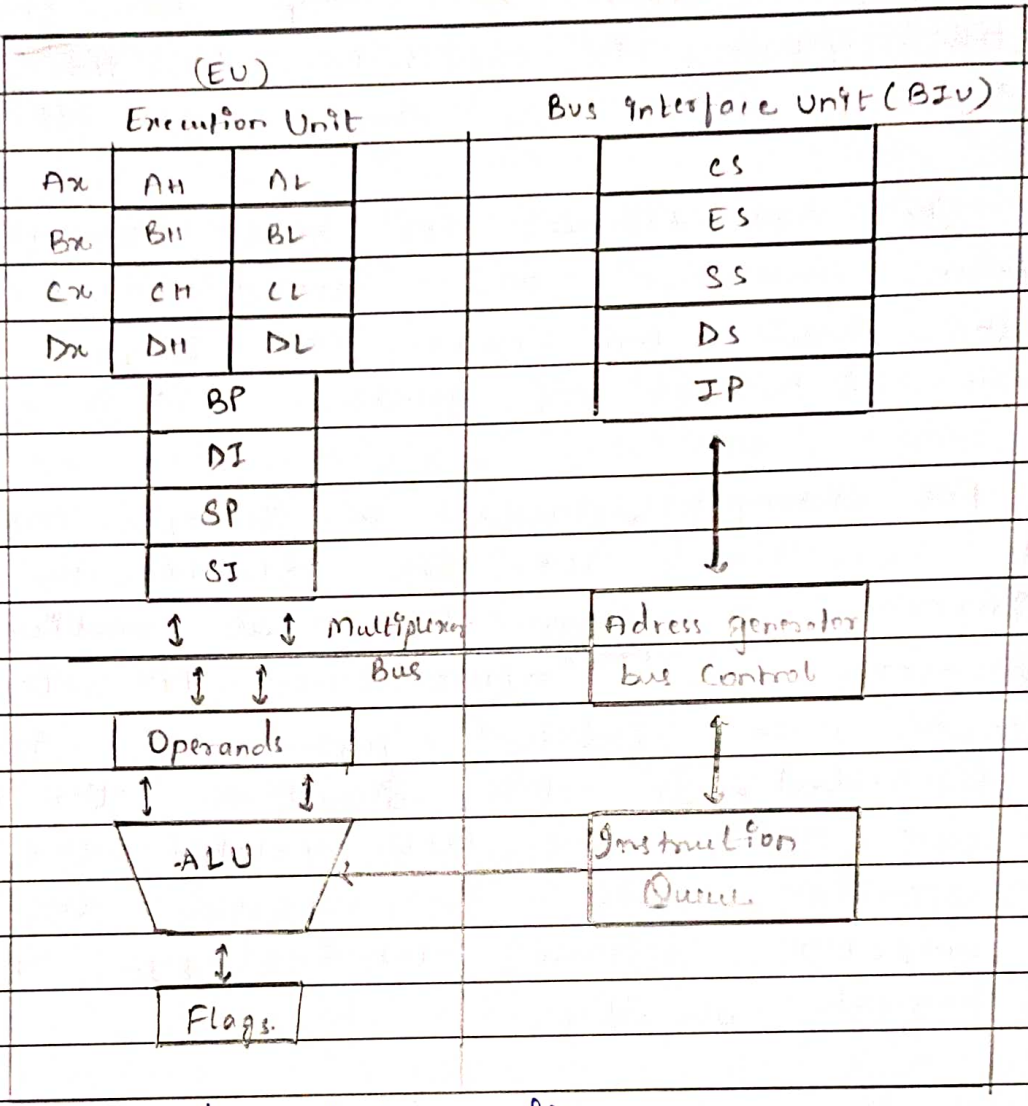
Assignment - 1

1) Explain with internal block diagram of 8086 Architecture.

→



Internal block diagram for 8086 CPU

Intel implemented the concepts of pipelining in the 8086 by splitting the internal structure of this microprocessor into these sections. the execution unit (EU) and bus interface unit [BIU]. These two section work simultaneously.

The BIU access memory and peripherals while the EU executes instructions previously fetched. this works only if the BIU keeps ahead of the EU, thus BIU of the 8086 has a

buffer or Queue. The buffer is 6 bytes in the 8086. If any instruction takes too long to execute, the queue is filled to its maximum capacity and the bus will sit idol.

