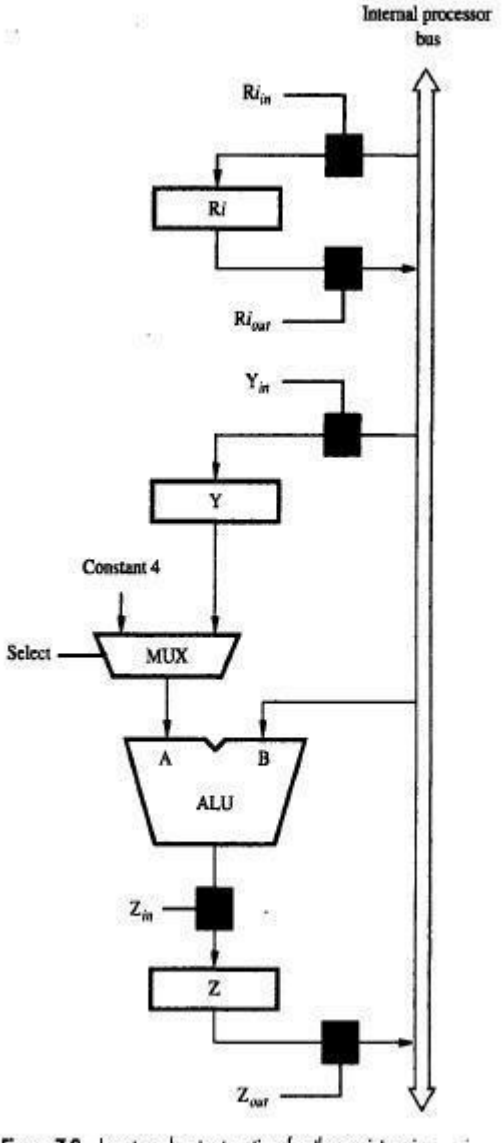



Figure 7.2 Input and output gating for the registers in Figure 7.1.

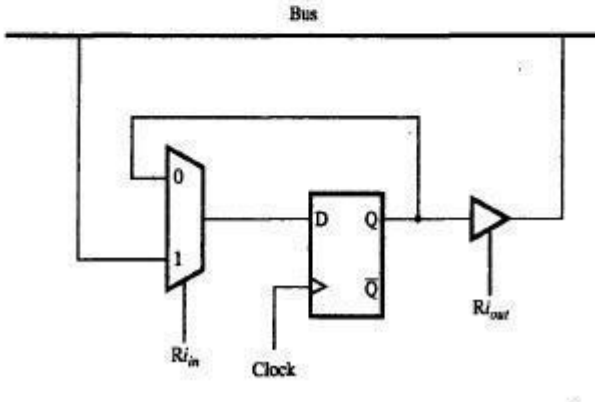



Figure 7.3 Input and output gating for one register bit.

**PERFORMING AN ARITHMETIC OR LOGIC OPERATION**

• The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs.

• One of the operands is output of MUX & the other operand is obtained directly from bus.

• The result (produced by the ALU) is stored temporarily in register Z.

• The sequence of operations for [R3]←[R1]+[R2] is as follows

1) R1out, Yin //transfer the contents of R1 to Y register

2) R2out, SelectY, Add, Zin //R2 contents are transferred directly to B input of ALU.
// The numbers are added. Sum stored in register Z

3) Zout, R3in //sum is transferred to register R3

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

***Write the complete control sequence for the instruction: Move (Rs), Rd***

• This instruction copies the contents of memory-location pointed to by Rs into Rd. This is a memory read operation. This requires the following actions

→ fetch the instruction

→ fetch the operand (i.e. the contents of the memory-location pointed by Rs).

→ transfer the data to Rd.

• The control-sequence is written as follows

1) PCout, MARin, Read, Select4, Add, Zin
2) Zout, PCin, Yin, WMFC
3) MDRout, IRin
4) Rs, MARin, Read
5) MDRinE, WMFC
6) MDRout, Rd, End

## FETCHING A WORD FROM MEMORY

• To fetch instruction/data from memory, processor transfers required address to MAR (whose output is connected to address-lines of memory-bus).

