IAT 3 Answer Sheet

Q. 1 Demonstrate the DMA (4) and its implementation (2) and show how the data is transferred between memory (2) and I/O devices using the DMA controller (2).

Answer: The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA.

DMA controller→ is a control circuit that performs DMA transfers. → is a part of the I/O device interface. → performs the functions that would normally be carried out by processor.

While a DMA transfer is taking place, the processor can be used to execute another program.

DMA interface has three registers: 1) First register is used for storing starting-address. 2) Second register is used for storing word-count. 3) Third register contains status- & control-flags.

The R/W bit determines direction of transfer. If R/W=1, controller performs a read-operation (i.e. it transfers data from memory to I/O), Otherwise, controller performs a write-operation (i.e. it transfers data from I/O to memory).

If Done=1, the controller has completed transferring a block of data and is ready to receive another command. (IE - Interrupt Enable).

If IE=1, controller raises an interrupt after it has completed transferring a block of data.

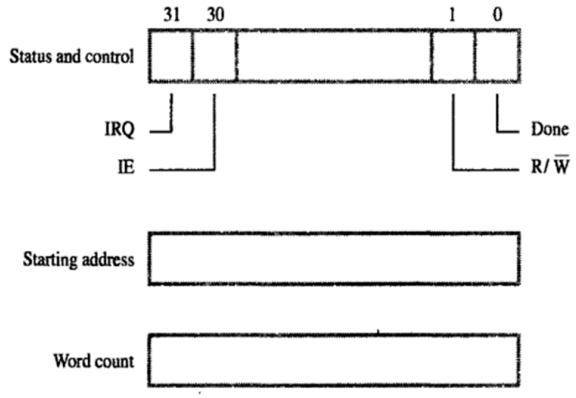
If IRQ=1, controller requests an interrupt.

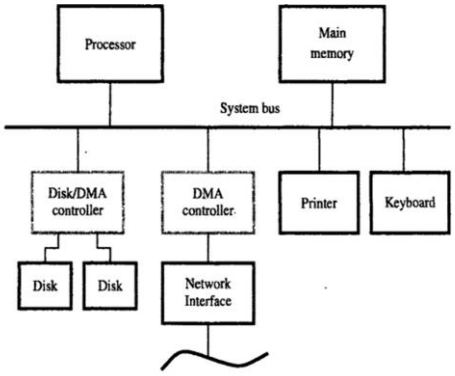
Requests by DMA devices for using the bus are always given higher priority than processor requests.

There are 2 ways in which the DMA operation can be carried out:

1)Processor originates most memory-access cycles. DMA controller is said to "steal" memory cycles from processor. Hence, this technique is usually called Cycle Stealing.

2)DMA controller is given exclusive access to main-memory to transfer a block of data without any interruption. This is known as Block Mode (or burst mode).

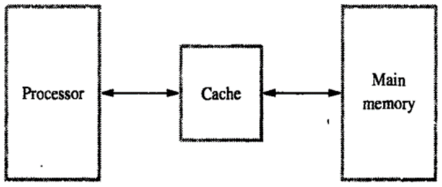




Use of DMA controllers in a computer system.

Q. 2 Discuss Cache memory (2) and Mapping functions (2). Explain the direct (2), associative (2) and set-associative mapping (2) with a labelled diagram.

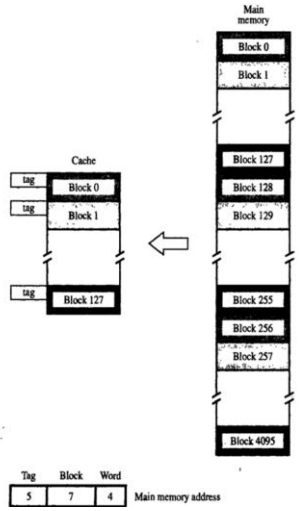
Answer: *Cache Memory-* The speed of the main memory is very low in comparison with the speed of modern processors. For good performance, the processor cannot spend much of its time waiting to access instructions and data in main memory. An efficient solution is to use a fast cache memory which makes the memory appear to the processor to be faster than it really is.



