Semester End Examination – July/August 2022
Scheme and Solution
Faculty-Divya Singh/Aparna N

Sub:	**Computer Organization & Architecture**					Sub Code:	21CS34	Branch:	ISE
Date:		Duration:	3hrs	Max Marks:	100	Sem/Sec:	III A, B & C		

		MARKS
1	**Operational Concepts of Computer System** Instruction consists of two parts 1) Operation code (Opcode) 2) Operands  • To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. • Individual instructions are brought from the memory into the processor, which executes the specified operations. • Data to be stored are also stored in the memory. Examples: - **Add LOCA, R0** • This instruction adds the operand at memory location LOCA, to operand in register R0 & places the sum into register. • This instruction requires the performance of several steps, 1. First the instruction is fetched from the memory into the processor. 2. The operand at LOCA is fetched and added to the contents of R0 3. Finally the resulting sum is stored in the register R0 The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons. **Load LOCA, R1** **Add R1, R0** The following steps are included to perform the above operation- 1. First the instruction is fetched from the memory into the processor. 2. The operand at LOCA is fetched from main memory into the register R1 3. Add the contents of the register R0 and R1 3. Finally the resulting sum is stored in the register R0 Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.	8(3 diagram)

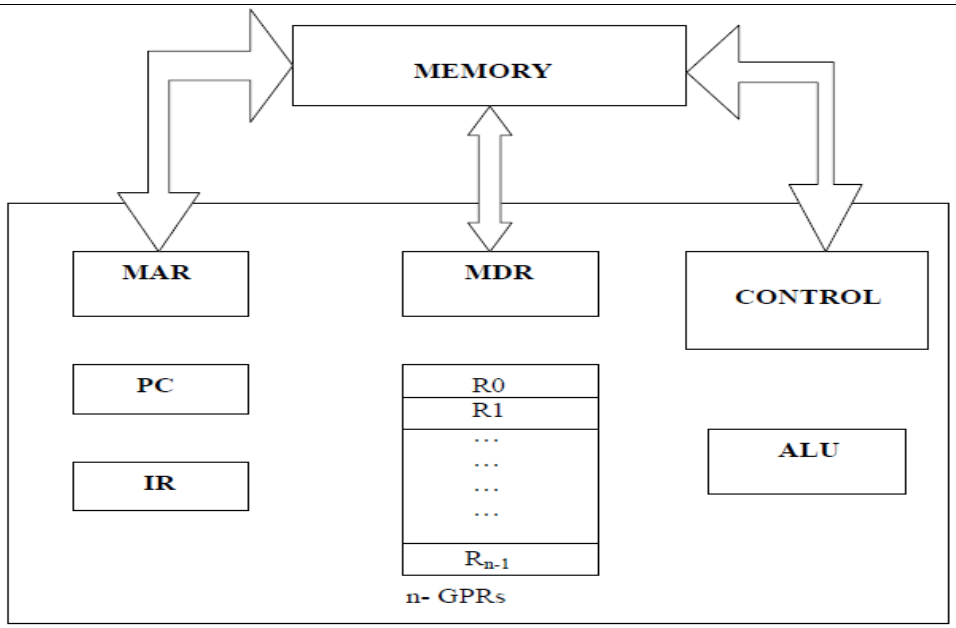


Fig b : Connections between the processor and the memory

The above fig. b shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains several registers used for several different purposes.

The instruction register (IR):-Holds the instructions that is currently being executed.Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.
The program counter PC:-This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.
Besides IR and PC, there are n-general purpose registers R0 through Rn-1. The other two registers which facilitate communication with memory are: -

1. MAR – (Memory Address Register):- It holds the address of the location to be accessed.
2. MDR – (Memory Data Register):- It contains the data to be written into or read out of the address location.

Operating steps are

1.Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiate a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.

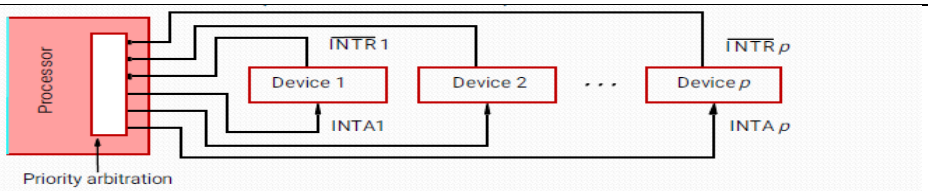
	9. After one or two such repeated cycles, the ALU can perform the desired operation. 10.If the result of this operation is to be stored in the memory, the result is sent to MDR. 11.Address of location where the result is stored is sent to MAR & a write cycle is initiated. 12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.	
1b	**Three address, two address, one address** S=A*B+C*D Load A MULTIPLY B Store X Load C MULTIPLY D Store Y Load X ADD Y Store S	8
	Basic performance equation • We now focus our attention on the processor time component of the total elapsed time. Let 'T' be the processor time required to execute a program that has been prepared in some high-level language. The compiler generates a machine language object program that corresponds to the source program. Assume that complete execution of the program requires the execution of N machine cycle language instructions. The number N is the actual number of instruction execution and is not necessarily equal to the number of machine cycle instructions in the object program. Some instruction may be executed more than once, which in the case for instructions inside a program loop others may not be executed all, depending on the input data used. Suppose that the average number of basic steps needed to execute one machine cycle instruction is S, where each basic step is completed in one clock cycle. If clock rate is 'R' cycles per second, the program execution time is given by $$T = \frac{N \times S}{R}$$ • this is often referred to as the basic performance equation. We must emphasize that N, S & R are not independent parameters changing one may affect another. Introducing a new feature in the design of a processor will lead to improved performance only if the overall result is to reduce the value of T.	4
2a	Addresssing modes.	10

