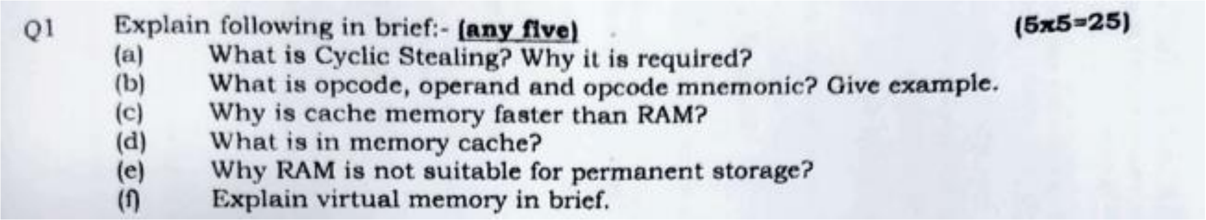


Q1 Explain following in brief:- **(any five)** (5x5=25)

(a) What is Cyclic Stealing? Why it is required?

(b) What is opcode, operand and opcode mnemonic? Give example.

(c) Why is cache memory faster than RAM?

(d) What is in memory cache?

(e) Why RAM is not suitable for permanent storage?

(f) Explain virtual memory in brief.

(a)

Cycle stealing is a technique used in computer architecture to improve the efficiency of the CPU by allowing other devices to use the system bus during certain idle cycles of the CPU.

In a computer system, the CPU and other devices communicate with each other through a shared system bus. When a device needs to transfer data to or from memory, it must request control of the system bus from the CPU. The CPU then suspends its current operation and relinquishes control of the system bus to the requesting device.

Cycle stealing allows certain devices, such as an input/output (I/O) controller, to take control of the system bus during idle cycles of the CPU, without requiring an explicit request for control. This allows the I/O controller to transfer data to or from memory without interrupting the CPU's processing of instructions.

Cycle stealing is required to improve the efficiency of the system, as it allows the CPU and I/O devices to work in parallel, without requiring the CPU to wait for the completion of an I/O transfer. This can improve overall system performance by reducing the time that the CPU spends waiting for data transfers to complete, and by allowing I/O devices to transfer data to or from memory more quickly.

Overall, cycle stealing helps to improve the efficiency of the computer system, which allows it to perform more work in less time.

(b)

In computer science, an opcode (short for "operation code") is a code that represents a specific instruction for a processor or microcontroller to perform. An opcode specifies the operation to be performed, and it is typically followed by one or more operands, which are the data or memory locations on which the operation is to be performed.

The operand specifies the specific data or memory location on which the operation is to be performed. Depending on the instruction, an operand can be a register, a memory address, a constant value, or some other data.

An opcode mnemonic is a human-readable representation of the opcode, which is easier to remember and use than the numeric value of the opcode. Mnemonics are used to make assembly language programming more intuitive and accessible.

For example, let's consider the instruction "ADD R1, R2", which adds the contents of register R2 to the contents of register R1 and stores the result in R1. In this example, "ADD" is the opcode mnemonic, and "R1" and "R2" are the operands. The corresponding opcode value for "ADD" might be 0x01, and the register values would be encoded in binary form as well.

In summary, an opcode is a code that represents a specific instruction for a processor to perform, an operand is the data or memory location on which the operation is to be performed, and an opcode mnemonic is a human-readable representation of the opcode.

(c)

Cache memory is faster than RAM for several reasons:

1. Proximity to the CPU: Cache memory is located on the CPU or very close to it, which means that it can be accessed much more quickly than RAM, which is typically located on a separate chip or module. The closer the memory is to the CPU, the less time it takes for the CPU to retrieve data from it.
2. Size: Cache memory is typically smaller than RAM, which means that it can be accessed more quickly because the CPU has to search through fewer bits to find the data it needs.
3. Organization: Cache memory is organized in a way that is optimized for the types of data that the CPU is likely to need. For example, modern CPUs use a hierarchy of cache levels, with each level optimized for different types of data and access patterns.
4. Cost: Cache memory is more expensive than RAM, which means that it is made from higher-quality components that can operate at higher speeds.
5. Caching algorithms: The algorithms used to manage cache memory are designed to keep the most frequently used data in the cache, which means that the CPU is more likely to find the data it needs in the cache and less likely to have to access slower RAM.

All of these factors combine to make cache memory much faster than RAM. However, cache memory is also more expensive and can hold less data than RAM, so it is typically used to store only the most frequently accessed data, while less frequently accessed data is stored in RAM.

(d)

In computer architecture, cache memory is a small, high-speed memory that stores frequently accessed data and instructions. The data stored in cache memory is a subset of the data stored in main memory, and is usually organized in blocks or lines.

The specific data stored in cache memory depends on how the cache is designed and how it is used by the computer system. For example, in a CPU cache, the cache stores recently used instructions and data from main memory that the processor is likely to use again in the near future. This can include instructions from the program being executed, data that is frequently accessed, or other data that is needed by the processor.

(e)

RAM (Random Access Memory) is a type of volatile memory that is not suitable for permanent storage of data. This is because RAM loses its contents when power is turned off, making it unsuitable for long-term data storage.

Another reason why RAM is not suitable for permanent storage is that it is relatively expensive compared to other storage technologies. The cost per unit of storage in RAM is much higher than in other types of storage, making it impractical for storing large amounts of data.

In summary, RAM is a fast, volatile memory that is used for temporary storage of data while the computer is running. It is not suitable for permanent storage of data because it loses its contents when power is turned off, and it is relatively expensive compared to other storage technologies.

(f)

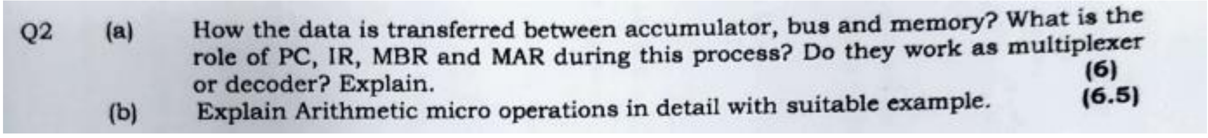
In a computer with virtual memory, the operating system maps a portion of the program's virtual address space to physical memory and another portion to a hard disk or solid-state drive. The portion of the program's address space that is in physical memory is called the resident set, while the portion that is on disk is called the paging file or swap file.

When the program accesses memory that is not currently in the resident set, a page fault occurs. The operating system then retrieves the required page from the paging file and places it in physical memory. This process is called paging, and it is transparent to the program, which can continue to access memory as if it were all present in physical memory.

The advantage of virtual memory is that it allows a computer to run larger programs or multiple programs simultaneously without running out of physical memory. It also provides memory protection, which prevents one program from accessing the memory of another program or the operating system.

However, virtual memory also has some disadvantages. Paging can cause performance problems if the computer has to swap a lot of data between physical memory and the paging file. Additionally, the hard disk or solid-state drive used for virtual memory is much slower than physical memory, so access to data stored on disk can be much slower than access to data stored in RAM.

In summary, virtual memory is a computer architecture technique that allows a computer to use more memory than it physically has available by using a paging file or swap file on a hard disk or solid-state drive. It allows larger programs or multiple programs to run simultaneously without running out of physical memory, but can also cause performance problems if the computer has to swap a lot of data between physical memory and the paging file.



Q2 (a) How the data is transferred between accumulator, bus and memory? What is the role of PC, IR, MBR and MAR during this process? Do they work as multiplexer or decoder? Explain. (6)

(b) Explain Arithmetic micro operations in detail with suitable example. (6.5)

(a)

The transfer of data between the accumulator, bus, and memory in a computer system is a fundamental operation that takes place during the execution of a program. The process typically involves the following steps:

1. The CPU sends a memory address to the Memory Address Register (MAR) to indicate where in memory the data is located.
2. The CPU then sends a read or write signal to the memory using the Control Bus, indicating whether it wants to read or write data.
3. The memory reads or writes the data from or to the memory location specified by the address in the MAR and sends it back to the CPU through the Data Bus.
4. The data is then temporarily stored in the Memory Buffer Register (MBR).
5. The CPU then sends the data to the accumulator or other registers for processing, typically through the Arithmetic and Logic Unit (ALU), using the Data Bus.
6. The Program Counter (PC) keeps track of the address of the next instruction to be executed, and the Instruction Register (IR) holds the instruction that is currently being executed by the CPU.