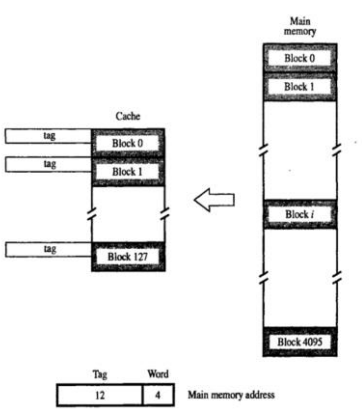
Use of a cache memory.

Mapping Functions:-

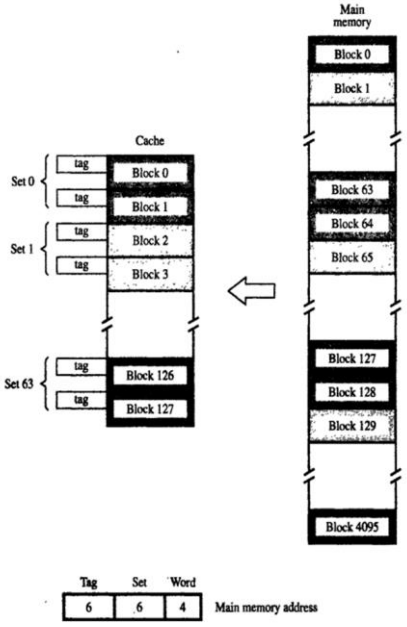
•Direct Mapping – The simplest way to determine cache locations in which to store memory blocks is the direct-mapping technique.



•Associative Mapping – It gives complete freedom in choosing the cache location in which to place the memory block.



•Set-Associative Mapping – It is the combination of direct and associative mapping techniques. Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set.



Q.3 (a) Give a detailed description of the execution of a complete instruction. (5)

Answer:

Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

$$IR \leftarrow [[PC]]$$

Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

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

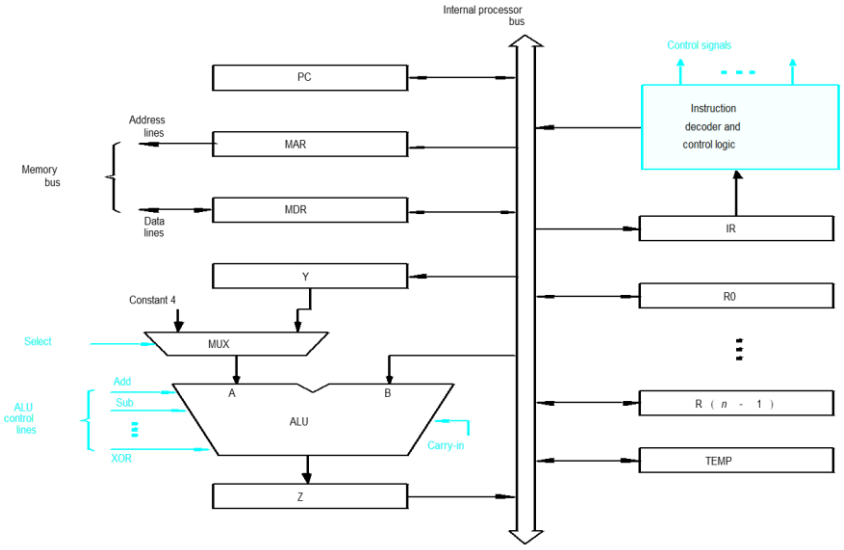


Figure 7.1. Single-bus organization of the datapath inside a processor.

For Example :

lAdd (R3), R1

lFetch the instruction

lFetch the first operand (the contents of the memory location pointed to by R3)

lPerform the addition

lLoad the result into R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
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.

(b) With a neat diagram, explain the centralized arbitration (2.5) and distributed bus arbitration scheme (2.5).

