Department of ISE
Internal Assessment Test 3 – March 2024
Evaluation scheme

Sub :	**Digital Design and Computer Organization**					Sub Code:	BCS302	Branch :	ISE	
Date:	04-03-2024	Duration :	90 min's	Max Marks:	50	Sem / Sec:	03/ A,B,C			OBE

Answer any FIVE FULL QUESTIONS		MARKS	CO	RBT
1(a)	Define hardware interrupt and software interrupt. Explain with suitable examples. 5 Differences-5 Marks	5	CO4	L2
1(b)	Explain the need of priority arbitration mechanism for interrupt handling. Explanation-3 Marks Diagram-2 Marks	5	CO4	L2
2	What is cache memory? Explain the types of cache mapping ? Cache memory- 4 Marks Three types of mapping- 6 Marks (each types carries 2 marks)	10	CO4	L2
3	Explain the working of DMA in detail with necessary diagram. DMA explanation- 6 Marks Diagrams-4 Marks	10	CO4	L2
4	Illustrate with a neat diagram the fundamentals of the basic processing unit Processing Unit explanation- 6 Marks Diagrams-4 Marks	10	CO5	L2
5	Analyze how the execution of complete instruction carries out. Execution explanation- 6 Marks Instructions-4 Marks	10	CO5	L2
6(a)	What is pipelining. Explain the effect of an execution operation in pipelining taking more than one clock cycle. Pipelining-2 Marks Effect of an execution operation-3 Marks	5	CO5	L2
6(b)	Explain the operations of 4 stage pipeline. Operation- 3 Marks Diagram-2 Marks	5	CO5	L2

CI CCI HOD

Department of ISE
Internal Assessment Test 3 – March 2024
Solution

1(a) Define hardware interrupt and software interrupt. Explain with suitable examples.

Sr. No.	Hardware Interrupt	Software Interrupt
1	Hardware interrupt is an interrupt generated from external device or hardware.	Software interrupt is the interrupt that is generated by any internal system of the computer.
2	It does not increment the program counter.	It increment the program counter.
3	Hardware interrupt can be invoked with some external device such as a request to start an I/O or an occurrence of a hardware failure.	Software interrupt can be invoked withthe help of INT instruction.
4	It has lowest priority than software interrupts	It has the highest priority among all interrupts.
5	A hardware interrupt is triggered by external hardware and is considered as one of the ways to communicate with the outsideperipherals, hardware.	A software interrupt is triggered by software and considered as one of the ways to communicate with kernel or to trigger system calls, especially during error or exception handling.
6	It is an asynchronous event.	It is a synchronous event.
7	Hardware interrupts can be classified into two types: 1. Maskable Interrupt. 2. Non Maskable Interrupt.	Software interrupts can be classified into two types they are: 1. Normal Interrupts. 2. Exception
8	Keystroke depressions and mouse movements are examples of hardware interrupt.	All system calls are examples of software interrupts
9	**Examples:** • When an I/O operation is completed, like reading some data into the computer from a tape drive. • An interrupt generated by a mouse when a button is clicked • An interrupt generated by a network card when data is received • An interrupt generated by a disk drive when a read or write operation is completed	**Examples:** • Output to the screen, execute file etc. • A system call to read or write data to a file. • A division by zero exception. • A page fault exception.

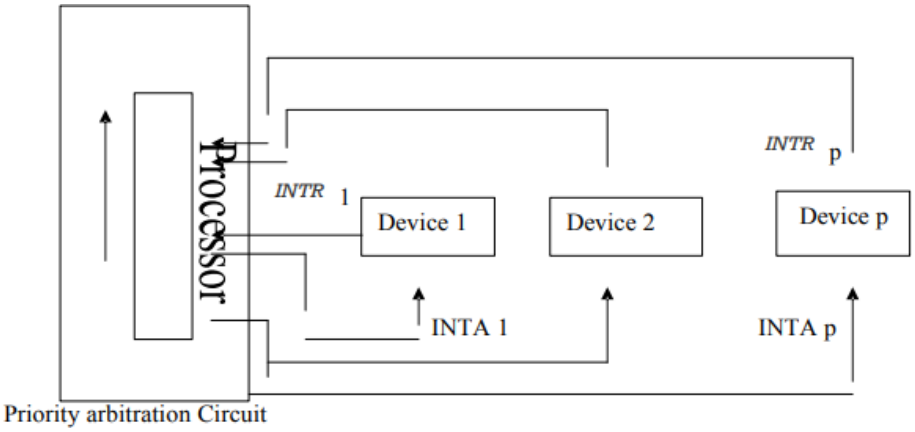
1(b) Explain the need of priority arbitration mechanism for interrupt handling.