During this process, the MAR and MBR work as a multiplexer and decoder, respectively. The MAR selects the memory location where data is to be read or written, while the MBR temporarily stores the data before it is processed by the CPU. The PC and IR work together to keep track of the current instruction being executed by the CPU, and the next instruction to be executed.

In summary, the transfer of data between the accumulator, bus, and memory in a computer system involves several key components, including the MAR, MBR, PC, IR, and Control Bus. These components work together to ensure that data is properly read from or written to memory and processed by the CPU, with the MAR and MBR working as a multiplexer and decoder, respectively.

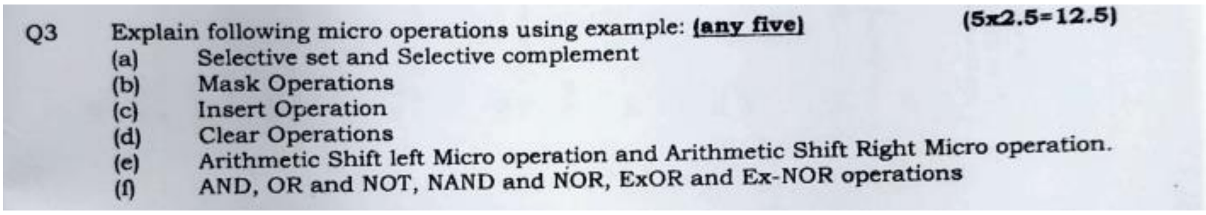
(b)

In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

The basic Arithmetic Micro-operations are classified in the following categories:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Shift

Symbolic Representation	**Description**
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one



Q3 Explain following micro operations using example: **(any five)** (5x2.5=12.5)

(a) Selective set and Selective complement

(b) Mask Operations

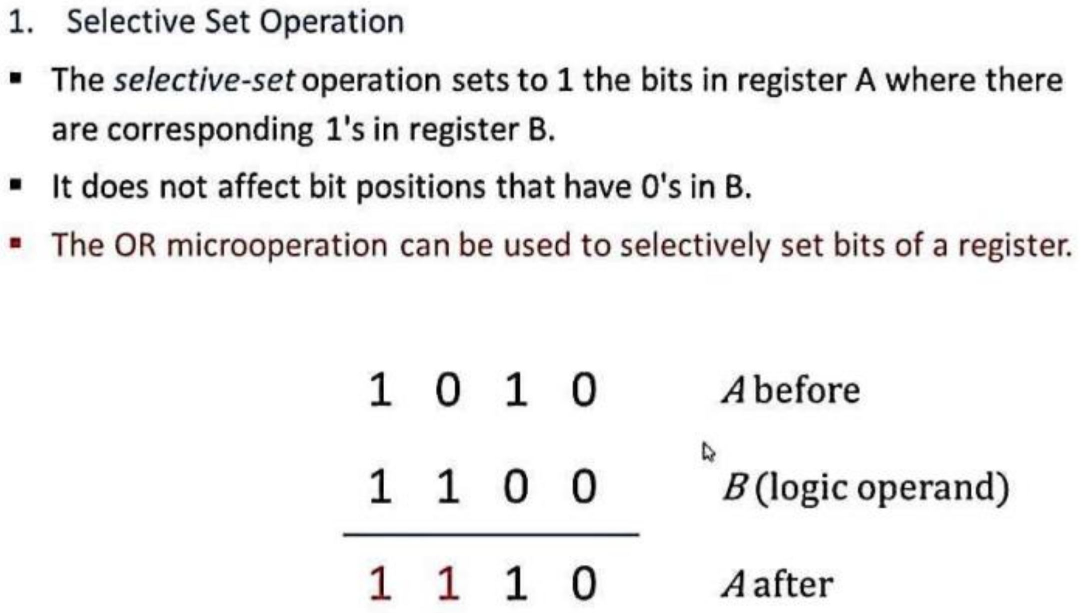
(c) Insert Operation

(d) Clear Operations

(e) Arithmetic Shift left Micro operation and Arithmetic Shift Right Micro operation.

(f) AND, OR and NOT, NAND and NOR, ExOR and Ex-NOR operations

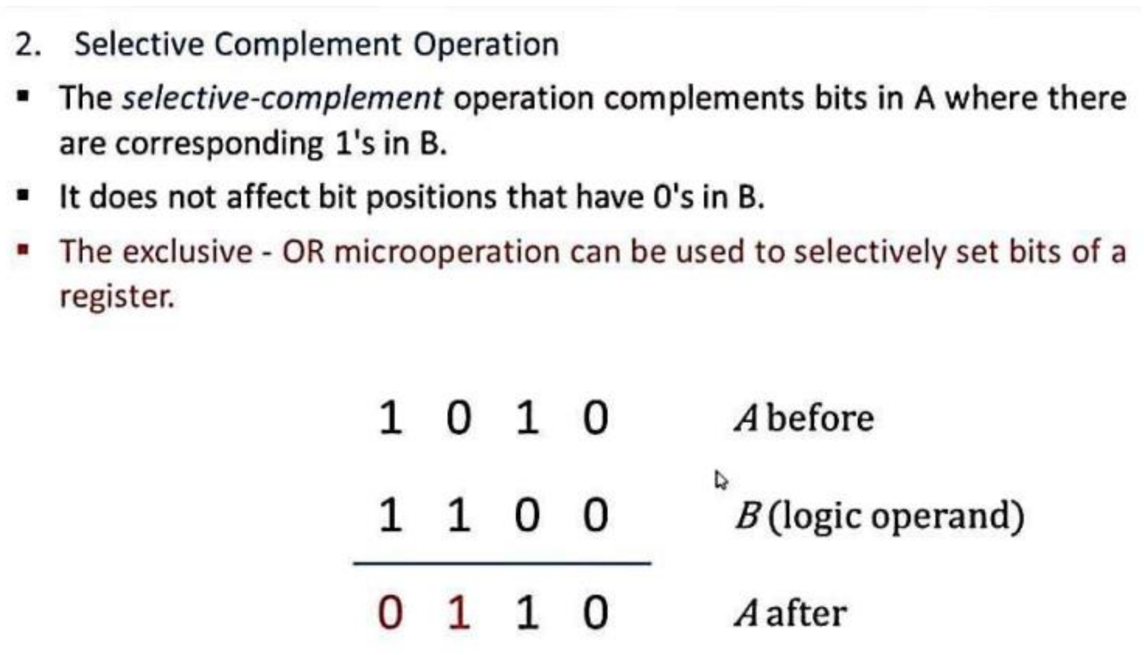
(a)



1. Selective Set Operation

- The *selective-set* operation sets to 1 the bits in register A where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- The OR microoperation can be used to selectively set bits of a register.

$$\begin{array}{cccc l} 1 & 0 & 1 & 0 & A\text{ before} \\ 1 & 1 & 0 & 0 & B\text{ (logic operand)} \\ \hline 1 & 1 & 1 & 0 & A\text{ after} \end{array}$$

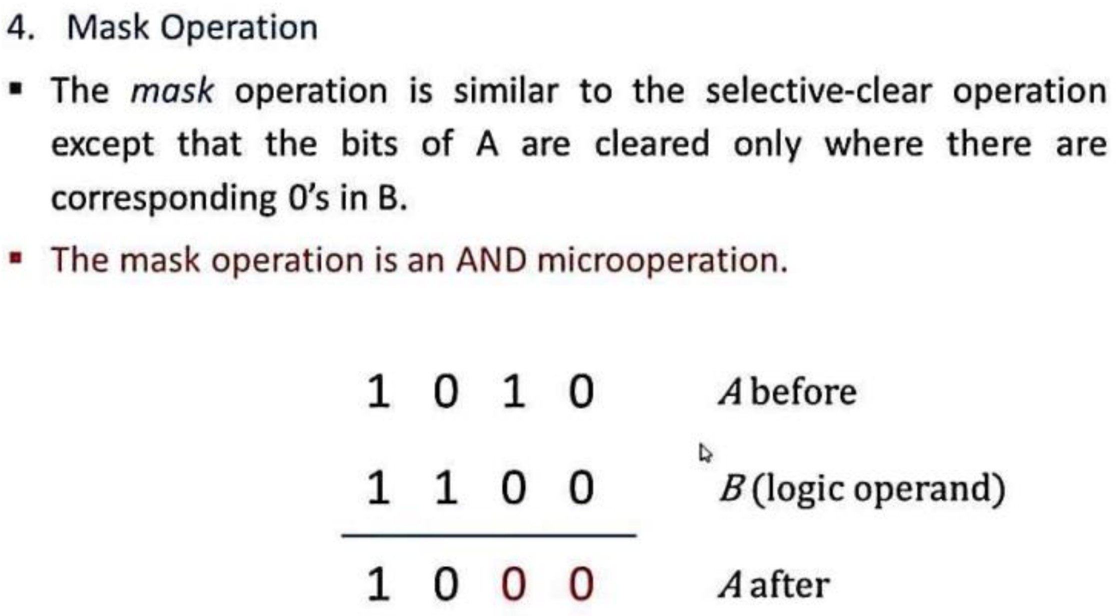


2. Selective Complement Operation

- The *selective-complement* operation complements bits in A where there are corresponding 1's in B.
- It does not affect bit positions that have 0's in B.
- The exclusive - OR microoperation can be used to selectively set bits of a register.

