Other ETE Notes

Practice Set -2 for 5th Semester COA ETE Examination

Q.1 Discuss the various pipeline conflicts or pipeline hazards.

Pipeline Hazards (1)

- ***Pipeline Hazards*** are situations that prevent the next instruction in the instruction stream from executing in its designated clock cycle
- Hazards reduce the performance from the ideal speedup gained by pipelining
- Three types of hazards
  - ***Structural hazards***
    - Arise from resource conflicts when the hardware can't support all possible combinations of overlapping instructions
  - ***Data hazards***
    - Arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline
  - ***Control hazards***
    - Arise from the pipelining of branches and other instructions that change the PC (Program Counter)

Q.2 How many 128 X 8 RAM chips are needed to provide a memory capacity of 2048 bytes?

To determine how many 128 x 8 RAM chips are needed for a memory capacity of 2048 bytes, we can follow these steps:

1. Calculate the capacity of one RAM chip: Each chip has a capacity of 128 x 8 bits. Since 1 byte is equal to 8 bits, this translates to 128 x 8 / 8 = 128 bytes per chip.
2. Calculate the total memory requirement in bytes: We need 2048 bytes.
3. Divide the total memory by the capacity of one chip: 2048 bytes / 128 bytes/chip = 16 chips.

Therefore, you would need 16 128 x 8 RAM chips to provide a memory capacity of 2048 bytes.

Q.3 The main memory has 3-page frames (frame 0, frame 1 and frame 2). Pages from virtual memory are required in the order 2,3,2,1,5,2,4,5,3,2,5,2. Calculate the Hit ratio using FIFO replacement Algorithm

Note: AI Answer

Frame 0	Frame 1	Frame 2	Page Faults
2	-	-	Yes
2	3	-	Yes
2	3	2	No (Hit)
1	3	2	Yes
1	5	2	Yes
4	5	2	Yes
4	5	2	No (Hit)
4	5	3	Yes
4	2	3	Yes
5	2	3	Yes
5	2	2	No (Hit)

we have 2 hits and 9 page faults. The hit ratio is therefore

112≈0.1818

Q.4 What is the significance of page replacement? How many pages fault occurs in FIFO and LRU for the reference string 1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,7,8,9,5,4,5,4,2 with 4-page frames?

Page replacement is a significant aspect of memory management in operating systems. When a page fault occurs (i.e., a requested page is not in memory), the operating system needs to choose a page to evict from memory to make room for the new page. The page replacement algorithm determines which page to replace.

Now, let's calculate the number of page faults using both the FIFO (First-In-First-Out) and LRU (Least Recently Used) algorithms for the given reference string with 4-page frames.

Frame 0	**Frame 1**	**Frame 2**	**Frame 3**	**Page Faults**
1	-	-	-	Yes
1	2	-	-	Yes
1	2	3	-	Yes
1	2	3	4	Yes
5	2	3	4	Yes
5	3	3	4	No (Hit)
5	3	4	4	No (Hit)
5	3	4	1	Yes
6	3	4	1	Yes
6	7	4	1	Yes
6	7	8	1	Yes
6	7	8	7	No (Hit)

Frame 0	Frame 1	Frame 2	Frame 3	Page Faults
6	7	8	8	No (Hit)
6	7	9	8	Yes
6	7	9	7	No (Hit)
6	8	9	7	Yes
6	8	9	8	No (Hit)
6	8	9	9	No (Hit)
5	8	9	9	Yes
5	4	9	9	Yes
5	4	5	9	Yes
5	4	5	4	No (Hit)
2	4	5	4	Yes

So, we have 15 page faults using the FIFO algorithm.

LRU

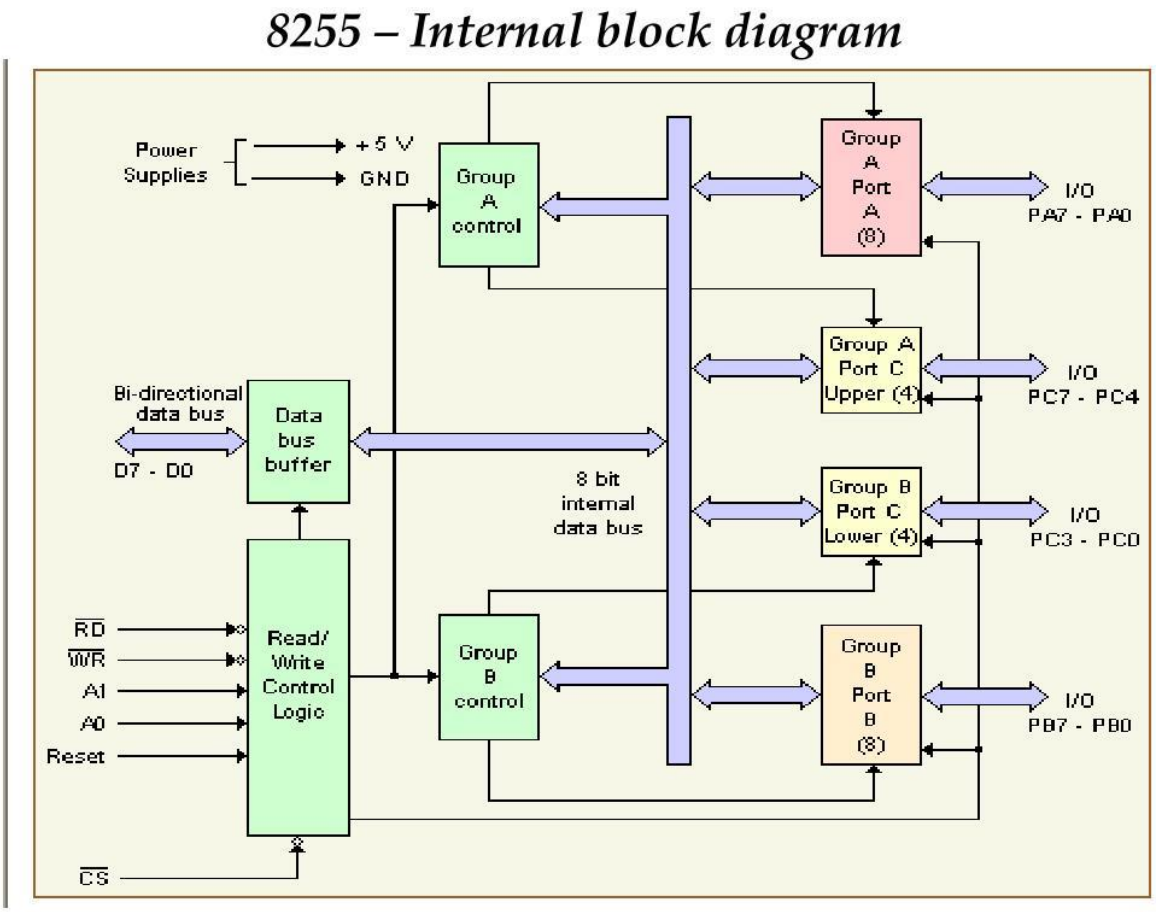
Frame 0	Frame 1	Frame 2	Frame 3	Page Faults
1	-	-	-	Yes
1	2	-	-	Yes
1	2	3	-	Yes
1	2	3	4	Yes
5	2	3	4	Yes
5	2	3	4	No (Hit)
5	2	3	4	No (Hit)
5	2	3	1	Yes
5	2	6	1	Yes
5	7	6	1	Yes
8	7	6	1	Yes

Frame 0	Frame 1	Frame 2	Frame 3	Page Faults
8	7	6	1	No (Hit)
8	7	6	1	No (Hit)
8	7	9	1	Yes
8	7	9	1	No (Hit)
8	7	9	1	No (Hit)
8	7	9	1	No (Hit)
8	7	9	1	No (Hit)
5	7	9	1	Yes
5	4	9	1	Yes
5	4	9	1	No (Hit)
5	4	9	1	No (Hit)
2	4	9	1	Yes

So, we have 7 hits and 16 page faults. The hit ratio is therefore

237≈0.3043

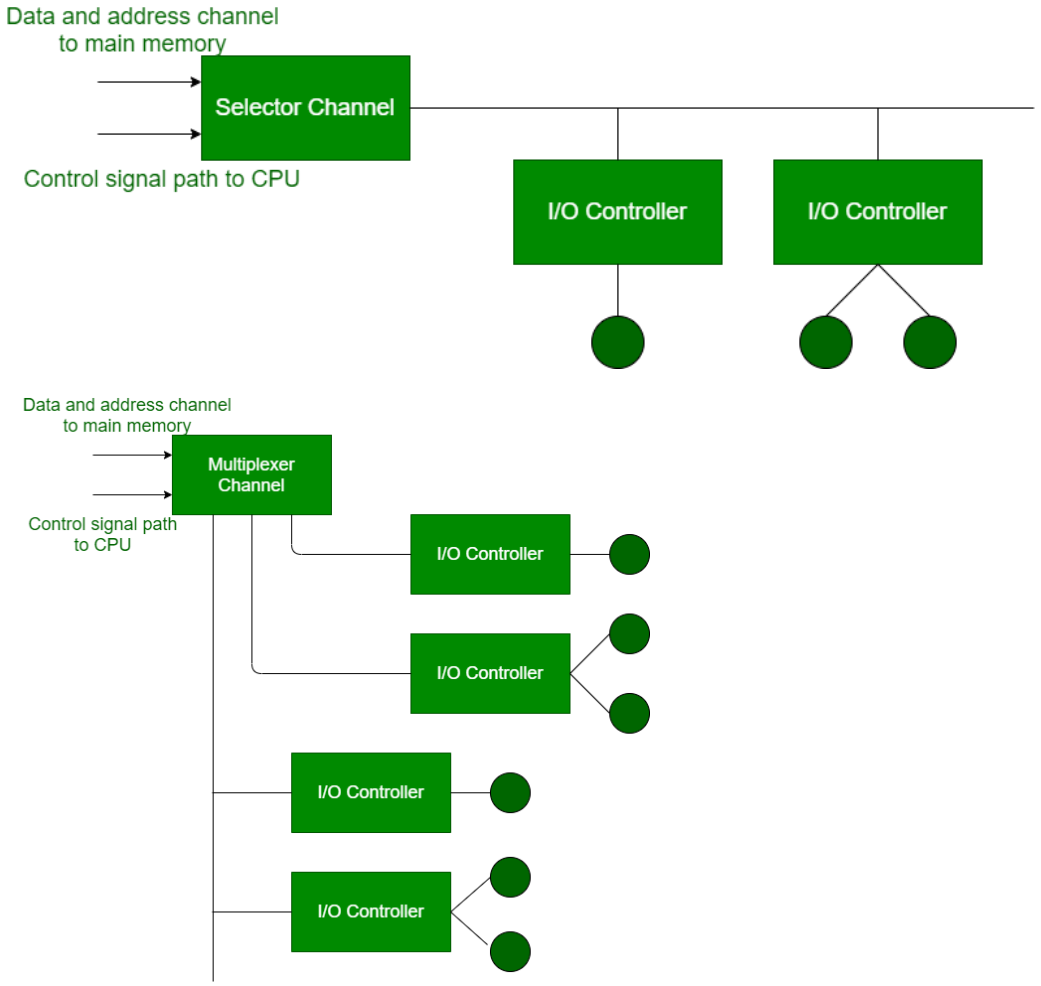
Q.5 Examine the 8-bit programmable parallel port using a clear and well-organized diagram.