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri) (LOC)	EA = [Ri] EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ri] + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	−(Ri)	Decrement Ri; EA = [Ri]

EA = effective address
Value = a signed number

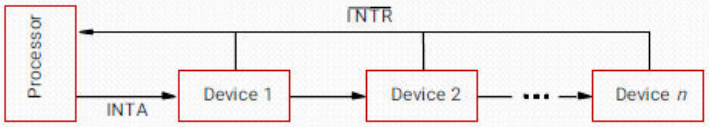
2b	**Big endian, little endian,** There are two ways in which byte-addresses are arranged (Figure2.3). **1) Big- Endian:** Lower byte-addresses are used for the more significant bytes of the word. **2) Little- Endian:** Lower byte-addresses are used for the less significant bytes of the word In both cases, byte-addresses 0,4,8.....are taken as the addresses of successive words in the memory.	6(3 M each)
2c	**Overall Spec Rating** Performance measurement is the measure of how well a processor operates for a given benchmark. 2 SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC-System Performance Evaluation Corporation). 3 SPEC Rating is given by SPEC Rating = (Running time of the reference Computer) / (Running time of the Computer Under test) $\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{\frac{1}{n}}$	4
3a	Explain the following methods of handling interrupt from multiple devices i) Interrupt nesting. ii) Daisy chain method Interrupt Nesting Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device. In general, same arrangement is used when multiple devices can send interrupt requests to the processor. During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device. Since the interrupt service routines are usually short, the delay that this causes is generally acceptable. However, for certain devices this delay may not be acceptable. I/O devices are organized in a priority structure: An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device. A priority level is assigned to a processor that can be changed under program control. Priority level of a processor is the priority of the program that is currently being executed. When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device. If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.	10(5 M each)



- *Each device has a separate interrupt-request and interrupt-acknowledge line.*
- *Each interrupt-request line is assigned a different priority level.*
- *Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.*
- *If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.*

ii)

Daisy chain scheme:



- *Devices are connected to form a daisy chain.*
- *Devices share the interrupt-request line, and interrupt-acknowledge line is connecte to form a daisy chain.*
- *When devices raise an interrupt request, the interrupt-request line is activated.*
- *The processor in response activates interrupt-acknowledge.*
- *Received by device 1, if device 1 does not need service, it passes the signal to devic*
- *Device that is electrically closest to the processor has the highest priority.*

3b

What is Bus arbitration? Explain centralized & distributed bus arbitration with neat diagram.

10(5 m each)

Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.

The device that is allowed to initiate transfers on the bus at any given time is called the bus master.

When the current bus master relinquishes its status as the bus master, another device can acquire this status.

The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called bus arbitration.

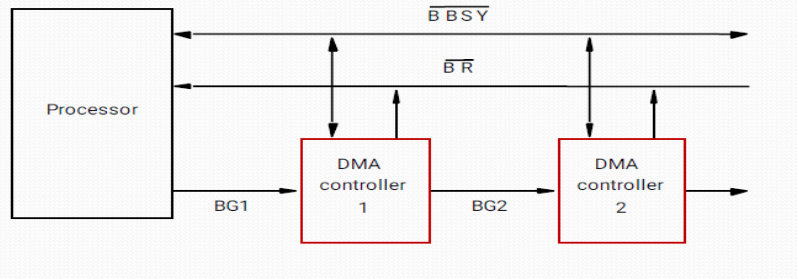
2 Approaches of Bus arbitration

Centralized arbitration:

A single bus arbiter performs the arbitration.

Distributed arbitration:

All devices participate in the selection of the next bus master.

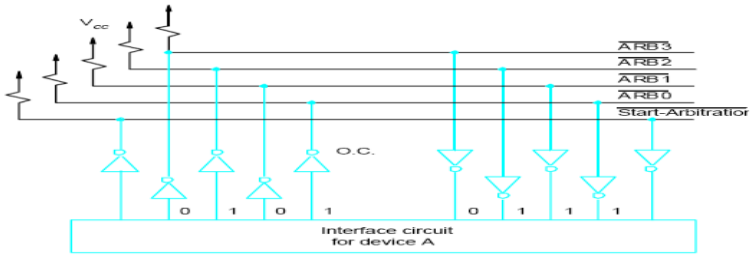


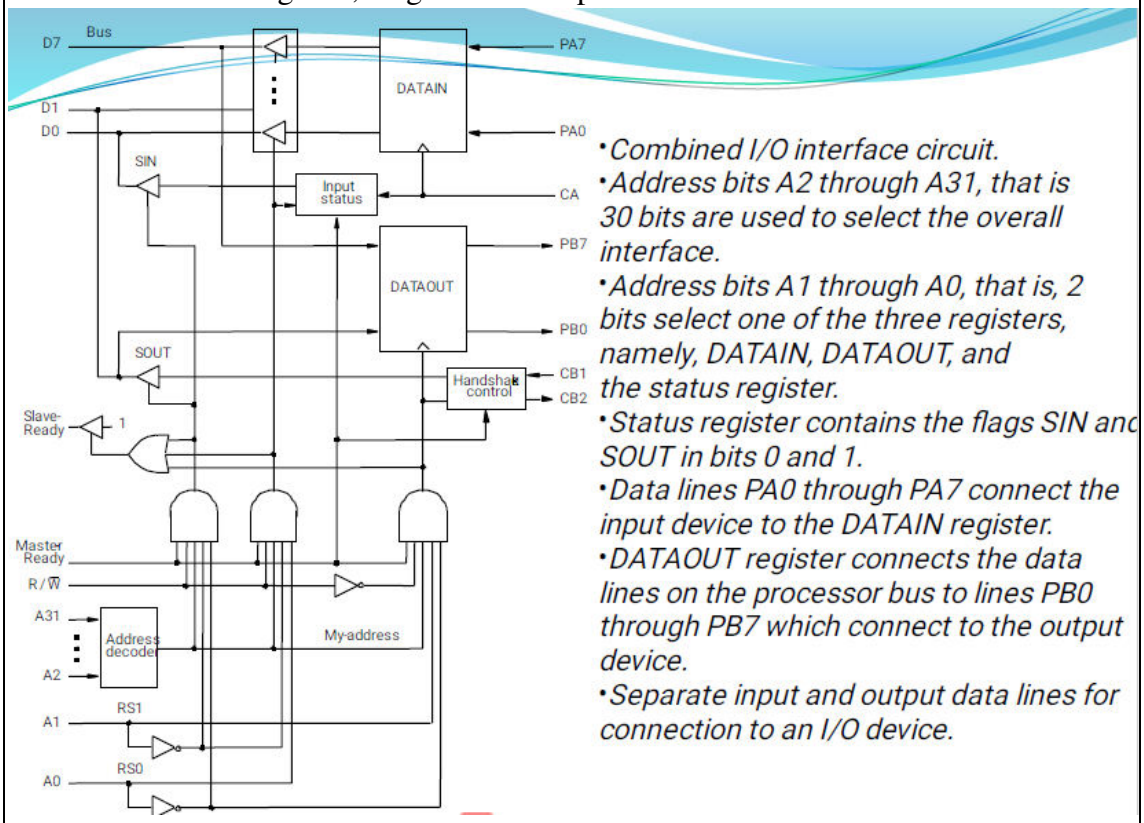
Bus arbiter may be the processor or a separate unit connected to the bus.