1	0	1	0	*A* before
1	1	0	0	*B* (logic operand)
0	1	1	0	*A* after

(b)

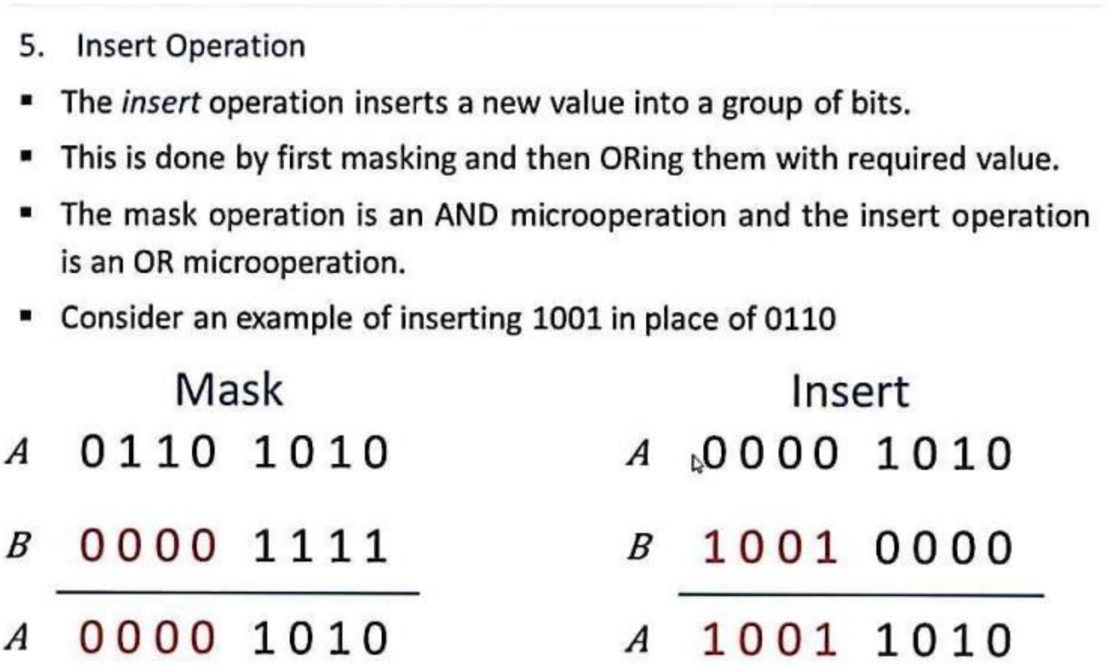


4. Mask Operation

- The *mask* operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B.
- The mask operation is an AND microoperation.

1	0	1	0	*A* before
1	1	0	0	*B* (logic operand)
1	0	0	0	*A* after

(c)

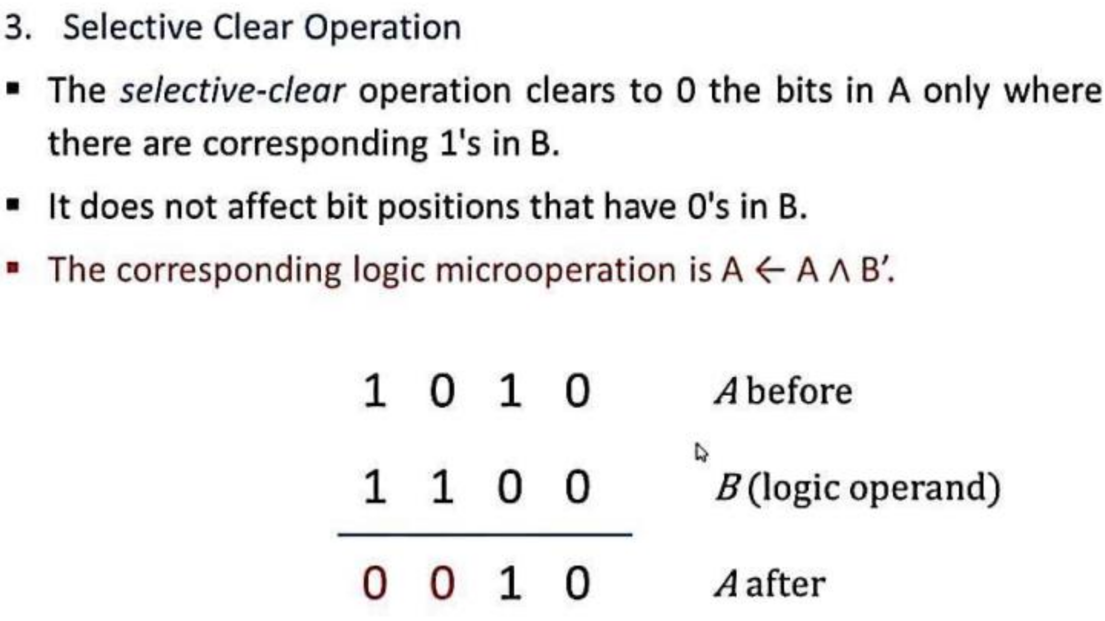


5. Insert Operation

- The *insert* operation inserts a new value into a group of bits.
- This is done by first masking and then ORing them with required value.
- The mask operation is an AND microoperation and the insert operation is an OR microoperation.
- Consider an example of inserting 1001 in place of 0110

	Mask		Insert
A	0110 1010	*A*	0000 1010
B	0000 1111	*B*	1001 0000
A	0000 1010	*A*	1001 1010

(d)



3. Selective Clear Operation

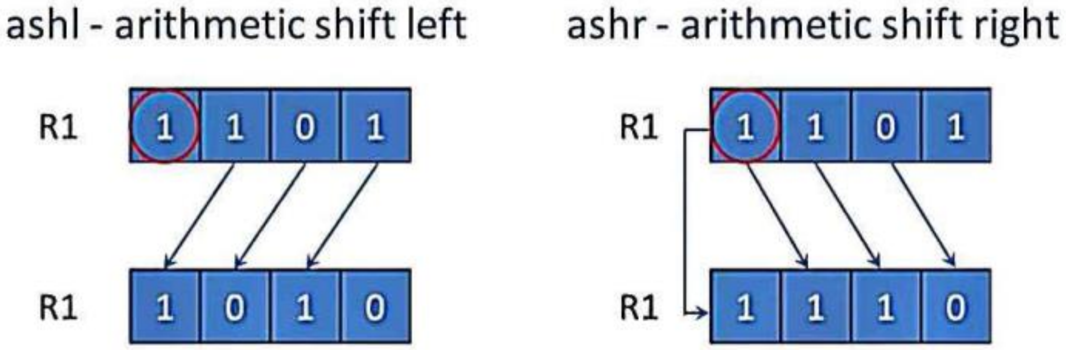
- The *selective-clear* operation clears to 0 the bits in A only where there are corresponding 1's in B.
- It does not affect bit positions that have 0's in B.
- The corresponding logic microoperation is A ← A ∧ B′.

