Course Code : MCS-202
Course Title : Computer Organisation
Assignment Number : PGDCA(I)/202/Assignment/2023
Last Dates for Submission : 30th April, 2023 (for January session)
31st October, 2023 (for July session)

There are four questions in this assignment, which carries 80 marks. Rest 20 marks are for viva voce. You April use illustrations and diagrams to enhance the explanations. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation. Answer to each part of the question should be confined to about 300 words. Make suitable assumption, if any.

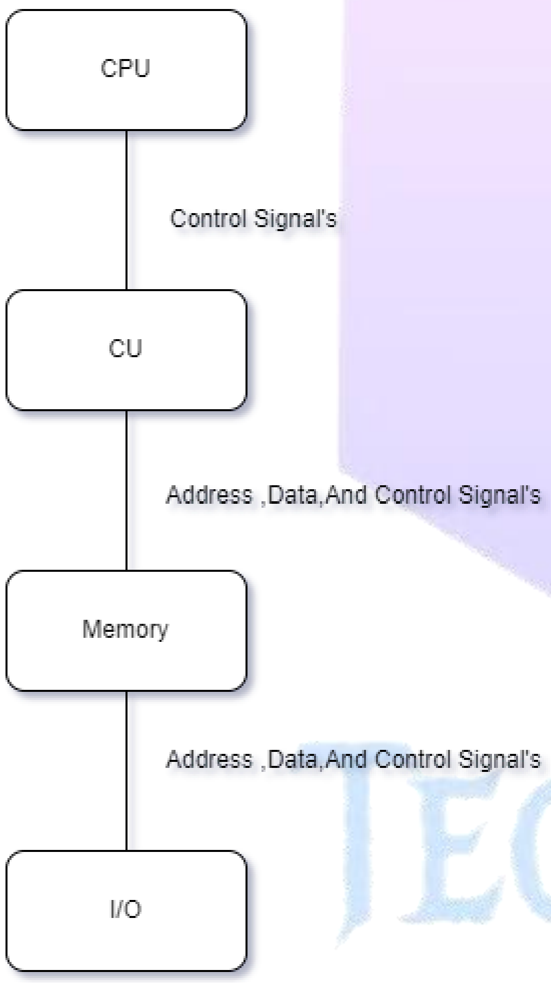
Question 1. (covers Block1) (2 marks each × 10 parts =20 Marks)

(a) Define the main features of von Neumann's architecture with the help of a diagram.

Ans:-
Von Neumann architecture is a model of computer architecture based on the idea of having a central processing unit (CPU), memory, input/output (I/O) devices, and a bus connecting them all. It was proposed by mathematician and computer scientist John von Neumann in the mid-1940s and became the dominant architecture for electronic computers.
The main features of von Neumann's architecture are:

1. A single processor or CPU, which executes instructions stored in memory.
2. A single address bus that connects the CPU to memory and allows the CPU to read and write data from and to memory.
3. A single data bus that carries data between the CPU, memory, and I/O devices.
4. A single control unit that manages the flow of data between the CPU, memory, and I/O devices.
5. Stored program concept: instructions and data are stored in the same memory, and both are accessed using the same address bus. This allows the CPU to execute programs stored in memory.
6. Instruction execution cycle: the CPU fetches an instruction from memory, decodes it, executes it, and then stores the result back in memory. This cycle repeats until the program is complete.

The following diagram illustrates the von Neumann architecture:


CPU
Control Signal's
CU
Address ,Data,And Control Signal's
Memory
Address ,Data,And Control Signal's
I/O

In this diagram, the CPU is connected to the control unit (CU) through control signals, and the CU is connected to memory and I/O devices through address, data, and control signals. The memory and I/O devices are connected to the CPU through the same data bus. The instruction execution cycle is also illustrated in this diagram.

(b) List the differences between von Neumann and Harvard architecture.

Ans:- Von Neumann and Harvard architectures are two different models of computer architecture. The main differences between them are:

1. Memory organization: In von Neumann architecture, both instructions and data are stored in the same memory unit and accessed through the same data bus. In Harvard architecture, instructions and data are stored in separate memory units and accessed through separate data buses.
2. Instruction fetching: In von Neumann architecture, the CPU fetches instructions from memory one at a time and executes them sequentially. In Harvard architecture, the CPU can fetch both instruction and data simultaneously from separate memories, allowing for faster execution.

3. Instruction execution: In von Neumann architecture, instructions and data compete for the same bus, leading to potential delays in execution. In Harvard architecture, the separate data buses allow for parallel execution of instruction and data, leading to faster execution.
4. Flexibility: Von Neumann architecture is more flexible than Harvard architecture because it can store and retrieve both instructions and data from the same memory. Harvard architecture is less flexible because it requires separate memory units for instructions and data.
5. Complexity: Harvard architecture is more complex than von Neumann architecture because it requires separate memory units and data buses for instructions and data. This can lead to increased hardware costs and more complicated software design.
6. Use cases: Von Neumann architecture is commonly used in general-purpose computers, where flexibility is important, while Harvard architecture is commonly used in embedded systems and specialized applications, where speed is more important than flexibility.

Overall, von Neumann architecture is simpler and more flexible, while Harvard architecture is more complex but can be faster in certain applications.

(c) Perform the following conversion of numbers:
(i) Decimal $(5432109876)_{10}$ to binary and hexadecimal.

Ans:-

To convert $(5432109876)_{10}$ to binary, we can use the method of repeated division by 2. We divide the decimal number by 2 and record the remainder (0 or 1) as the least significant bit (LSB) of the binary number. We then divide the quotient by 2 and record the remainder as the next bit of the binary number. We repeat this process until the quotient is 0.

5432109876 / 2 = 2716054938, remainder 0 (LSB)
2716054938 / 2 = 1358027469, remainder 0
1358027469 / 2 = 679013734, remainder 1
679013734 / 2 = 339506867, remainder 0
339506867 / 2 = 169753433, remainder 1
169753433 / 2 = 84876716, remainder 1
84876716 / 2 = 42438358, remainder 0
42438358 / 2 = 21219179, remainder 0
21219179 / 2 = 10609589, remainder 1
10609589 / 2 = 5304794, remainder 1
5304794 / 2 = 2652397, remainder 0

Therefore, $(5432109876)_{10}$ = $(10100001100100100111011100001100)_2$.

To convert $(5432109876)_{10}$ to hexadecimal, we can use the method of repeated division by 16. We divide the decimal number by 16 and record the remainder as a hexadecimal digit (0-9 or A-F) of the hexadecimal number. We then divide the quotient by 16 and record the remainder as the next digit of the hexadecimal number. We repeat this process until the quotient is 0.

5432109876 / 16 = 339506867, remainder 12 (C)
339506867 / 16 = 21219179, remainder 11 (B)
21219179 / 16 = 1326198, remainder 11 (B)
1326198 / 16 = 82887, remainder 6 (6)
82887 / 16 = 5180, remainder 7 (7)
5180 / 16 = 323, remainder 12 (C)
323 / 16 = 20, remainder 3 (3)
20 / 16 = 1, remainder 4 (4)
1 / 16 = 0, remainder 1 (1)

Therefore, $(5432109876)_{10}$ = $(CBB66C3471)_{16}$.

(ii) Hexadecimal $(FCEB9A86)_h$ to Octal.

Ans:-

To convert (FCEB9A86)h to octal, we can first convert it to binary and then group the binary digits into groups of three (starting from the right-most digit) and convert each group into its octal equivalent.

$(FCEB9A86)_h$ = $(11111100111010111001101010000110)_2$

Grouping into threes: 011 111 100 111 010 111 001 101 010 000 110

Converting each group to octal: 317 757 151 652

Therefore, $(FCEB9A86)_h$ = $(317757151652)_8$.

(iii) String “ABCD987$@&” to UTF 16

Ans:- To convert the string "ABCD987$@&" to UTF-16, we need to first assign a Unicode code point to each character in the string, and then convert the code points to their UTF-16 representation.

The Unicode code points for the characters in the string "ABCD987$@&" are:
A: U+0041
B: U+0042
C: U+0043
D: U+0044
9: U+0039
8: U+0038
7: U+0037
$: U+0024
@: U+0040
&: U+0026
To convert each code point to its UTF-16 representation, we can use the following rules:
For code points in the Basic Multilingual Plane (BMP), which includes most commonly used characters, the UTF-16 representation is simply the code point expressed as a 16-bit integer (with the most significant byte first).
For code points outside the BMP, the UTF-16 representation is a surrogate pair, consisting of two 16-bit integers.
Applying these rules, we get the following UTF-16 representation for the string "ABCD987$@&":
A: U+0041 → 0041 (in UTF-16)
B: U+0042 → 0042
C: U+0043 → 0043
D: U+0044 → 0044
9: U+0039 → 0039
8: U+0038 → 0038
7: U+0037 → 0037
$: U+0024 → 0024
@: U+0040 → 0040
&: U+0026 → 0026
Therefore, the UTF-16 representation of the string "ABCD987$@&" is the sequence of 16-bit integers:

0041 0042 0043 0044 0039 0038 0037 0024 0040 0026

(iv) Octal $(10235647)_0$ to Decimal

To convert $(10235647)_0$ to decimal, we can use the positional notation system with base 8. Each digit in the octal number represents a power of 8.

