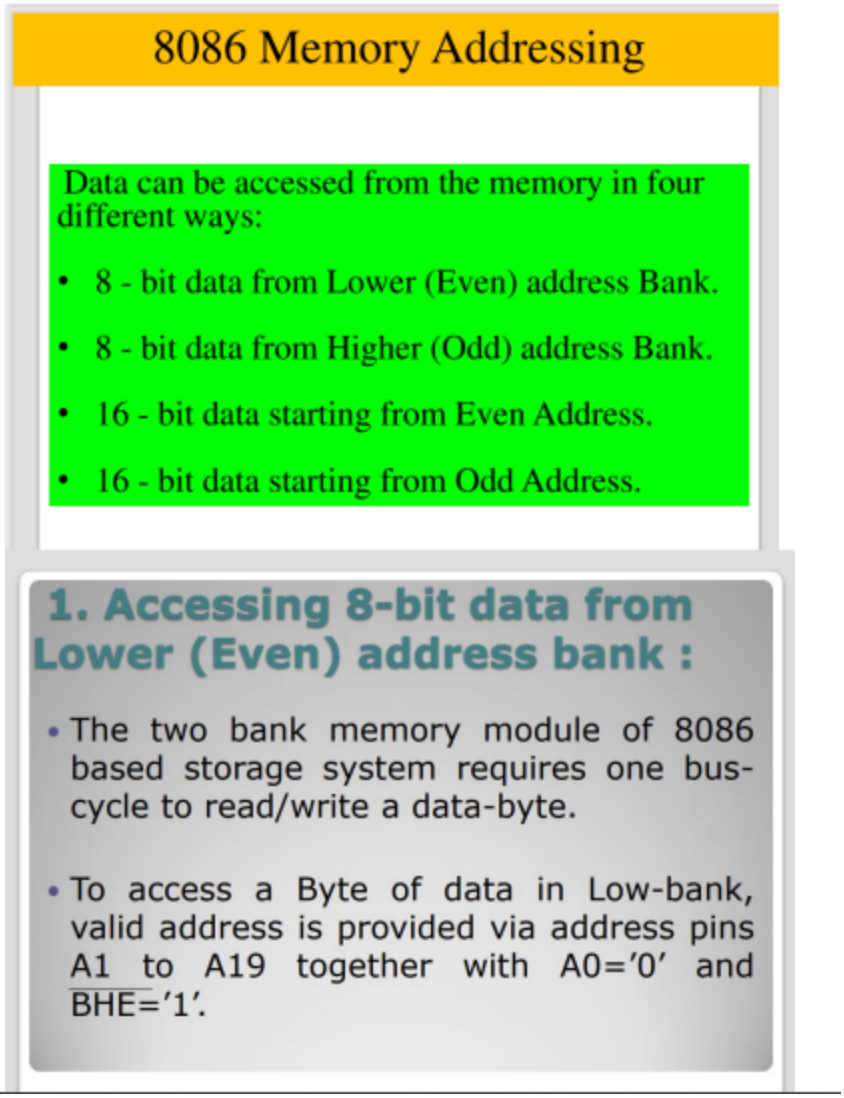
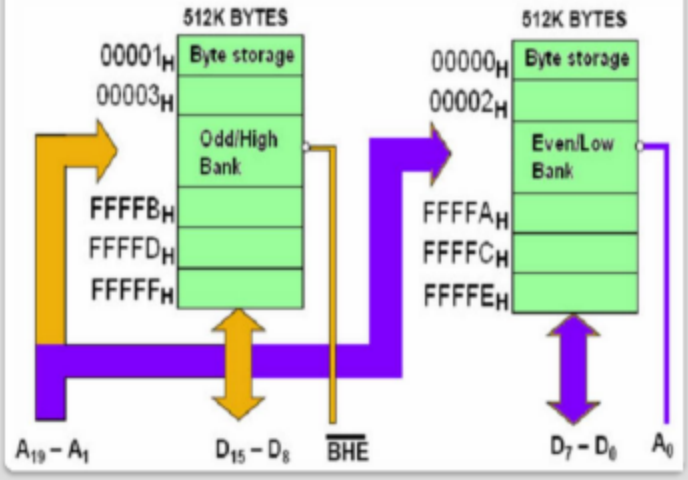
4th+5th batch final 4