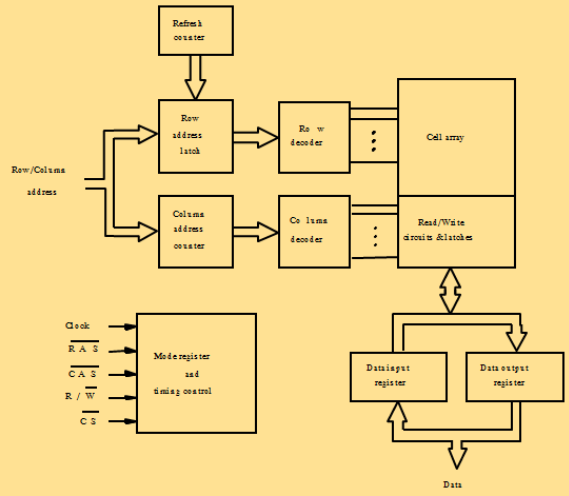
Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.

DMA controller requests the control of the bus by asserting the Bus Request (BR) line.

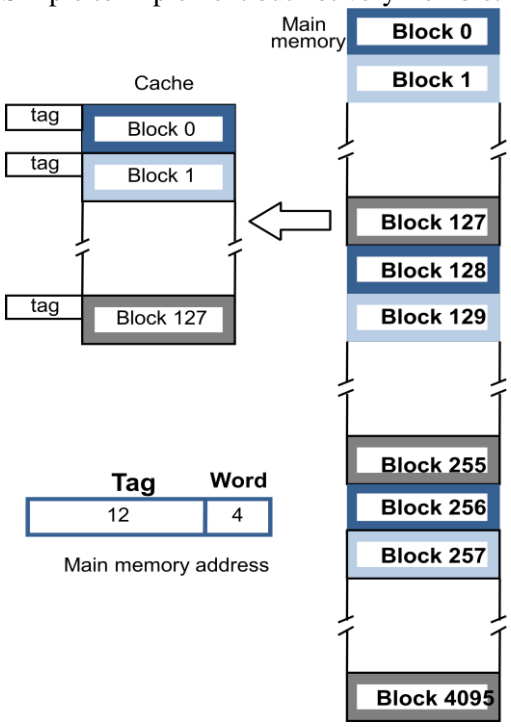
In response, the processor activates the Bus-Grant1 (BG1) line, indicating that the controller may use the

	bus when it is free. BG1 signal is connected to all DMA controllers in a daisy chain fashion. BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted BR can acquire control of the bus. All devices waiting to use the bus share the responsibility of carrying out the arbitration process. Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability. Each device is assigned a 4-bit ID number. All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal. To request the bus a device: Asserts the Start-Arbitration signal. Places its 4-bit ID number on the arbitration lines. The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.  Arbitration process: Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB. If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions. The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines. Device A has the ID 5 and wants to request the bus: - Transmits the pattern 0101 on the arbitration lines. Device B has the ID 6 and wants to request the bus: - Transmits the pattern 0110 on the arbitration lines. Pattern that appears on the arbitration lines is the logical OR of the patterns: - Pattern 0111 appears on the arbitration lines. Arbitration process: Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB. If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions. Device A compares its ID 5 with a pattern 0101 to pattern 0111. It detects a difference at bit position 0, as a result, it transmits a pattern 0100 on the arbitration lines. The pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110. This pattern is the same as the device ID of B, and hence B has won the arbitration.	
4a	Illustrate a program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display	10

	`` Move #LINE, R0 Initialize memory pointer WAITK TestBit #0,STATUS Test SIN Branch=0 WAITK Wait for character to be entered Move DATAIN,R1 Read character WAITD TestBit #1,STATUS Test SOUT Branch=0 WAITD Wait for display to become ready Move R1,DATAOUT Send character to display Move R1,(R0)+ Store character and advance pointer Compare #$0D,R1 Check if Carriage Return Branch≠0 WAITK If not, get another character Move #$0A,DATAOUT Otherwise, send Line Feed Call PROCESS Call a subroutine to process the input line ``` The example described above illustrates program controlled I / O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor polls the devices	
4b	Discuss with neat diagram ,the general 8-bit parallel interface circuit.  •Combined I/O interface circuit. •Address bits A2 through A31, that is 30 bits are used to select the overall interface. •Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register. •Status register contains the flags SIN and SOUT in bits 0 and 1. •Data lines PA0 through PA7 connect the input device to the DATAIN register. •DATAOUT register connects the data lines on the processor bus to lines PB0 through PB7 which connect to the output device. •Separate input and output data lines for connection to an I/O device.	10(Diagram-4)
5a	Explain the internal organization of a 16 Megabits DRAM chip configured as 2MX8 Cells	8(4-Diagram)
5b	Explain synchronous DRAM with neat diagram.	8(3-Diagram)