Starting from the rightmost digit, the powers of 8 are 8^0 = 1, 8^1 = 8, 8^2 = 64, 8^3 = 512, and so on.

Therefore, we can calculate the decimal equivalent of $(10235647)_0$ as follows:

(18^7) + (08^6) + (28^5) + (38^4) + (58^3) + (68^2) + (48^1) + (78^0)
= 8,388,607 + 0 + 262,144 + 12,288 + 2,040 + 384 + 32 + 7
= 8,665,402

Therefore, (10235647)O = $(8665402)_{10}$ in decimal.

(d) Simplify the following function using K-map:F(A, B, C, D) = Σ (1, 3, 5, 7, 13, 15). Draw the resultant circuit for the function using NAND gates.

Ans:-

CD AB	00	01	11	10
00	1		1	
01	1		1	
11	1	1	1	
10				

We group the adjacent 1's to form the largest possible groups. In this case, we can form two groups:

A'B'C' + A'BC' + AB'C' + ABC' + A'B'C + AB'C
A'BD' + A'BCD' + ABCD' + AB'CD'
Simplifying each group using Boolean algebra, we get:

A'C' + BC' + AB'
A'D' + CD' + AB'CD
Therefore, the simplified function is:

F(A,B,C,D) = (A'C' + BC' + AB') + (A'D' + CD' + AB'CD)

To draw the circuit using NAND gates, we first convert each term to NAND form:

A'C' + BC' + AB' = (A + C)(B + C')(A + B')
A'D' + CD' + AB'CD = (A + D)(C + D')(A + B')(C + D)
Then, we implement each term using NAND gates:

(e) Consider the Adder-Subtractor circuit given in Unit 3 of Block 1. Explain how this circuit will perform subtraction (A-B) if the value of A is 0001 and B is 1101. You must list all the bit values including Cin and Cout and overflow condition.
Ans:-
The Adder-Subtractor circuit given in Unit 3 of Block 1 can perform both addition and subtraction using a single circuit. The circuit consists of a 4-bit adder and a control input called Op which determines whether the circuit performs addition or subtraction.

When Op is 0, the circuit performs addition, and when Op is 1, the circuit performs subtraction. In subtraction mode, the circuit performs 2's complement subtraction.

To perform the subtraction (A-B) when A = 0001 and B = 1101, we need to first convert B to its 2's complement form. To do this, we invert all the bits of B and then add 1 to the result. The 2's complement of B is therefore:

B' = NOT(B) + 1
= 0010 + 1
= 0011

Now, we can use the Adder-Subtractor circuit with Op = 1 to perform the subtraction (A-B') = 0001 - 0011.

To perform the subtraction, we need to add A and the 2's complement of B (B') together, and set the Op input to 1. We can represent the numbers using 4-bit binary notation and the addition can be performed as follows:

```
  0001    (A)
+  0011    (B' = 2's complement of B)
 -------
  0100    (sum)
```

The result of the addition is 0100 in binary notation, which is equal to 4 in decimal notation. Therefore, (A-B) = (0001-1101) = -3 in decimal notation.

Now, to determine the overflow condition, we need to check the most significant bit (MSB) of both the operands and the result. In this case, the MSB of A is 0, the MSB of B' is also 0, and the MSB of the result (sum) is 0. Since the signs of the operands are different and the MSB of the result is 0, there is no overflow condition.

The bit values of Cin and Cout are 0 and 0 respectively, since there is no carry or borrow from the highest bit position.

(f) Explain the functioning of a 3X8decoder. You should draw its truth table and explain its logic diagram with the help of example input.
Ans:-
A 3x8 decoder is a combinational logic circuit that has three inputs and eight outputs. It decodes a three-bit binary input into one of the eight possible outputs. The decoder selects only one output line based on the input combination. The output is usually active low, which means that the output will be low (0) only for the selected output line, and all other outputs will be high (1).

Here is the truth table for a 3x8 decoder:

A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0

1 1 1 0 0 0 0 0 0 0 1

The three inputs are labeled A, B, and C, and the eight outputs are labeled Y0 through Y7. Each row in the truth table shows the binary input values and the corresponding output values.

The logic diagram of a 3x8 decoder consists of three AND gates and eight inverters. The three inputs A, B, and C are connected to the inputs of the AND gates. The output of each AND gate is connected to one of the eight inverters. The output of each inverter is one of the eight outputs.

Here is an example input for a 3x8 decoder: A=1, B=0, C=1

Using the truth table, we can determine that the output should be Y5=1, and all other outputs should be 0. This means that the output line for Y5 will be low (0), and all other output lines will be high (1).

(g) Assume that a source data value of 1110 was received at a destination as 1100. Show how Hamming's Error-Correcting code will be appended to source data, so this error of one bit is identified and corrected at the destination. You April assume that error while transmission occurs only in the source data and not in the code

Ans:-

To use Hamming's Error-Correcting Code to identify and correct the error in the received data, we need to append additional bits to the original data. The number of additional bits required depends on the size of the original data. In this case, we need to append 4 bits to the original 4 bits of data to make a total of 8 bits. The additional bits will be used to create a parity check matrix that will allow the receiver to identify and correct the error in the received data.

Here are the steps to append the Hamming Code to the original data:

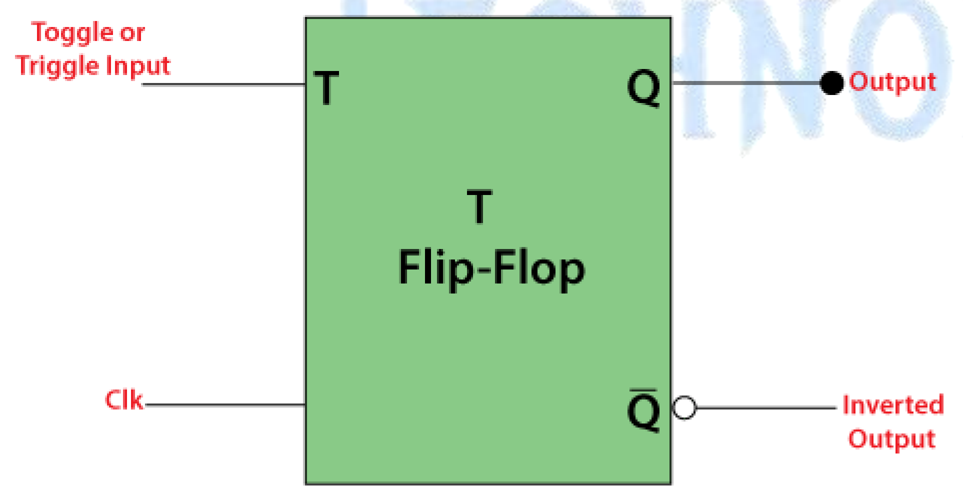
Write the data bits in binary format: 1110
Determine the number of parity bits required by finding the smallest number that satisfies the following equation: 2^r >= m + r + 1, where r is the number of parity bits and m is the number of data bits. In this case, m = 4, so we need to find the smallest r that satisfies 2^r >= 4 + r + 1. It can be seen that r = 3 satisfies this equation.
Reserve the positions for the parity bits by placing a 'P' at the positions of powers of 2: _ P _ 1 _ 1 0 _ 1 1 1 0
Fill in the data bits in the remaining positions: 0 0 1 1 P 1 0 0 P 1 1 1 0
Calculate the value of each parity bit by counting the number of 1s in the data bits that it covers, including itself. Write the value of the parity bit in binary format. For example, P1 covers bits 1,3,5,7, so we count the number of 1s in those bits and write the value of P1 as the binary equivalent of that count, which is 2 in this case.
Write the final code by placing the calculated parity bits in the reserved positions: 0 0 1 1 0 1 0 0 1 1 1 1 0
The final code that is transmitted is 001101001110. Now, if one bit is changed during transmission, the receiver can identify and correct the error by using the parity bits. For example, if the received code is 001101001100, the receiver can identify the error in bit 8 by calculating the parity bits using the received data and comparing them to the parity bits that were transmitted. The receiver can then correct the error by flipping the incorrect bit from 0 to 1 or from 1 to 0, and the original data of 1110 will be recovered.

(h) Explain the functioning of the T flip-flop with the help of a logic diagram and characteristic table. Also, explain the excitation table of this flip-flop.

Ans:-

The T flip-flop, also known as a toggle flip-flop, is a type of flip-flop that can change its output state based on the value of a toggle input (T). It has two stable states: the output is either 0 or 1, and it can be toggled between these two states by applying a pulse to the T input.

The logic diagram of a T flip-flop is shown below:



The T input is connected to the inputs of two logic gates: an XOR gate and a NOT gate. The output of the XOR gate is connected to the input of the NOT gate, and the output of the NOT gate is the output of the flip-flop (Q). The other output of the XOR gate is the complement of Q (NOT Q).