• 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

• MFC (Memory Function Completed): Addressed-device sets MFC to 1 to indicate that the contents of the specified location

→ have been read &

→ are available on data-lines of memory-bus

• Consider the instruction Move (R1), R2. The sequence of steps are:

1) R1out, MARin, Read ;desired address is loaded into MAR & Read command is issued

2) MDRinE, WMFC; load MDR from memory bus & Wait for MFC response from memory

3) MDRout, R2in ; load R2 from MDR

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

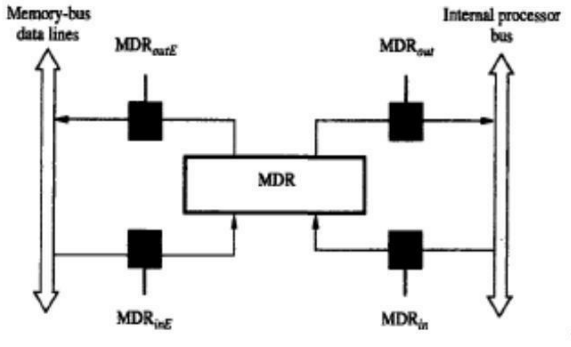



**Figure 7.4** Connection and control signals for register MDR.

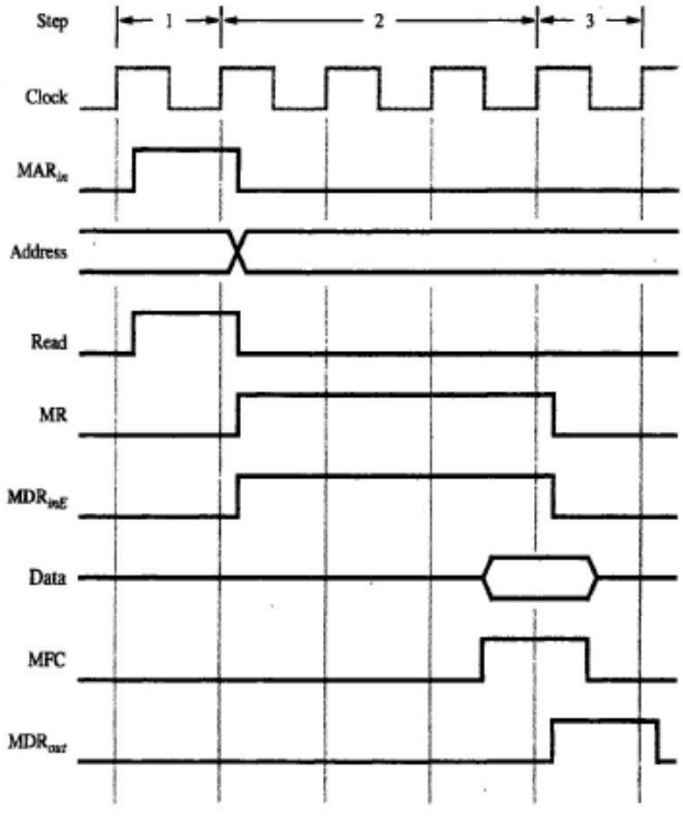



**Figure 7.5** Timing of a memory Read operation.

**Storing a Word in Memory**

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

1) R1out, MARin ;desired address is loaded into MAR
2) R2out, MDRin, Write ;data to be written are loaded into MDR & Write command is issued
3) MDRoutE, 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.

• Control sequence for execution of this instruction is as follows

1) PCout, MARin, Read, Select4, Add, Zin

2) $Z_{out}$, $PC_{in}$, $Y_{in}$, WMFC
3) MDRout, IRin

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

• Instruction execution proceeds as follows:

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.
Step3--> Fetched instruction is moved into MDR and then to IR.
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, and the addition is performed.

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.