- An 8-bit programmable parallel port, such as the 8255 programmable peripheral interface, is a versatile device used to interface the CPU with its outside world, such as ADC, DAC, keyboard, etc[1][2]. It consists of three 8-bit bidirectional I/O ports: Port A (PA), Port B (PB), and Port C (PC)[1][2] **Port A (PA)** and **Port B (PB)** can be used as 8-bit input/output ports[2].

- **Port C (PC)** can be used as an 8-bit input/output port or as two 4-bit input/output ports[2]. It can be further divided into two 4-bit ports named Cupper (Cu) and Clower (Cl)[1].
- The 8255 operates in a +5V regulated power supply[1].
- Depending on the value of the Control word, the 8255 can work in different modes[1].

Q.6 Explain various types of I/O channels with the help of diagram.



Input/Output (I/O) channels are used to manage data transfers between the CPU and peripheral devices. They are an extension of the Direct Memory Access (DMA) concept, as they have direct access to the main memory[1]. Here are the main types of I/O channels:

1. **Selector Channel**: This type of channel controls multiple high-speed devices but is dedicated to the transfer of data with one of the devices at a time[2]. Each device is handled by a controller or I/O module[2].
2. **Multiplexer Channel**: This is a DMA controller that can handle multiple devices simultaneously[2]. It can perform block transfers for several devices at once[2]. There are two types of multiplexers used in this channel[2]:
    - **Byte Multiplexer**: Used for low-speed devices. It transmits or accepts characters and interleaves bytes from several devices[2].
    - **Block Multiplexer**: Accepts or transmits blocks of characters. Interleaves blocks of bytes from several devices and is used for high-speed devices[2].

Q.7 Determine the number of clock cycles that it takes to process 100 tasks in a six-segment pipeline.

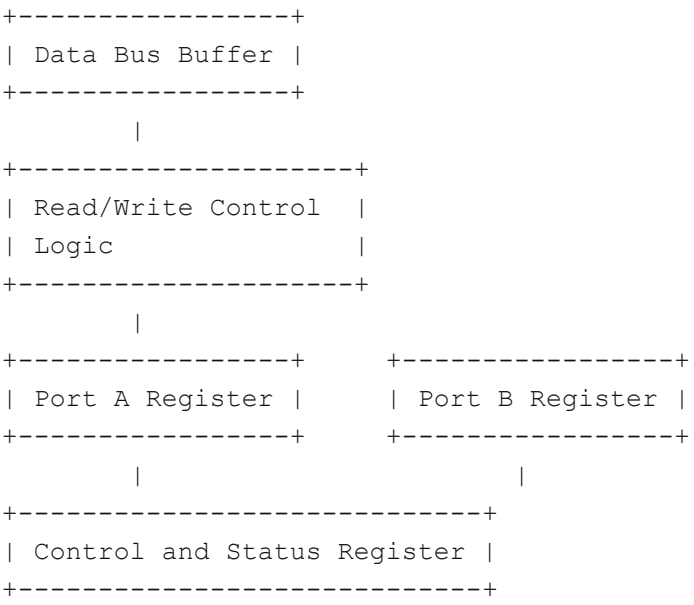
In a pipeline with six stages, the first task will take 6 clock cycles to pass through all the stages (from stage 1 to stage 6). After the first task has entered stage 2, the second task can enter stage 1. This means that starting from the 6th clock cycle, a new task can be introduced into the pipeline at each clock cycle.

So, for 100 tasks, the total number of clock cycles needed will be the 6 cycles needed for the first task to pass through all the stages plus 99 cycles for the remaining tasks (since after the first task, a new task can be introduced at each clock cycle).

Therefore, the total number of clock cycles needed to process 100 tasks in a six-segment pipeline is

6+99=105 clock cycles.

Q.8 Discuss I/O interface with the help of its block diagram. What is the major requirement of an I/O interface?



The blocks in the diagram represent the following components[2]:

1. **Data Bus Buffer**: This is used to communicate with the CPU. All control word data and status information between the interface unit and CPU are transferred through the data bus[2].
2. **Read/Write Control Logic**: This block generates necessary control signals for overall device operations. All commands from the CPU are accepted through this block[2].
3. **Port A and Port B Registers**: These are used to transfer data between the I/O device and the Interface Unit. Each port consists of bi-directional data input and output buffers[2].
4. **Control and Status Register**: The CPU gives control information to the control register. The interface unit controls input and output operation between the CPU and I/O devices. The status register indicates the status of data registers, ports, and also records errors that may occur during the transfer of data[2].

The major requirements of an I/O interface are[3]:

1. **Control and Timing**: To manage the operations of the I/O devices.
2. **Processor Communication**: To communicate with the CPU and transfer data.

3. **Device Communication**: To communicate with the I/O devices and manage their operations.
4. **Data Buffering**: To temporarily store data while it is being transferred.
5. **Error Detection**: To detect and manage errors during data transfer.

Q.9 Describe Magnetic Disks and Magnetic Tapes in detail.

Sr. No.	Key	Magnetic Tape Memory	Magnetic Disk Memory
1	Definition	Magnetic tape is type of non-volatile memory uses thin plastic ribbon is used for storing data and as data use to be stored on ribbon so data read/write speed is slower due to which is mainly used for data backups.	On other hand Magnetic Disk is also type of non-volatile memory uses circular disk used for storing data.
2	Constituent	As mentioned above Magnetic tape memory is having plastic ribbon as main constituent.	On other hand Magnetic disk memory has metallic or plastic circular disk coated with magnetic oxide, as main constituent.
3	Cost Concern	Cost concern in case of Magnetic tape memory is less as plastic ribbon is much cheaper as compared to the circular disk as in case of magnetic disk memory.	While on other hand in case of Magnetic disk memory the cost concern is more as compared to that of Magnetic tape memory.
4	Reliability	As plastic ribbon is not much reliable and chances of its breakdown is more hence Magnetic tape memory is less reliable as compared to Magnetic Disk memory.	On other hand due to more breakdown prone circular disk Magnetic disk memory becomes more reliable than Magnetic tape memory.
5	Performance	Data store and retrieval is much slower in case of Magnetic tape memory due to which it is less efficient and has comparatively degrade performance as compared to Magnetic disk memory.	On other hand data storage and retrieval is faster in case of Magnetic disk memory and hence has better performance and more efficient as compared to Magnetic tape memory.
6	Usage	Due to low data transmission rate and less portability Magnetic tape memory mainly used for Data backup purposes.	On other hand Magnetic disk memory can be used as secondary storage.

Q.10 How many lines of the address bus must be used to access 2048 bytes of memory? How many of these lines will be common to all chips? How many lines must be decoded for chip select? Specify the size of the decoders.

1. **Number of Address Lines (n):** $n = \log_2(\text{Number of Addresses})$. For 2048 bytes of memory, $n = \log_2(2048) = 11$ address lines.
2. **Common Lines Among Memory Chips:** If all memory chips are of the same size, all address lines will be common to all chips.
3. **Number of Lines for Chip Select (CS):** The number of lines to be decoded for chip select (d) depends on the number of memory chips (m). $d = \log_2(m)$.
4. **Size of Decoders for Chip Select:** The size of the decoders is determined by the number of lines being decoded (d). A decoder with n inputs and 2^n outputs is needed. For example, if decoding 2 lines for chip select, a 2-to-4 decoder is required.

Q.11 A non-pipeline system takes 50 ns to process a task. The same task can process in a six-segment pipeline with a dock cycle of 10 ns. Determine the speedup ratio of the pipeline for 100 tasks. What is the maximum speedup that can achieved? What is the maximum speedup that can achieved?

The speedup of a pipeline can be calculated using the formula:

Speedup=Execution time with pipelineExecution time without pipeline

Let's calculate the execution times:

1. **Execution time without pipeline**: Since the non-pipeline system takes 50 ns to process a task, for 100 tasks, the total time would be

   100×50ns=5000ns

   .

2. **Execution time with pipeline**: In a pipeline system, the first task will take 6 clock cycles (one for each pipeline stage) and each subsequent task will take 1 clock cycle (since a new task can be introduced at each clock cycle). So, for 100 tasks, the total time would be

   (6+99)×10ns=1050ns

   .

So, the speedup ratio of the pipeline for 100 tasks is:

Speedup=1050ns5000ns≈4.76

The maximum speedup that can be achieved by pipelining is ideally equal to the number of pipeline stages. However, in practice, it's less than that due to pipeline stalls, branch penalties, etc. In this case, the pipeline has 6 stages, so the maximum theoretical speedup is 6.

Q.12. A cache memory has access time of 40 ns and main memory access time is 160 ns. Calculate the average access time of CPU if the hit ratio is 80%.

The average access time of the CPU can be calculated using the formula:

Average Access Time=Hit Time+Miss Rate×Miss Penalty

where:

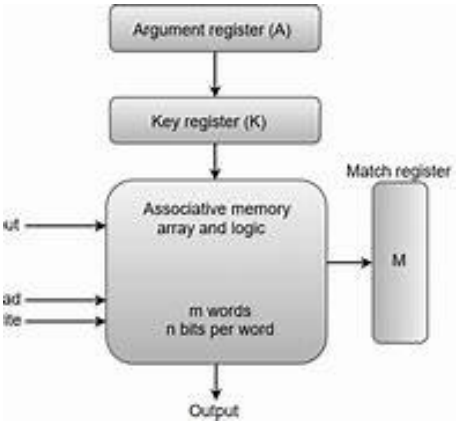
- Hit Time is the time to access the cache (40 ns in this case),
- Miss Rate is the rate at which a required data is not found in the cache (1 - Hit Ratio = 1 - 0.8 = 0.2 in this case), and
- Miss Penalty is the time to access the main memory (160 ns in this case).

Substituting these values into the formula gives:

Average Access Time=40ns+0.2×160ns=40ns+32ns=72ns

So, the average access time of the CPU is 72 ns.

Q.13 Discuss hardware organization of associative memory with the help of its block diagram.



Associative memory, also known as Content Addressable Memory (CAM), is a type of memory that is used to search for data based on its content rather than its address[1][2]. It's often used in applications that require high-speed searches, such as database management systems and networking[2].

1. **Argument Register (A)**: This register holds the word that is to be searched in memory[1].
2. **Key Register (K)**: This register provides a mask for choosing a particular field or key in the argument word[1]. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word[1].
3. **Match Register (M)**: This register consists of m bits, one for each memory word[1]. If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit Mi in the match register is set to 1[1].
4. **Memory Array**: This is where the data is stored. Each cell in the array is compared with the corresponding bit in the argument register[1].