The characteristic table of the T flip-flop is shown below:

T	Q(t)	Q(t+1)
0	Q	Q
1	NOT Q	Q'

The characteristic table shows the relationship between the input (T), the current state of the flip-flop (Q(t)), and the next state of the flip-flop (Q(t+1)). When T is 0, the output of the flip-flop remains in its current state, which means that Q(t+1) is equal to Q(t). When T is 1, the output of the flip-flop toggles to the opposite state, which means that Q(t+1) is the complement of Q(t), denoted as Q'.
The excitation table of the T flip-flop shows the input required to make the flip-flop change its state from its current state to the desired next state. The excitation table for the T flip-flop is shown below:

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

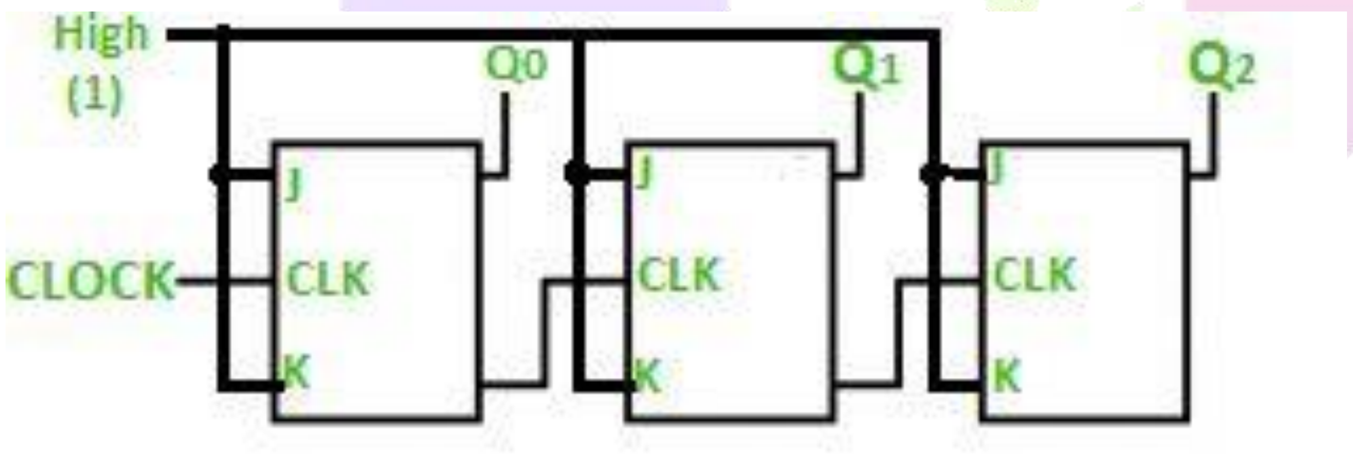
The excitation table shows that if the current state of the flip-flop is 0 and we want to set it to 1, we need to apply a pulse to the T input. Similarly, if the current state is 1 and we want to set it to 0, we need to apply a pulse to the T input. If we want to keep the current state of the flip-flop, we need to keep the T input at 0.

In summary, the T flip-flop is a useful building block in digital circuits, especially in circuits that require the toggling of a signal between two states. Its characteristic table and excitation table provide a convenient way to design and analyze circuits that use T flip-flops.

(i) Explain the functioning of a 3-bitripple counter with the help of a diagram.

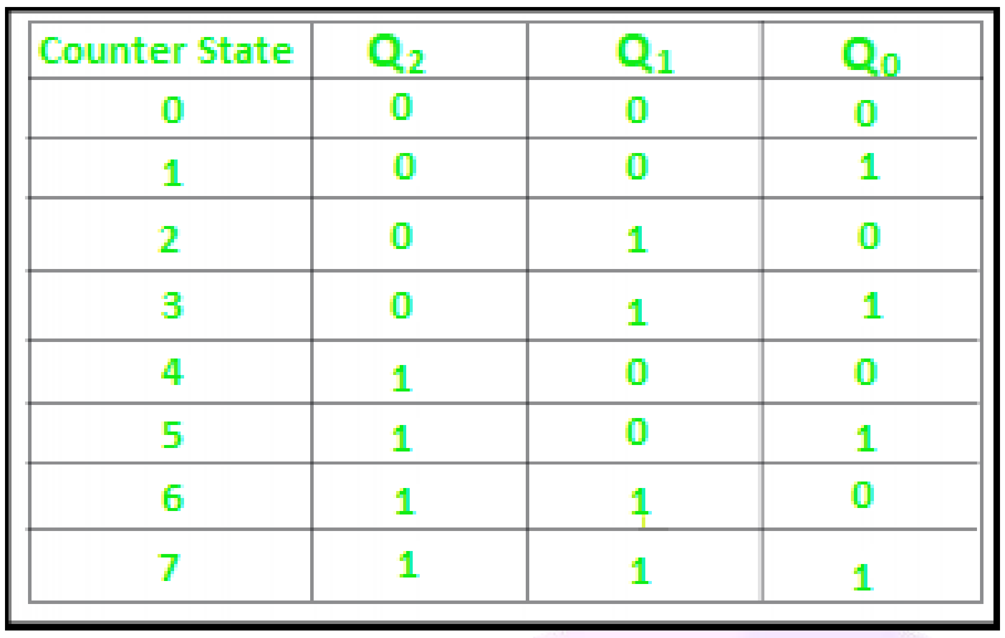
Ans:- A 3-bit ripple counter is a type of digital circuit that counts from 0 to 7 (in binary) and then repeats the counting sequence. It consists of three flip-flops connected in a cascade, where the output of each flip-flop is connected to the clock input of the next flip-flop.

The diagram of a 3-bit ripple counter is shown below:



The circuit is driven by a clock signal (C) that is connected to the clock input of the first flip-flop (FF1). When the clock signal changes from 0 to 1, FF1 changes its output state based on its characteristic table (which can be either a D, JK or T flip-flop depending on the design). The output of FF1 (Q0) is connected to the clock input of the second flip-flop (FF2), causing it to change its output state on the next clock pulse. Similarly, the output of FF2 (Q1) is connected to the clock input of the third flip-flop (FF3), causing it to change its output state on the next clock pulse.

The three flip-flops work together to count from 0 to 7 (in binary). The binary count sequence of the three bits is shown below:



Counter State	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The count sequence repeats after 7, as the flip-flops continue to toggle in a binary sequence.

In summary, a 3-bit ripple counter is a simple digital circuit that uses three flip-flops to count from 0 to 7 and then repeat the counting sequence. Its operation is based on the principle of cascading flip-flops, where the output of one flip-flop is connected to the clock input of the next flip-flop.

(j) Represent (-78.25)10 and (0.03125)10 in IEEE 754 single precision format.
Ans:-
The IEEE 754 single precision format is a binary representation of a floating-point number using 32 bits, with 1 bit for the sign, 8 bits for the exponent, and 23 bits for the significand.

To represent a decimal number in the IEEE 754 format, we need to convert it to binary and then normalize it to have a single nonzero digit before the decimal point. We can then express the number in scientific notation as follows:

number = (-1)^significand x 1.significand x 2^exponent

where the significand is the binary fraction after normalization, the exponent is the power of 2 needed to shift the radix point to the right place, and the sign bit is 0 for positive numbers and 1 for negative numbers.

Let's first convert (-78.25)10 to binary:

Convert the integer part to binary: 78 = 1001110
Convert the fractional part to binary: 0.25 = 0.01
Concatenate the two binary numbers: 1001110.01
We can now normalize this binary number by shifting the radix point to the left until we have a single nonzero digit before the decimal point:

1.00111001 x 2^6

The sign bit is 1 because the number is negative. The exponent is 6 because we had to shift the radix point 6 places to the right to get a single nonzero digit before the decimal point. The significand is 00111001 (with a leading 1 implied by the format).

We can now represent (-78.25)10 in the IEEE 754 single precision format as follows:

Sign bit: 1 (because the number is negative)
Exponent bits: 6 + 127 = 133 (we add 127 to the biased exponent to get the true exponent)
Significand bits: 00111001000000000000000 (with 23 bits after the leading 1)
Therefore, (-78.25)10 in IEEE 754 single precision format is:

1 10000101 00111001000000000000000

Let's now convert (0.03125)10 to binary:

Convert the fraction to binary: 0.03125 = 0.00001
We can now normalize this binary number by shifting the radix point to the right until we have a single nonzero digit before the decimal point:

1.0000 x 2^-5

The sign bit is 0 because the number is positive. The exponent is -5 because we had to shift the radix point 5 places to the left to get a single nonzero digit before the decimal point. The significand is 0000 (with a leading 1 implied by the format).