The BIU fetches the new instruction from memory and stored in IQR, whenever the queue has room for 2 bytes in the 6-byte 8086 queue.

For example, when a jump instruction is executed the BIU starts to fetch information from the new location in memory and information in the queue that was fetched previously is discarded. In this situation the EU must wait until the BIU fetches the new instruction. This is refered to in Computer Science terminology as branch penalty.

Pipeling in 8086 has 2 stages, fetch and execute.

Memory divided into four segments

i) CS → Code Segment
ii) ES → Extra Segment
iii) DS → Data Segment
iv) SS → Stack Segment

Ip → Instution pointer.

Special purpose registers :-

i) Bp → Base pointer
ii) DI → Destination index
iii) SI → Source index
iv) Sp → Stack pointer.

All registers size is 16 bit only

The cocept of pipelining combined with an increasing number of data bus pins has in recent years, led to the design of very powerfull procesror

Assignment No. - 02

1) Define pipeling ? and explain MOV and ADD instruction.

→

Pipeling is the process of accumulating instruction from the processor a pipeline. Pipelining is a technique where multiple instructions are overlapped during execution.

Assembly language programming :-
An assembly language program consists of, among other things a series of lines of assembly language instructions. An assembly instruction consists of a memonic, optionally followed by one or two operands.

Two widely used instructions : the move and add instructions.

Mov instruction : Copies the data from one location to another.
Syntax :
Mov destination, Source ;
↳ It copies source operand to destination

Opcode operand 1, operand 2

This instruction tells the CPU to move (In reality, copy) the source operand to the destination operand.
Moving a value that is two large into a register will cause an error.

Eg:- MOV AL, 10h
MOV DL, AL
DL = 10h
} Example for 8 bit

Example for 16 bit :

MOV Ax, 5678h
MOV Dx, Ax

Ax = Dx = 5678h

ADD instruction :- The ADD instruction has the following format.

Syntax :- ADD instruction tells the CPU to add the source and the distination operands and put the result in the destination.

To add two numbers such as 502h and 03h, each can be moved to a register and then added together.

Move AL, 02h ; move 02h into AL
Move BL, 03h ; move 09h into BL
ADD AL, BL ; AL = AL + BL

Executing the program above results in AL = 02h + 03h = 05h and BL = 03h, the program above can be written in many ways depending on the registers used. Another way might be :

move DH, 02h ;
move CL, 03h ;
ADD DH, CL ;

The program results in DH = 05h and CL = 03h

Look at following variations.

```
MOVE    DH, 02h ;  Local one operand into DH
ADD     DH, 03h ;  Add the second operand to
```

Assignment - 3

3) Assume that DS is 5000 and offset is 1950 and calculate the physical address.

→ DS = 5000 , offset = 1950

```
  50000
+  1950
-------
  51950.
```

Physical address = 51950 //

DS [5|0|0|0] offset [1|9|5|0]

The physical address will be

1. Start with DS — 5000
2. Shift DS left — 50000
3. Add the offset — + 1950
4. Physical address → 51950

Assignment 04

Q4. Show how the flag register is affected by the addition of 9CH and 64H

→

```
MOV   BH, 9CH
MOV   AL, 64H
Add   BH, AL
```

```
9 → 1 0 0 1      C → 1 1 0 0
6 → 0 1 1 0      4 → 0 1 0 0
  (1) 0 0 0 0        0 0 0 0
↳ carry out.
```

CF = 1 : Since there is a carry beyound D7

AF = 1 : Since there is a carry from d3 to d4

PF = 1 : Since there is an even number of 1's in the result.

ZF = 1 : Since the result is zero

SF = 0 : Since d7 of the result is zero

Show how the flag register is affected by the addition of 34F5H and 95EBH.

```
MOV   BH, 34F5H
MOV   AL, 95EBH
Add   BH, AL
```

34F5	0011	0100	1111	0101
95EB	1001	0101	1110	1011
CAE0	1100	1010	1110	0000

CF = 0 ; Since there is no carry beyound d15.

AF = 1 ; Since there is a carry from d3 to d4

PF = 0 ; Since there is a odd number of 1's in the lower byte.

ZF = 0 ; Since the result is not zero.

SF = 1 ; Since d15 of the result is zero.

11/6 Assignment - 5

Q5 Define addressing mode, write a short note on Immediate and Direct addressing mode.

→ The CPU can access operands [data] in various ways.