Answer: The device that is allowed to initiate data-transfers on bus at any given time is called bus master. There can be only one bus-master at any given time. Bus Arbitration is the process by which the next device to become the bus-master is selected & bus-mastership is transferred to that device.

•The two approaches are:

1) **Centralized Arbitration:** A single bus-arbiter performs the required arbitration.

•A single bus-arbiter performs the required arbitration.

•Normally, processor is the bus-master.

•Processor may grant bus-mastership to one of the DMA controllers.

•A DMA controller indicates that it needs to become bus-master by activating BR line.

•The signal on the BR line is the logical OR of bus-requests from all devices connected to it.

•Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free.

•BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.

•If DMA controller-1 is requesting the bus, Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2.

•Current bus-master indicates to all devices that it is using bus by activating BBSY line.

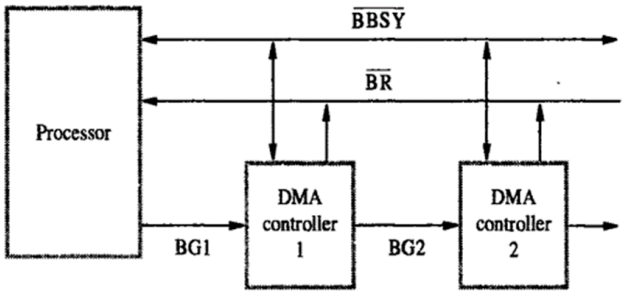
•The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers.

•Arbiter ensures that only 1 request is granted at any given time according to a priority scheme. (BR-Bus-Request, BG- Bus-Grant, BBSY-Bus Busy).

•The timing diagram shows the sequence of events for the devices connected to the processor.

•DMA controller-2 requests and acquires bus-mastership and later releases the bus.

After DMA controller-2 releases the bus, the processor resources bus-mastership.



Bus arbitration using a daisy chain

2) **Distributed Arbitration**: All devices participate in selection of next bus-master.

All device participate in the selection

of next bus-master

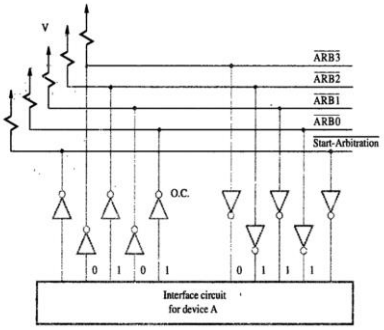
•Each device on bus is assigned a 4-bit identification number (ID).

•When 1 or more devices request bus, they → assert Start-Arbitration signal & place their 4-bit ID numbers on four open-collector lines 0 BRA through 3 BRA .

•A winner is selected as a result of interaction among signals transmitted over these lines.

•Net-outcome is that the code on 4 lines represents request that has the highest ID number.

•Advantage: This approach offers higher reliability since operation of bus is not dependent on any single device.

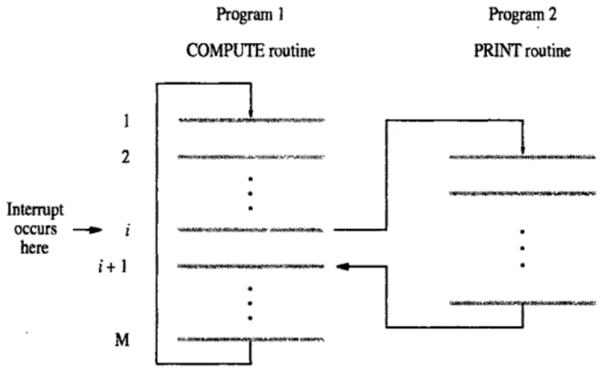


A distributed arbitration scheme

Q.4 (a) What is an interrupt (2)? What are interrupt service routines (1) and what are vectored interrupts (1)? Explain with an example (1)

Answer: I/O device initiates the action instead of the processor. This is done by sending a special hardware signal to the processor called as interrupt (INTR), on the interrupt-request line. The processor can be performing its own task without the need to continuously check the I/O device. When device gets ready, it will "alert" the processor by sending an interrupt-signal. The routine executed in response to an interrupt-request is called ISR (Interrupt Service Routine).