We can now represent (0.03125)10 in the IEEE 754 single precision format as follows:

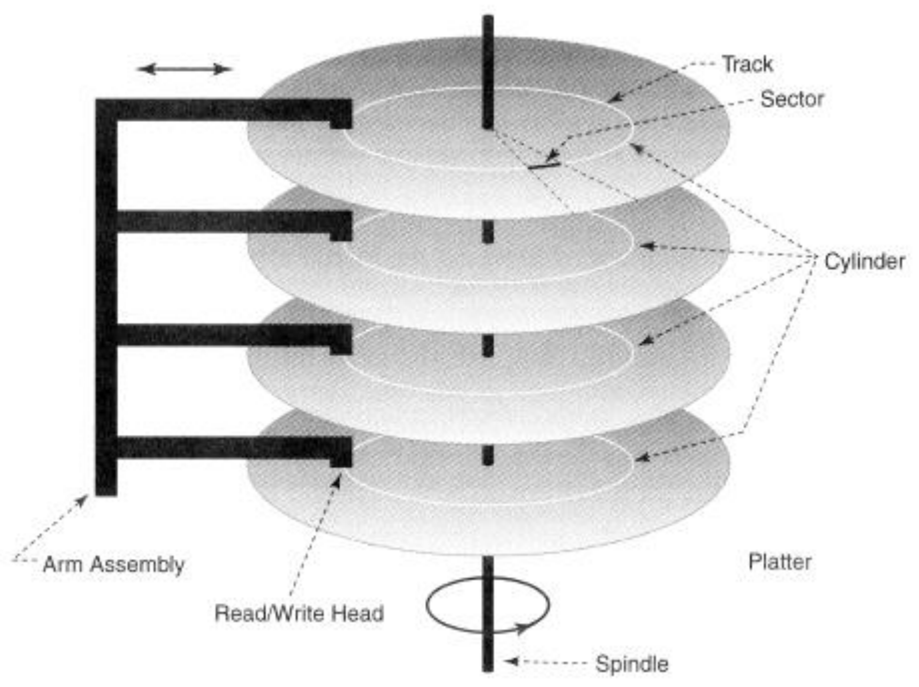
Sign bit: 0 (because the number is positive)
Exponent bits: -5 + 127 = 122 (we add 127 to the biased exponent to get the true exponent)
Significand bits: 00000000000000000000000 (with 23 bits after the leading 1)
Therefore, (0.03125)10 in IEEE 754 single precision format is:

0 01111010 00000000000000000000000

Question 2. (covers Block 2) (4 marks each × 5 parts =20 Marks)
(a) What is DRAM? Why is it used as the main memory of a computer? How is it different that the cache memory? Explain the data organisation of a Hard disk with the help of a diagram. What is CLV? What are its advantages and disadvantages? How can you overcome the disadvantages of CLV?
Ans.
DRAM, or Dynamic Random Access Memory, is a type of computer memory that stores data and instructions temporarily while the computer is in operation. DRAM is used as the main memory of a computer because it is fast, cheap, and can be easily expanded. It is also volatile, which means that it requires a constant supply of power to maintain its contents.

Cache memory is a type of memory that is used to store frequently accessed data and instructions so that the computer can access them more quickly. Cache memory is typically smaller and faster than DRAM, but also more expensive. Cache memory is usually located closer to the CPU, while DRAM is further away.

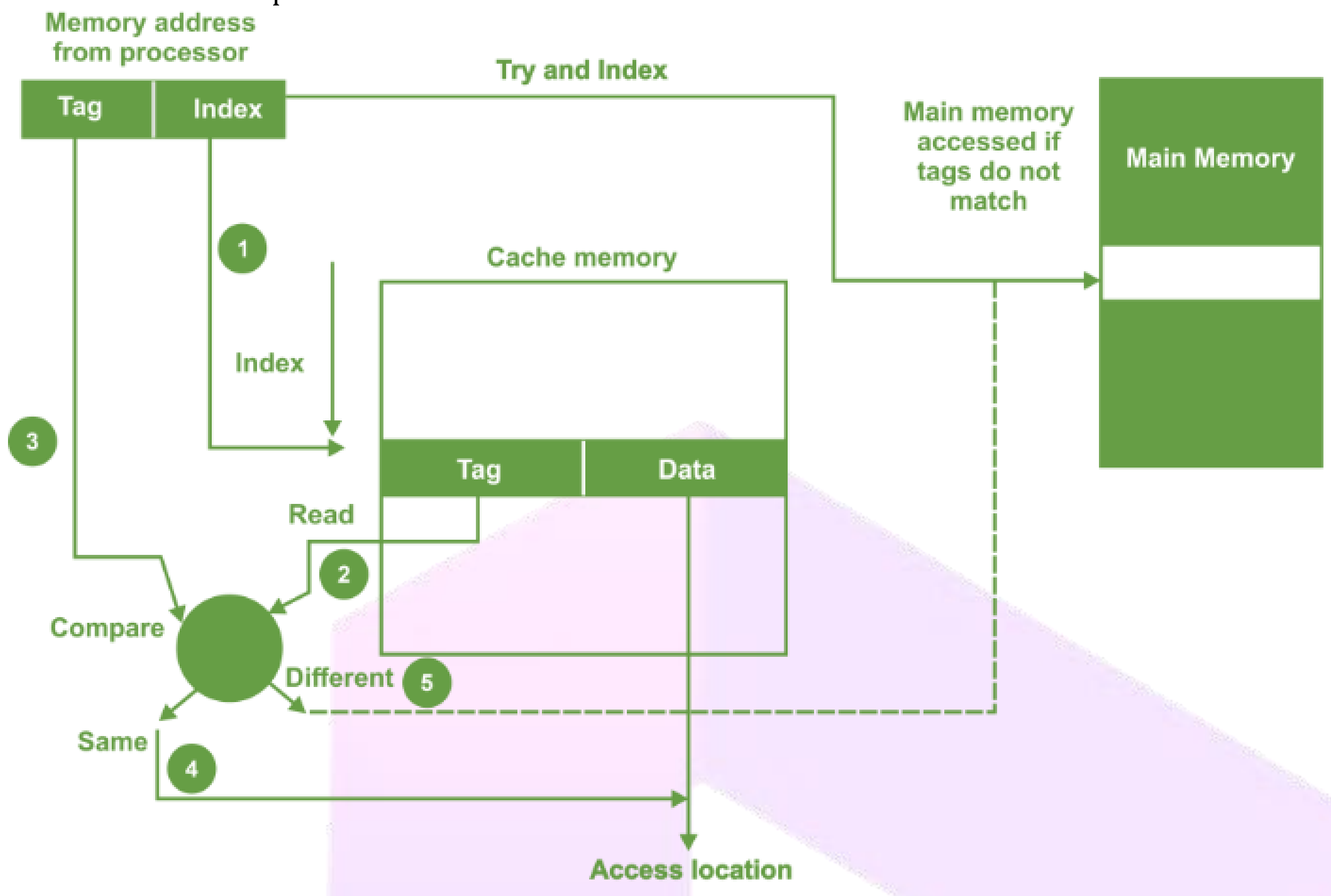
A hard disk is typically divided into multiple platters, each of which is divided into sectors and tracks. The data is stored in concentric circles on the platters, with each circle representing a track. Each track is divided into sectors, which are the smallest units of data that can be read or written. The data is stored on the platters using magnetic fields, which can be read by a read/write head attached to an actuator arm. The following diagram illustrates the data organization of a hard disk:



CLV, or Constant Linear Velocity, is a technique used to write data to a disk at a constant speed. This is in contrast to CAV, or Constant Angular Velocity, where the disk spins at a constant speed while the head moves in and out to read or write data. The advantages of CLV include improved data transfer rates and more consistent read and write performance. The disadvantages of CLV include slower access times for data stored towards the center of the disk and increased wear on the motor due to the need to maintain a constant linear velocity. To overcome these disadvantages, some systems use a combination of CLV and CAV, or use a technique called Zoned Constant Linear Velocity (Z-CLV) where the disk is divided into multiple zones, each with its own constant linear velocity.

(b) Explain the following cache to the main memory mapping scheme with the help of an example and a suitable diagram.
(i) Direct cache mapping
(ii) Four-way set associative cache mapping
Ans.
Direct Mapping -
In direct mapping, the cache consists of normal high-speed random-access memory. Each location in the cache holds the data, at a specific address in the cache. This address is given by the lower significant bits of the main memory address. This enables the block to be selected

directly from the lower significant bit of the memory address. The remaining higher significant bits of the address are stored in the cache with the data to complete the identification of the cached data.