1	0	1	0	*A* before
1	1	0	0	*B* (logic operand)
0	0	1	0	*A* after

(e)

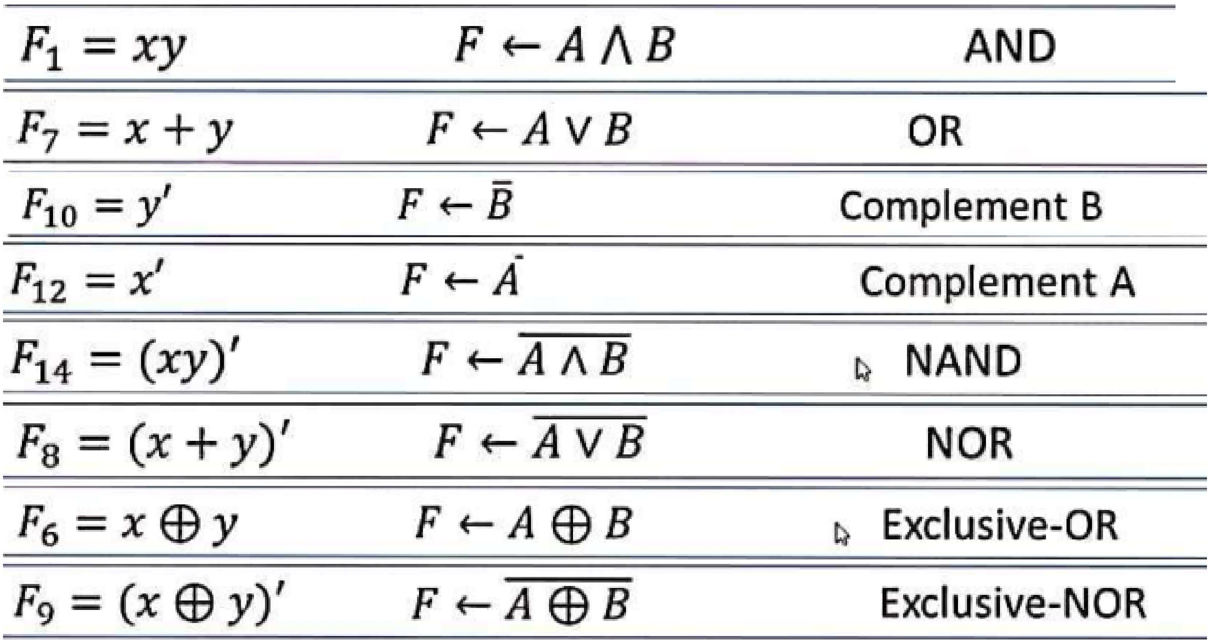
3. Arithmetic Shift

- An *arithmetic shift* is a micro-operation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.

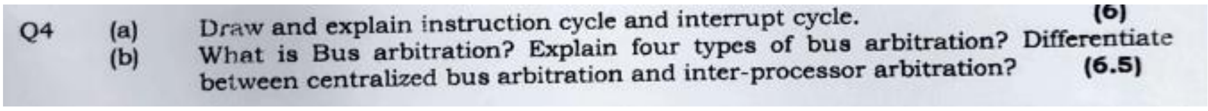

ashl - arithmetic shift left

ashr - arithmetic shift right

(f)



$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR



Q4 (a) Draw and explain instruction cycle and interrupt cycle. (6)

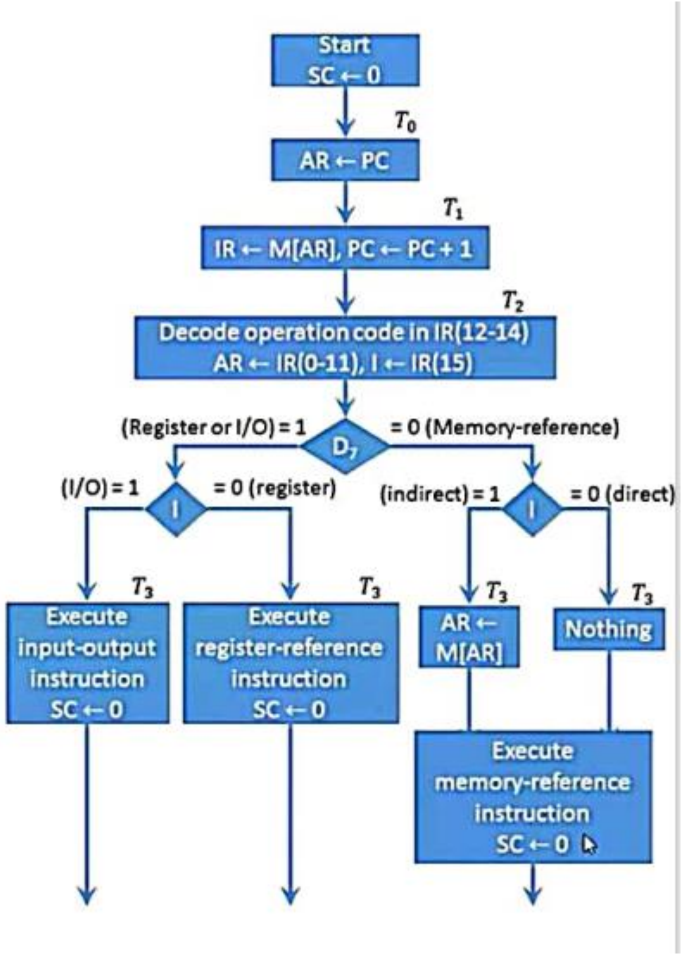
(b) What is Bus arbitration? Explain four types of bus arbitration? Differentiate between centralized bus arbitration and inter-processor arbitration? (6.5)

(a)

The instruction cycle, also known as the fetch-decode-execute cycle, is the basic sequence of steps that a central processing unit (CPU) goes through to fetch, decode, and execute an instruction from memory. The instruction cycle is repeated for each instruction in a program, and it is the fundamental operation that drives the execution of computer programs. The steps in the instruction cycle are as follows:

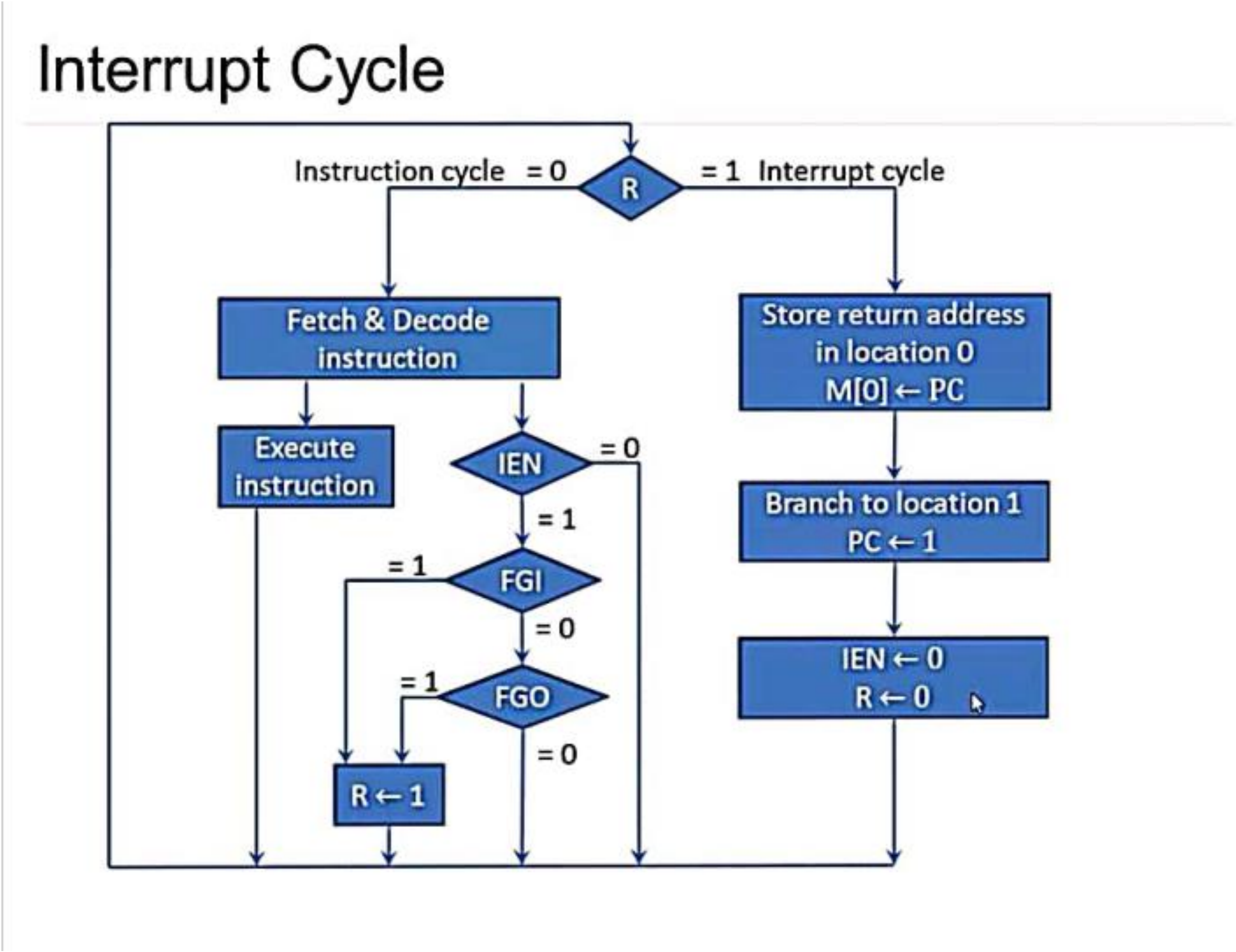
1. Fetch: The CPU fetches the next instruction from memory into its instruction register.
2. Decode: The CPU decodes the instruction by determining the operation to be performed and the operands that are involved.
3. Execute: The CPU performs the operation specified by the instruction, using the operands from registers or memory.
4. Store: The CPU stores the result of the operation back into a register or memory.