	 •*Operation is directly synchronized with processor clock signal.* •*The outputs of the sense circuits are connected to a latch.* •*During a Read operation, the contents of the cells in a row are loaded onto the latches.* •*During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches.* •*Data held in the latches correspond to the selected columns are transferred to the output.* •*For a burst mode of operation, successive columns are selected using column address counter and clock. CAS signal need not be generated externally. A new data is placed during raising edge of the clock*	
5c	Explain any 2 types of Read only memory (ROM). ■ Read-Only Memory: ▪ Data are written into a ROM when it is manufactured. ■ Programmable Read-Only Memory (PROM): ▪ Allow the data to be loaded by a user. ▪ Process of inserting the data is irreversible. ▪ Storing information specific to a user in a ROM is expensive. ▪ Providing programming capability to a user may be better. ■ Erasable Programmable Read-Only Memory (EPROM): ▪ Stored data to be erased and new data to be loaded. ▪ Flexibility, useful during the development phase of digital systems. ▪ Erasable, reprogrammable ROM. ▪ Erasure requires exposing the ROM to UV light.	4(2 M each)
6a	Describe the different mapping functions in cache. ■ Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is. ■ Cache memory is based on the property of computer programs known as "locality of reference". ■ Three mapping functions: ▪ Direct mapping ▪ Associative mapping ▪ Set-associative mapping. • 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. • May lead to contention for cache blocks even if the cache is not full. • 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.	12(4 M each)

- Simple to implement but not very flexible.



•Main memory block can be placed into any cache position.

•Memory address is divided into two fields:

- Low order 4 bits identify the word within a block.

- High order 12 bits or tag bits identify a memory block when it is resident in the cache.

•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.

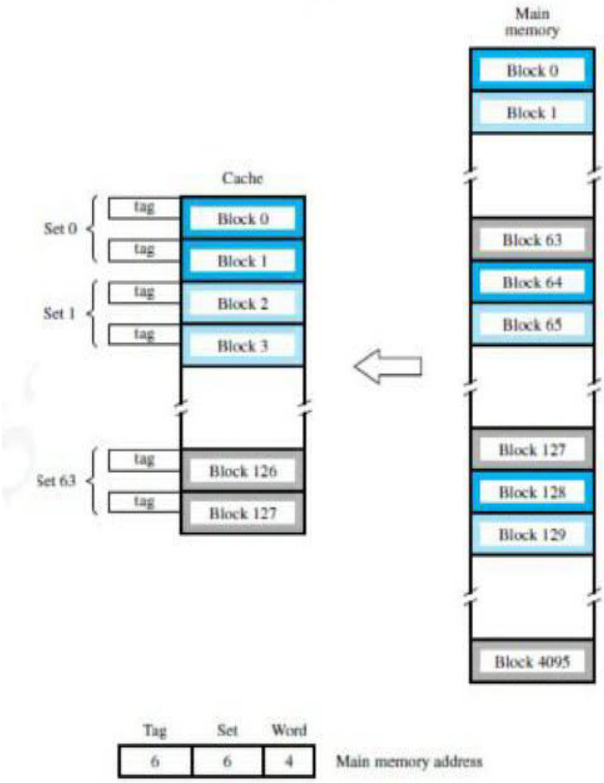


Figure 8.18 Set-associative-mapped cache with two blocks per set.

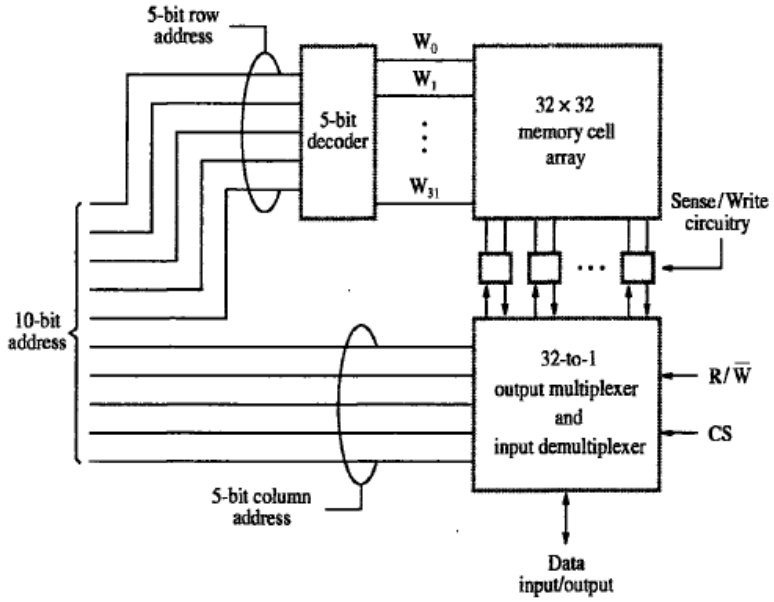
Blocks of cache are grouped into sets.
Mapping function allows a block of the main memory to reside in any block of a specific set.
Divide the cache into 64 sets, with two blocks per set.
Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.
Memory address is divided into three fields:

- *6 bit field determines the set number.*
- *High order 6 bit fields are compared to the tag fields of the two blocks in a set.*

Set-associative mapping combination of direct and associative mapping.
Number of blocks per set is a design parameter.