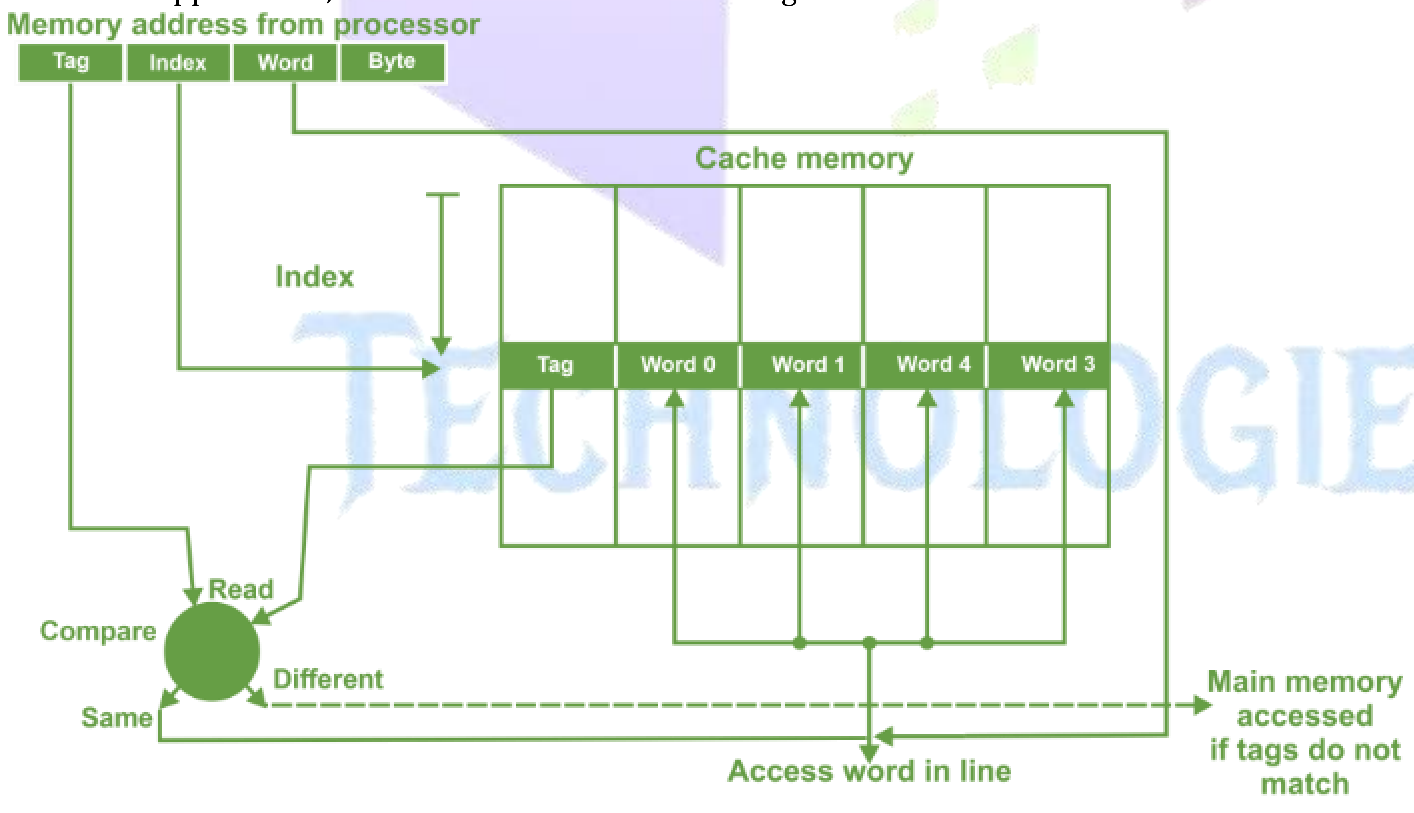


Cache with direct mapping

As shown in the above figure, the address from processor is divided into two field a tag and an index.
The tag consists of the higher significant bits of the address and these bits are stored with the data in cache. The index consists of the lower significant b of the address. Whenever the memory is referenced, the following sequence of events occurs

1. The index is first used to access a word in the cache.
2. The tag stored in the accessed word is read.
3. This tag is then compared with the tag in the address.
4. If two tags are same this indicates cache hit and required data is read from the cache word.
5. If the two tags are not same, this indicates a cache miss. Then the reference is made to the main memory to find it.

For a memory read operation, the word is then transferred into the cache. It is possible to pass the information to the cache and the process simultaneously.
In direct mapped cache, there can also be a line consisting of more than one word as shown in the following figure

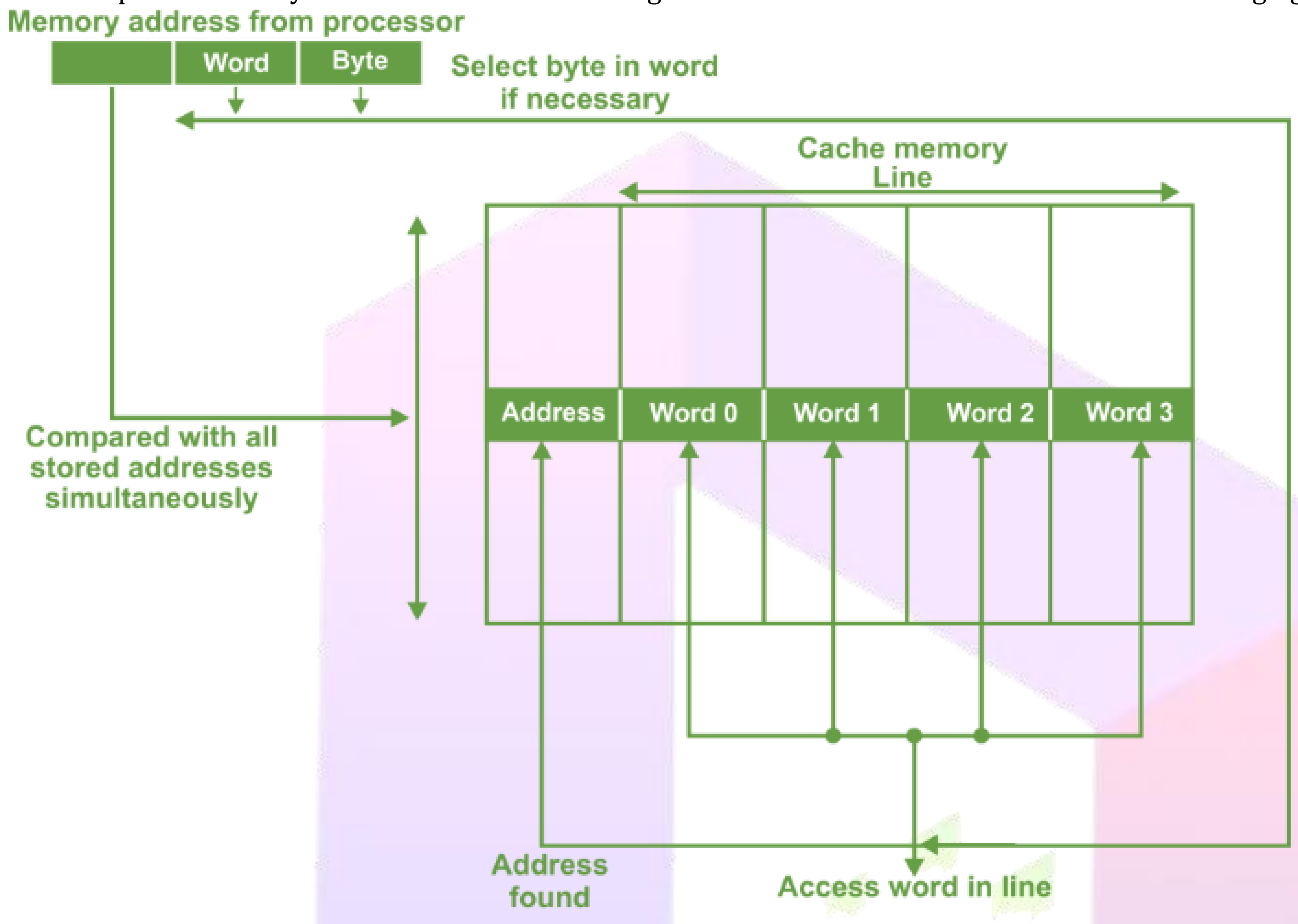


Direct mapped cache with a multi-word block.

In such a case, the main memory address consists of a tag, an index and a word within a line. All the words within a line in the cache have the same stored tag
The index part in the address is used to access the cache and the stored tag is compared with required tag address.
For a read operation, if the tags are same, the word within the block is selected for transfer to the processor. If tags are not same, the block containing the required word is first transferred to the cache. In direct mapping, the corresponding blocks with the same index in the

main memory will map into the same block in the cache, and hence only blocks with different indices can be in the cache at the same time. It is important that all words in the cache must have different indices. The tags may be the same or different.

(ii)Set Associative Mapping -
In set associative mapping a cache is divided into a set of blocks. The number of blocks in a set is known as associativity or set size. Each block in each set has a stored tag. This tag together with index completely identify the block.
Thus, set associative mapping allows a limited number of blocks, with the same index and different tags.
An example of four way set associative cache having four blocks in each set is shown in the following figure


Memory address from processor
Word
Byte
Select byte in word if necessary
Cache memory Line
Address
Word 0
Word 1
Word 2
Word 3
Compared with all stored addresses simultaneously
Address found
Access word in line

Fuly associative mapped cache with multi-word lines.

In this type of cache, the following steps are used to access the data from a cache:

1. The index of the address from the processor is used to access the set.
2. Then the comparators are used to compare all tags of the selected set with the incoming tag.
3. If a match is found, the corresponding location is accessed.
4. If no match is found, an access is made to the main memory.