Q.14 Consider a fully associative mapped cache of size 512 KB with block size 1 KB. There are 17 bits in the tag. Find-

a) Size of main memory
b) Tag directory size

a) **Size of main memory:**

In a fully associative cache, the tag size is determined by the number of blocks in the main memory. Given that there are 17 bits in the tag, this means there are

217

blocks in the main memory.

Given that each block is 1 KB (or 1024 bytes), the total size of the main memory is

217×1KB=217×1024bytes=227bytes=128MB

.

b) **Tag directory size:**

The size of the tag directory is determined by the number of entries in the cache and the size of each entry. In a fully associative cache, each entry corresponds to a block in the cache.

Given that the cache size is 512 KB and each block is 1 KB, there are

1KB/block512KB=512

blocks in the cache, and thus 512 entries in the tag directory.

Each entry in the tag directory contains a tag and some additional information for cache management (such as a valid bit). If we assume that each entry contains only the tag (which is 17 bits), the total size of the tag directory is

512×17bits=8704bits=1088bytes

Q.15 Distinguish between Strobe Control and Handshaking techniques for asynchronous data transfer.

Strobe Control Method[1][2]:

- Strobe control is a method used in asynchronous data transfer that synchronizes data flow between two devices[1].
- Bits are transmitted one at a time, independently of one another, and without the aid of a clock signal in asynchronous communication[1].
- Strobe control involves sending data along with a different signal known as the strobe signal[1].
- The strobe signal alerts the receiving device that the data is valid and ready to be read[1].
- The receiving device waits for the strobe signal before reading the data to ensure it is synchronized with its clock[1].

- The strobe signal is usually generated by the transmitting device and is sent either before or after the data[1].
- If the strobe signal is sent before the data, it is called a leading strobe. If it is sent after the data, it is called a trailing strobe[1].

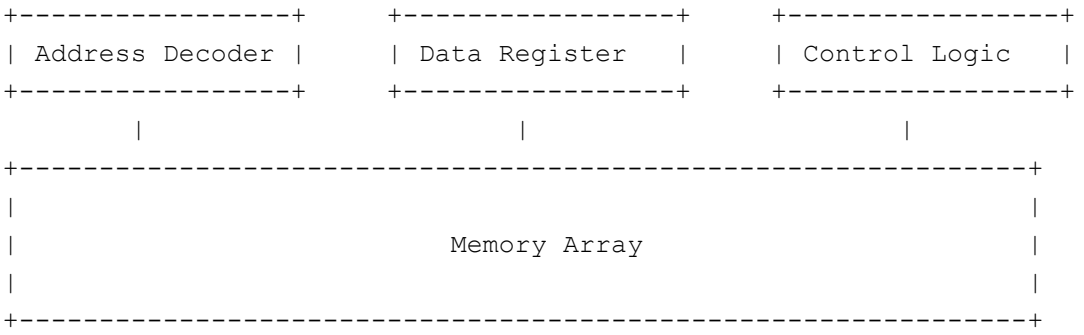
Handshaking Method[12]:

- Handshaking is a method used in asynchronous data transfer where two devices manage their communication[1].
- It ensures that the transmitting and receiving devices are prepared to send and receive data[1].
- Handshakes are essential in asynchronous communication since there is no clock signal to synchronize the data transfer[1].
- During handshaking, two types of signals are mostly used: request-to-send (RTS) and clear-to-send (CTS)[1].
- The receiving device is notified by an RTS signal when the transmitting equipment is ready to provide data[1].
- The receiving device responds with a CTS signal when it is ready to accept data[1].

Q.16 Demonstrate the block diagrams and functionalities of integrated circuit chips for RAM and ROM.

RAM (Random Access Memory) Integrated Circuit Chips[12]:

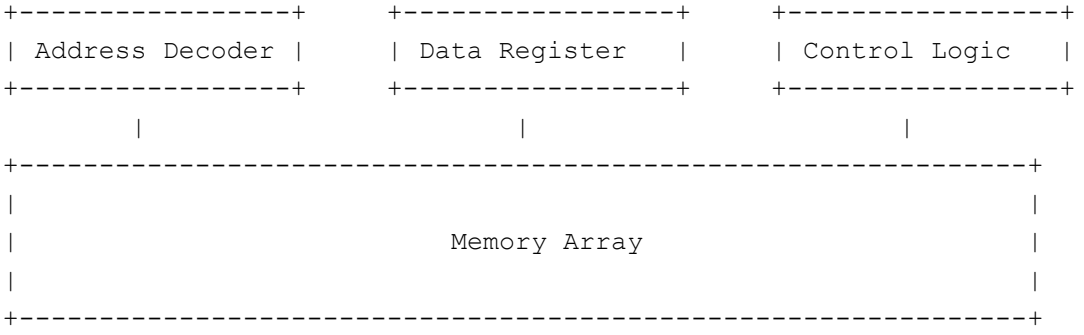
RAM chips are available in a variety of sizes and are used as per the system requirement[1]. The following block diagram demonstrates the chip interconnection in a 128 * 8 RAM chip[1].



A 128 * 8 RAM chip has a memory capacity of 128 words of eight bits (one byte) per word[1]. This requires a 7-bit address and an 8-bit bidirectional data bus[1]. The 8-bit bidirectional data bus allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation[1].

ROM (Read Only Memory) Integrated Circuit Chips[13]:

ROM chips are also available in a variety of sizes and are also used as per the system requirement[1]. The following block diagram demonstrates the chip interconnection in a 512 * 8 ROM chip[1].

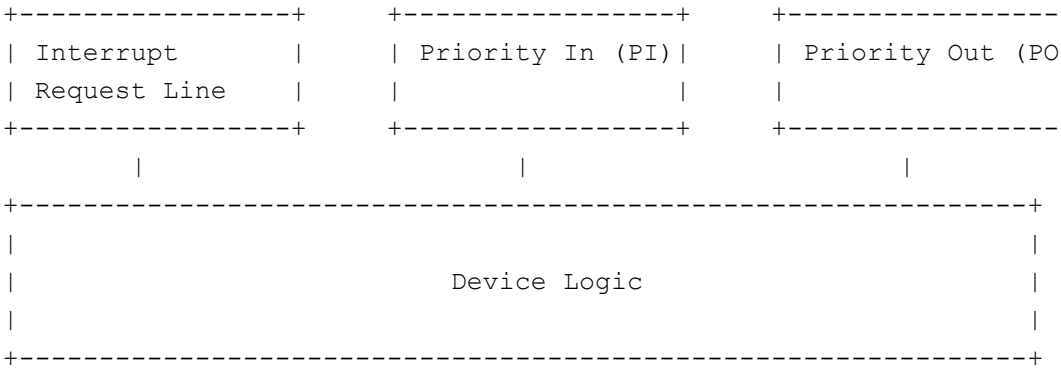


A ROM chip has a similar organization as a RAM chip[1]. However, a ROM can only perform read operation; the data bus can only operate in an output mode[1].

Q.17 Describe the concept of daisy-chaining. Devise a single-stage daisy-chaining priority logic circuit and elucidate its operation, detailing how it manages interrupt-initiated I/O requests.

Daisy-Chaining is a method used in computer architecture to establish priority among devices that request an interrupt. This method involves connecting all the devices that can request an interrupt in a serial manner. The device with the highest priority is placed first, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain.

Here's a simplified representation of a single-stage daisy-chaining priority logic circuit:



The blocks in the diagram represent the following components:

1. **Interrupt Request Line**: This line is common to all devices and goes into the CPU. When no interrupts are pending, the line is in HIGH state. But if any of the devices raises an interrupt, it places the interrupt request line in the LOW state.
2. **Priority In (PI)**: This signal is received at the PI input of a device. If the device has not requested the interrupt, it passes this signal to the next device through its PO (priority out) output.
3. **Priority Out (PO)**: If the device had requested the interrupt, the device consumes the acknowledge signal and blocks its further use by placing 0 at its PO (priority out) output. The device then proceeds to place its interrupt vector address (VAD) into the data bus of the CPU.

Operation of Daisy-Chaining Priority Logic Circuit:

The CPU acknowledges an interrupt request from the line and then enables the interrupt acknowledge line in response to the request. This signal is received at the PI (Priority in) input of the first device. If the device has not requested the interrupt, it passes this signal to the next device through its PO (priority out) output. However, if the device had requested the interrupt, the device consumes the acknowledge signal and blocks its further use by placing 0 at its PO (priority out) output. The device then proceeds to place its interrupt vector address (VAD) into the data bus of the CPU.

In this way, the daisy chain arrangement provides the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority. This method is particularly useful for managing interrupt-initiated I/O requests in a system with multiple I/O devices.

Q.18 Design parallel priority interrupt hardware for a system with four interrupt sources using priority encoder.

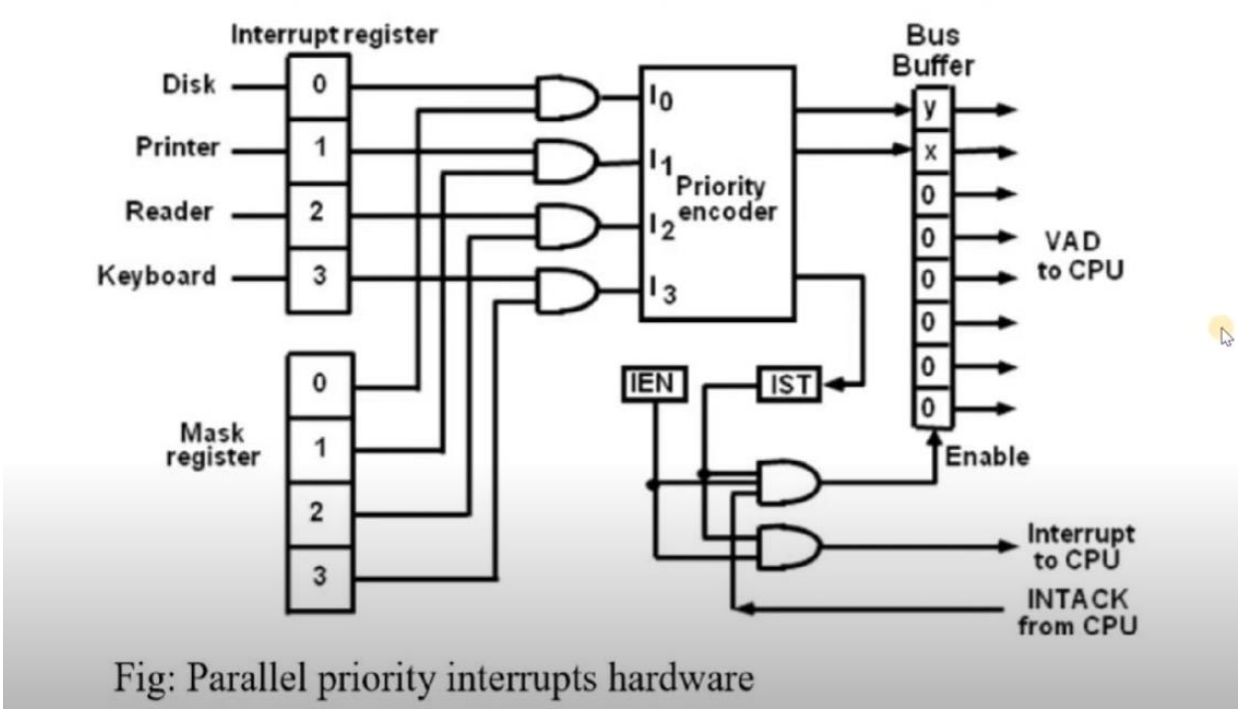


Fig: Parallel priority interrupts hardware

Priority Encoder

- Determines the highest priority interrupt when more than one interrupts take place

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' I_1'$
0	0	0	1	1	1	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$

Fig: Priority Encoder Truth Table

1. **nterrupt Request Lines (I0, I1, I2, I3)**: These lines are connected to the devices that can request an interrupt. When a device wants to interrupt the CPU, it sends a signal on its corresponding interrupt request line.
2. **Priority Encoder (4-to-2)**: This block receives the interrupt requests from the devices and outputs the binary code of the highest-priority active input signal. For example, if I1 and I2 are active, the priority encoder will output the binary code of I2 because it has a higher priority.
3. **Interrupt Vector (IV0, IV1)**: These lines carry the binary code output from the priority encoder to the CPU. The CPU uses this code to determine which interrupt service routine to execute.