For some devices, initiating an interrupt, a long delay in responding to an interrupt request may lead to erroneous operation. Consider, for example, a computer that keeps track of the time of day using a real-time clock. This is a device that sends interrupt requests to the processor at regular intervals. For each of these requests, the processor executes a short interrupt-service routine to increment a set of counters in the memory that keep track of time in seconds, minutes, and so on. Proper operation requires that the delay in responding to an interrupt request from the real-time clock be small in comparison with the interval between two successive requests. To ensure that this requirement is satisfied in the presence of other interrupting devices, it may be necessary to accept an interrupt request from the clock during the execution of an interrupt-service routine for another device. This example suggests that I/O devices should be organized in a priority structure. An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.

A multiple-level priority organization means that during execution of an interrupt-service routine, interrupt requests will be accepted from some devices but not from others, depending uponthe device's priority. To implement this scheme, we can assign a priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts only from devices that have priorities higher than its own.

The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the PS. These are privileged instructions, which can be executed only while the processor is running in the supervisor mode. The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application programs. Thus, a user program cannot accidentally, or intentionally, change the priority of the processor and disrupt the system's operation. An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privileged instruction. A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt- acknowledge lines for each device, as shown in Figure below.

Each of the interrupt-request lines is assigned a different priority level. Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor. A request is accepted only if it has a higher priority level than that currently assigned to the processor.

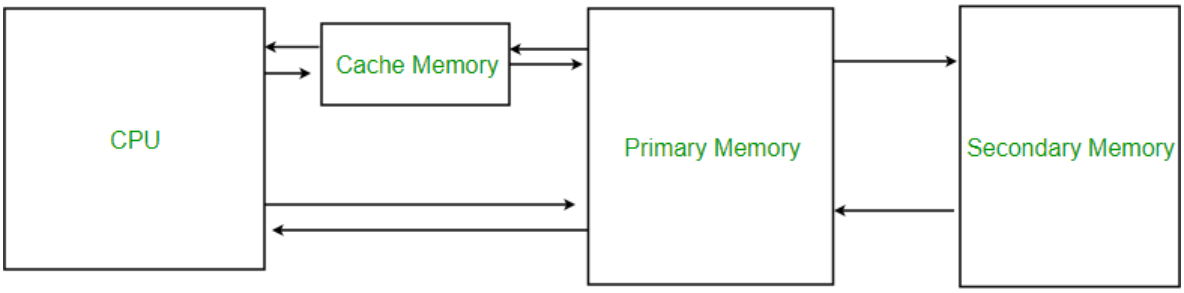


2 What is cache memory? Explain the types of cache mapping

Cache Memory is a special very high-speed memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data. The most important use of cache memory is that it is used to reduce the average time to access data from the main memory.

Characteristics of Cache Memory:

- Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
- Cache Memory holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is costlier than main memory or disk memory but more economical than CPU registers.
- Cache Memory is used to speed up and synchronize with a high-speed CPU.



Types of Cache Mapping:

1. **Direct mapping**

It is the Simplest mapping technique where each block of main memory maps to only one cache line i.e. if a block is in cache, it must be in one specific place. Main memory locations can only be copied into one location in the cache, accomplished by dividing main memory into blocks that correspond in size with the cache.