The tag address bits are always chosen to be the most significant bits of the full address, the block address bits are the next significant bits and the word/byte address bits are the least significant bits. The number of comparators required in the set associative cache is given by the number of blocks in a set. The set can be selected quickly and all the blocks of the set can be read out simultaneously with the tags before waiting for the tag comparisons to be made. After a tag has been identified, the corresponding block can be selected.

(c) What are the different types of Interrupts? How is an interrupt processed? What is the need for DMA even though there exists an Interrupt driven I/O mechanism? Explain the functions of DMA.
Ans.
Different types of Interrupts:

1. Hardware Interrupts: These interrupts are generated by hardware devices such as keyboard, mouse, or network card when they need attention from the CPU.
2. Software Interrupts: These interrupts are triggered by software instructions or system calls.
3. Maskable Interrupts: These interrupts can be disabled or masked by the CPU.
4. Non-Maskable Interrupts: These interrupts cannot be disabled or masked by the CPU.

Processing of Interrupts:
When an interrupt occurs, the CPU temporarily stops executing the current instruction and switches to the interrupt service routine (ISR) to handle the interrupt. The ISR saves the current state of the CPU and performs the necessary actions to handle the interrupt. Once the ISR is completed, the CPU returns to the main program.

Need for DMA:
Even though an interrupt-driven I/O mechanism is available, it can consume a lot of CPU cycles and slow down the system, especially when transferring large amounts of data. DMA (Direct Memory Access) is a mechanism that enables data transfer between devices and memory without the need for CPU intervention. It allows the CPU to perform other tasks while the data transfer is in progress.
Functions of DMA:

1. DMA enables high-speed data transfer between devices and memory without the need for CPU intervention.
2. DMA can transfer data in both directions between memory and devices.
3. DMA reduces the load on the CPU and increases system performance.
4. DMA supports various data transfer modes such as block transfer, cycle-stealing, and demand transfer.
5. DMA can transfer data to or from multiple devices simultaneously.

(d) Explain the functioning of Programmed I/O. What are its advantages and disadvantages? For what kind of architecture would you like to use Programmed I/O? Explain when an I/O processoris needed in a computer system.
Ans.

Programmed I/O is a method of data transfer between a computer's CPU and an I/O device. In programmed I/O, the CPU directly controls the transfer of data between itself and the I/O device without the intervention of an interrupt controller or DMA controller. The CPU issues an I/O command to the I/O device and then waits for the I/O device to complete the command before proceeding with the next instruction.

Advantages of programmed I/O include simplicity and low hardware requirements. Programmed I/O does not require any specialized hardware, which makes it a cost-effective solution for simple I/O operations. Programmed I/O is also easy to implement and does not require complex software drivers or interrupt handlers.

Disadvantages of programmed I/O include slow transfer speeds and increased CPU utilization. Since the CPU controls the transfer of data, it must wait for each I/O operation to complete before proceeding with the next instruction. This can result in slow transfer speeds, particularly for large data transfers. Additionally, programmed I/O can increase CPU utilization, as the CPU must actively manage each I/O operation.

Programmed I/O is best suited for small-scale systems where the I/O requirements are not complex and transfer speeds are not critical. It is typically used in embedded systems and microcontrollers where simplicity and low cost are more important than high performance. An I/O processor is needed in a computer system when the I/O requirements are complex or when high-performance I/O operations are required. An I/O processor is a specialized processor that is dedicated to managing I/O operations. It can perform data transfers between the CPU and I/O devices without the intervention of the CPU, allowing the CPU to focus on other tasks. I/O processors are typically used in high-performance servers and storage systems where fast data transfers are critical.

(e) Explain the following in the context of I/O technologies:
(i) Video Memory
(ii) LCD monitor
(iii) Voice-based input
(iv) Scanners
Ans.

(i) Video Memory: Video memory, also known as VRAM (Video Random Access Memory), is a type of memory that is used by the computer's graphics processing unit (GPU) to store the image data that is displayed on the computer's screen. Video memory is different from the computer's main memory (RAM), and it is optimized for handling the large amount of data that is required for rendering complex graphics and video. Video memory is essential for displaying high-resolution images and videos, and it is used by a variety of I/O technologies, including graphics cards, video game consoles, and video playback devices.

(ii) LCD monitor: LCD (Liquid Crystal Display) monitors are a type of display technology that uses liquid crystals to create images on the screen. LCD monitors are thinner and lighter than traditional CRT (Cathode Ray Tube) monitors, and they consume less power. LCD monitors are commonly used in desktop and laptop computers, televisions, and other devices that require high-quality visual displays. LCD monitors typically connect to the computer using a VGA, DVI, HDMI, or DisplayPort cable.

(iii) Voice-based input: Voice-based input is a type of I/O technology that allows users to input data into a computer or other device using their voice. Voice-based input is commonly used in smartphones and other mobile devices, as well as in virtual assistant technologies such as Siri and Alexa. Voice-based input relies on software that is designed to recognize and interpret spoken commands, and it requires a microphone or other input device that can capture the user's voice.

(iv) Scanners: Scanners are a type of I/O device that are used to create digital images of physical documents or images. Scanners use a variety of technologies to capture images, including flatbed scanners, sheet-fed scanners, and handheld scanners. Scanners typically connect to the computer using a USB or other data cable, and they require specialized software that can interpret the scanned images and convert them into a digital format that can be stored on the computer's hard drive or other storage device. Scanners are commonly used in home and office settings for tasks such as document scanning, image archiving, and photo restoration.

Question 3. (Covers Block 3)
(a) Explain the following instructions for a computer system:
i) Shift Instructions
ii) Branch and Jump Instructions
iii) Subroutine call and return instructions
iv) Bit manipulation instructions
Ans.

i) Shift Instructions: Shift instructions are a type of computer instruction that allows the computer system to move data within a register or memory location by a certain number of bit positions to the left or right. These instructions can be used to perform operations such as multiplication or division by powers of two or to manipulate data in binary form. The shift instructions can be classified into two types: logical shift and arithmetic shift. In logical shift, the bits are shifted in the specified direction and the empty bits are filled with zeros. In arithmetic shift, the most significant bit is preserved during a right shift operation to maintain the sign of the number being shifted.

ii) Branch and Jump Instructions: Branch and Jump instructions are used to change the order of execution of instructions in a computer system. A branch instruction causes the program to jump to a different location in the program, while a jump instruction causes the program to jump to an absolute location in the program. These instructions are used in programming languages to implement control flow statements such as if-else statements, loops, and function calls.

iii) Subroutine call and return instructions: Subroutine call and return instructions are used to implement functions and procedures in programming languages. A subroutine call instruction transfers control to a subroutine, which is a sequence of instructions that performs a specific task. The return instruction is used to transfer control back to the calling program after the subroutine has completed its task. The return instruction can also pass values back to the calling program.

iv) Bit manipulation instructions: Bit manipulation instructions are used to manipulate individual bits in a register or memory location. These instructions can be used to set, clear, or toggle individual bits or groups of bits, and to perform logical operations such as AND, OR, and XOR on the bits. Bit manipulation instructions are often used in low-level programming tasks such as device drivers and embedded systems.

(b) Assume that a computer has 16-bit memory words. The size of instructions on this machine is 32 bits, which consists of an 8-bit operation code, a 4-bit addressing mode, one4-bit register address and one 16-bit memory address. Some of the opcodes and addressing modes for this machine are shown below:

Opcode: 10001111 – Add the content of the operand into the Accumulator register
Opcode: 10001110 - Subtract the content of the operand from the Accumulator register
Addressing Mode: 0011 Direct addressing using a memory address
Addressing Mode: 0100 Base Register Addressing; where the 4-bit register address is the address of the register and the 16-bit memory address contains the offset from the base register.

Show the content of memory containing two add instructions one using addressing mode 0011 and the other using addressing mode 0100 along with operands and operand addresses. You should show instructions, operandsand memory addresses in the diagram.
Ans.

To show the content of memory containing two add instructions, we need to provide the opcode, addressing mode, register address, memory address, and operand value for each instruction.
For the first instruction using addressing mode 0011, we have:

- Opcode: 10001111
- Addressing mode: 0011
- Register address: 0000 (Assuming that the Accumulator register is 0)
- Memory address: 0000000000001001 (Assuming that the operand is stored in memory location 9)
- Operand value: 0010110101011011

Thus, the content of memory location 9 would be:

Address	Memory Content
9	0010110101011011

For the second instruction using addressing mode 0100, we have:

- Opcode: 10001111
- Addressing mode: 0100
- Register address: 0000 (Assuming that the Accumulator register is 0)
- Memory address: 0000000000010110 (Assuming that the base register is 22 and the offset is 20)
- Operand value: 1110000111010011

Thus, the content of memory location (22 + 20) = 42 would be:

Address	Memory Content
42	1110000111010011

The memory diagram for the two instructions:

Address	Memory Content
9	0010110101011011
10	(unused)
11	(unused)
12	(unused)
13	(unused)
14	(unused)
15	(unused)
16	(unused)
17	(unused)
18	(unused)
19	(unused)
20	(unused)
21	(unused)
22	(unused)
23	(unused)
24	(unused)
25	(unused)
26	(unused)
27	(unused)
28	(unused)
29	(unused)
30	(unused)
31	(unused)
32	(unused)
33	(unused)
34	(unused)
35	(unused)
36	(unused)
37	(unused)
38	(unused)
39	(unused)
40	(unused)
41	(unused)
42	1110000111010011
43	(unused)
44	(unused)
45	(unused)

Address	Memory Content
46	(unused)
47	(unused)
48	(unused)
49	(unused)
50	(unused)
51	(unused)
52	(unused)
53	(unused)
54	(unused)
55	(unused)
56	(unused)
57	(unused)
58	(unused)
59	(unused)
60	(unused)
61	(unused)
62	(unused)
63	(unused)

(c) With the help of micro-operations, explain how an instruction that uses direct addressing mode can be fetched, decoded and executed by a machine. You April follow the conventions as given in Unit 10 of Block 3.