The operation of this system is as follows:

1. When a device wants to interrupt the CPU, it sends a signal on its corresponding interrupt request line.
2. The priority encoder receives the interrupt requests and determines which one has the highest priority.
3. The priority encoder outputs the binary code of the highest-priority active input signal.
4. The CPU receives the interrupt vector and uses it to determine which interrupt service routine to execute.

Q.19 Explain the Direct Memory Transfer (DMA) techniques used for data transfer in a computer system in detail.

Direct Memory Access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as a DMA controller (DMAC).

DMA Controller The DMA controller is a hardware device that enables I/O devices to directly access memory with minimal participation from the processor[1]. It serves as an interface for the data bus and the I/O devices[1]. The DMA controller contains an address unit, which generates the address and selects an I/O device for the transfer of data[1].

Types of DMA There are four popular types of DMA[1]:

1. **Single-Ended DMA**: Operates by reading and writing from a single memory address.
2. **Dual-Ended DMA**: Can read and write from two memory addresses.
3. **Arbitrated-Ended DMA**: Works by reading and writing to several memory addresses.
4. **Interleaved DMA**: Reads from one memory address and writes from another memory address.

Working of DMA Controller The DMA controller has three registers[1]:

1. **Address register**: Contains the address to specify the desired location in memory.
2. **Word count register**: Contains the number of words to be transferred.
3. **Control register**: Specifies the transfer mode.

The CPU can both read and write into the DMA registers under program control via the data bus[1]. The DMA controller seizes the memory bus and CPU momentarily, preventing the CPU from accessing main memory[2]. This cycle stealing improves the total system performance[2].

Advantages of DMA

1. **Speed**: The data transfers are faster than the CPU-managed data transfers because it doesn't require CPU involvement in each byte[2].
2. **Efficiency**: It reduces the CPU overhead, by allowing it to focus on other tasks[2].
3. **Parallelism**: Multiple DMA channels work simultaneously which improves the system performance in terms of throughput[2].

Q.20 Differentiate between the following:

a) RISC vs CISC Processor
b) Hardwired Control Unit vs Micro-programmed Control Unit
c) Vectored and Non- vectored Interrupts
d) Write -Through and Write Back policy of Cache

RISC	CISC
Instruction Set	Small, simple instructions	Large, complex instructions
Instruction Length	Uniform, usually executed in one clock cycle	Variable, may require multiple clock cycles
Registers	More general-purpose registers	Fewer general-purpose registers
Instruction Decoding	Simpler	Complex
Power Consumption	Lower	Higher
Execution Speed	Faster	Slower
Instructions for Complex Tasks	More	Fewer
Memory Usage	Increased	Decreased
Architecture	Based on the hardwired control unit	Based on the microprogrammed control unit

Table

	Hardwired Control Unit	**Micro-programmed Control Unit**
Control Signals	Generated by hardware using combinational logic	Generated by a program (microcode)
Operation Speed	Faster	Slower

	Hardwired Control Unit	Micro-programmed Control Unit
Design Complexity	More complex and difficult to design	Easier and less complex to design
Flexibility	Less	More
Usage	Used in RISC architecture	Used in CISC architecture

Table

	Vectored Interrupts	Non-Vectored Interrupts
Vector Address	Have a fixed vector address	Do not have a predefined vector address
Program Control Transfer	After executing, program control is transferred to that address	The interrupting device gives the address of the sub-routine
ISR Address Knowledge	The CPU is aware of the address of the ISR when the interrupt occurs	The CPU doesn't know the address of the ISR nor the source of the IRQ when the interrupt occurs

Table

	Write-Through	Write-Back
Data Update	Data is simultaneously updated to cache and memory	The data is updated only in the cache and updated into the memory at a later time
Process	Simpler and more reliable	Data is updated in the memory only when the cache line is ready to be replaced
Latency	A data write will experience latency (delay) as we have to write to two locations	Write Back is designed to reduce write operation to a memory
Inconsistency Problem	Solves the inconsistency problem	If Cache fails or if the System fails or power outages the modified data will be lost

	Write-Through	Write-Back
Usage	Used when there are no frequent writes to the cache	

Q.21 Compare and contrast the following page replacement policies.

a) LRU page replacement
b) FIFO page replacement
c) Optimal page replacement

Assuming three frames, determine how many page faults would occur for the following page reference string: 1,2,3,4,2,1,5,6,2,1,2,1,5,6 in each of the mentioned replacement algorithm.

**Policy	Description	Page Faults for Reference String 1,2,3,4,2,1,5,6,2,1,2,1,5,6**
LRU (Least Recently Used)	Replaces the page that hasn't been used for the longest time.	9
FIFO (First In, First Out)	Replaces the page that has been in memory the longest.	10
Optimal	Replaces the page that won't be used for the longest time (impossible to implement in practice, but a good theoretical benchmark).	7

LRU:

1. 1 (PF) 2 (PF) 3 (PF) 4 (PF) 2 (hit) 1 (hit) 5 (PF) 6 (PF) 2 (hit) 1 (hit) 2 (hit) 1 (hit) 5 (PF) 6 (PF)

FIFO:

1 (PF) 2 (PF) 3 (PF) 4 (PF) 2 (hit) 1 (PF) 5 (PF) 6 (PF) 2 (PF) 1 (PF) 2 (PF) 1 (hit) 5 (PF) 6 (PF)

Optimal:

1 (PF) 2 (PF) 3 (PF) 4 (PF) 2 (hit) 1 (hit) 5 (PF) 6 (PF) 1 (hit) 2 (hit) 1 (hit) 2 (hit) 5 (hit) 6 (hit)

Key Differences:

- LRU focuses on recent usage, prioritizing pages that have been actively used.
- FIFO is simpler but can evict frequently used pages if they were loaded early.
- Optimal is theoretically ideal but impractical due to requiring future knowledge of page references.

General Performance:

- Optimal is typically the best, followed by LRU, then FIFO.
- However, performance depends on workload patterns and memory constraints.

Additional Considerations:

- Implementation overhead for LRU can be higher than FIFO.
- Other algorithms like Clock or Second-Chance exist with different trade-offs.
- Page replacement strategies are often combined with other memory management techniques.

Q.22 Explain the concept of Virtual memory. Also, elaborate how memory management is done by using paging and segmentation.

Virtual Memory: Virtual memory is a memory management technique used by operating systems (OS). It allows a computer to temporarily increase the capacity of its main memory — RAM — by using secondary memory such as a hard drive or solid-state drive (SSD). Virtual memory is an illusion of a memory that is larger than the real memory. The basis of virtual memory is the noncontiguous memory allocation model. The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations. It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time.

Paging and Segmentation: Paging and segmentation are both memory management techniques used by operating systems to manage the allocation of memory to processes.

Paging: Paging is a method or technique which is used for non-contiguous memory allocation. It is a fixed-size partitioning scheme. In paging, both main memory and secondary memory are divided into equal fixed-size partitions. The partitions of the

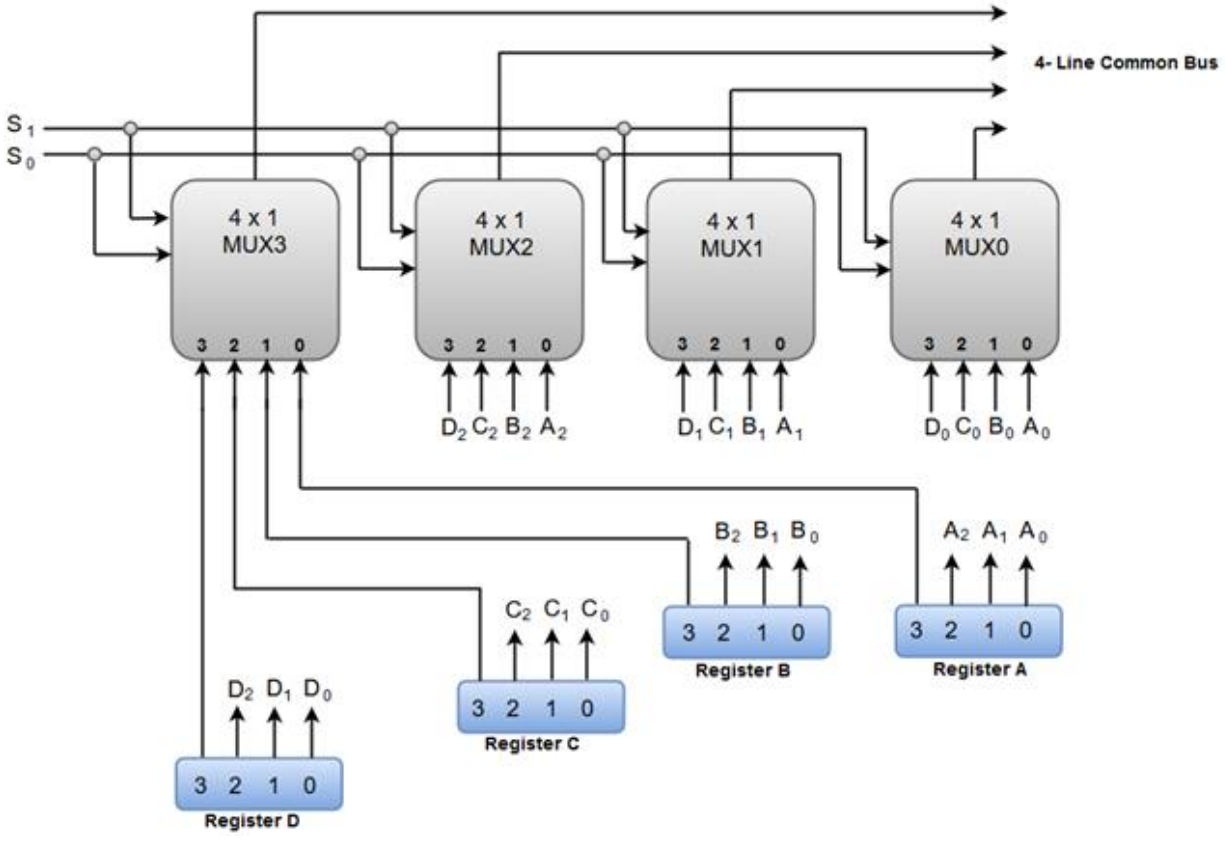
secondary memory area unit and main memory area unit are known as pages and frames respectively. Paging is a memory management method accustomed fetch processes from the secondary memory into the main memory in the form of pages. In paging, each process is split into parts where the size of every part is the same as the page size.