Formula to map a memory block to a cache line: i = j mod c

i=Cache Line Number (block address of cache)

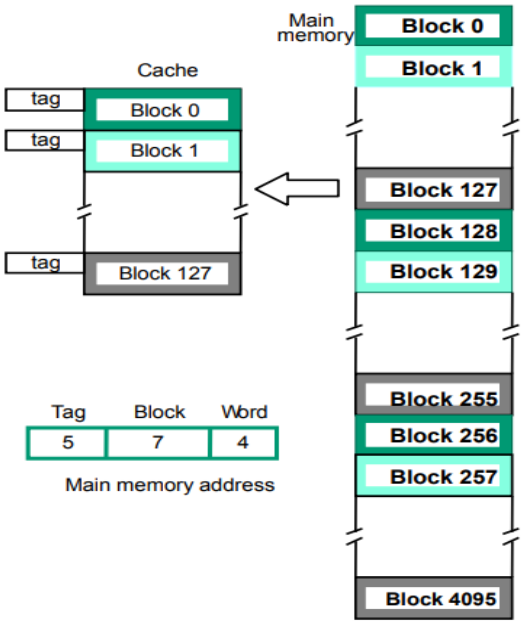
j=Main Memory Block Number(block address of main memory)

c=Number of Lines in Cache(no.of cache blocks) i.e. we divide the memory block by the number of cache lines and the remainder is the cache line address

In the below figure, Block j of the main memory maps to j modulo 128 of the cache. 0 maps to 0, 129 maps to 1. More than one memory block is mapped onto the same position in the cache. This may lead to contention for cache blocks even if the cache is not full.We can resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.

Memory address is divided into three fields: Low order 4 bits determine one of the 16 words in a block. - When a new block is brought into the cache, the the next 7 bits determine which cache block this new block is placed in. - High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.

This mapping is simple to implement but not very flexible.



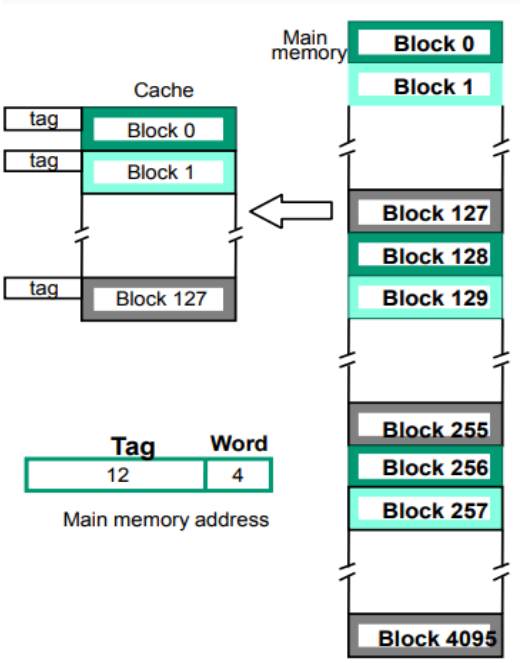
The direct mapping technique is simple and inexpensive to implement. When the CPU wants to access data from memory, it places a address. The index field of CPU address is used to access address. The tag field of CPU address is compared with the associated tag in the word read from the cache. If the tag-bits of CPU address is matched with the tag-bits of cache, then there is a hit and the required data word is read from cache. If there is no match, then there is a miss and the required data word is stored in main memory. It is then transferred from main memory to cache memory with the new tag.

2. Fully Associative Mapping:

An associative mapping uses an associative memory. This memory is being accessed using its contents. Each line of cache memory will accommodate the address (main memory) and the contents of that address from the main memory. That is why this memory is also called Content Addressable Memory (CAM). It allows each block of main memory to be stored in the cache.

A fully associative mapping scheme can overcome the problems of the direct mapping scheme. A main memory block can load into any line of cache. Memory address is interpreted as tag and word. Tag uniquely identifies block of memory. Every line's tag is examined for a match which also need a Dirty and Valid bit. But Cache searching gets expensive! Ideally need circuitry that can simultaneously examine all tags for a match. Hence lots of circuitry needed, high cost.

It is more Flexible, and uses cache space efficiently. Replacement algorithms can be used to replace an existing block in the cache when the cache is full. Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.



3.Set Associative:

It is a Compromise between **fully-associative and direct-mapped cache.**

Cache is divided into a number of sets. Each set contains a number of lines. A given block maps to any line in a specific set .Use direct-mapping to determine which set in the cache corresponds to a set in memory . Memory block could then be in any line of that set .