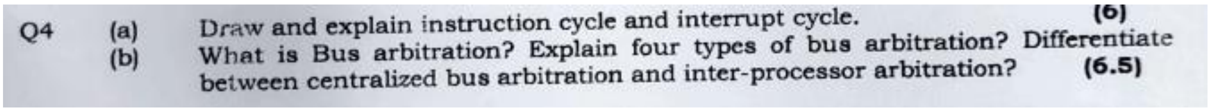
After the execution of an instruction, the CPU increments the program counter to point to the next instruction in memory, and the instruction cycle starts again.



The steps in the interrupt cycle are as follows:

1. Interrupt Request: An interrupt request is generated by an I/O device or another hardware component to request the CPU's attention.
2. Save State: The CPU saves the current state of the program, including the program counter, register values, and other important data, to a dedicated area in memory called the interrupt stack.
3. Interrupt Handler Routine: The CPU switches to the interrupt handler routine, which is a separate program that handles the interrupt.
4. Interrupt Service: The interrupt handler routine performs the necessary operations to service the interrupt, such as transferring data between the I/O device and memory.
5. Restore State: After the interrupt is serviced, the CPU restores the saved state of the interrupted program, including the program counter and register values.
6. Resume Execution: The CPU resumes the execution of the interrupted program from where it left off before the interrupt occurred.


Interrupt Cycle


Q4 (a) Draw and explain instruction cycle and interrupt cycle. (6)
(b) What is Bus arbitration? Explain four types of bus arbitration? Differentiate between centralized bus arbitration and inter-processor arbitration? (6.5)

(b)

Bus arbitration is the process of determining which device on a shared bus can access the bus at a given time when multiple devices try to access the bus simultaneously. Bus arbitration is necessary to avoid conflicts and ensure that only one device accesses the bus at a time, thereby preventing data corruption and ensuring proper data transfer. There are four types of bus arbitration:

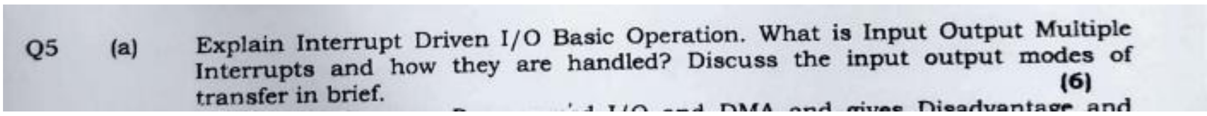
1. Centralized Bus Arbitration: In centralized bus arbitration, a single device, typically a bus controller, is responsible for arbitrating bus access. The bus controller receives requests from all devices on the bus and grants access to one device at a time based on a predetermined priority scheme.
2. Distributed Bus Arbitration: In distributed bus arbitration, each device on the bus is responsible for arbitrating access to the bus. When a device wants to access the bus, it sends a request to all other devices on the bus, and each

device responds with its own priority. The device with the highest priority is granted access to the bus.

3. Request/Grant Bus Arbitration: In request/grant bus arbitration, each device on the bus sends a request for bus access to a centralized controller. The controller then grants access to one device at a time based on a predetermined priority scheme.
4. Time Division Multiplexed Bus Arbitration: In time division multiplexed bus arbitration, the bus is divided into time slots, and each device is granted access to the bus for a predetermined time slot. The time slots are allocated based on a predetermined priority scheme.

The main differences between centralized bus arbitration and inter processor arbitration are:

1. Control: In centralized bus arbitration, a single controller is responsible for managing access to the shared bus. This controller arbitrates requests from all the processors and grants access to the bus. In contrast, interprocessor arbitration involves multiple processors coordinating access to shared resources amongst themselves without the involvement of a central controller.
2. Overhead: Centralized bus arbitration can introduce a significant amount of overhead since all requests must pass through the central controller. In contrast, interprocessor arbitration can be more efficient since processors can coordinate access to resources amongst themselves without the need for a central controller.
3. Scalability: Centralized bus arbitration can become a bottleneck in a system with a large number of processors. As the number of processors increases, the centralized controller can become overwhelmed, leading to delays and contention for access to the shared resource. Interprocessor arbitration is more scalable, since processors can coordinate access amongst themselves without requiring a centralized controller.
4. Complexity: Centralized bus arbitration is generally easier to implement and manage since all requests are handled by a single controller. In contrast, interprocessor arbitration can be more complex since processors must coordinate amongst themselves, and additional mechanisms are required to resolve conflicts.



Q5 (a) Explain Interrupt Driven I/O Basic Operation. What is Input Output Multiple Interrupts and how they are handled? Discuss the input output modes of transfer in brief. (6)

I/O and DMA and gives Disadvantage and

(a)

Input/output (I/O) operations involve transferring data between the CPU and external devices, such as disks, printers, keyboards, and network adapters. Interrupt-driven I/O is a technique that allows the CPU to perform other tasks while waiting for an I/O operation to complete. The following are the basic operations involved in interrupt-driven I/O in computer architecture:

1. Initialization: The first step is to initialize the I/O device and the interrupt mechanism. This involves setting up the device controller, initializing registers and buffers, and enabling interrupts.
2. Issuing the I/O request: The CPU issues an I/O request to the device controller. The request contains the address of the data to be transferred and the type of operation to be performed (read or write).
3. Executing other instructions: While waiting for the I/O operation to complete, the CPU executes other instructions from the program or switches to another task, if available.
4. Interrupt handling: When the I/O operation is completed, the device controller generates an interrupt signal to the CPU. The CPU suspends the current task and transfers control to the interrupt handler, which is a small routine that performs the necessary operations to handle the interrupt. The interrupt handler typically reads or writes data from or to the I/O device, updates status flags, and performs any necessary synchronization with the main program.
5. Resuming the interrupted task: After the interrupt has been handled, the CPU resumes the interrupted task at the point where it was suspended and continues to execute instructions from the program.

Input/output (I/O) multiple interrupts occur when multiple devices generate interrupts at the same time. This can happen when multiple I/O devices complete their operations simultaneously and generate interrupt signals to the CPU. Handling multiple interrupts is important for system performance and efficiency because it allows the CPU to process all the pending interrupts in a timely and efficient manner.

The following are the basic steps involved in handling multiple interrupts in computer architecture:

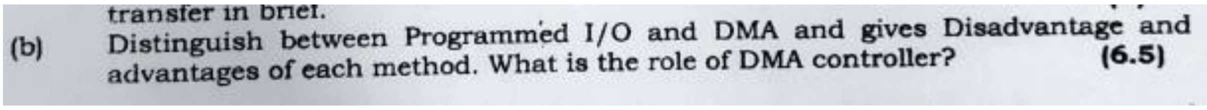
1. Interrupt handling: When an interrupt is generated, the CPU suspends the current task and transfers control to the interrupt handler, which is a small

routine that performs the necessary operations to handle the interrupt. The interrupt handler typically reads or writes data from or to the I/O device, updates status flags, and performs any necessary synchronization with the main program.