Segmentation: Segmentation is another non-contiguous memory allocation scheme like paging. Like paging, in segmentation, the process isn't divided indiscriminately into fixed (fixed) size pages. It is a variable-size partitioning scheme. Like paging, in segmentation, secondary and main memory are not divided into partitions of equal size. The partitions of secondary memory area units are known as segments. The details concerning every segment are held in a table known as segmentation table. Segment table contains two main data concerning segment, one is Base, which is the base address of the segment and another is Limit, which is the length of the segment. In segmentation, the CPU generates a logical address that contains the Segment number and segment offset.

In summary, virtual memory, paging, and segmentation are all crucial components of memory management in modern operating systems. They each play a role in ensuring efficient use of memory resources, improving system performance, and providing a level of abstraction between applications and the underlying hardware.

Q.23 Design a common bus system for a digital computer system having four registers of 4-bit using 3- state buffers.

Bus System for 4 Registers:



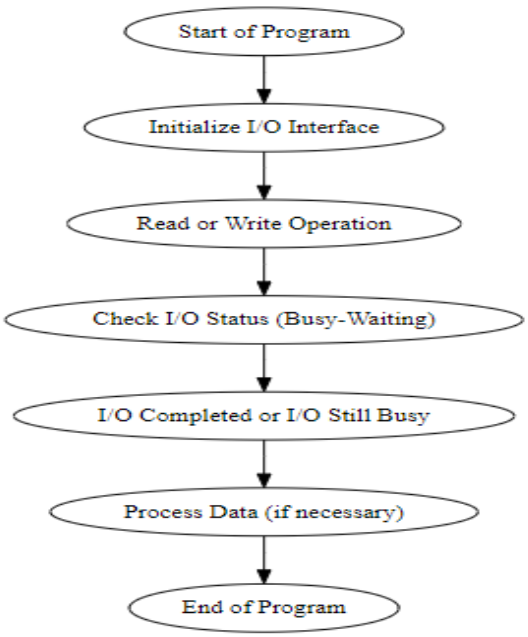
1. **Registers**: Create four 8-bit registers, let's call them A, B, C, and D. These registers will hold the data that is to be transferred.

2. **3-State Buffers**: Connect each register to a 3-state buffer. A 3-state buffer is a type of digital logic circuit that can have three states: logic 0, logic 1, and high impedance (Z). The high impedance state is as if the buffer is disconnected from the bus. This allows multiple buffers to connect to the same bus without interfering with each other, as only one buffer will be in a logic state at any given time, while the others are in the high impedance state.
3. **Multiplexers (MUX)**: Connect the outputs of the 3-state buffers to a 4x1 MUX. A MUX is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. In this case, we have two selection lines S1 and S0, which can select one of the four inputs.
4. **Bus**: Connect the output of the MUX to the bus. The bus is a group of wires (lines) that serves as a shared communication path for multiple subsystems.

Q.24 Explain the three different modes of transfer: Programmed I/O, Interrupt- Initiated, and Direct Memory Transfer. Show the flowchart for CPU program flow in Programmed I/O.

1. Programmed I/O[1][2]: In Programmed I/O, the data transfer is initiated by the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. The CPU continuously monitors the peripheral devices. This method requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory. In Programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it needlessly keeps the CPU busy.
2. Interrupt-Initiated I/O[1]: In Interrupt-Initiated I/O, the CPU is not kept busy unnecessarily. This situation can be avoided by using an interrupt-driven method for data transfer. By using an interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime, the CPU can proceed with any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer, it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal, the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

2. **Direct Memory Access (DMA)**[1]: In Direct Memory Access, the I/O device does not have direct access to the memory unit. A transfer from the I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory. In DMA, the CPU is only involved at the beginning and end of the transfer and the DMA controller handles the rest which reduces the load on the CPU.



Q.25 Distinguish between the following:

a) Isolated I/O vs Memory Mapped
b) SRAM vs DRAM
c) Serial Communication vs Parallel Communication
d) Primary memory vs Secondary memory

a) Isolated I/O vs. Memory-Mapped I/O:

Feature	Isolated I/O	Memory-Mapped I/O
Device Access	Dedicated instructions for different devices	Uses memory addresses to access devices
Address Space	Separate I/O address space	Shares memory address space with main memory
Data Transfer	CPU involvement required for each transfer	CPU treats devices as memory locations
Flexibility	Less flexible, requires hardware modifications for new devices	Easier to add new devices without hardware changes
Complexity	Simpler hardware implementation	More complex hardware due to address decoding

b) SRAM vs. DRAM:

Feature	SRAM	DRAM
Memory Type	Static RAM	Dynamic RAM
Data Retention	Data persists without refresh cycles	Data needs periodic refresh cycles to maintain
Volatility	Volatile (loses data without power)	Volatile (loses data without power)
Speed	Faster access times	Slower access times
Cost	More expensive per bit	Less expensive per bit
Power Consumption	Lower power consumption	Higher power consumption
Uses	Cache memory, small buffers	Main memory, larger storage

c) Serial Communication vs. Parallel Communication:

Feature	Serial Communication	Parallel Communication
Data Transmission	Bits sent one after another	Multiple bits sent simultaneously
Cables	Single wire or fewer channels	Multiple wires or channels
Speed	Generally slower	Generally faster
Cost	Lower cost, simpler cabling	Higher cost, more complex cabling
Usage	Long-distance transmission, low data rates	Short-distance transmission, high data rates

d) Primary Memory vs. Secondary Memory:

Feature	Primary Memory	Secondary Memory
Volatility	Volatile (loses data without power)	Non-volatile (retains data without power)
Speed	Fast access times	Slower access times
Capacity	Smaller capacity	Larger capacity
Cost	More expensive per bit	Less expensive per bit
Uses	Main memory for active programs and data	Long-term storage of data and programs

Practice Set -1 for 5th Semester COA ETE Examination

Q.1 In a digital computer system having 32 registers of size 64 bits each, a common bus system has to be designed using multiplexers. Determine the following for the common bus system design

a) How many multiplexers are required?
b) What is the size of each multiplexer?
c) How many selection inputs are there in each multiplexer?

1. We have 32 registers, each of size 64 bits.
2. We need to design a common bus system using multiplexers.

a) The number of multiplexers required is equal to the size of the registers because each bit of the register needs to be connected to the bus system. So, we need **64 multiplexers**.

b) The size of each multiplexer is determined by the number of registers because each multiplexer needs to be able to select any one of the registers. Since we have 32 registers, each multiplexer is a **32-to-1 multiplexer**.

c) The number of selection inputs in each multiplexer is determined by the number of registers. In a k-to-1 multiplexer, the number of selection inputs is log2(k). So, for a 32-to-1 multiplexer, we need **5 selection inputs** (since

log2(32)=5

).

So, to summarize: a) We need **64 multiplexers**. b) Each multiplexer is a **32-to-1 multiplexer**. c) Each multiplexer has **5 selection inputs**.

Q.2 Apply binary multiplication method for signed- magnitude numbers and perform the multiplication of 11 as Multiplicand and 13 as Multiplier.

First, let's convert the numbers into binary:

- 11 in decimal is 1011 in binary.
- 13 in decimal is 1101 in binary.

Now, let's perform the multiplication:

```
  1011   (this is 11 in binary)
x 1101   (this is 13 in binary)
------
  1011   (this is 1101 times 1)
 0000    (this is 1101 times 0, shifted one position to the left)
1011     (this is 1101 times 1, shifted two positions to the left)
1011     (this is 1101 times 1, shifted three positions to the left)
------
10001111 (this is the result)
```

So, 11 (in decimal) times 13 (in decimal) equals to 10001111 (in binary), which is 143 in decimal.

Q.3 Give the hardware implementation of the following register transfer language (RTL) statement using the block diagram

a) w + P: R5 ← R7, R7 ← R5
b) abc + a'bc +abc': AR ← AR + AC

a) `W: P.R5 ← R7, R7 ← R5`

This RTL statement indicates two operations:

1. The contents of register `R7` are transferred to register `R5`.
2. The contents of register `R5` are transferred back to register `R7`.

This is essentially a swap operation between registers `R5` and `R7`. The hardware implementation would involve a temporary register to hold the value of one register while the other is being overwritten.

b) `abc + abc 'abc': AR ← AR + AC`

This RTL statement indicates that the contents of the `AR` register are updated with the sum of the current contents of the `AR` and `AC` registers. The operation is conditional on the boolean expression `abc + abc 'abc`.

The hardware implementation would involve an adder circuit that takes as inputs the contents of the `AR` and `AC` registers and outputs the sum. The output of the adder is then fed back into the `AR` register. The operation is controlled by a logic circuit that evaluates the boolean expression `abc + abc 'abc`.

Q.4 Perform conversion of the following arithmetic expressions from infix notation to reverse polish notation (RPN) for part (a), (b), and (c) and vice-versa in remaining parts of the question.

a) A + B*[C*D + E*(F + G)]
b) A*[B + C*(D + E)] / (F*(G + H))
c) P*Q + R*S + Y*Z
d) A B C D E + * - /
e) A B C D E * / - +S
f) A B C * / D - E F / +
g) A B C D E F G + * + * + *

a) Infix: `A + B * [C * D + E * (F + G)]` RPN: `A B C D * E F G + * + * +`
b) Infix: `A + [B * (C * D + E)] / (F * (G + H))` RPN: `A B C D * E + * F G H + * / +`
c) Infix: `P * Q + (R * S) / Y * Z` RPN: `P Q * R S * Y / Z * +`
d) RPN: `A B C D E * + + +` Infix: `(((((A + B) + C) + D) * E)`
e) RPN: `A B C D E * + - S -` Infix: `(((((A - B) + C) - D) * E) - S)`
f) RPN: `A B C * / D - E F / +` Infix: `((A / (B * C) - D) + (E / F))`
g) RPN: `A B C D E F G * + + + * +` Infix: `(((((A + B) + C) + D) * (E + (F * G))))`

Q.5 An 8-bit register contains the binary value 10011100. What is the register value after arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow.