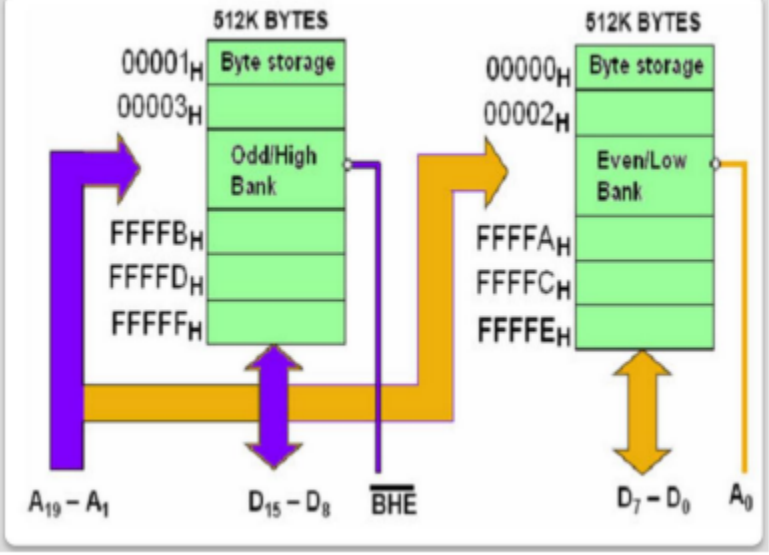
4 a) ans:


8086 Memory Addressing
Data can be accessed from the memory in four different ways:
• 8 - bit data from Lower (Even) address Bank.
• 8 - bit data from Higher (Odd) address Bank.
• 16 - bit data starting from Even Address.
• 16 - bit data starting from Odd Address.
1. Accessing 8-bit data from Lower (Even) address bank :
• The two bank memory module of 8086 based storage system requires one bus-cycle to read/write a data-byte.
• To access a Byte of data in Low-bank, valid address is provided via address pins A1 to A19 together with A0='0' and BHE='1'.



2. Accessing 8-bit data from Higher (Odd) address bank

- Similarly to access a Byte of data in High-bank, valid address in pins A1 to A19, A0='1' and $\overline{BHE}$='0' are required to access the data through D8 to D15 of the data-bus.
- These signals disable the Low bank and enable the High bank to transfer (in/out) data through D8 to D15 of the data-bus.



4 b) ans:

Looping structure:

A loop is sequence of instructions that is repeated . The number of times to repeat may be known in advance, or it depend on condition. There are three types of loop. And they are:

1.For loop
2.While loop
3. Repeat

For loop

This is a loop structure in which the loop statements are repeated a number of times (a count controlled loop). In pseudo code,

For loop_ count times Do
Statements
END-FOR

Example

* Write a count-controlled loop to display a row of 80 stars.

```
For 80 times Do
        display '*'
END_FOR
```

Code: -

```
        MOV   CX, 80     ;number of stars to display
        MOV   AH,2       ; display character function
        MOV   DL, '*'    ; character to display
TOP:
        INT    21h       ;display a star
        LOOP   TOP       ; repeat 80 times
```

The counter for the loop is the register CX which is initialized to loop _ count . Execution of the LOOP instruction causes CX to be decrement automatically, and if CX is not 0, control transfer s to destination_ label. Must precede the LOOP instruction by no more than 126 bytes.

While loop

This loop depends on a condition. In Pseudocode,

WHILE condition DO
Statements
END_WHILE

The condition is checked at the top of the loop. If true, the statements are executed; if false, the program goes on to whatever follows. It is possible that the condition will be false initially , in which case the loop body is not executed at all. The loop executes as long as the condition is true.