Once the interrupt-request signal comes from the device, the processor has to inform the device that its request has been recognized and will be serviced soon. This is indicated by a special control signal on the bus called interrupt-acknowledge (INTA).



VECTORED INTERRUPTS

•A device requesting an interrupt identifies itself by sending a special-code to processor over bus.

•Then, the processor starts executing the ISR.

•The special-code indicates starting-address of ISR.

•The special-code length ranges from 4 to 8 bits.

•The location pointed to by the interrupting-device is used to store the staring address to ISR.

•The staring address to ISR is called the interrupt vector.

•Processor loads interrupt-vector into PC & executes appropriate ISR.

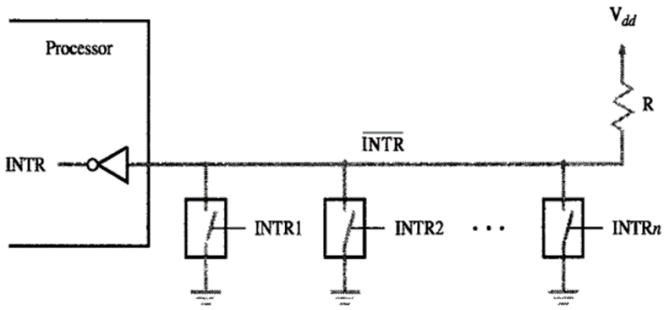
•When processor is ready to receive interrupt-vector code, it activates INTA line.

•Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal.

•The interrupt vector also includes a new value for the Processor Status Register.

(b) Draw an equivalent circuit of interrupt hardware and explain. (5)

Answer: An I/O device requests an interrupt by activating a bus-line called interrupt-request (IR). A single IR line can be used to serve n devices. All devices are connected to IR line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all IR signals are inactive (i.e. if all switches are open), the voltage on the IR line will be equal to Vdd. When a device requests an interrupt by closing its switch, the voltage on the line drops to 0, causing the INTR received by the processor to go to 1. The value of INTR is the logical OR of the requests from individual devices INTR=INTR1+ INTR2+ . . . . . +INTRn. A special gate known as open-collector or open-drain are used to drive the INTR line. Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.



An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

Q.5 (a) What is accessing I/O devices (2)? Draw and explain the I/O interface for an input device (3).

Answer: There are 2 ways to deal with I/O devices

1. **Memory mapped I/O Memory**

I/O devices share a common address space. Any data-transfer instruction (like Move, Load) can be used to exchange information. For example, Move DATAIN, R0; this instruction reads data from DATAIN (input-buffer associated with keyboard) and stores them into processor-register R0.

2. **I/O mapped I/O**

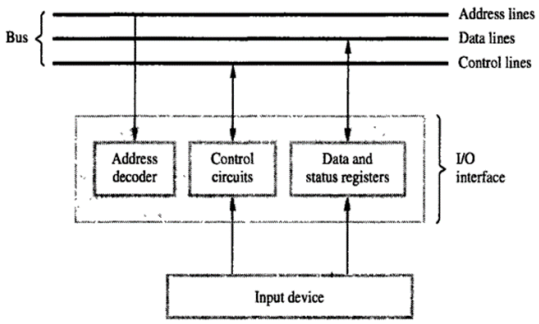
Memory and I/O address spaces are different. Special instructions named IN and OUT are used for data transfer. Advantage of separate I/O space: I/O devices deal with fewer address lines.

• Address decoder: enables the device to recognize its address when this address appears on the address lines.

• Data register: holds data being transferred to or from the processor.

• Status register: contains information relevant to the operation of the I/O device.

• Address decoder, data and status registers, and control circuits required to coordinate I/O transfers constitute the device's interface circuit.