Immediate Addressing mode :-

In immediate addressing mode; the source operand is a constant. In immediate addressing mode, as this name implies, when the instruction is assembled, the operand comes immediately after the op code. for this reason, this addressing mode executes quickly, however in programming it has limited use.

Immediate addressing mode can be used to load information into any of the register except the segment register and the flag registers.

```
MOV  AX, 2550H ;
MOV  CX, 625 ;
```

To move information to the segment registers the data must first be moved to a general purpose register and then to the segment register.

Example :

```
MOV  AX, 2550H
MOV  DS, AX
MOV  DS, 0123H ; Illegal!
                 Cannot move data
                 into segment reg.
```

Direct addressing mode :- In the direct addressing mode the data is in some memory location and the address of the data in memory comes immediately addressing the operand itself is provided with the instructions, whereas in direct addressing mode, the address of the operand is provided with the instruction. This address is the offset address and one can calculate the physical address by shifting left the DS register and adding it to the offset as follows;

```
MOV  DL, [2400] ; move contents of DS:2400H
                  into DL
```

In this case physical address is calculated by combining the contents of offset location 2400H with DS, the data segment register. Notice the bracket around the address, In the absence of braces executing the command will give an error. Since it is interputed to move the value 2400 (16-bit data) into register DL, an 8-bit register

Assignment 6

Q6] Lilith an example explain assembler data directives.

→ All the assemblers designed for the x86 microprocessor have standardized the directives for data representation.

The following are some of the data directives are

1. ORG (origin) : ORG is used to indicate the begining of the offset address. The number that comes after ORG can be either in hex or in decimal of the number is not, followed by tt, it is decimal and the assembler will convert it to hex.

2. DB (define byte) : It allows allocation of memory in byte - sized chunks, DB can be used to define numbers in decimal, binary, hex and ASCII. DB is the only directive that can be used to define ASCII strings larger than two characters ∴ it should be used for all ASCII data definations

List file for DB examples :

```
DATA1   DB    25          ; DECIMAL
DATA2   DB    10001001B   ; BINARY
DATA3   DB    12H         ; HEX

        ORG   0010H       ;
DATA4   DB    '2591'      ; ASCII NUMBERS.
```

3. DUP (duplicate) : DUP is used to duplicate a given number of characters. This can avoid a lot of typing.

```
DATA7   DB   OFFH, OFFH, OFFH, OFFH ; FF
             ORG 38H

DATA8   DB   6DUP(OFFH) ; Fill 6 Bytes with ff
```

4. DT (define ten bytes) : DT is used for memory allocation of packed BCD numbers. This directive allocates 10 bytes, but a maximum of 18 digit can be entered.

```
DATA23   DT   8679435649829  ; BCD
DATA24   DT   ?              ; NOTHING

DEC      DT   6553d ; the assembler will
                      convert the decimal
                      number to hex and
                      store it.
```

5. DW (define word)

DW is used to allocate memory two bytes (one word) at a time.

eg:

```
DATA11   DW   954    ; DECIMAL
DATA12   DW   253FH  ; HEX
         ORG  78H
```

6. EQU (equate) : This is used to define a constant without occupying a memory location EQU can also be used outside the data segment, even in the middle of a code segment.

Using EQU for the counter constant in the immediate addressing mode :

```
COUNT   EQU   25
```

when executing the instructions "MOV cx, COUNT", the register CX will be loaded with the value 25. This is in contract to using DB

```
COUNT   DB   25.
```

7. DD (define double word)

The DD directive is used to allocate memory locations that are 4 bytes in size.

ex :-

```
DATA16   DD   1023
```

8. DQ (define quad word)

DQ is used to allocate memory 8 bytes (four words) in size.

```
DATA20   DQ   'HI'
```

Assignment 4

1. Explain types of logic instructions

→ 8 Logic instruction are :

1. AND 2. OR 3. XOR 4. SHIFT 5. COMPARE

→ AND instruction

AND destination, Source

This instruction will perform a logical AND on the operands and place the result in the destination. AND will automatically change the 'CF and OF to zero' and PF, ZF and SF are set according to the result. The rest of the flags are either undecided or unaffected.

Ex:

MOV	BC, 35h	0011	0101
AND	BC, 0Fh	0000	1111
		0000	0101

→ OR instruction

OR destination, Source

The distination and Source operands are OR'ed and the result is placed in the destination.