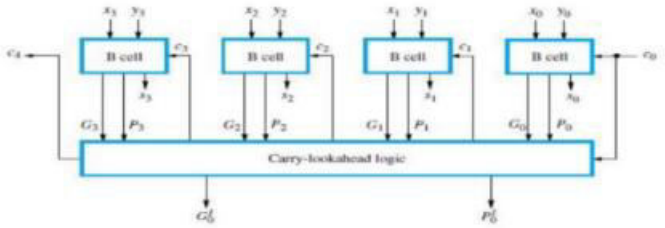
- *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
- *Other extreme is to have one block per set, is the same as direct mapping.*

6b	Draw for lK x 1memory chip with neat figure.	8Diagram-4)



The memory circuit in Figure 5.2 stores 128 bits and requires 14 external connections for address, data, and control lines. Of course, it also needs two lines for power supply and ground connections. Consider now a slightly larger memory circuit, one that has 1K (1024) memory cells. This circuit can be organized as a 128 × 8 memory, requiring a total of 19 external connections. Alternatively, the same number of cells can be organized into a 1K × 1 format. In this case, a 10-bit address is needed, but there is only one data line, resulting in 15 external connections. Figure 5.3 shows such an organization. The required 10-bit address is divided into two groups of 5 bits each to form the row and column addresses for the cell array. A row address selects a row of 32 cells, all of which are accessed in parallel. However, according to the column address, only one of these cells is connected to the external data line by the output multiplexer and input demultiplexer.

	Module 4	
7a	**7a).With the help of a diagram explain 4 bit carry look ahead adder and it operation**	10M(4M – Diagram)



The diagram of a carry look ahead adder is as shown above,

In ripple carry adders, the carry propagation time is the major speed limiting factor. Most other arithmetic operations like multiplication and division are implemented using several add/subtract steps. Thus, improving the speed of addition will improve the speed of all other arithmetic operations.
Carry-lookahead adder improves the speed by reducing carry propagation delay. It calculates the carry signals in advance, based on input signals instead of waiting for them to ripple through the adders.

$$S_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = xy_i + x_i c_i + y_i c_i$$

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

~~$c_{i+1} = x_i y_i$~~

$$c_{i+1} = G_i + P_i c_i$$

where

$G_i = x_i y_i$ → Generate function

$P_i = x_i + y_i$ → Propagate function

Consider the design of 4-bit adder.

$C_1 = G_0 + P_0 C_0$

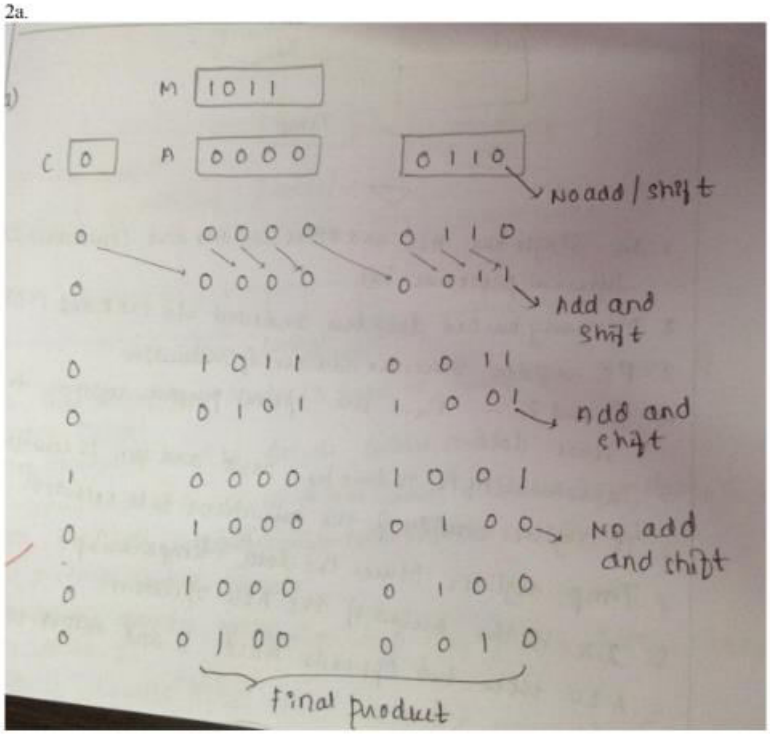
$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

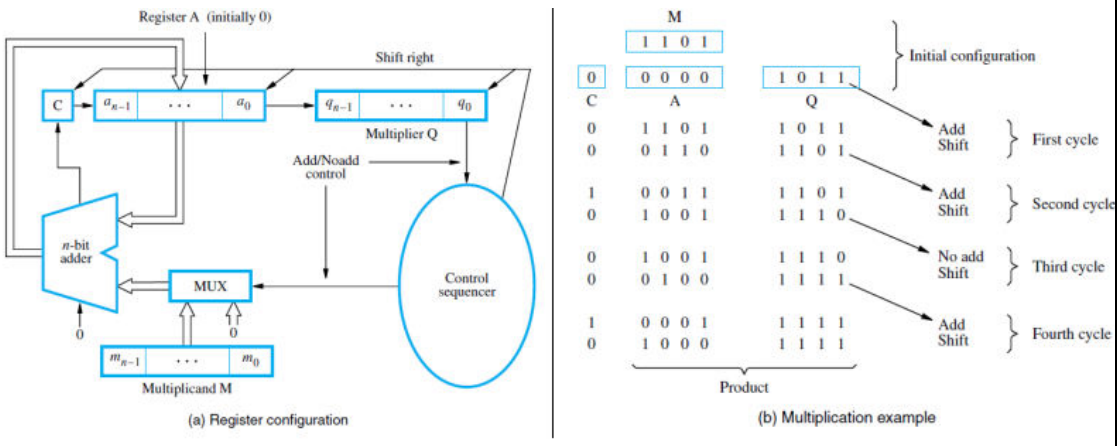
$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

Each carry signal is expressed as a direct Sum of Product (SOP) of C_0 rather than its preceding carry signal.