(b) Discuss Register transfers (2), performing an Arithmetic or logic operation (2), Fetching a word from memory (2) and storing a word in memory (1)

Answer: Register Transfers: The input and output gates for register Ri are controlled by signals is Ri_{in} and Ri_{out} .

Ri_{in} Is set to 1 – data available on common bus are loaded into Ri.

Ri_{out} Is set to 1 – the contents of register are placed on the bus.

Ri_{out} Is set to 0 – the bus can be used for transferring data from other registers .

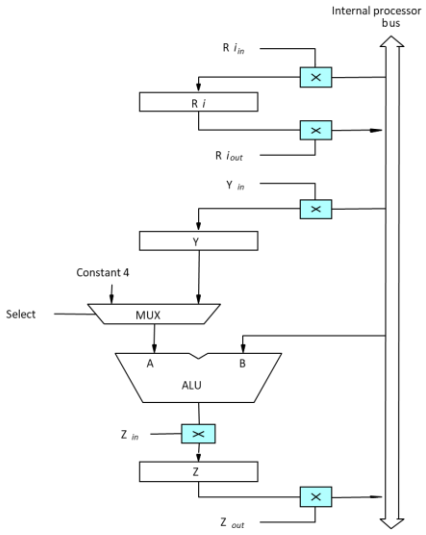


Figure 7.2. Input and output gating for the registers in Figure 7.1.

2.Performing an Arithmetic or Logic Operation

The ALU is a combinational circuit that has no internal storage. ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z. What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

1.R1out, Yin

2.R2out, SelectY, Add, Zin

3.Zout, R3in

Step 1: Output of the register R1 and input of the register Y are enabled, causing the contents of R1 to be transferred to Y.

Step 2: The multiplexer's select signal is set to select Y causing the multiplexer to gate the contents of register Y to input A of the ALU.
Step 3: The contents of Z are transferred to the destination register

3. Fetching a Word from Memory

Address into MAR; issue Read operation; data into MDR. The response time of each memory access varies (cache miss, memory-mapped I/O,…). To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).

Move (R1), R2
MAR ← [R1]

Start a Read operation on the memory bus

Wait for the MFC response from the memory

Load MDR from the memory bus

R2 ← [MDR]

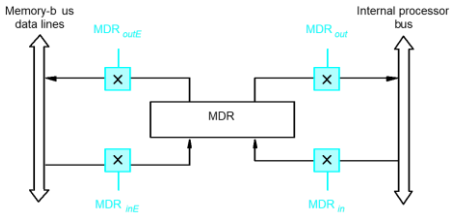


Figure 7.4. Connection and control signals for re gister MDR.

4. Storing a word in memory

Address is loaded into MAR

Data to be written loaded into MDR.

Write command is issued.

Example: Move R2,(R1)

$R1_{out}$,MAR_{in}

$R2_{out}$,MDR_{in},Write

MDR_{outE}, WMFC

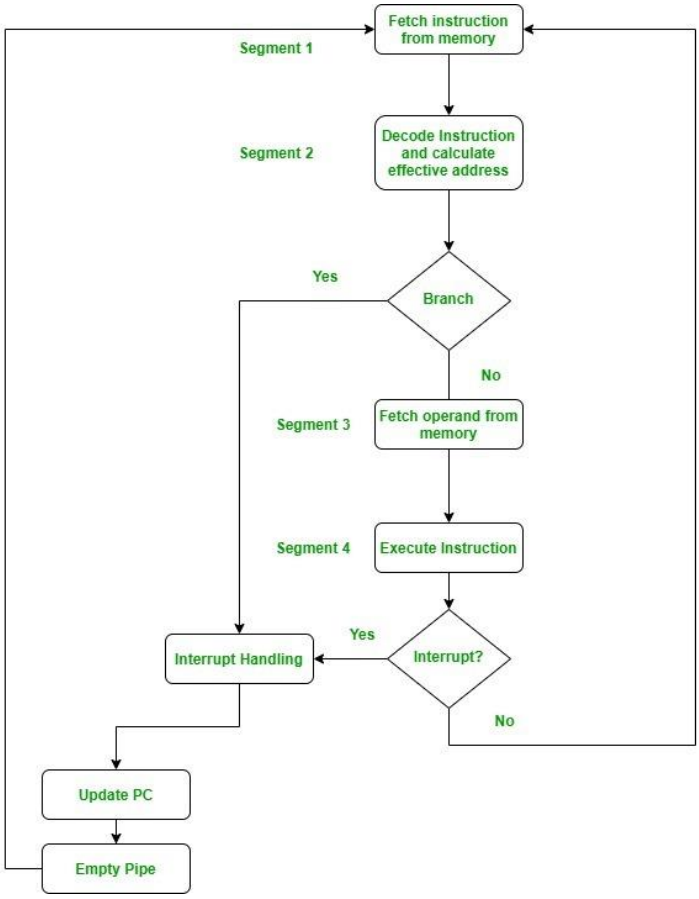
Q.6 (a) What is instruction pipelining? (4)