Ans.

When an instruction using direct addressing mode is executed by a machine, the following micro-operations are performed:

1. Fetch the instruction from memory:
   - The program counter (PC) is used to access the memory location where the instruction is stored.
   - The instruction is transferred from memory to the instruction register (IR) using the memory data register (MDR).
2. Decode the instruction:
   - The opcode and addressing mode fields of the instruction are extracted from the IR.
   - The control unit generates the appropriate control signals based on the opcode field.
3. Calculate the operand address:
   - If the addressing mode is direct, the operand address is simply the memory address field of the instruction.
   - The address is transferred to the address register (AR) using the MDR.
4. Fetch the operand from memory:
   - The address in AR is used to access the memory location where the operand is stored.
   - The operand is transferred from memory to the MDR.
5. Execute the instruction:
   - The ALU performs the specified operation on the accumulator and the operand from the MDR.
   - The result is stored back in the accumulator.
6. Update the program counter:
   - The PC is incremented to point to the next instruction in memory.

Each of these micro-operations is controlled by the control unit, which generates the appropriate control signals for each step. The micro-operations are executed in sequence, allowing the machine to fetch, decode, and execute instructions using direct addressing mode.

(d) What is a micro-instruction? Explain the organisation of control unit. How microinstruction-based control unit work? Explain with the help of a diagram.

Ans.

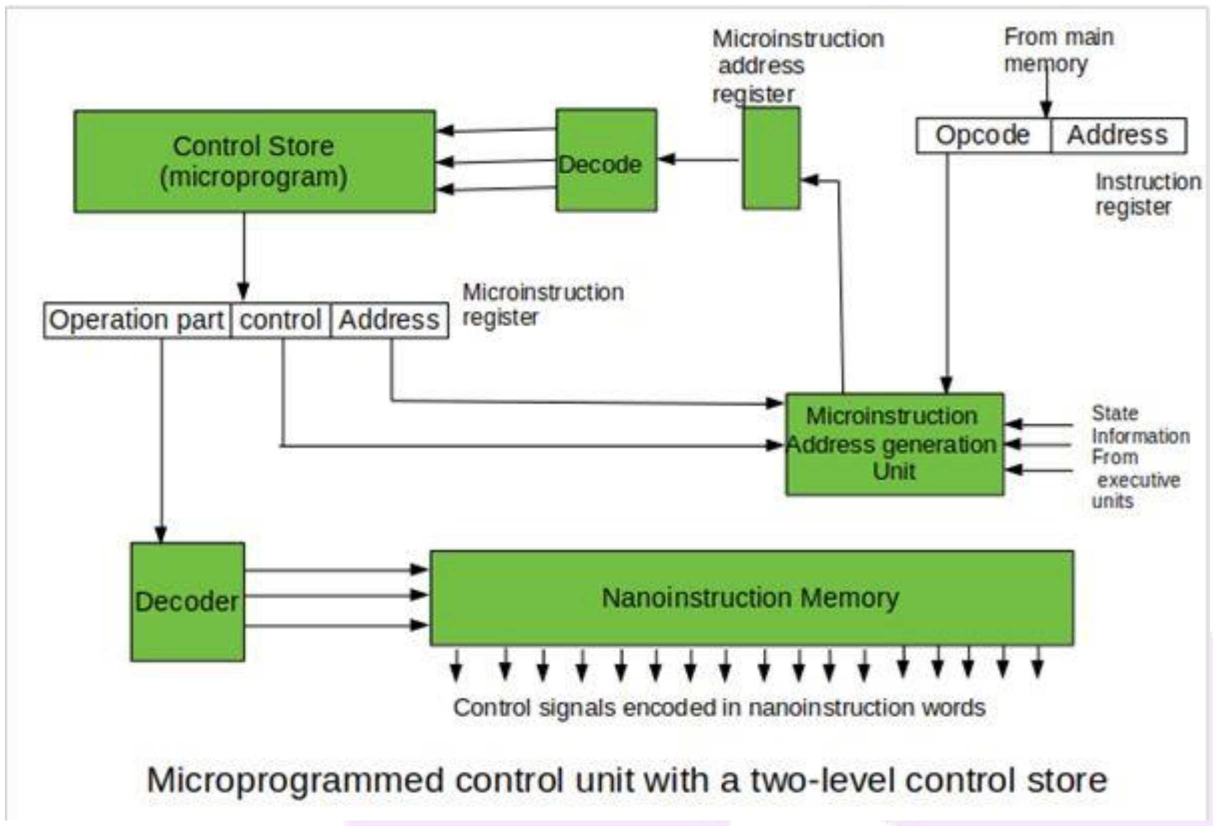
A micro-instruction is a low-level instruction used by a control unit to control the operations of a computer's CPU. It is a machine-level instruction that specifies the individual steps that must be taken by the CPU to execute a single instruction in a computer program.

The control unit is responsible for generating the control signals that direct the CPU to execute the appropriate micro-instructions. The organization of a control unit typically consists of three main components: the instruction register (IR), the sequencing logic, and the control memory.

The IR is a register that holds the current instruction being executed by the CPU. The sequencing logic determines the order in which the micro-instructions are executed, and the control memory contains the micro-instructions themselves.

A microinstruction-based control unit works by breaking down a single machine-level instruction into a series of micro-instructions that the CPU can execute. The control unit retrieves the micro-instructions from the control memory based on the instruction currently in the

IR. Each micro-instruction specifies a single operation that the CPU must perform, such as reading data from a register, performing an arithmetic operation, or storing data to memory.
Here's a diagram that illustrates how a microinstruction-based control unit works:


Microinstruction address register
From main memory
Control Store (microprogram)
Decode
Opcode
Address
Instruction register
Operation part
control
Address
Microinstruction register
Microinstruction Address generation Unit
State Information From executive units
Decoder
Nanoinstruction Memory
Control signals encoded in nanoinstruction words
Microprogrammed control unit with a two-level control store

The control unit receives the instruction code from the instruction register, which is used to access the appropriate micro-instructions from the control memory. The sequencing logic determines the order in which the micro-instructions are executed and generates the appropriate control signals to perform each operation.
Each micro-instruction specifies a single operation that the CPU must perform, such as reading data from a register, performing an arithmetic operation, or storing data to memory. The control signals generated by the sequencing logic activate the appropriate circuits in the CPU to perform each operation.
In summary, a microinstruction-based control unit works by breaking down complex machine-level instructions into a series of simple micro-instructions that can be executed by the CPU. The control unit retrieves the micro-instructions from the control memory based on the instruction in the IR and generates the appropriate control signals to execute each operation.

(e) Explain how the circular buffer in RISC helps in implementation of efficient procedure calls. Also, explain the concept of RISC pipelining and optimisation of RISC pipelining.
Ans.
Circular buffer in RISC: One of the major advantages of RISC (Reduced Instruction Set Computing) architecture is its support for efficient procedure calls. RISC architectures typically use a circular buffer for storing the return addresses of procedures that have been called. This circular buffer is known as a "call stack" or "stack."

When a procedure is called, the address of the instruction following the call instruction is pushed onto the call stack. This address is used later to return to the instruction after the call instruction. When the called procedure is finished executing, the return address is popped from the call stack and used to return to the calling procedure.

The circular buffer is used because it allows for efficient allocation and deallocation of memory for procedure calls. The buffer can be pre-allocated with a fixed size, and as procedure calls are made, the return addresses are simply pushed onto the buffer. When the buffer is full, the oldest return address is overwritten with the newest one, effectively wrapping around to the beginning of the buffer.
RISC Pipelining: RISC architectures are designed to support pipelining, which is a technique used to increase processor efficiency by overlapping the execution of multiple instructions. In a pipelined architecture, the processor is divided into a series of stages, and each stage performs a different operation on the instruction being executed.

For example, a typical RISC pipeline might include the following stages:
- Instruction Fetch (IF): Fetch the next instruction from memory.
- Instruction Decode (ID): Decode the instruction and retrieve any necessary operands.
- Execute (EX): Perform the operation specified by the instruction.
- Memory Access (MEM): Access memory if necessary.
- Write Back (WB): Write the result of the operation back to a register.