Example

*Write some code to count the number of characters in an input line.

```
Code:   MOV   DX, 0      ; DX count characters
        MOV   AH, 1      ; prepare to read
        INT   21H        ; character in AL
WHILE_:
        CMP   AL,0DH          ;CR?
        JE    END_WHILE       ; yes , exit
        INC   DX                ; not  CR, increment count
        INT   21H               ; read a character
        JMP   WHILE _           ; loop back
END _WHILE:
```

c

How instruction affect the flags:
SUB AX,BX [AX=50h, BX=70h];
ADD AL, BL [AL=50h,BL=70h]

Sol: The instruction "SUB AX, BX" in the x86 instruction set will subtract the contents of the BX register from the AX register, storing the result in AX. Given the values in the

square brackets, AX will contain 50h (hexadecimal) - 70h = 80h after this instruction is executed.

The instruction "ADD AL, BL" will add the contents of the AL and BL registers and store the result in AL. Given the values in the square brackets, AL will contain 50h + 70h = C0h (192 in decimal) after this instruction is executed.

[how it occure in 50h-70h=80h?
The subtraction of hexadecimal values works similarly to decimal subtraction. To subtract 70h from 50h, we need to convert both values to binary, perform the subtraction, and then convert the result back to hexadecimal.

The conversion of 50h to binary is:
50h = 0101 0000 (binary)

The conversion of 70h to binary is:
70h = 0111 0000 (binary)

Now, we perform the subtraction:
0101 0000 (50h in binary)

0111 0000 (70h in binary)
0001 0000 (result in binary)

Finally, we convert the result back to hexadecimal:
0001 0000 (binary) = 10h (hexadecimal)

So, the result of 50h - 70h = 80h in hexadecimal.
]
[how it occurs in 50h +70h=C0h?
The addition of hexadecimal values works similarly to decimal addition. To add 50h and 70h, we need to convert both values to binary, perform the addition, and then convert the result back to hexadecimal.

The conversion of 50h to binary is:
50h = 0101 0000 (binary)

The conversion of 70h to binary is:
70h = 0111 0000 (binary)

Now, we perform the addition:
0101 0000 (50h in binary)

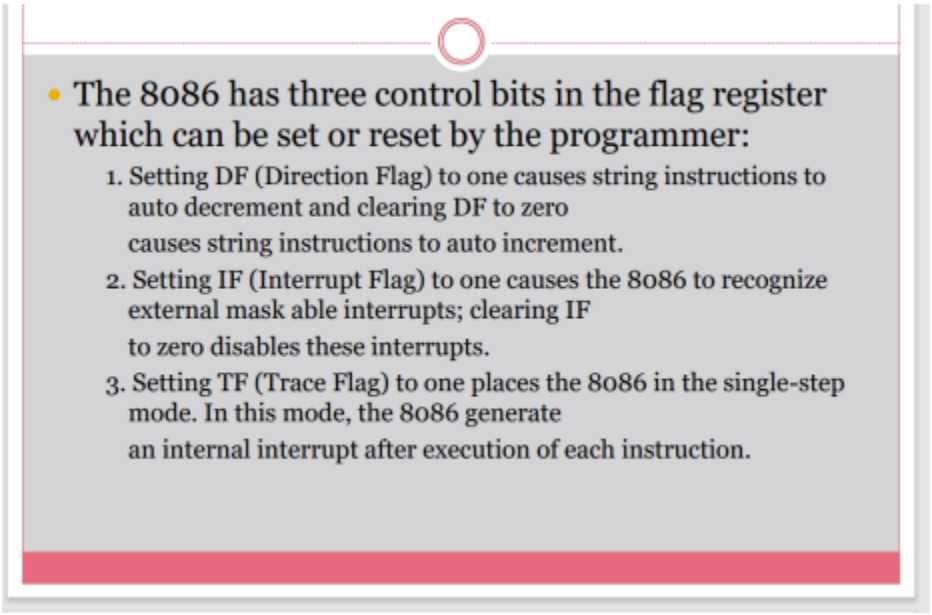
0111 0000 (70h in binary)
1110 0000 (result in binary)

Finally, we convert the result back to hexadecimal:
1110 0000 (binary) = E0h (hexadecimal)