## BRANCHING INSTRUCTIONS

• Control sequence for an unconditional branch instruction is as follows:

1) PCout, MARin, Read, Select4, Add, Zin

2) $Z_{out}$, $PC_{in}$, $Y_{in}$, WMFC
3) MDRout, IRin

4) Offset-field-of-$IR_{out}$, Add, $Z_{in}$
5) $Z_{out}$, $PC_{in}$, End

• The processing starts, as usual, the fetch phase ends in step3.

• In step 4, the offset-value is extracted from IR by instruction-decoding circuit.

• Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.
• In step 5, the result, which is the branch-address, is loaded into the PC.

• The offset X used in a branch instruction is usually the difference between the branch target-address and the address immediately following the branch instruction. (For example, if the branch instruction is at location 1000 and branch target-address is 1200, then the value of X must be 196, since the PC will be containing the address 1004 after fetching the instruction at location 1000).

• In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC.

e.g.: Offset-field-of-IRout, Add, Zin,
If N=0 then End
If N=0, processor returns to step 1 immediately after step 4.

If N=1, step 5 is performed to load a new value into PC.

(For reference only
1000 BI
1004 PC
/
/
/
1200 BTA

1200-1004=196)

# Pipelining

**Introduction**

The basic building blocks of a computer are introduced in preceding chapters. In this chapter, we discuss in detail the concept of pipelining, which is used in modern computers toachieve high performance. We begin by explaining the basics of pipelining and how it can lead to improved performance. Then we examine machine instruction features that facilitate pipelined execution, and we show that the choice of instructions and instruction sequencingcan have a significant effect on performance. Pipelined organization requires sophisticated compilation techniques, and optimizing compilers have been developed for this purpose.

Among other things, such compilers rearrange the sequence of operations to maximize the benefits of pipelined execution.

**BASIC CONCEPTS**

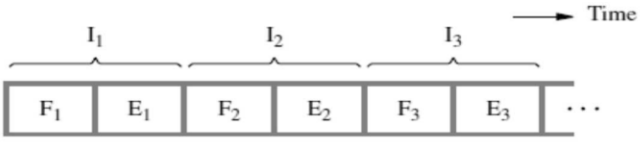
The speed of execution of programs is influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performedat the same time. In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed. We have encountered concurrent activities several times before. The concept of multiprogramming and explained how it is possible for I/O transfers and computational activities to proceed simultaneously. DMA devices make this possible because they can perform I/O transfers independently once these transfers are initiated by the processor.

Pipelining is a particularly effective way of organizing concurrent activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturingplants, where pipelining is commonly known as an assembly-line operation. Readers are undoubtedly familiar with the assembly line used in car manufacturing. The first station in anassembly line may prepare the chassis of a car, the next station adds the body, the next one installs the engine, and so on. While one group of workers is installing the engine on

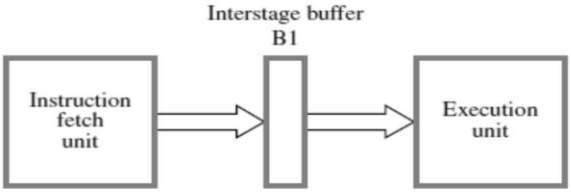
one car, another group is fitting a car body on the chassis of another car, and yet another group is preparing a new chassis for a third car. It may take days to complete work on a given car, butit is possible to have a new car rolling off the end of the assembly line every few minutes.

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.

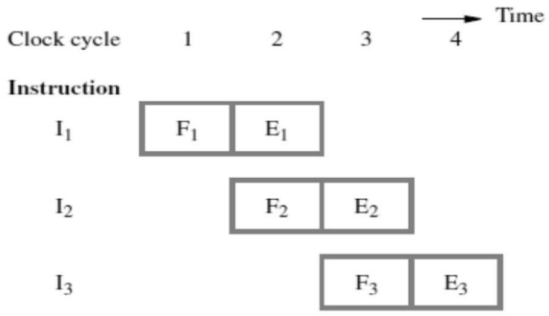
Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown in Figure b. The instruction fetched by the fetch unit is deposited in an intermediate storage buffer, B1. This buffer is needed to enable the execution unit to execute 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. For the purposes of this discussion, we assume that both the source and the destination of the data operated on by the instructions are inside the block labelled "Execution unit."