Optimisation of RISC Pipelining: There are several techniques used to optimise the performance of RISC pipelining. One such technique is instruction-level parallelism, which involves executing multiple instructions in parallel within the pipeline.

Another technique is branch prediction, which is used to reduce the impact of branch instructions on pipeline performance. Branch instructions can cause the pipeline to stall while the target address of the branch is determined. Branch prediction involves predicting the target address of the branch before it is executed, so that the pipeline can continue to execute instructions while the target address is being determined.

Instruction reordering is also used to optimise RISC pipelining. By reordering instructions, the pipeline can be filled with instructions that do not depend on the results of previous instructions, allowing for more efficient execution.

Finally, forwarding is a technique used to reduce pipeline stalls caused by data dependencies. Forwarding involves bypassing the pipeline stages and directly forwarding data between stages to eliminate the need for a stall.

In summary, the circular buffer in RISC architectures helps in the implementation of efficient procedure calls by using a pre-allocated buffer to store return addresses, while RISC pipelining is a technique used to increase processor efficiency by overlapping the execution of multiple instructions. Optimisations such as instruction-level parallelism, branch prediction, instruction reordering, and forwarding are used to further improve the performance of RISC pipelining.

Question 4. (Covers Block 4) (5 marks each × 4 parts =20 Marks)

(a) Explain the structure of 8086 micro-processor. What are segment registers? Explain how segment registers can be used for computation of physical address in 8086 micro-processor.

Ans.

The Intel 8086 microprocessor is an 16-bit microprocessor, which was first introduced in 1978. It has a 20-bit address bus and can access up to 1MB of memory. It is composed of two main units: the Execution Unit (EU) and the Bus Interface Unit (BIU).

The BIU is responsible for fetching instructions and data from memory and transferring them to the EU. It contains the Instruction Pointer (IP) register, which holds the address of the next instruction to be executed. The BIU also contains four segment registers: Code Segment (CS), Data Segment (DS), Stack Segment (SS), and Extra Segment (ES). These segment registers are used to compute the physical address of the data or instructions being accessed by the processor.

The EU is responsible for executing instructions. It contains general purpose registers such as AX, BX, CX, and DX, which can be used for arithmetic and logical operations. It also contains index and pointer registers such as SI, DI, BP, and SP, which are used for memory addressing.

Segment registers in 8086 microprocessor are 16-bit registers that contain a segment address, which is added to the offset address of a memory location to compute the physical address. The physical address is computed using the following formula:

Physical Address = Segment Address * 16 + Offset Address

For example, if the DS register contains the segment address 0x1234 and an instruction is trying to access data at offset 0x5678, then the physical address would be computed as:
Physical Address = 0x1234 * 16 + 0x5678 = 0x1F678

Segment registers are used to allow the 8086 to address up to 1MB of memory, even though it only has a 20-bit address bus. The segment registers are also used to implement memory protection, since each segment can be assigned different access permissions.

In summary, segment registers in the 8086 microprocessor are used to compute the physical address of the data or instructions being accessed by the processor. They allow the 8086 to address up to 1MB of memory and implement memory protection.

(b) How does 8086 microprocessor handle interrupts? Explain with the help of an example. Write a program using 8086 assembly language to output a string "Computer Organisation includes assembly programming"

Ans.

The 8086 microprocessor can handle interrupts through the Interrupt Vector Table (IVT) and Interrupt Service Routine (ISR). The IVT is a table of addresses located in the first 1KB of memory, which contains the addresses of the ISR for each interrupt type. When an interrupt occurs, the 8086 saves the current state of the program and jumps to the ISR specified in the IVT.

Here is an example of how the 8086 microprocessor handles interrupts:

1. A device sends an interrupt signal to the 8086 microprocessor.
2. The 8086 completes the current instruction and saves the state of the program, including the values of registers and the IP.
3. The 8086 loads the interrupt number from the interrupt signal into the IP and sets the interrupt flag.
4. The 8086 jumps to the address specified in the IVT for the interrupt number and begins executing the ISR.
5. When the ISR is finished, it clears the interrupt flag and returns to the saved state of the program.

Here is an example program in 8086 assembly language to output a string "Computer Organisation includes assembly programming" using the DOS interrupt:

```
.MODEL SMALL
.STACK 100h

.DATA
   MESSAGE DB 'Computer Organisation includes assembly programming$'

.CODE
.STARTUP
   MOV AX, @DATA
   MOV DS, AX  ; Initialize the data segment

   LEA DX, MESSAGE ; Load the address of the message into DX
   MOV AH, 9 ; Set the function code for outputting a string
   INT 21h ; Call the DOS interrupt to output the string

.EXIT
END
```

In this program, we define a string MESSAGE and load its address into the DX register using the LEA instruction. We then set the AH register to 9, which is the function code for outputting a string using the DOS interrupt. Finally, we call the DOS interrupt using INT 21h to output the string.

When this program is executed, it outputs the string "Computer Organisation includes assembly programming" to the console.

(c) Write a program in 8086 assembly language, which converts a four-digitpacked BCD number to ASCII digits and then prints them. Explain each step of the program.The program should take the BCD value from the memory.

Ans.

A program in 8086 assembly language to convert a four-digit packed BCD number to ASCII digits and print them:

```
.MODEL SMALL
.STACK 100h

.DATA
   BCD_NUM DB 12h ; Packed BCD number to convert and print
   ASCII_NUM DB 4 DUP ('$') ; Buffer to store ASCII digits

.CODE
.STARTUP
   MOV AX, @DATA
   MOV DS, AX  ; Initialize the data segment
```

```
  ; Convert BCD to ASCII
  MOV AL, BCD_NUM
  AAM ; Divide AL by 10 to separate tens and ones digits
  ADD AX, 3030h ; Convert tens and ones digits to ASCII
  MOV BYTE PTR ASCII_NUM[0], AL ; Store ones digit in buffer
  MOV BYTE PTR ASCII_NUM[1], AH ; Store tens digit in buffer

  MOV AL, BCD_NUM+1
  AAM ; Divide AL by 10 to separate hundreds and thousands digits
  ADD AX, 3030h ; Convert hundreds and thousands digits to ASCII
  MOV BYTE PTR ASCII_NUM[2], AL ; Store thousands digit in buffer
  MOV BYTE PTR ASCII_NUM[3], AH ; Store hundreds digit in buffer

  ; Print ASCII digits
  LEA DX, ASCII_NUM ; Load address of buffer into DX
  MOV AH, 9 ; Set the function code for outputting a string
  INT 21h ; Call the DOS interrupt to output the string

.EXIT
END
```

Explanation of each step of the program:

1. Define the BCD_NUM variable to hold the packed BCD number to convert and the ASCII_NUM buffer to store the converted ASCII digits.
2. Initialize the data segment.
3. Load the packed BCD number from memory into the AL register.
4. Divide the value in AL by 10 using the AAM instruction, which separates the tens and ones digits and stores them in the AL and AH registers, respectively.
5. Add 3030h to the AX register to convert the tens and ones digits to their ASCII values.
6. Store the ones digit in the first byte of the ASCII_NUM buffer and the tens digit in the second byte.
7. Load the next byte of the packed BCD number into the AL register and repeat steps 4-6 for the hundreds and thousands digits.
8. Print the converted ASCII digits stored in the ASCII_NUM buffer using the DOS interrupt.

When this program is executed with a packed BCD number stored in the BCD_NUM variable, it will convert the number to ASCII digits and print them to the console.

(d) Explain the following in the context of parallel processing:
(i) Arithmetic pipelining
(ii) Array processing
(iii) Inter-processor arbitration
(iv) Cache coherence
Ans.

(i) Arithmetic pipelining: Arithmetic pipelining is a technique used in parallel processing where arithmetic operations are divided into multiple stages, with each stage performing a specific operation on the data. The input data is passed through each stage of the pipeline, with each stage performing its operation and passing the result to the next stage. This allows multiple operations to be performed simultaneously, resulting in faster processing times.

(ii) Array processing: Array processing is a type of parallel processing where data is processed in parallel across multiple processors. In this technique, a large data set is divided into smaller parts, and each part is processed by a separate processor. This allows for faster processing times and is commonly used in scientific and engineering applications where large data sets need to be processed.

(iii) Inter-processor arbitration: Inter-processor arbitration is the process of resolving conflicts between processors in a parallel processing system. In this technique, multiple processors may try to access a shared resource, such as memory or I/O devices, at the same time. Inter-processor arbitration is used to ensure that only one processor at a time can access the shared resource, preventing conflicts and ensuring that the system operates correctly.

(iv) Cache coherence: Cache coherence is the process of ensuring that all processors in a parallel processing system have consistent views of shared data stored in caches. In a multi-processor system, each processor may have its own cache, and if one processor modifies shared data stored in its cache, the other processors may not immediately see the change. Cache coherence protocols ensure that all processors see the same values for shared data, which is necessary for correct operation of parallel processing systems.