However, since the result is greater than FFh (255 in decimal), the value stored in the AL register will be the least significant 8 bits of the result, which is E0h & 255 = C0h (hexadecimal). So, the result of 50h + 70h = C0h in hexadecimal
]

6(a)

6 a) ans:



6(b)

Write short notes on computer architecture:
i)Logical Shift ii) ROL iii) ROR iv) RCL v) arithmetic shift vi) RCR

i) Logical Shift: Logical Shift is a type of bitwise operation in which the binary representation of a number is shifted left or right by a specified number of positions. During a logical shift, the bits that are shifted out of the end are discarded, and zeros are added to the beginning.

ii) ROL (Rotate Left): ROL is a type of bitwise operation in which the binary representation of a number is rotated to the left by a specified number of positions. During a ROL operation, the bits that are shifted out of the left end are shifted back to the right end.

iii) ROR (Rotate Right): ROR is a type of bitwise operation in which the binary representation of a number is rotated to the right by a specified number of positions. During a ROR operation, the bits that are shifted out of the right end are shifted back to the left end.

iv) RCL (Rotate Carry Left): RCL is a type of bitwise operation in which the binary representation of a number is rotated to the left by a specified number of positions, with the carry bit being rotated along with the other bits.

v) Arithmetic Shift: Arithmetic Shift is a type of bitwise operation in which the binary representation of a number is shifted left or right by a specified number of positions. During an arithmetic shift, the bits that are shifted out of the end are filled with copies of the most significant bit.

vi) RCR (Rotate Carry Right): RCR is a type of bitwise operation in which the binary representation of a number is rotated to the right by a specified number of positions, with the carry bit being rotated along with the other bits.

C

Mention the mejor pins functionality of 8257. Differentiate between 8253 and 8254.

The 8257 is a Programmable DMA Controller, which provides direct memory access (DMA) services to peripheral devices. Some of the major pins and their functionality of the 8257 are:

A0-A1: These are address pins that are used to select one of the four DMA channels.

D0-D7: These are data pins that are used to transfer data between the microprocessor and the peripheral device.

READY: This is an input pin that is used by the peripheral device to signal the 8257 that it is ready to receive or transmit data.

HLDA: This is an output pin that is used by the 8257 to signal the peripheral device that the data transfer is complete.

On the other hand, the 8253 and 8254 are Programmable Interval Timers, which are used to generate precise time intervals in microprocessor-based systems. The main difference between the 8253 and 8254 is that the 8254 has a higher input clock frequency, which makes it more precise. Additionally, the 8254 has an extra mode of operation called "one-shot" mode, which generates a single time pulse, while the 8253 does not have this capability.

5th batch 3(a)

Short notes on various types of jumping instruction.

In microprocessors, jumping instructions are used to change the flow of execution of a program. There are various types of jumping instructions, including:

Unconditional Jump: An unconditional jump instruction transfers control to a specified memory location, regardless of the current status of the flags or other conditions.

Conditional Jump: A conditional jump instruction transfers control to a specified memory location based on the state of one or more flags or other conditions.

Relative Jump: A relative jump instruction transfers control to a memory location that is a specified number of bytes away from the current instruction.

Direct Jump: A direct jump instruction transfers control to a specified memory location by using an absolute address.

Register Jump: A register jump instruction transfers control to a memory location that is stored in a register.

Subroutine Jump: A subroutine jump instruction transfers control to a subroutine, which is a block of code that performs a specific task. The subroutine jump instruction also saves the return address so that the program can return to the next instruction after the subroutine is completed.

Interrupt Jump: An interrupt jump instruction transfers control to an interrupt service routine, which is a block of code that is executed in response to an interrupt request. The interrupt jump instruction saves the current state of the program so that it can be resumed after the interrupt service routine is completed.