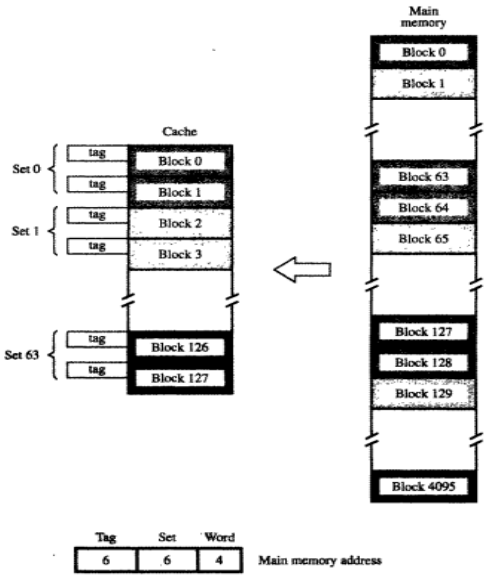
e.g. 2 lines per set - 2 way associative mapping

A given block can be in either of 2 lines in a specific set

e.g. K lines per set -K way associative mapping

A given block can be in one of K lines in a specific set

Much easier to simultaneously search one set than all lines



3 Explain the working of DMA in detail with necessary diagram.

Blocks of data are often transferred between the main memory and I/O devices such as disks. Direct Memory Access is a technique for controlling such transfers without frequent, program-controlled intervention by the processor.

Data are transferred by executing instruction such as,

Move DATAIN, R0

An instruction to transfer input or output data is executed only after the processor determines that the I/O device is ready, either by polling its status register or by waiting for an interrupt request. In either case, considerable overhead is incurred, because several program instructions must be executed involving many memory accesses for each data word transferred.

When transferring a block of data, instructions are needed to increment the memory address and keep track of the word count. The use of interrupts involves operating system routines which incur additional overhead to save and restore processor registers, the program counter, and other state information.

An alternative approach is used to transfer blocks of data directly between the main memory and I/O devices, such as disks. A special control unit is provided to manage the transfer, without continuous intervention by the processor. This approach is called direct memory access, or DMA.

DMA controller:

The unit that controls DMA transfers is referred to as a DMA controller. It may be part of the I/O device interface, or it may be a separate unit shared by a number of I/O devices. The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.

The Figure 4.10 below shows the block diagram of the DMA controller. The DMA controller have three registers as follows.

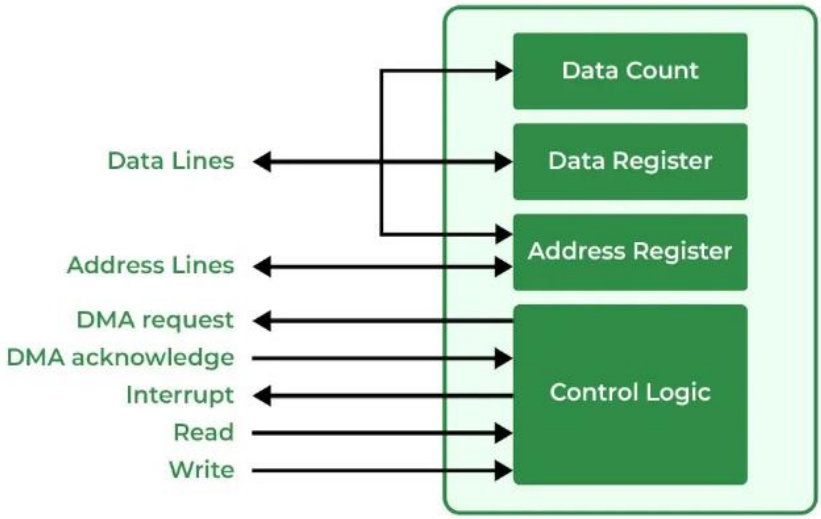
Address register – It contains the address to specify the desired location in memory.

Word count register – It contains the number of words to be transferred.

Control register – It specifies the transfer mode.