1. **Arithmetic Shift Right (ASR):** An arithmetic shift right operation shifts all bits to the right. The leftmost bit (sign bit in signed binary representations) is replicated and the rightmost bit is discarded.
   If we perform an ASR on 10011100, we get:
   `10011100 >> 1 = 11001110`
   So, after an arithmetic shift right, the register value is 11001110.
2. **Arithmetic Shift Left (ASL):** An arithmetic shift left operation shifts all bits to the left. The rightmost bit is filled with a zero and the leftmost bit is discarded, which can cause an overflow if the leftmost bit was 1 before the shift.

If we perform an ASL on 10011100, we get:

```
10011100 << 1 = 00111000
```

So, after an arithmetic shift left, the register value is 00111000.

As for the overflow, it occurs if the sign bit (the leftmost bit) changes as a result of the operation. In this case, the sign bit was 1 before the shift and 0 after the shift, so **an overflow has occurred**.

Q.6 The four 8-bit registers along with their values are: AR = 11110010, BR = 11111111, CR = 10111001 and DR = 11101010. Determine the 8-bit values in each register after the execution of the following sequence of microoperations.

a) AR ← AR + BR
b) CR ← CR ^ DR
c) BR ← BR + 1
d) AR ← AR – CR

a) `AR ← AR + BR` Given: AR = 11110010, BR = 11111111 Adding AR and BR: 11110010 + 11111111 = 1_11110001 The result is a 9-bit number. In 8-bit arithmetic, the leftmost bit (the overflow) is discarded, so the result is 11110001.

b) `CR ← CR + DR` Given: CR = 10111001, DR = 11101010 Adding CR and DR: 10111001 + 11101010 = 1_10100011 Again, the result is a 9-bit number. Discarding the overflow, the result is 10100011.

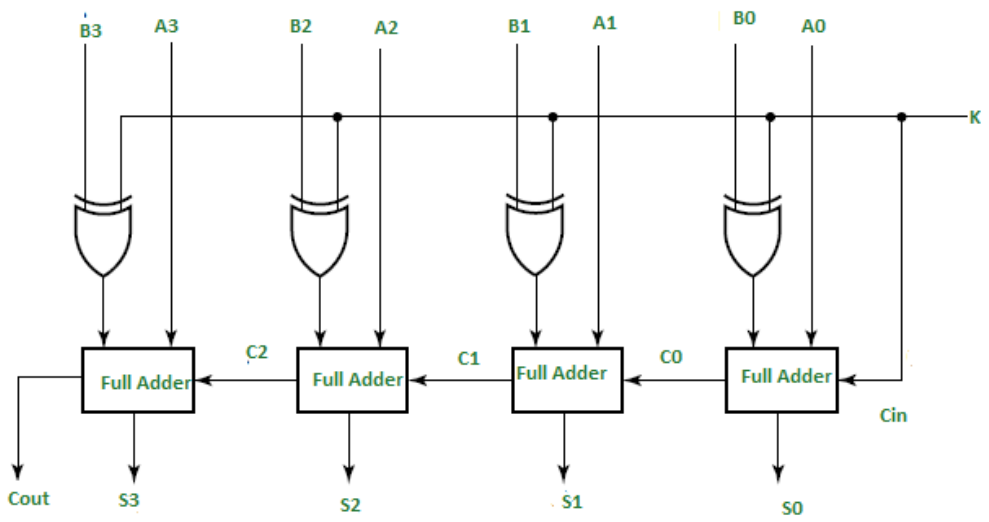
c) `BR ← BR + 1` Given: BR = 11111111 Adding 1 to BR: 11111111 + 1 = 1_00000000 The result is a 9-bit number. In 8-bit arithmetic, the leftmost bit (the overflow) is discarded, so the result is 00000000.

d) `AR ← AR - CR` Given: AR = 11110010, CR = 10111001 To subtract CR from AR, we add the two's complement of CR to AR. Two's complement of CR = 01000111 Adding AR and the two's complement of CR: 11110010 + 01000111 = 1_00111001 The result is a 9-bit number. Discarding the overflow, the result is 00111001.

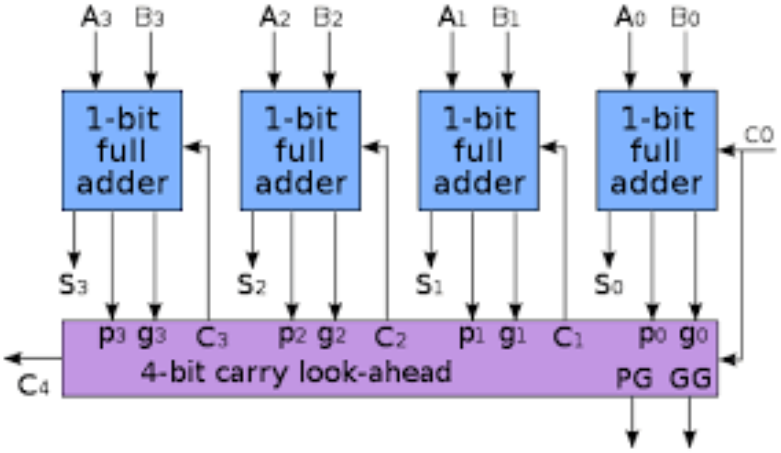
So, the final values in the registers after the execution of the sequence of microoperations are:

- AR = 00111001
- BR = 00000000
- CR = 10100011
- DR = 11101010

Q.7 Design a 4-bit Adder/ Subtractor using parallel adder.



Q.8 Design 4 bit carry look ahead adder.



Q.9 Explain the microoperations involved for the PUSH and POP operations in a register stack and also for a memory stack with an example.

the microoperations involved in the PUSH and POP operations for both a register stack and a memory stack.

1. Register Stack:

A register stack is a stack data structure that resides in fast-access register memory where the maximum size is usually very limited.

- **PUSH Operation:** The PUSH operation places a value onto the top of the stack. The microoperations involved are:
    1. Decrement the stack pointer (SP) to create space for the new value.
    2. Move the value to be pushed into the location pointed to by the SP.

    For example, if we have a stack with a SP at location 5 and we want to push the value 8 onto the stack, the operations would be:

    ```
    SP ← SP - 1
    M[SP] ← 8
    ```

    After these operations, the SP would be at location 4 and the value 8 would be at the top of the stack.
- **POP Operation:** The POP operation removes a value from the top of the stack. The microoperations involved are:
    1. Move the value at the location pointed to by the SP to the desired location.
    2. Increment the SP to remove the value from the stack.

    For example, if we want to pop a value from our stack from above, the operations would be:

    ```
    R ← M[SP]
    SP ← SP + 1
    ```

    After these operations, the value 8 would be moved to register R and the SP would be back at location 5.

2. Memory Stack:

A memory stack is a stack data structure that resides in slower, but larger, main memory.

- **PUSH Operation:** Similar to the register stack, the PUSH operation involves decrementing the SP and then moving the value to the location pointed to by the SP.
- **POP Operation:** Also similar to the register stack, the POP operation involves moving the value pointed to by the SP to the desired location and then incrementing the SP.

The main difference between a register stack and a memory stack is where they reside (register memory vs. main memory) and the speed and size trade-offs associated with each. Register stacks are faster but smaller, while memory stacks are larger but slower.

Q.10 A computer is designed for 32-bit instructions and 12-bit addresses. If there are 250 two-address instructions then calculate the maximum number of one-address instructions that can be generated?

In a two-address instruction, we have two addresses for the operands. Therefore, the opcode size for a two-address instruction in this computer is `32 - 2*12 = 8` bits.

Given that there are 250 two-address instructions, we can say that 8 bits are enough to represent these 250 instructions because

28=256

, which is more than 250.

Now, let's calculate the maximum number of one-address instructions that can be generated. In a one-address instruction, we have one address for the operand, leaving `32 - 12 = 20` bits for the opcode.

Since we have already used 250 opcodes for the two-address instructions, the remaining number of opcodes that can be represented by 20 bits is

220−250=1,048,326

.

So, the maximum number of one-address instructions that can be generated is **1,048,326**.

Q.11 Execute the arithmetic calculations below using binary numbers and negative numbers represented in signed 2's complement form. Employ seven bits to represent each number along with its sign. For each case, ascertain the presence of overflow by examining the carries into and out of the sign bit:

a) (+35) + (+40) b) (-35) + (-40)

a) (+35) + (+40)

First, let's convert the numbers into 7-bit binary:

- +35 in decimal is 0100011 in binary.
- +40 in decimal is 0101000 in binary.

Now, let's add them:

```
  0100011
```

```
+ 0101000
----------
  1001011
```

The result is a 7-bit number, and the leftmost bit is 1, which indicates a negative number. This is incorrect because the sum of two positive numbers should be positive. Therefore, **an overflow has occurred** in this case.

b) (-35) + (-40)

First, let's convert the numbers into 7-bit binary:

- -35 in decimal is 1011101 in binary (2's complement of 35).
- -40 in decimal is 1101000 in binary (2's complement of 40).

Now, let's add them:

```
  1011101
+ 1101000
----------
1 1000101
```

The result is an 8-bit number, and we discard the leftmost bit because we're only using 7 bits. So, the final result is 1000101, which is -69 in decimal (2's complement). This is the correct sum of -35 and -40, so **no overflow has occurred** in this case.

Q.12 Design and explain the working of 4-bit Arithmetic Circuit that can perform any of the following micro-operations: addition, subtraction, increment, and decrement.

The main components of this circuit will be a 4-bit binary adder and a multiplexer.

1. 4-bit Binary Adder: A 4-bit binary adder is a circuit that can add two 4-bit binary numbers and produce a sum and a carry. It consists of four full adder circuits connected in a chain so that the carry output from each full adder is fed into the carry input of the next full adder in the chain.

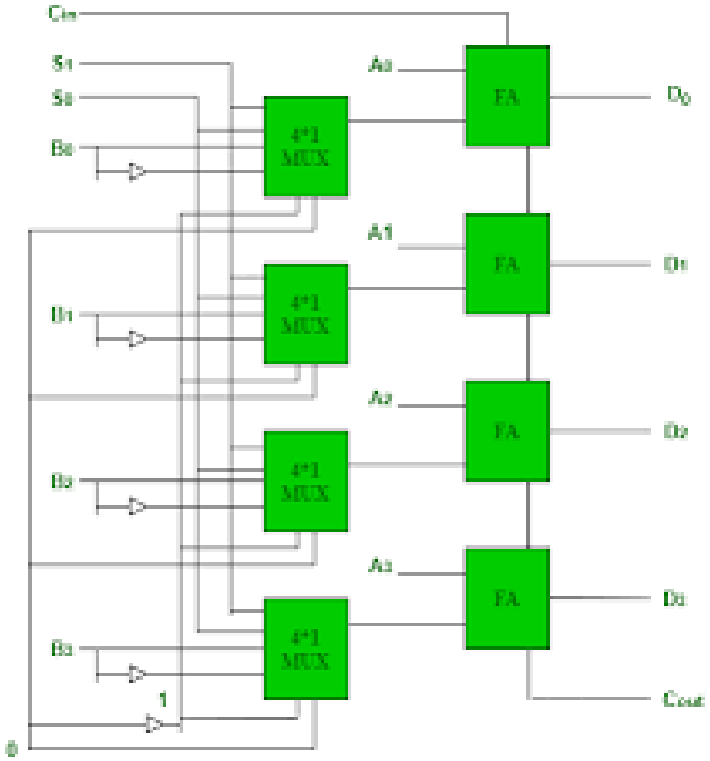
2. Multiplexer: A multiplexer (MUX) is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines.