2a.

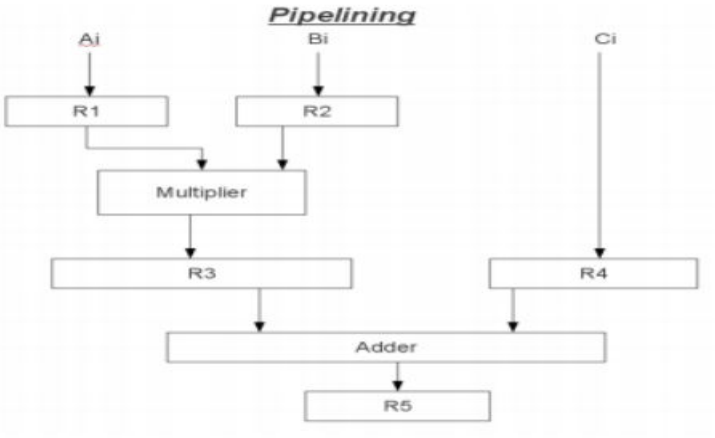


7b	**Illustrate the hardware arrangement of sequential Multiplication** Multiplication is performed as a series of (n) conditional addition and shift operation such that if the given bit of the multiplier is 0 then only a shift operation is performed, while if the given bit of the multiplier is 1 then addition of the partial products and a shift operation are performed. The combinational array multiplier uses a large number of logic gates for multiplying numbers. Multiplication of two n-bit numbers can also be performed in a sequential circuit that uses a single n bit adder. The block diagram in Figure shows the hardware arrangement for sequential multiplication. This circuit performs multiplication by using single n-bit adder n times to implement the spatial addition performed by the n rows of ripple-carry adders in Figure. Registers A and Q are shift registers, concatenated as shown. Together, they hold partial product PPi while multiplier bit qi generates the signal Add/Noadd. This signal causes the multiplexer MUX to select 0 when qi = 0, or to select the multiplicand M when qi = 1, to be added to PPi to generate PP(i + 1). The product is computed in n cycles. The partial product grows in length by one bit per cycle from the initial vector, PP0, of n 0s in register A. The carryout from the adder is stored in flipflop C, shown at the left end of the register C. **Algorithm:** (1) The multiplier and multiplicand are loaded into two registers Q and M. Third	10 M(Diagram-5M)

	register A and C are cleared to 0. (2) In each cycle it performs 2 steps: (a) If LSB of the multiplier qi =1, control sequencer generates Add signal which adds the multiplicand M with the register A and the result is stored in A. (b) If qi =0, it generates Noadd signal to restore the previous value in register A. (3) Right shift the registers C, A and Q by 1 bit  	
8a	Draw the single bus architecture and explain the control sequence for execution of the instruction ADD (R3), R1. 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 R3.	10(Diagram-4)
8b	With a neat block diagram, explain hardwired control unit.	10(Diagram-4)

	Explanation	
	Module 5	
9a	**With a suitable example .Explain the concept of pipeline processing** Performance of a computer can be increased by increasing the performance of the CPU. This can be done by executing more than one task at a time. This procedure is referred to as pipelining. The concept of pipelining is to allow the processing of a new task even though the processing of previous task has not ended. Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments. A pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments. Consider the following operation: Result=(A+B)*C First the A and B values are Fetched which is nothing but a "Fetch Operation". The result of the Fetch operations is given as input to the Addition operation, which is an Arithmetic operation. The result of the Arithmetic operation is again given to the Data operand C which is fetched from the memory and using another arithmetic operation which is Multiplication in this scenario is executed. Finally the Result is again stored in the "Result" variable. In this process we are using up-to 5 pipelines which are Fetch Operation (A) Fetch Operation(B) Addition of (A & B), Fetch Operation(C)	10((Example-4)

Multiplication of ((A+B), C)
Load ((A+B)*C)



Now consider the case where a k-segment pipeline with a clock cycle time t, is used to execute n tasks. The first task T1 requires a time equal to k t, to complete its operation since there are k segments in the pipe.
The remaining n - 1 tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n - 1)t, . Therefore, to complete n tasks using a k-segment pipeline requires k + (n - 1) clock cycles.
For example, the diagram of Fig. shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.

TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

9b **Draw and Explain the pipeline for floating point addition and subtraction** — 10(Diagram-3+example-4)

Arithmetic Pipelines are commonly used in various high-performance computers. They are used in order to implement floating-point operations, fixed-point multiplication, and other similar kinds of calculations that come up in scientific situations.

Let's look at an example to better understand the ideas of an arithmetic pipeline. We perform addition and subtraction of floating points on a unit of the pipeline here.

The inputs in the floating-point adder pipeline refer to two different normalized floating-point binary numbers. These are defined as follows:

$A = X * 2^x = 0.9504 * 10^3$

$B = Y * 2^y = 0.8200 * 10^2$

Where x and y refer to the exponents and X and Y refer to two fractions representing the mantissa.