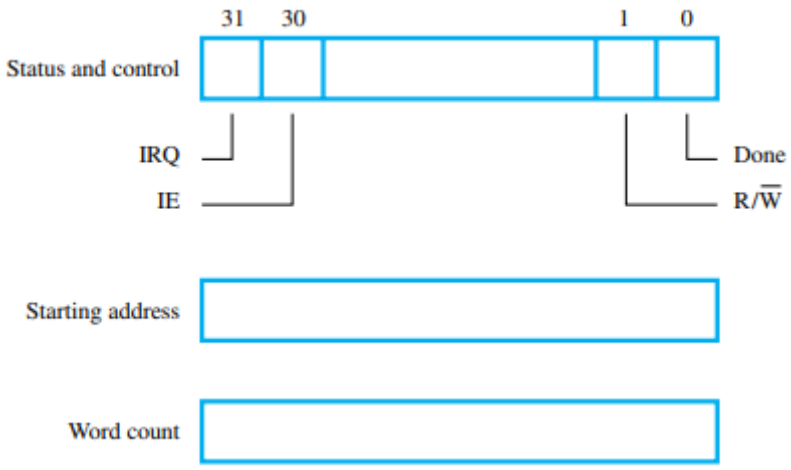
For each word transferred, it provides the memory address and generates all the control signals needed. It increments the memory address for successive words and keeps track of the number of transfers. Although a DMA controller transfers data without intervention by the processor, its operation must be under the control of a program executed by the processor, usually an operating system routine. To initiate the transfer of a block of words, the processor sends to the DMA controller the starting address, the number of words in the block, and the direction of the transfer. The DMA controller then proceeds to perform the requested operation. When the entire block has been transferred, it informs the processor by raising an interrupt.

Figure shows an example of the DMA controller registers that are accessed by the processor to initiate data transfer operations. Two registers are used for storing the starting address and the word count. The third register contains status and control flags. The R/W bit determines the direction of the transfer. When this bit is set to 1 by a program instruction, the controller performs a Read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a Write operation.



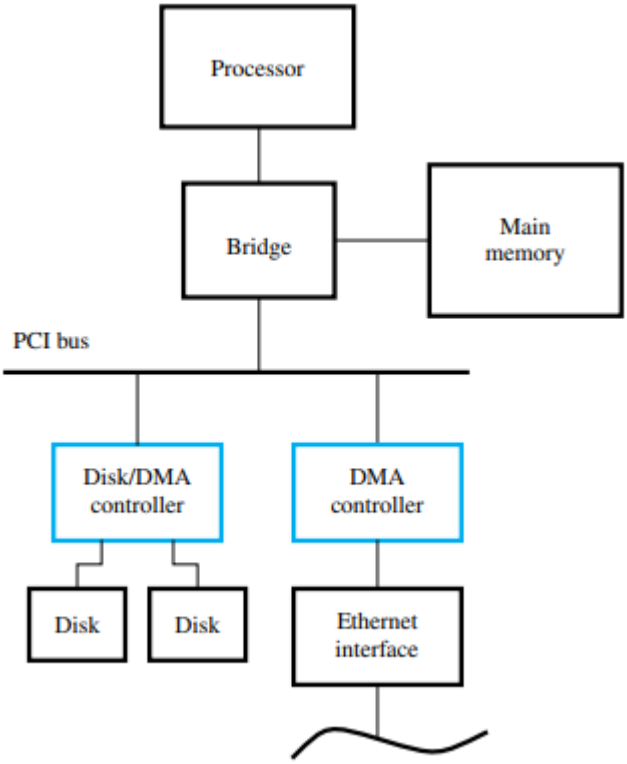
When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1. Bit 30 is the Interrupt-enable flag, IE. When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data. Finally, the controller sets the IRQ bit to 1 when it has requested an interrupt.

Figure shows how DMA controllers may be used in a computer system. A DMA controller connects a high-speed Ethernet to the computer's I/O bus.The disk controller, which controls two disks, also has DMA capability and provides two DMA channels. It can perform two independent DMA operations, as if each disk had its own DMA controller. The registers needed to store the memory address, the word count, and so on, are duplicated, so that one set can be used



with each disk.

To start a DMA transfer of a block of data from the main memory to one of the disks, an OS routine writes the address and word count information into the registers of the disk controller. The DMA controller proceeds independently to implement the specified operation. When the transfer is completed, this fact is recorded in the status and control register of the DMA channel by setting the Done bit. At the same time, if the IE bit is set, the controller sends an interrupt request to the processor and sets the IRQ bit. The status register may also be used to record other information, such as whether the transfer took place correctly or errors occurred.



Modes of Data Transfer in DMA

There are 3 modes of data transfer in DMA that are described below.