Answer: 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 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

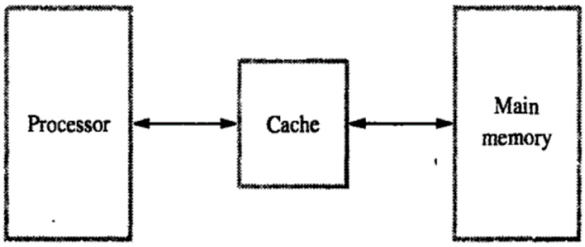
The flowchart for instruction pipeline is shown below.



(b) What is the role of cache memory (2)? Discuss pipeline performance (3)

Answer: Cache Memory: The speed of the main memory is very low in comparison with the speed of modern processors. For good performance, the processor cannot spend much of its time waiting to access instructions and data in main memory. An efficient solution is to use a

fast cache memory which makes the memory appear to the processor to be faster than it really is.



Use of a cache memory.

Pipeline performance-

The pipelined processor in Figure 8.2 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 8.2*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 8.2*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 8.3 shows an example in which the operation specified in instruction I_2 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 D_4 and F_5 must be postponed as shown.

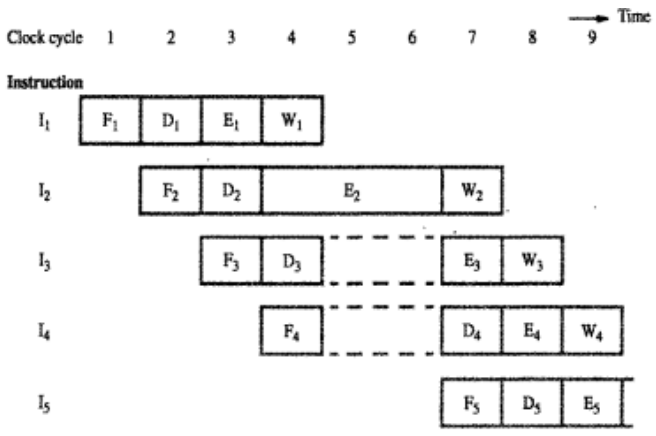
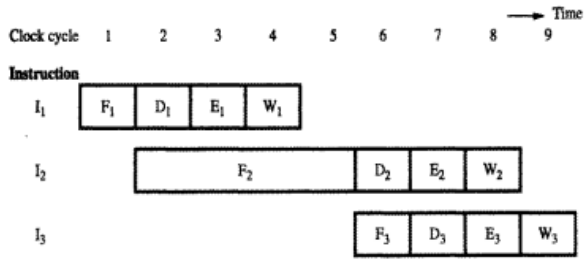


Figure 8.3 Effect of an execution operation taking more than one clock cycle.

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*. 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 8.4. Instruction I_1 is fetched from the cache in cycle 1, and its execution proceeds normally. However, the fetch operation for instruction I_2, 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 I_2 to arrive. We assume that instruction I_2 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

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.