2. Interrupt priority: Each interrupt is assigned a priority level, which determines the order in which the interrupts are processed. The CPU maintains a list of pending interrupts, sorted by priority level. The highest priority interrupt is processed first, followed by the lower priority interrupts in order.
3. Interrupt masking: During the interrupt handling process, the CPU disables interrupts of lower priority levels to avoid interrupt nesting and ensure proper synchronization. This is known as interrupt masking.
4. Interrupt acknowledgement: After the interrupt has been handled, the CPU sends an acknowledgement signal to the interrupting device to inform it that the interrupt has been processed and that it can resume normal operation.
5. Interrupt re-entry: If another interrupt occurs while the CPU is processing an interrupt, the CPU suspends the current interrupt handling routine and transfers control to the new interrupt handler, following the same priority and masking rules.

The four-primary input-output modes of transfer:

1. Programmed I/O: In this mode, the CPU communicates with the input/output devices by sending commands and receiving status information. The CPU waits for the device to complete the operation before moving on to the next instruction.
2. Interrupt-driven I/O: In this mode, the CPU initiates the transfer of data to or from the I/O device and then continues to execute other instructions. When the I/O device is ready to transfer data, it interrupts the CPU, and the CPU handles the transfer.
3. Direct memory access (DMA): In this mode, a separate DMA controller manages the transfer of data between the I/O device and the memory, without involving the CPU. The CPU initiates the transfer and then continues executing other instructions while the DMA controller manages the transfer.
4. Channel I/O: In this mode, a dedicated channel controller handles the transfer of data between the I/O device and the memory, without involving the CPU. The channel controller has its own memory and instructions, and it manages the transfer independently of the CPU.



transfer in brief.

(b) Distinguish between Programmed I/O and DMA and gives Disadvantage and advantages of each method. What is the role of DMA controller? (6.5)

1. Programmed I/O:

In programmed I/O, the CPU controls the entire data transfer process. It initiates the transfer, waits for the transfer to complete, and then retrieves the data from the I/O device. The advantages and disadvantages of programmed I/O are as follows:

Advantages:

- Simple and easy to implement
- Useful for small amounts of data transfer
- Can handle various types of I/O devices

Disadvantages:

- CPU utilization is high and can lead to reduced system performance
- Not efficient for large amounts of data transfer
- Wastes CPU cycles during I/O operations

2. Direct Memory Access (DMA):

In DMA, the data transfer process is controlled by a dedicated DMA controller, which manages the transfer independently of the CPU. The CPU sets up the DMA controller with the necessary transfer parameters, and then the controller handles the actual data transfer. The advantages and disadvantages of DMA are as follows:

Advantages:

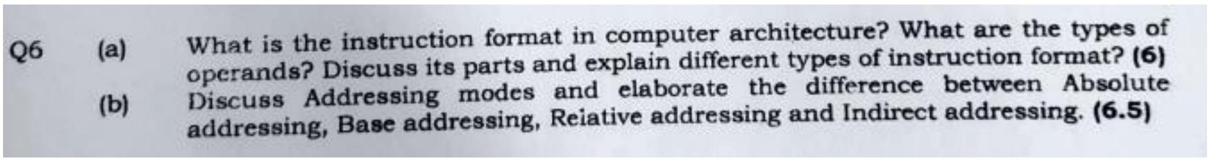
- Efficient for large amounts of data transfer
- Reduces CPU utilization and improves system performance
- Allows the CPU to perform other tasks while data transfer is in progress

Disadvantages:

- More complex to implement than programmed I/O
- Requires dedicated hardware, such as a DMA controller
- Can lead to synchronization issues if not implemented properly

The roles and functions of a DMA controller are:

1. Data transfer initiation: The DMA controller initiates the data transfer between the I/O device and memory without any intervention from the CPU. The CPU only needs to set up the DMA controller with the necessary transfer parameters, such as the source and destination addresses, transfer length, and transfer mode.
2. Data transfer management: The DMA controller manages the data transfer by continuously transferring data from the I/O device to memory or from memory to the I/O device until the transfer is complete. The DMA controller can also handle various types of data transfer, such as block transfers, scatter/gather transfers, and cycle stealing transfers.
3. Interrupt handling: The DMA controller generates an interrupt signal to the CPU when the data transfer is complete or when an error occurs during the transfer. The interrupt signal informs the CPU that the data transfer is complete and that the data is available in memory for further processing.
4. Multiple DMA channels: Most modern DMA controllers have multiple DMA channels that can handle multiple data transfers simultaneously. This allows the system to perform multiple I/O operations in parallel, improving system performance and efficiency.



Q6 (a) What is the instruction format in computer architecture? What are the types of operands? Discuss its parts and explain different types of instruction format? **(6)**

(b) Discuss Addressing modes and elaborate the difference between Absolute addressing, Base addressing, Relative addressing and Indirect addressing. **(6.5)**

(a)

In computer architecture, an instruction format refers to the structure of an instruction in machine language. It specifies the layout and size of an instruction and how the operands are represented. An operand is a value that an instruction operates on, such as a register or a memory location.

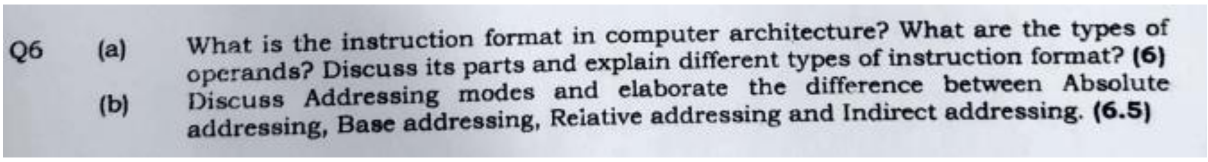
There are two main types of operands:

1. Register Operand: The operand is a register in the CPU, which holds a data value.
2. Memory Operand: The operand is a memory location in the main memory of the computer.

The instruction format is typically divided into two parts: the opcode (operation code) and the operands. The opcode specifies the operation that the instruction performs, such as add, subtract, or jump. The operands specify the data that the instruction operates on.

Different types of instruction formats are:

1. Register to Register: In this format, both operands are registers. The opcode specifies the operation to be performed on the operands, and the result is stored in one of the registers.
2. Register to Memory: In this format, one operand is a register, and the other is a memory location. The opcode specifies the operation to be performed, and the result is stored in the memory location.
3. Immediate: In this format, one operand is an immediate value, which is a constant value included in the instruction. The other operand is a register. The opcode specifies the operation to be performed, and the result is stored in the register.
4. Direct Addressing: In this format, both operands are memory locations. The opcode specifies the operation to be performed, and the result is stored in one of the memory locations.
5. Indirect Addressing: In this format, one operand is a memory location that contains the address of the other operand. The opcode specifies the operation to be performed, and the result is stored in the memory location pointed to by the address.



Q6 (a) What is the instruction format in computer architecture? What are the types of operands? Discuss its parts and explain different types of instruction format? **(6)**

(b) Discuss Addressing modes and elaborate the difference between Absolute addressing, Base addressing, Relative addressing and Indirect addressing. **(6.5)**

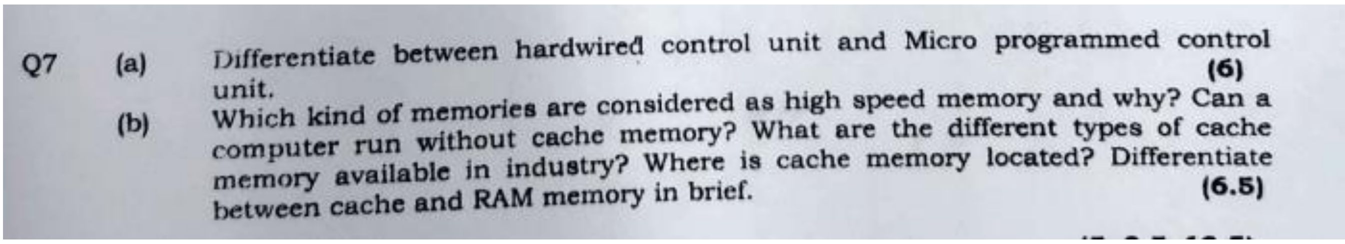
(b)

In computer architecture, an addressing mode is a technique used by the CPU to access memory or registers in order to retrieve or store data. Different addressing modes provide flexibility to the programmer to access memory or register operands in different ways, which can be helpful in optimizing program performance and memory usage.