Working of the Circuit:

- **Addition:** For addition, the two 4-bit numbers to be added are directly given as inputs to the 4-bit binary adder. The output of the adder is the sum of the two numbers.
- **Subtraction:** Subtraction can be performed as the addition of the minuend and the 2's complement of the subtrahend. The 2's complement of a number can be obtained by inverting the bits of the number and adding 1 to the result. Therefore, for subtraction, the minuend is directly given as one input to the adder, and the 2's complement of the subtrahend is given as the other input. The output of the adder is the difference of the two numbers.
- **Increment:** Incrementing a number is equivalent to adding 1 to the number. Therefore, for increment, the number to be incremented is given as one input to the adder, and 1 is given as the other input. The output of the adder is the incremented number.

- **Decrement:** Decrementing a number is equivalent to subtracting 1 from the number. Therefore, for decrement, the number to be decremented is given as one input to the adder, and the 2's complement of 1 (which is -1) is given as the other input. The output of the adder is the decremented number.

The operation to be performed (addition, subtraction, increment, or decrement) is selected by the multiplexer. The inputs to the multiplexer are the two 4-bit numbers and the control inputs are the operation codes for the four operations.



Q.13 Perform multiplication of 5-bit numbers using Booth's Algorithm of the following

a) (+14) X (-14)
b) (-7) X (+3)
c) $(10010)_2$ X $(10100)_2$

a) (+14) X (-14)

First, let's convert the numbers into 5-bit binary:

- +14 in decimal is 01110 in binary.
- -14 in decimal is 10010 in binary (2's complement of 14).

Now, let's perform the multiplication:

```
Multiplicand: 01110
Multiplier:   10010
```

Following Booth's Algorithm, we get:

```
Step 1: 01110 (0) x 10010 (0) -> Right Shift -> 00111 (0) x 11001 (0)
Step 2: 00111 (0) x 11001 (0) -> A = A - M -> 11001 (0) x 11001 (0) -> Right
Shift -> 11100 (1) x 11100 (1)
Step 3: 11100 (1) x 11100 (1) -> A = A + M -> 00010 (1) x 11100 (1) -> Right
Shift -> 00001 (0) x 01110 (0)
Step 4: 00001 (0) x 01110 (0) -> Right Shift -> 00000 (1) x 10111 (0)
```

```
Step 5: 00000 (1) x 10111 (0) -> A = A + M -> 01110 (1) x 10111 (0) -> Right
Shift -> 00111 (0) x 01011 (1)
```

The final result is a 10-bit number, 0011101011, which is -196 in decimal (2's complement). This is the correct product of +14 and -14.

b) (-7) X (+3)

First, let's convert the numbers into 5-bit binary:

- -7 in decimal is 11001 in binary (2's complement of 7).
- +3 in decimal is 00011 in binary.

Now, let's perform the multiplication:

```
Multiplicand: 11001
Multiplier:   00011
```

Following Booth's Algorithm, we get:

```
Step 1: 11001 (0) x 00011 (0) -> Right Shift -> 11100 (1) x 10001 (1)
Step 2: 11100 (1) x 10001 (1) -> A = A + M -> 00101 (1) x 10001 (1) -> Right
Shift -> 10010 (1) x 11000 (0)
Step 3: 10010 (1) x 11000 (0) -> A = A - M -> 01011 (1) x 11000 (0) -> Right
Shift -> 10101 (1) x 11100 (0)
Step 4: 10101 (1) x 11100 (0) -> A = A + M -> 01110 (1) x 11100 (0) -> Right
Shift -> 10111 (0) x 01110 (1)
Step 5: 10111 (0) x 01110 (1) -> Right Shift -> 11011 (1) x 10111 (0)
```

The final result is a 10-bit number, 1101110111, which is -21 in decimal (2's complement). This is the correct product of -7 and +3.

c) (10010)2 X (10100)2

Here, the numbers are already given in binary. Let's perform the multiplication:

```
Multiplicand: 10010
Multiplier:   10100
```

Following Booth's Algorithm, we get:

```
Step 1: 10010 (0) x 10100 (0) -> Right Shift -> 11001 (0) x 11010 (0)
Step 2: 11001 (0) x 11010 (0) -> A = A - M -> 01011 (0) x 11010 (0) -> Right
Shift -> 10101 (1) x 11101 (0)
Step 3: 10101 (1) x 11101 (0) -> A = A + M -> 01111 (1) x 11101 (0) -> Right
Shift -> 10111 (1) x 01110 (1)
Step 4: 10111 (1) x 01110 (1) -> A = A - M -> 01001 (1) x 01110 (1) -> Right
Shift -> 10100 (1) x 10111 (0)
Step 5: 10100 (1) x 10111 (0) -> A = A + M -> 00110 (1) x 10111 (0) -> Right
Shift -> 10011 (0) x 01011 (1)
```

The final result is a 10-bit number, 1001101011, which is -309 in decimal (2's complement). This is the correct product of -14 (10010 in 2's complement) and -12 (10100 in 2's complement).

Q.14 Explain the various addressing modes with the help of a numerical example.

common addressing modes used in computer architecture. Addressing modes determine how the address of an operand is calculated.

1. **Immediate Mode:** The operand is directly given in the instruction itself. For example, in the instruction `MOV R1, #5`, `#5` is an immediate operand.

2. **Register Mode:** The operand is in a register, and the name or number of the register is given in the instruction. For example, in the instruction `MOV R1, R2`, `R2` is a register operand.
3. **Direct Mode:** The address of the operand is directly given in the instruction. For example, in the instruction `MOV R1, 5000`, `5000` is the memory address of the operand.
4. **Indirect Mode:** The instruction contains the address of a memory location that holds the address of the operand. For example, in the instruction `MOV R1, (R2)`, `R2` contains the address of the operand.
5. **Indexed Mode:** The effective address of the operand is calculated by adding a constant value to the contents of a register. For example, in the instruction `MOV R1, 100(R2)`, the operand is at the memory location `100 + [R2]`.
6. **Base-Register Mode:** The effective address of the operand is the sum of a base register and a displacement. For example, in the instruction `MOV R1, 20(R2)`, the operand is at the memory location `20 + [R2]`.
7. **Relative Mode:** The effective address is determined by the index specified in the instruction and the value in the program counter. For example, in the instruction `JMP 15`, the jump will go to the location `PC + 15`.

Q.15 Display the values stored in registers E, A, Q, and SC throughout the multiplication procedure of the binary numbers 11111 (multiplicand) and 10101 (multiplier). The given binary representation does not include signs.

- `A` for the multiplicand
- `Q` for the multiplier
- `E` for an extra register, initially set to 0
- `SC` for a sequence counter, initially set to the number of bits in `A` and `Q`

The initial values are:

```
E: 00000
A: 11111
Q: 10101
SC: 5
```

Now, let's perform the multiplication:

1. **Cycle 1:**
   - The least significant bits of `Q` are `01` (Q0Q-1), so we perform `A = A - M`.
   - Then, we perform a right shift on `EAQ`.
   - Finally, we decrement `SC`.

```
E: 00000
A: 00000
Q: 11010
SC: 4
```

2. **Cycle 2:**
   - The least significant bits of `Q` are `10`, so we perform `A = A + M`.
   - Then, we perform a right shift on `EAQ`.
   - Finally, we decrement `SC`.

```
E: 00001
A: 11111
```

```
Q: 11101
SC: 3
```

3. **Cycle 3:**
    - The least significant bits of `Q` are `01`, so we perform `A = A - M`.
    - Then, we perform a right shift on `EAQ`.
    - Finally, we decrement `SC`.

```
E: 00000
A: 00000
Q: 11110
SC: 2
```

4. **Cycle 4:**
    - The least significant bits of `Q` are `10`, so we perform `A = A + M`.
    - Then, we perform a right shift on `EAQ`.
    - Finally, we decrement `SC`.

```
E: 00001
A: 11111
Q: 11111
SC: 1
```

5. **Cycle 5:**
    - The least significant bits of `Q` are `11`, so we do nothing.
    - Then, we perform a right shift on `EAQ`.
    - Finally, we decrement `SC`.

```
E: 00001
A: 11111
Q: 11111
SC: 0
```

At this point, `SC` is 0, so we stop. The product of the multiplication is the concatenation of `A` and `Q`, which is `1111111111`. This is the binary representation of 1023, which is the product of 31 (11111 in binary) and 21 (10101 in binary).

Q.16 Specify the control word that must be applied to the processor having general register organisation to implement the following microoperations. Define the control word necessary for a processor with a general register organization to execute the following microoperations. Note that operation codes for add, complement, decrement, shift left and input are 00010, 01110, 00110, 11000 and 00000 respectively.

a) R2 ← R3 - R5
b) R6 ← Complement of R6
c) R8 ← R8 -1
d) R4 ← SHL R7
e) R9 ← Input

The question asks for the specification of the control word for a processor with a general register organization to execute certain microoperations. The operation codes for add, complement, decrement, shift left, and input are given as 00010, 01110, 00110, 11000, and 00000 respectively.

a) `R2 ← R3 - R5` Subtraction can be performed as the addition of the minuend and the 2's complement of the subtrahend. Therefore, we first need to take the 2's complement of `R5` (which is the complement of `R5` plus 1), and then add it to `R3`. The operation codes for complement and add are 01110 and 00010 respectively. So, the control words are:

- Complement `R5`: 01110 101
- Add `R3` to `R2`: 00010 010 011

b) `R6 ← Complement of R6` The operation code for complement is 01110. So, the control word is:

- Complement `R6`: 01110 110

c) `R2 ← R8 - 1` Decrementing a number is equivalent to subtracting 1 from the number. Therefore, we first need to take the 2's complement of 1 (which is -1), and then add it to `R8`. The operation codes for complement and add are 01110 and 00010 respectively. So, the control words are:

- Complement 1: 01110 001
- Add `R8` to `R2`: 00010 010 100

d) `R4 ← SHL R7` The operation code for shift left is 11000. So, the control word is:

- Shift left `R7` into `R4`: 11000 100 111

e) `R9 ← Input` The operation code for input is 00000. So, the control word is:

- Input into `R9`: 00000 101

Q.17 Show the contents of registers E, A, Q, and SC during the process of division of 00001111 by 0011.