OR can be used to set certain bits of an operand to '1', CF and OF will be reset to zero, and SF, ZF and PF will be set according to the result.

Ex

Mov	Ax, 0504h	0000	0101	0000	0100
OR	Ax, DA68h	1101	1010	0110	1000
		1101	1111	0110	1100
		D	F	6	C

→ XOR instruction

XOR destination, Source

XOR instruction works similar to AND, OR instructions. The XOR instruction can be used to clear the content of register by XOR'ing it with itself.

Ex,

```
MOV   DH, 54h     0101   0100
XOR   DH, 78h     0111   1000
                  0010   1100
```

XOR sets the result bits to 1 if they are not equal ; otherwise, they are reset to 0.

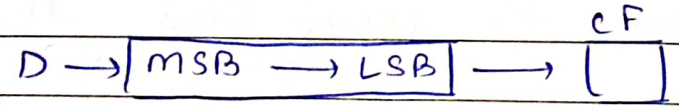
→ SHIFT instruction :

There are two kinds of shift operations

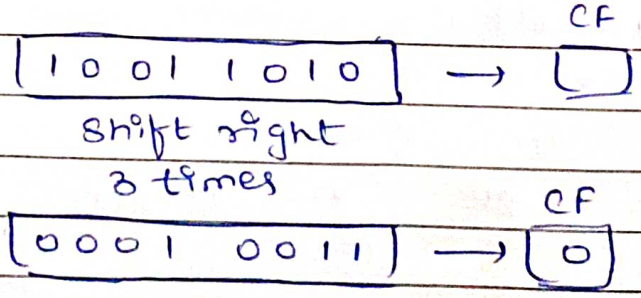
→ logical → Arithematic

The logical shift is for unsigned operands and arithematic shift signed operands.

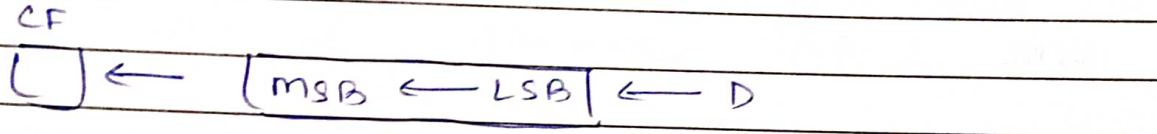
SHR (Shift right) : This is logical shift right, the operand is shifted right bit by bit and for every shift the LSB will go to the carry flag and MSB is filled with zeros



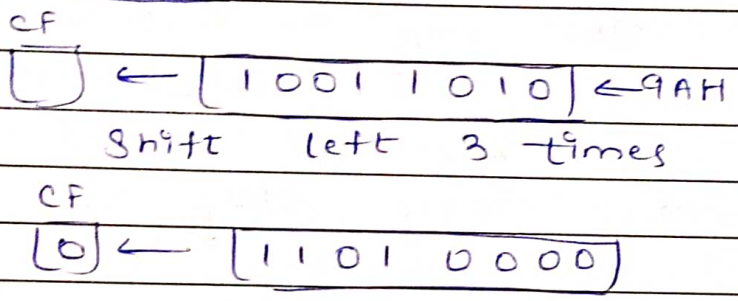

```
MOV   AL, 9AH
MOV   CL, 3
SHR   AL, CL
```



SHL (Shift left) : This is logical shift left, the operand is shifted ~~right~~ left bit by bit and for every shift the MSB will go the carry flag and LSB is filled with zeros.




```
MOV AL, 9AH
MOV CL, 3
SHL AL, CL
```



→ COMPAIR instruction (cmp)

cmp destination, Source

The cmp instruction compairs two operands and change the flags according to the results of the operation

The cmp instruction substracts the Source value from the destination and the result is not stored anywhere only the flags are updated. according to the result of the substraction.

Compare operands	CF	ZF
destination > Source	0	0
destination = Source	0	1
destination < Source	1	0

28/06/12

```
Ex      mov     bx, 7888h
        mov     cx, 9FFFh
        cmp     bx, cx
        jNC     next
        add     Bx, 4000h
Next :  Add     cx, 250h
```

The cmp instruction sets the CF flag and resets the ZF flag, jNC jmps to desired lable only when the carry flag is not set.

Assignment -8

1) write a short note on Substraction of unsigned numbers

→ Substraction of unsigned numbers :

SUB → Substracts the source operand from the destination operand, the result is stored in destination operand.

destination operand
SUB AX, BX
source operand

In Substraction, the processor uses the 2's Complement method and the adder circuitry for performing substraction command. Following are the steps taken by the hardware of the processor.