Burst OR Block Mode: In Burst Mode, buses are handed over to the CPU by the DMA if the whole data is completely transferred, not before that. (i.e the DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption)

Cycle Stealing Mode: In Cycle Stealing Mode, buses are handed over to the CPU by the DMA after the transfer of each byte. Continuous request for bus control is generated by this Data Transfer Mode. It works more easily for higher-priority tasks.

Transparent Mode: Transparent Mode in DMA does not require any bus in the transfer of the data as it works when the CPU is executing the transaction.

4 Illustrate with a neat diagram the fundamentals of the basic processing unit

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 P,← [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.**

The operations specified by an instruction can be carried out by performing one or more of the following actions:

1.Read the content of a given memory-location and load them into a register.

2.Read data from one or more register

3.Perform ALU operations and place the result into the register.

4.Store data from a register into a given memory location.

Below Figure shows the single bus organization. ALU and all the registers are interconnected via a single common bus. Data and address line of the external memory bus is connected to the internal processor bus via MDR and MAR respectively(MDR-Memory Data Register and 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, 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**.

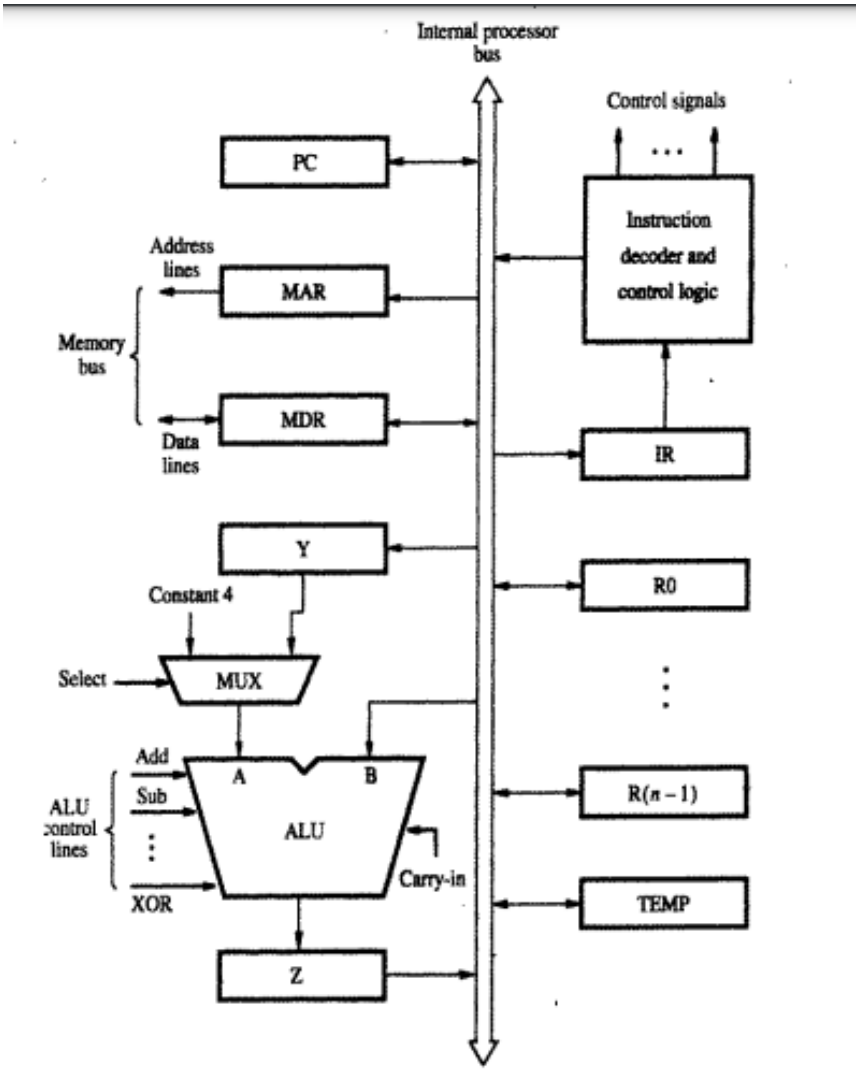


Figure:Single bus organization of the data path inside a processor

Disadvantage: Only one data word can be transferred over the bus in a clock cycle.

Solution: Providing multiple data-paths allows several data transfer to take place in parallel.