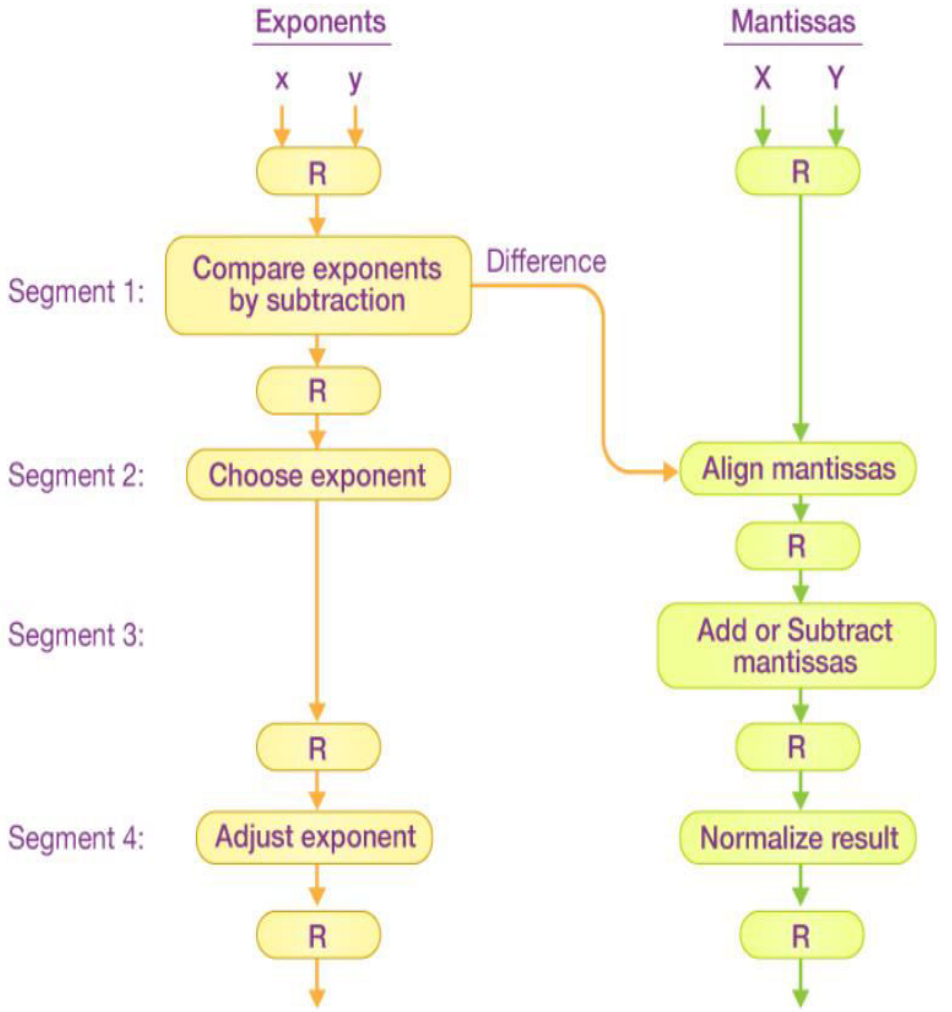
The floating-point addition and subtraction process is broken into four pieces. The matching sub-operation to be executed in the specified pipeline is contained in each segment. The four segments depict the following sub-operations:

1. Comparing the exponents using subtraction

2. Aligning the mantissa

3. Adding or subtracting the mantissa

4. Normalizing the result



1. Comparing Exponents by Subtraction

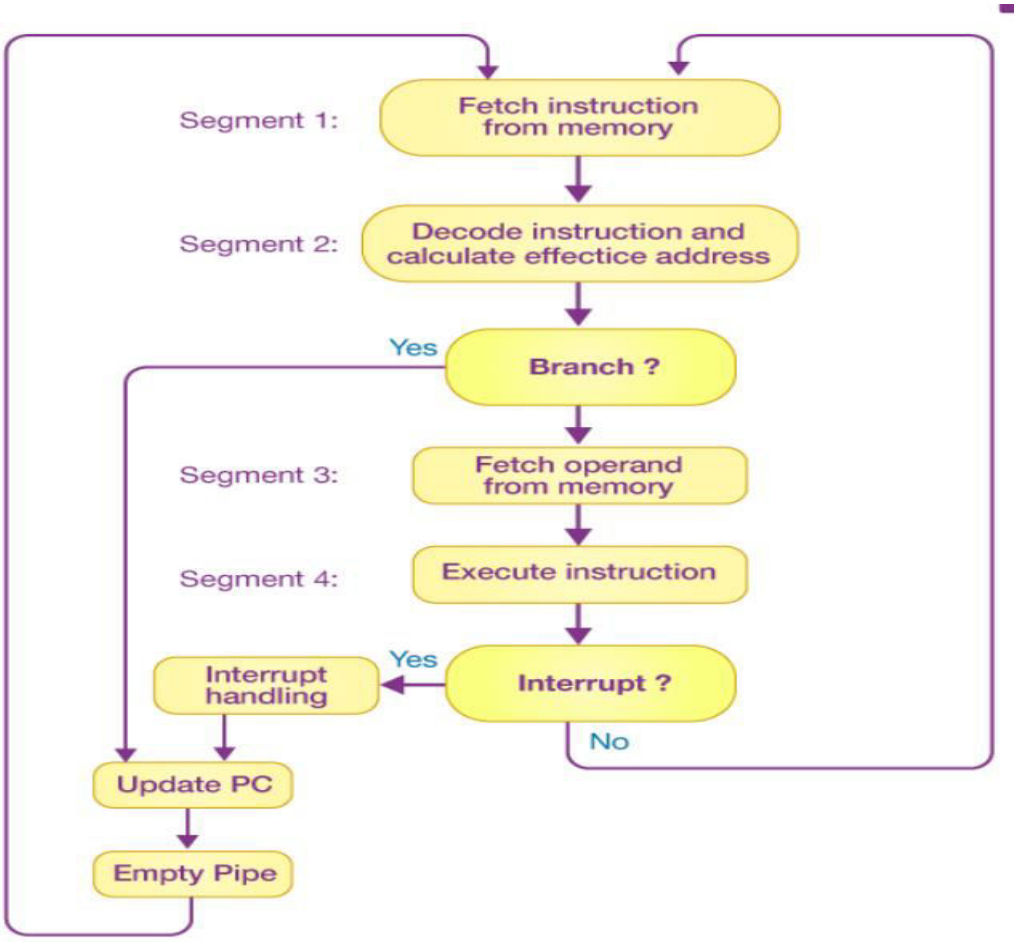
The difference between the exponents is calculated by subtracting them. The result's