4th batch 8 num:

a)

Short notes on various types of jumping instruction:
i) Signed Conditional Jumps
ii) Unsigned Conditional Jumps
iii) Single Flag jumps

Sol:
i) Signed Conditional Jumps: Signed conditional jumps are conditional jump instructions that compare the contents of a register to zero and make a jump based on whether the contents are positive, negative or zero. These types of jumps are used to make decisions based on the signed value of a register.

ii) Unsigned Conditional Jumps: Unsigned conditional jumps are conditional jump instructions that compare the contents of a register to zero and make a jump based on whether the contents are zero or not. These types of jumps are used to make decisions based on the unsigned value of a register.

iii) Single Flag Jumps: Single flag jumps are conditional jump instructions that make a jump based on the state of a single processor flag, such as the zero flag or the carry flag. These types of jumps are used to make decisions based on the results of arithmetic or logical operations, such as whether a value is zero or whether a carry occurred during an operation.

b) Write an assembly language program for comparing two string and concating two string (eta 5th batch er 5 er c tew ashche)

Here's an example of an assembly language program for comparing two strings and concatenating two strings, written in x86 Assembly language:

```
; Comparing two strings
section .data
string1 db 'Hello, World!', 0
string2 db 'Hello, World!', 0
result db 0

section .text
global _start

_start:
; Compare the two strings
mov eax, 0
mov ebx, 0
mov ecx, 0
mov edx, 0
```

```
mov eax, [string1]
mov ebx, [string2]
cld
repe cmpsb
jz strings_are_equal ; Jump if the strings are equal

; Strings are not equal
mov [result], 1
jmp end_of_program
```

```
strings_are_equal:
; Strings are equal
mov [result], 0

end_of_program:
; Concatenating two strings
mov eax, 4
mov ebx, 1
mov ecx, string1
mov edx, 14
int 0x80
```

```
mov eax, 4
mov ebx, 1
mov ecx, string2
mov edx, 14
int 0x80

; Exit the program
mov eax, 1
xor ebx, ebx
int 0x80
```

4th+5th batch 5 no

A

```
;program to reverse a string
 .model small
 .stack 100h
 .data
  s db 'Bangladesh'



 .code
 main proc

   mov ax,@data
   mov ds,ax

   mov si,offset s ;the address of 1st index of 's' is now into the source index

   mov cx,7

   l1:                          ;l1->level 1
   mov bx,[si]                  ;stored in a stack
   push bx
   inc si
   loop l1
```

```
  mov cx,7

  l2:
   pop dx
   mov ah,2
   int 21h
  loop l2


 mov ah,4ch
 int 21h
 main endp
 end main
```

(b)

Write assembly code statement for each of the high level language assignments statements:

i) A= C*A-7/B

ii)B=(A-B)*B/10

sol:

Here's an example of assembly code statements for each of the high-level language assignment statements, written in x86 Assembly language:

```
i) A = C * A - 7 / B
mov eax, C ; Move C into eax register
mul A ; Multiply eax with A and store result in eax
mov ebx, 7 ; Move 7 into ebx register
div B ; Divide eax by B and store result in eax
sub eax, ebx ; Subtract ebx from eax and store result in eax
mov A, eax ; Move result from eax to A

ii) B = (A - B) * B / 10
mov eax, A ; Move A into eax register
sub eax, B ; Subtract B from eax and store result in eax
mov ebx, B ; Move B into ebx register
mul ebx ; Multiply eax with ebx and store result in eax
mov ebx, 10 ; Move 10 into ebx register
div ebx ; Divide eax by ebx and store result in eax
mov B, eax ; Move result from eax to B
```

C

```
section .data

section .text

global _start

_start:

mov cx, 16 ; Set the loop counter to 16 (the number of bits in a register)

mov dx, 0 ; Set the count of 0 bits to 0

count_0_bits:

shr bx, 1 ; Shift the value in BX right by 1 bit

jnc skip ; If the carry flag is not set, jump to the skip label

inc dx ; If the carry flag is set, increment the count of 0 bits

skip:

loop count_0_bits ; Repeat the loop 16 times

mov ah, 0x0E ; Set the value of AH to 0x0E for the display interrupt

mov al, dl ; Copy the count of 0 bits to AL

int 0x10 ; Call the display interrupt

mov ax, 0x4C00 ; Set the AX register to 0x4C00 to end the program

int 0x21 ; Call the end program interrupt
```