(a) Sequential execution




(b) Hardware organization




(c) Pipelined execution

The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be 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.

In summary, the fetch and execute units in Figure b constitute a two-stage pipeline in which each stage performs one step in processing an instruction. An inter-stage storage buffer,

B1, is needed to hold the information being passed from one stage to the next. New informationis loaded into this buffer at the end of each clock cycle.

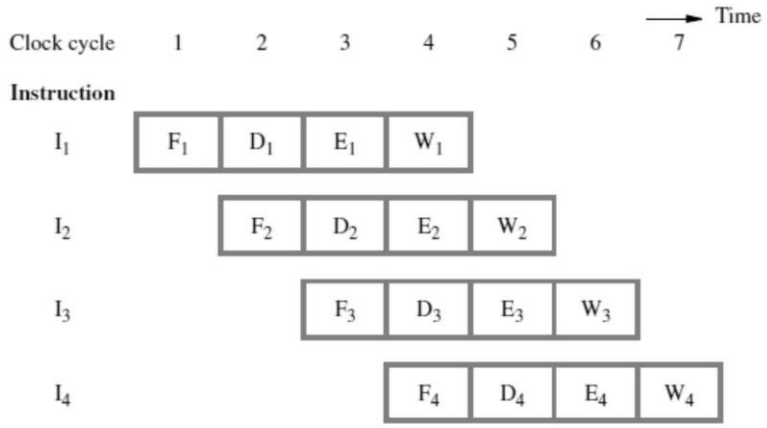
The processing of an instruction need not be divided into only two steps. For example,a pipelined processor may process each instruction in four steps, as follows:

F Fetch: read the instruction from the memory.

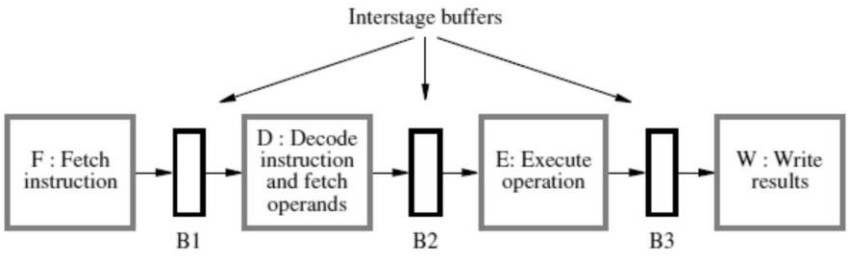
D Decode: decode the instruction and fetch the source operand(s).

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.

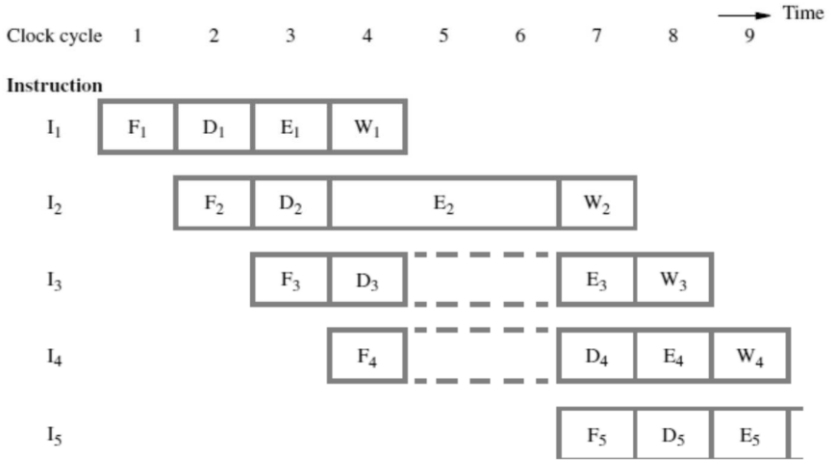
The use of cache memories solves the memory access problem. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor. This makes it possible to divide instruction fetching 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 pipelined processor in Figure completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation. 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