	exponent is chosen to be the larger exponent. The exponent difference, 3 – 2 = 1, defines the total number of times the mantissa associated with the lesser exponent should be shifted to the right. **2. Aligning the Mantissa** As per the difference of exponents calculated in segment one, the mantissa corresponding with the smaller exponent would be moved. $A = 0.9504 * 10^3$ $B = 0.08200 * 10^3$ **3. Adding the Mantissa** Both the mantissa would be added in the third segment. $C = A + B = 1.0324 * 10^3$ **4. Normalizing the Result** After the process of normalization, the result would be written as follows: $C = 0.1324 * 10^4$	
10a	**With the help of a timing diagram explain 4 segment instruction pipeline** Pipeline processing can happen not only in the data stream but also in the instruction stream. To perform tasks such as fetching, decoding and execution of instructions, most digital computers with complicated instructions would require an instruction pipeline. In general, each and every instruction must be processed by the computer in the following order: 1. Fetching the instruction from memory 2. Decoding the obtained instruction 3. Calculating the effective address 4. Fetching the operands from the given memory 5. Execution of the instruction 6. Storing the result in a proper place Each step is carried out in its own segment, and various segments may take different	10((Diagram-4)

amounts of time to process the incoming data. Furthermore, there are occasions when multiple segments request memory access at the very same time, requiring one segment to wait unless and until the memory access of another is completed.

If the instruction cycle is separated into equal-length segments, the organisation of an instruction pipeline will become much more efficient. A four-segment type of instruction pipeline refers to one of the most common instances of this style of organisation.

A four-segment instruction pipeline unifies two or more distinct segments into a single unit. For example, the decoding of the instruction and the calculation of the effective address can be merged into a single segment.



A four-segment instruction pipeline is illustrated in the block diagram given above. The instructional cycle is divided into four parts:

Segment 1

The implementation of the instruction fetch segment can be done using the FIFO or first-in, first-out buffer.

Segment 2

In the second segment, the memory instruction is decoded, and the effective address is

then determined in a separate arithmetic circuit.

Segment 3

In the third segment, some operands would be fetched from memory.

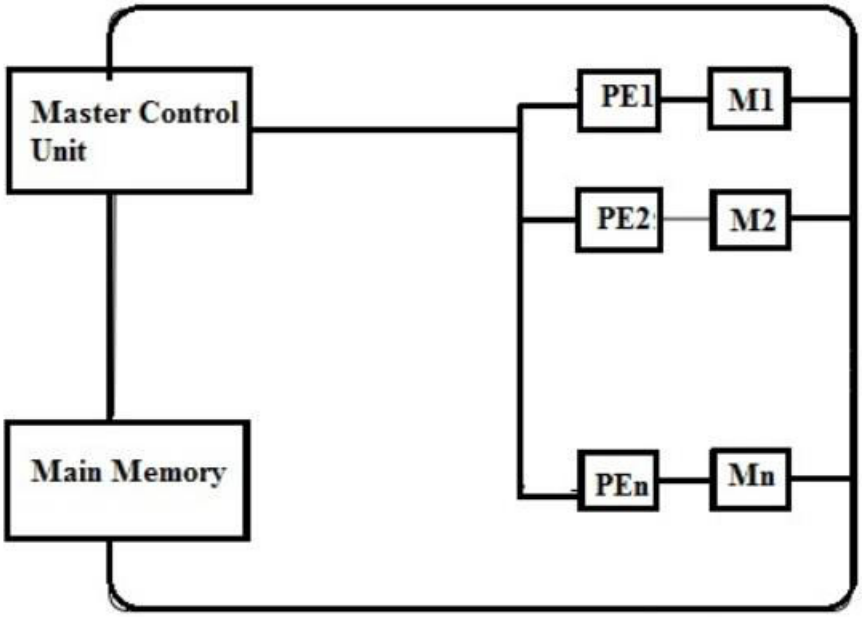
Segment 4

The instructions would finally be executed in the very last segment of a pipeline organisation.

	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

10b **Explain the organization of SIMD array processor with appropriate diagram?** 10(Diagram-4)

SIMD ('Single Instruction and Multiple Data Stream') processors is a computers with several processing units which operate in parallel. These processing units perform the same operation in synchronizing under the supervision of the common control unit (CCU). The SIMD processor includes a set of identical PEs (processing elements) where each PES has a local memory.



This processor includes a master control unit and main memory. The master control unit in the processor controls the operation of the processing elements. And also, decodes the instruction & determines how the instruction is executed. So, if the instruction is program control or scalar then it is executed directly in the master control unit. Main

memory is mainly used to store the program while every processing unit uses operands that are stored in its local memory.

Advantages

The advantages of an array processor include the following.

- Array processors improve the whole instruction processing speed.
- These processors run asynchronously from the host CPU the overall capacity of the system is improved.
  These processors include their own local memory that provides extra memory to systems. So this is an important consideration for the systems through a limited address space or physical memory.
- These processors simply perform computations on a huge array of data.
- These are extremely powerful tools that help in handling troubles with a high amount of parallelism.
- This processor includes a number of ALUs that permits all the array elements to be processed simultaneously.
- Generally, the I/O devices of this processor-array system are very efficient in supplying the required data to the memory directly.
- The main advantage of using this processor with a range of sensors is a slighter footprint.