There are several types of addressing modes, including:

1. Absolute addressing: In absolute addressing, the operand address is a fixed value. The address is directly specified in the instruction and remains constant throughout the program execution. For example, if the address of a variable is 0x1000, an instruction with absolute addressing would contain that specific memory address to fetch or store data.
2. Base addressing: In base addressing, the operand address is calculated by adding an offset to a base register. The base register stores the starting address of a memory block, and the offset specifies the displacement of the operand from the base address. For example, to access the third element of an array stored in memory starting at address 0x1000, a base register would be set to 0x1000 and the offset would be set to 2 (since arrays are zero-indexed). The instruction would then use the value in the base register (0x1000) plus the offset (2) to calculate the address of the third element (0x1006).
3. Relative addressing: In relative addressing, the operand address is specified as a fixed offset from the current program counter (PC). The PC is the register that holds the address of the current instruction being executed. Relative addressing is used when the operand address is known relative to the current program counter. For example, a jump instruction that specifies a relative offset will jump to an address that is relative to the current instruction.
4. Indirect addressing: In indirect addressing, the operand address is specified indirectly through a pointer. The instruction contains the memory address of the pointer, and the pointer contains the actual operand address. Indirect addressing is useful when the operand address is not known in advance or needs to be determined dynamically at runtime.

The main differences between these addressing modes are how the operand address is calculated or specified. Absolute addressing uses a fixed address, while base addressing and relative addressing use an offset value. Indirect addressing uses a pointer to indirectly specify the operand address.



Q7 (a) Differentiate between hardwired control unit and Micro programmed control unit. **(6)**

(b) Which kind of memories are considered as high speed memory and why? Can a computer run without cache memory? What are the different types of cache memory available in industry? Where is cache memory located? Differentiate between cache and RAM memory in brief. **(6.5)**

(a)

Hardwired Control Unit	Microprogrammed Control Unit
Hardwired control unit generates the control signals needed for the processor using logic circuits	Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
Faster	Slower
Difficult to modify	Easy to modify
More costly	Less costly
Cannot handle complex instructions	Can handle complex instructions
Only limited number of instructions are used	Control signals for many instructions can be generated
Used in computer that makes use of Reduced Instruction Set Computers (RISC)	Used in computer that makes use of Complex Instruction Set Computers (CISC)

(b)

In computer architecture, the kind of mnemonics that are considered high-speed memory are those that can be accessed quickly and efficiently by the processor. These mnemonics are typically stored in a special type of memory known as cache memory, which is much faster than the main system memory (RAM).

There are several types of cache memory, including:

1. Level 1 (L1) cache: This is the smallest and fastest cache memory in a computer, and it is built directly into the processor. L1 cache stores the most

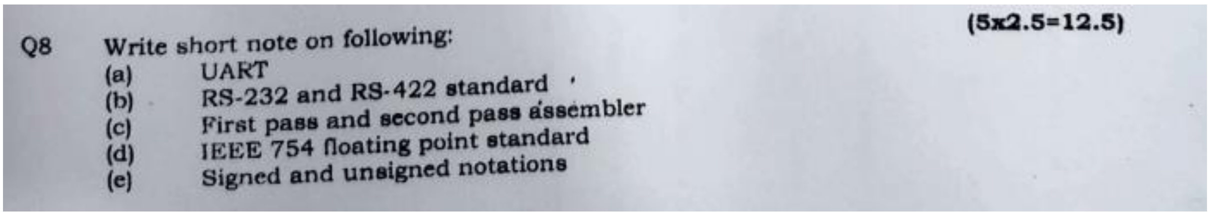
frequently accessed data and instructions, so the processor can quickly access them without having to wait for them to be retrieved from main memory.

2. Level 2 (L2) cache: This is a larger cache memory that is typically located on the processor chip or on a separate chip on the motherboard. L2 cache is slower than L1 cache, but still faster than main memory, and it stores additional frequently accessed data and instructions.
3. Level 3 (L3) cache: Some processors have a third level of cache memory, which is even larger than L2 cache but slower. L3 cache is typically shared among multiple processor cores, and it stores data and instructions that are less frequently accessed than those stored in L1 and L2 cache.

L1 cache is typically located inside the processor chip itself, whereas L2 and L3 cache are located outside the processor but still on the motherboard. L2 cache is often integrated into the processor package, while L3 cache is typically found on the motherboard itself.

No.	RAM	CACHE
1.	**RAM is a volatile memory that could store the data as long as the power is supplied.**	**Cache is a smaller and fast memory component in the computer.**
2.	**The size of RAM is greater.**	**The size of cache memory is less.**
3.	**It is expensive but not as expensive as Cache.**	**It is expensive than RAM.**
4.	**It holds programs and data that are currently executed by the CPU.**	**It holds frequently used data by the CPU.**
5.	**It is not fastest as compared to cache.**	**It is faster.**
6.	**RAM is faster than a hard disk, floppy disk, compact disk, or just any form of secondary storage media.**	**Cache memory increase the accessing speed of CPU.**
7.	**CPU reads Cache Memory data before reading RAM.**	**CPU reads RAM data after reading Cache Memory.**
8.	**It can be internal and external both.**	**It is generally internal.**

9.	Types of RAM- 1. Static RAM (SRAM) 2. Dynamic RAM (DRAM)	Types of Cache- 1. L1 (Level 1) cache or Primary cache 2. L2 (Level 2) cache or Secondary cache 3. L3 (Level 3) cache
10.	RAM is a short-term digital storage used to keep data and software that is now being used by the CPU.	It is located in close proximity to CPU, thus storing copies of data or instructions from frequently accessed locations of main memory in order to provide high-speed access by the processor.



Q8 Write short note on following: (5x2.5=12.5)

(a) UART

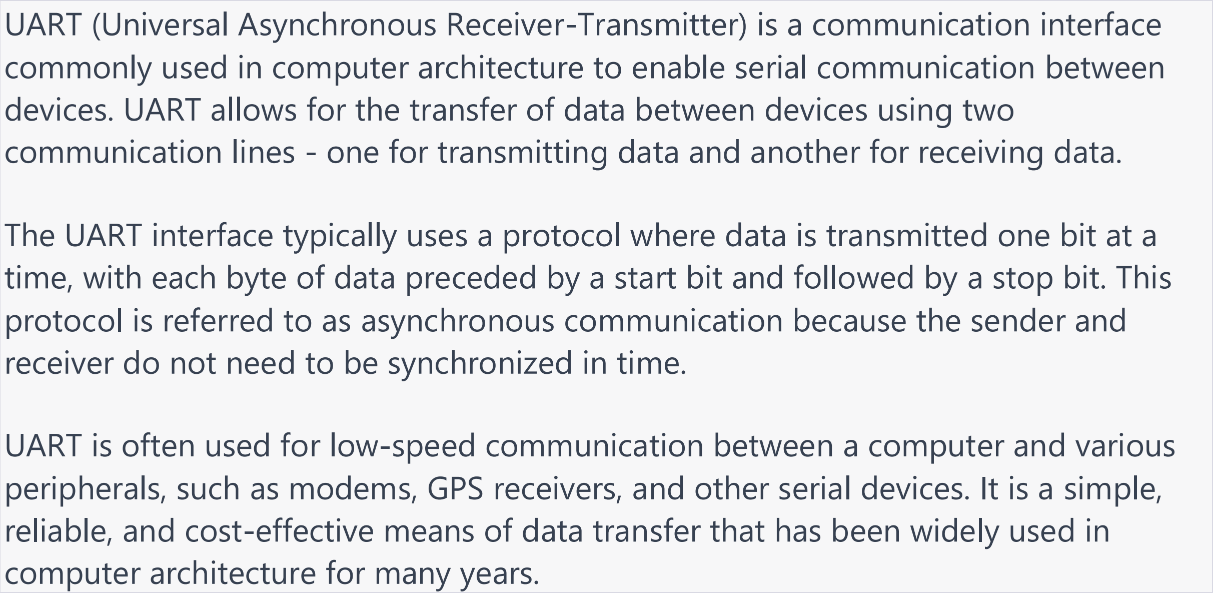
(b) RS-232 and RS-422 standard

(c) First pass and second pass assembler

(d) IEEE 754 floating point standard

(e) Signed and unsigned notations

(a)