5 Analyze how the execution of complete instruction carries out.

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) 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

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) Zout, PCin, Yin, WMFC

3) MDRout, IRin

4) Offset-field-of-IRout, Add, Zin

5) Zout, PCin, 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

6(a) What is pipelining. Explain the effect of an execution operation in pipelining taking more than one clock cycle.

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

1. One way to improve performance is to use faster circuit technology to implement the processor and the main memory.
2. 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 time needed to perform any one operation is not changed.

Pipelining is a particularly effective way of organizing concurrent activity in a computer system. 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.**

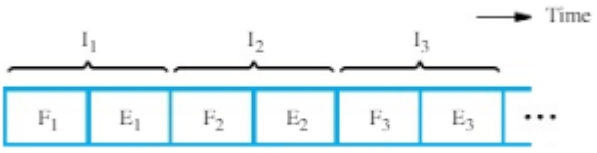


Figure Sequential execution

.Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown in **Figure**

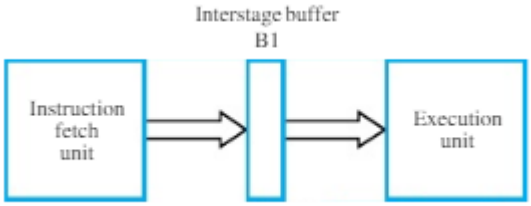


Figure Hardware organization

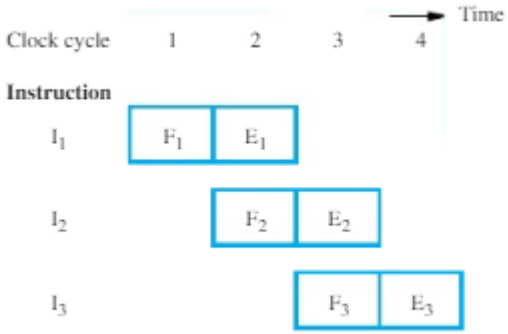


Figure Pipelined execution (2 stage)

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.

Operation of the computer proceeds as in Figure 5.9. 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 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. StepE2 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.

In summary, the fetch and execute units in Figure 5.3 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 information is loaded into this buffer at the end of each clock cycle.

6(b) Explain the operations of 4 stage pipeline.

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.

The sequence of events for this case is shown in the **Figure**

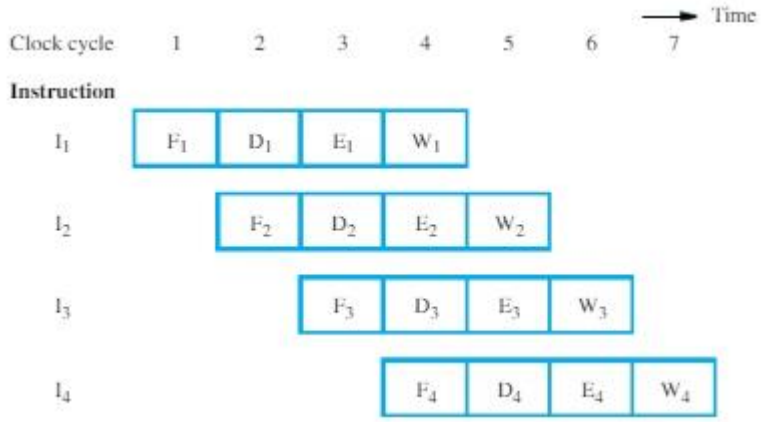


Figure Pipelined execution (4 stage)

Four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in the **Figure**

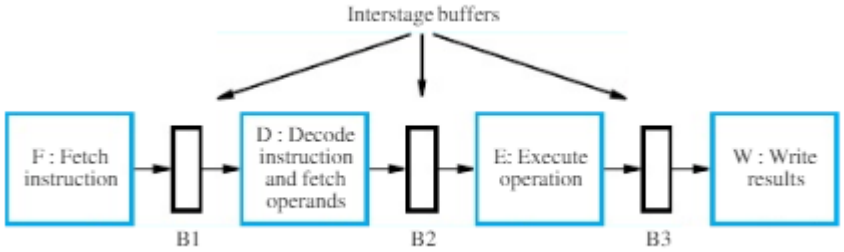


Figure Hardware organization

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 cycle3.Thebuffer 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.