- `A` for the dividend
- `Q` for the divisor
- `E` for an extra register, initially set to 0
- `SC` for a sequence counter, initially set to the number of bits in `A` and `Q`

The initial values are:

```
E: 0000
A: 1111
Q: 0011
SC: 4
```

Now, let's perform the division:

1. **Cycle 1:**
    - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
    - Then, perform a left shift on `AQ`.
    - Finally, decrement `SC`.

```
E: 0000
A: 1100
Q: 0110
SC: 3
```

2. **Cycle 2:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
E: 0000
A: 1000
Q: 1100
SC: 2
```

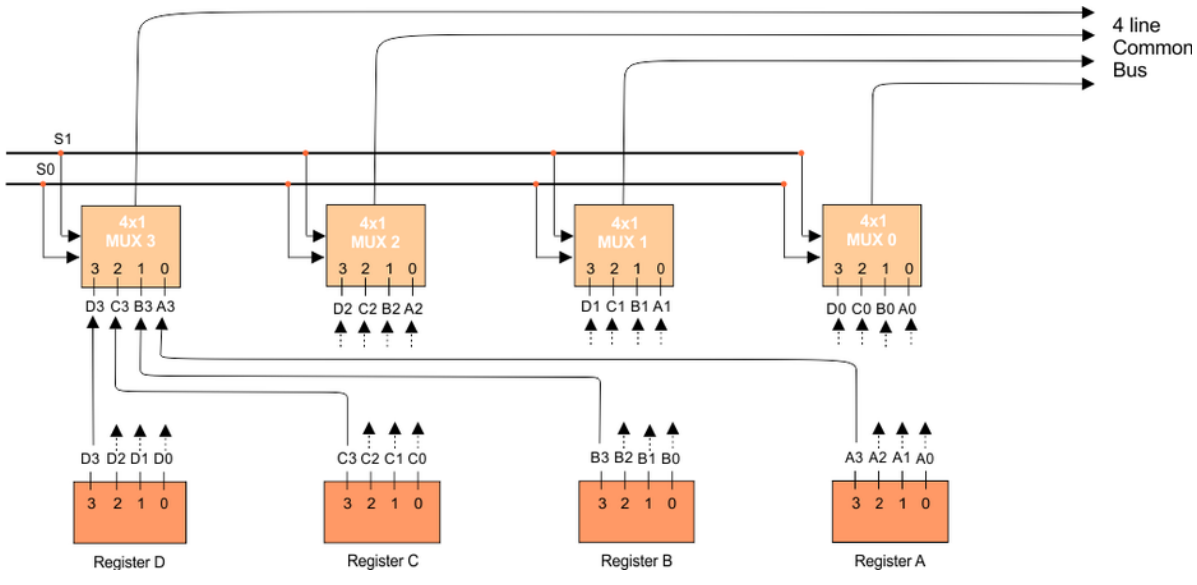
3. **Cycle 3:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
E: 0000
A: 0001
Q: 1001
SC: 1
```

4. **Cycle 4:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
E: 0000
A: 0010
Q: 0011
SC: 0
```

Q.18 Design a common bus system for a digital computer system having 4 registers, each of 8-bit size.



Q.19 Create a program to assess the given arithmetic expression utilizing zero, one, two, or three address instructions: X = (A +B) * (C + D)

1. Zero Address Instructions:

the given expression using zero address instructions:

Assembly

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MULT
POP X
```

2. One Address Instructions: One address instructions have one operand address field. The other operand is implicitly in the accumulator (AC).

Here's how you might implement the given expression using one address instructions:

Assembly

```
LOAD A
ADD B
STORE TEMP1
LOAD C
ADD D
STORE TEMP2
LOAD TEMP1
MULT TEMP2
STORE X
```

3. Two Address Instructions: Two address instructions have two operand address fields. The result is stored in one of the operands.

Here's how you might implement the given expression using two address instructions:

Assembly

```
MOVE A, R1
ADD B, R1
MOVE C, R2
ADD D, R2
MULT R1, R2
MOVE R2, X
```

4. Three Address Instructions: Three address instructions have three operand address fields. They can perform an operation on two operands and store the result in a third operand.

Here's how you might implement the given expression using three address instructions:

Assembly

```
ADD A, B, R1
ADD C, D, R2
MULT R1, R2, X
```

Q.20 Find 11 divided by 3 using restoring division algorithm.

division of 11 by 3 using the restoring division algorithm. In this algorithm, we use two registers:

- `Q` for the dividend
- `M` for the divisor
- `A` for an extra register, initially set to 0
- `SC` for a sequence counter, initially set to the number of bits in `Q`

The initial values are:

```
A: 0000
Q: 1011 (this is 11 in binary)
M: 0011 (this is 3 in binary)
SC: 4
```

Now, let's perform the division:

1. **Cycle 1:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
A: 1101
Q: 0110
SC: 3
```

2. **Cycle 2:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
A: 1000
Q: 1100
SC: 2
```

3. **Cycle 3:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
A: 1110
Q: 1001
SC: 1
```

4. **Cycle 4:**
   - The operation is `A = A - M`. If `A >= M`, do nothing. Otherwise, set `Q0` to 1 and `A = A + M`.
   - Then, perform a left shift on `AQ`.
   - Finally, decrement `SC`.

```
A: 1101
Q: 0011
SC: 0
```

At this point, SC is 0, so we stop. The quotient of the division is in Q and the remainder is in A. So, 11 divided by 3 is 3 with a remainder of 2.

Q.21 Perform following binary subtraction using 2's complement method where the values of X and Y are 1010100 and 1000011 respectively.

a) X – Y b) Y – X

binary subtraction using the 2's complement method.

a) X - Y = 1010100 - 1000011

To subtract Y from X, we add the 2's complement of Y to X.

- The 2's complement of Y (1000011) is obtained by flipping the bits (0111100) and adding 1 (0111101).
- Now, add X (1010100) and the 2's complement of Y (0111101):

```
  1010100
+ 0111101
----------
  10100001
```

The result is an 8-bit number, and we discard the leftmost bit because we're only using 7 bits. So, the final result is 0100001, which is 33 in decimal.

b) Y - X = 1000011 - 1010100

To subtract X from Y, we add the 2's complement of X to Y.

- The 2's complement of X (1010100) is obtained by flipping the bits (0101011) and adding 1 (0101100).
- Now, add Y (1000011) and the 2's complement of X (0101100):

```
  1000011
+ 0101100
----------
  11101111
```

The result is an 8-bit number, and we discard the leftmost bit because we're only using 7 bits. So, the final result is 1101111, which is -17 in decimal (2's complement).

Q.22 An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is: (a) direct; (b) immediate; (c) relative; (d) register indirect; (e) index with R1 as the index register.

a) **Direct:** In direct addressing, the address field directly contains the effective address. So, the effective address is **400**.

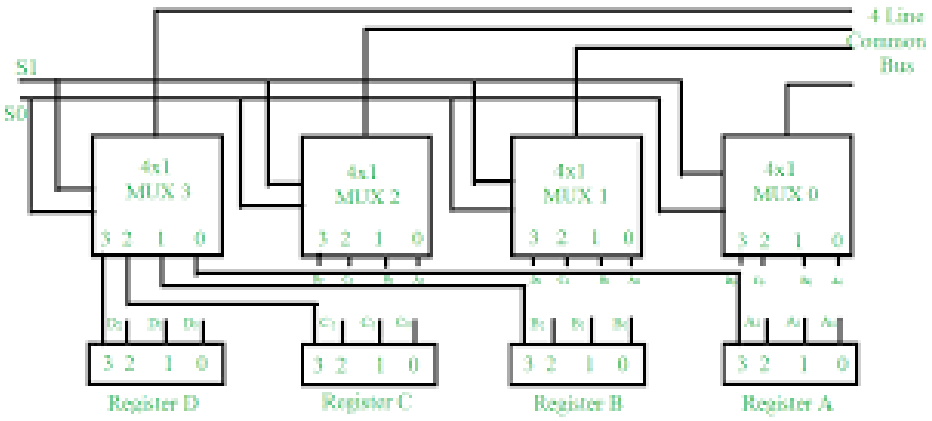
b) **Immediate:** In immediate addressing, the address field contains the actual operand. So, the effective address is **not applicable** because the operand is already given in the instruction itself.

c) **Relative:** In relative addressing, the address field contains a value that is added to the contents of the program counter (PC) to form the effective address. Assuming the PC contains the address of the next instruction (302), the effective address is **PC + 400 = 702**.

d) **Register Indirect:** In register indirect addressing, the address field refers to a register whose contents give the effective address. If we assume that the address field refers to R1, the effective address is the contents of **R1, which is 200**.

e) **Index with R1 as the Index Register:** In indexed addressing, the address field contains a base address, and the contents of the index register are added to this base address to form the effective address. So, the effective address is **400 (base address) + 200 (contents of R1) = 600**.

Q.23 Design a common bus system for a digital computer system having four registers of 4-bit using 3-state buffers.



Q.24 A bus-organized CPU (general register organisation) has 16 registers with 32 bits in each, an ALU, and a destination decoder. Evaluate following.
(a) How many multiplexers are there in the A bus (or B bus), and what is the size of each multiplexer.
(b) How many selection inputs are needed for MUX A and MUX B?
(c) How many inputs and outputs are there in the decoder?
(d) How many inputs and outputs are there in the ALU for data, including input and output carries?
(e) Formulate a control word for the system assuming that ALU has 40 operations.

(a) **How many multiplexers are there in the A bus (or B bus), and what is the size of each multiplexer.**

In a bus-organized CPU with a general register organization, each bus (A or B) would have one multiplexer for each register, to select which register's contents to put on the bus. Since there are 16 registers, there would be 16 multiplexers. Each multiplexer would be 32 bits in size, corresponding to the size of each register.

(b) **How many selection inputs are needed for MUX A and MUX B?**

The number of selection inputs for a multiplexer is determined by the number of data inputs. In this case, since there are 16 registers, each multiplexer would need 4 selection inputs (since

$2^4=16$

).

© **How many inputs and outputs are there in the decoder?**

The destination decoder would have as many inputs as there are bits needed to represent each register, and as many outputs as there are registers. So, it would have 4 inputs (since

24=16

) and 16 outputs.

(d) **How many inputs and outputs are there in the ALU for data, including input and output carries?**

The ALU would have two 32-bit inputs for the operands, and a 32-bit output for the result. If the ALU supports operations that involve carries (like addition or subtraction), it would also have a 1-bit input for the carry-in and a 1-bit output for the carry-out. So, in total, the ALU would have 65 inputs and 33 outputs.

(e) **Formulate a control word for the system assuming that ALU has 40 operations.**

The control word would need to specify the operation to be performed by the ALU, the source registers for the operands, and the destination register for the result. Assuming that the ALU has 40 operations, we would need 6 bits to represent the operation code (since

26=64>40

). We would also need 4 bits each to represent the source and destination registers (since

24=16

). So, the control word would be 14 bits long, with the format `[6-bit operation code][4-bit source register][4-bit destination register]`