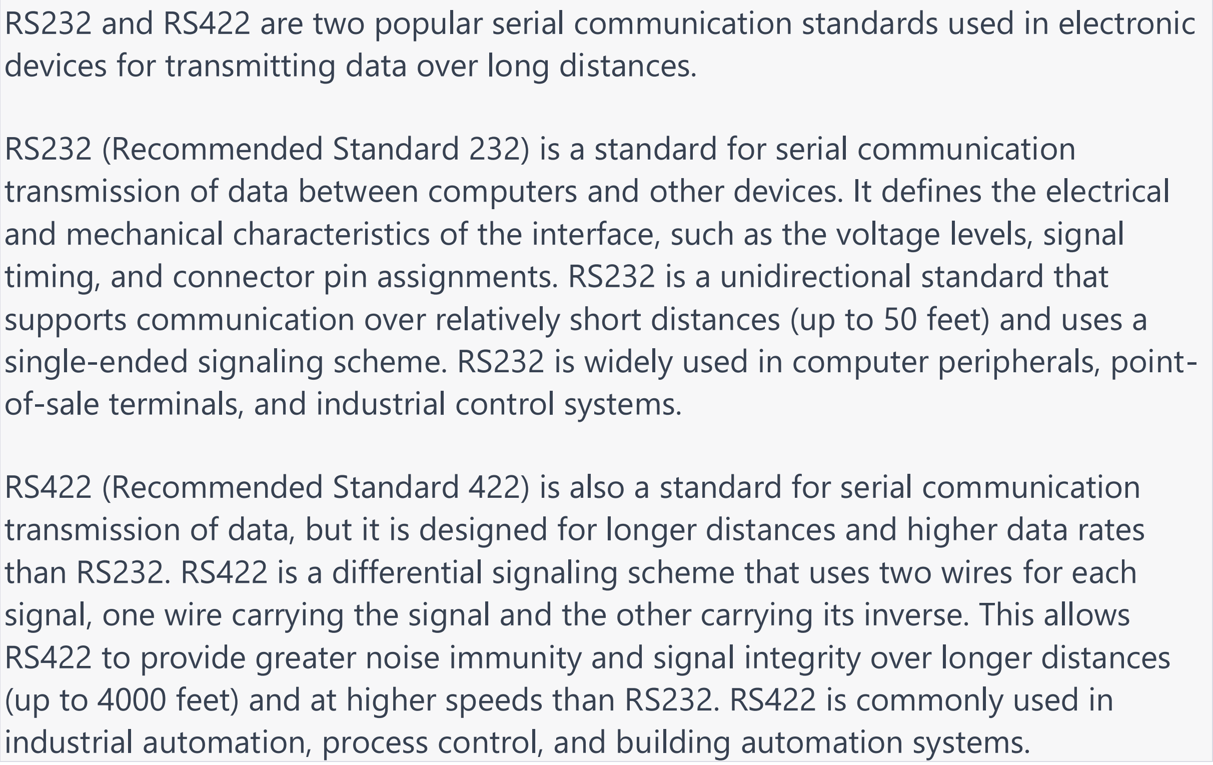


UART (Universal Asynchronous Receiver-Transmitter) is a communication interface commonly used in computer architecture to enable serial communication between devices. UART allows for the transfer of data between devices using two communication lines - one for transmitting data and another for receiving data.

The UART interface typically uses a protocol where data is transmitted one bit at a time, with each byte of data preceded by a start bit and followed by a stop bit. This protocol is referred to as asynchronous communication because the sender and receiver do not need to be synchronized in time.

UART is often used for low-speed communication between a computer and various peripherals, such as modems, GPS receivers, and other serial devices. It is a simple, reliable, and cost-effective means of data transfer that has been widely used in computer architecture for many years.

(b)



RS232 and RS422 are two popular serial communication standards used in electronic devices for transmitting data over long distances.

RS232 (Recommended Standard 232) is a standard for serial communication transmission of data between computers and other devices. It defines the electrical and mechanical characteristics of the interface, such as the voltage levels, signal timing, and connector pin assignments. RS232 is a unidirectional standard that supports communication over relatively short distances (up to 50 feet) and uses a single-ended signaling scheme. RS232 is widely used in computer peripherals, point-of-sale terminals, and industrial control systems.

RS422 (Recommended Standard 422) is also a standard for serial communication transmission of data, but it is designed for longer distances and higher data rates than RS232. RS422 is a differential signaling scheme that uses two wires for each signal, one wire carrying the signal and the other carrying its inverse. This allows RS422 to provide greater noise immunity and signal integrity over longer distances (up to 4000 feet) and at higher speeds than RS232. RS422 is commonly used in industrial automation, process control, and building automation systems.

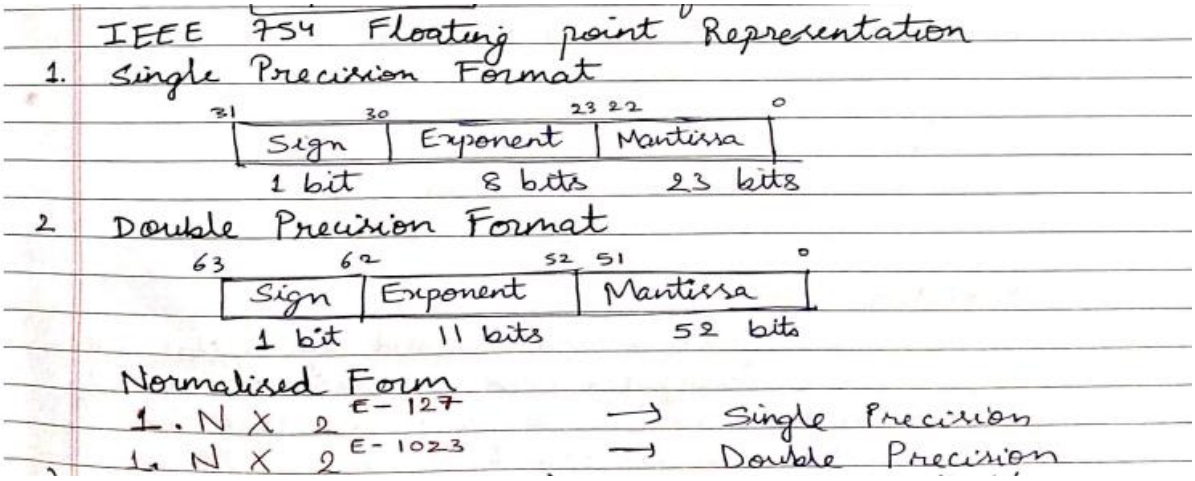
In summary, RS232 is a unidirectional standard designed for short distances and low data rates, while RS422 is a differential signalling scheme that supports bidirectional communication over longer distances and at higher speeds. Both standards have their own advantages and disadvantages and are used in a variety of applications depending on the requirements of the system.

(c)

A first pass assembler is an assembler program that scans the source code only once, generating a symbol table that stores the addresses of all symbols used in the program. It also generates a intermediate code that contains any instructions or directives that require the symbol table. However, it does not generate any machine code at this stage. The first pass assembler is mainly used to resolve forward references and detect any syntax errors in the program.

A second pass assembler is an assembler program that reads the intermediate code generated by the first pass assembler and generates the actual machine code. The second pass assembler uses the symbol table generated by the first pass to resolve all references to memory locations and symbols in the program. The second pass assembler also performs additional checks and optimizations on the program, such as error checking and generating code for pseudo-operations.

(d)



IEEE 754 Floating point Representation

1. Single Precision Format

31	30 … 23	22 … 0
Sign	Exponent	Mantissa
1 bit	8 bits	23 bits

2 Double Precision Format

63	62 … 52	51 … 0
Sign	Exponent	Mantissa
1 bit	11 bits	52 bits

Normalised Form

$1.N \times 2^{E-127}$ → Single Precision

$1.N \times 2^{E-1023}$ → Double Precision

(e)

https://www.tutorialspoint.com/unsigned-and-signed-binary-numbers#:~:text=Signed%20numbers%20use%20sign%20flag,numbers%20but%20not%20negative%20numbers.