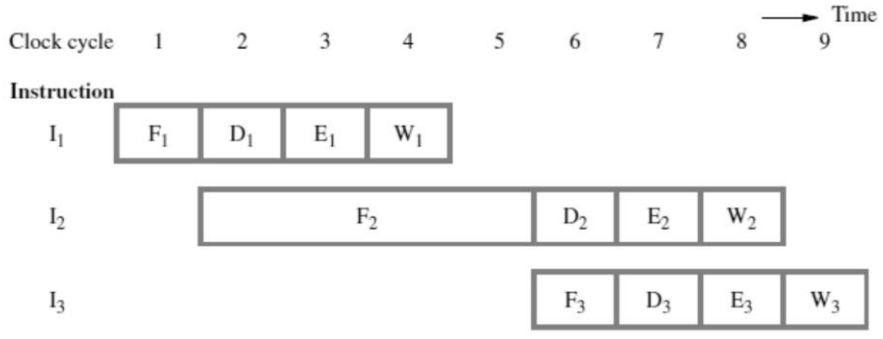
not the case.

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.




*Effect of an execution operation taking more than one clock cycle*

Pipelined operation in Figure 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. 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. 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, requiring the instruction to be fetched from the main memory. Such hazards are often called control hazards or instruction hazards. 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.




(a) Instruction execution steps in successive clock cycles


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

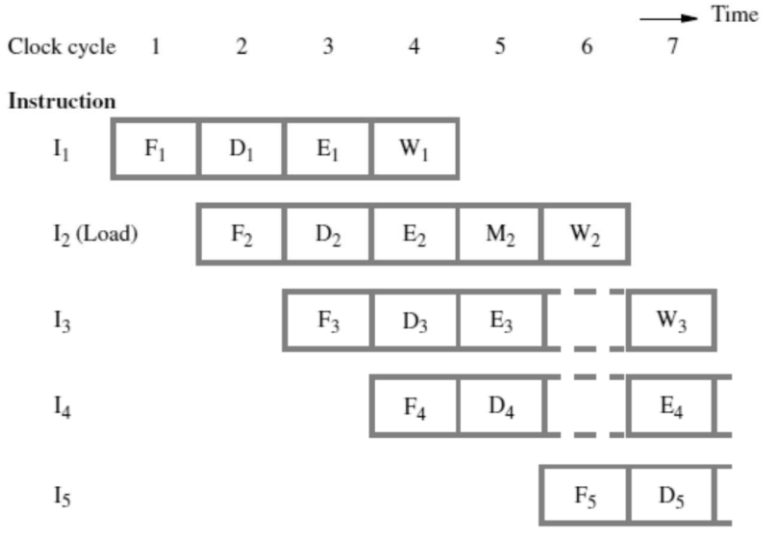
An alternative representation of the operation of a pipeline in the case of a cache miss is shown in Figure b. This figure gives the function performed by each pipeline stage in each clock cycle. Note that the Decode unit is idle in cycles 3 through 5, the Execute unit is idle in cycles 4 through 6, 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. Once created as a result of a delay in one of the pipeline stages, a bubble moves downstream until it reaches the last unit.

A third type of hazard that may be encountered in pipelined operation is 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 most common case in which this hazard may arise is in access to memory. One instruction may need to access memory as part of the Execute or Write stage while another instruction is being fetched. 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*

can be accommodated in our example 4-stage pipeline. 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 thoughthe 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.




*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. Any time one of the stages in thepipeline cannot complete its operation in one clock cycle, the pipeline stalls, and somedegradation in performance occurs. Thus, the performance level of one instruction completionin each clock cycle is actually the upper limit for the throughput achievable in a pipelinedprocessor organized as in Figure b. An important goal in designing processors is to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.