→ Take the 2's complement
→ Add it to minuend (destination operand)
→ Invert the carry.

These three steps are performed for every SUB instruction, then only the flags are set.

eg:
```
MOV AL, 3FH   ; AL = 0011 1111
MOV BL, 23H   ; BL = 0010 0011
SUB AL, BL    ; 2's BL = 1101 1101

; o/p AL = 0011 1111
  2's BL = 1101 1101
AL = AL + BL = 0001 1100
```

Then flag status is CF=0, ZF=0, AF=0, PF=0, SF=0. The programmer must look at the carry flag (not the sign flag) to determine if the result is positive or negative

Assignment - 9

9) Explain in detail DAA and DAS

→ DAA (Decimal Adjusted for Addition)
This is used after adding two decimal numbers which are represented in packed BCD form.

DAA works as follows:

→ If the LS hex digit in AL is <=9 and AC flag is 0, then there is no change.

→ If LS hex digit is > 9 or if AF flag is 1, It adds 6 (110-a) to the LS hex digit of AL and increment the MS hex digit if the addition generates a carry. The carry flag is set if the MS hex digit is incremented from F to 0.

ex: MOV AL, 88H
ADD AL, 48H
DAA

1000 1000 = 88
0100 1000 = +48
1101 0000

Here AL = 1 and MSB > 9, so add 6 to LSB and MSB

1101 0000
+ 110 110
(1) 0011 0110 = 136 in Ax (D0 in hex)

→ If the MS hex digit <=9 & carry = 0 the MS hex digit is not allowed.
→ If the MS hex digit is > 9 or if Carray = 1, it adds 6(110) to the MS hex digit of AL and sets carry to 1

ex 1001 1000 = 98 H
0110 0100 = 64 H
1111 1100 = FC H, here C>9, so add 6 to LSB
+ 110
0000 0010 here MSB is changed from
62 ⇒ 01100010 F to 0. So carry is set to 1.
Hence add 6 to MSB

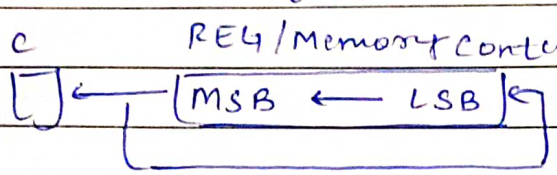
So finally 62 is the result Stored in AX.

DAS : Decimal adjusted for Subtraction
This is used after Subtracting two decimal numbers which are represented in packed BCD form.
DAS works as follows:
→ If the LS hex digit >9, or if AF flag is 1, it Subtracts 6(110) from the LS hex digit of AL and increment the MS hex digit if the addition generates a carry.
→ If the MS hex digit is >9 or if CF = 1, it Subtracts 6(110) from MSI hex digit of AL, & sets carry to 1.

Assignment 10

10) Write a short note on ROR (Rotate right) and ROL (Rotate left)

→ ROL (Rotate left)



It rotates left the content of a register or memory location which consists of a byte / word, The rotation may be one bit or may be the count as given in CL.

After rotation the MSB is moved to carry flag and also to the vacant position created at LSB.

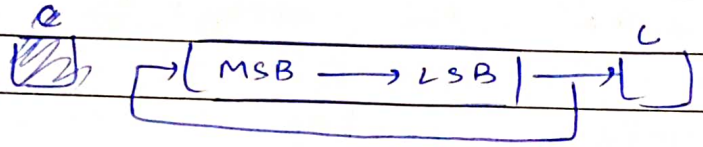
Only CF & OF flags are affected. The ZF flag will be set to 1, if there is a change in MSB after rotation.

The ROL instruction is used for swapping the hex digit in a byte by using the count value of 4, or for swapping the bytes in a word by using the cout of 8 in CL. It can also be used for to test the value of MSB.

Ex: MOV CX, 2360
ROL CX, 1

Before: CX = 2360
After: CX = 46C0 with CF=0, OF=0

ROR (Rotate Right)



It rotates right the contents of a reg / Memory location bit by bit. The data can be a byte/word

The rotation may be once, or CL times. After the rotation, the LSB is moved to carry and also to MSB. Only CF & OF flags are affected.

Ex: ROR AX, CL ; The count in CL & data is in AX.

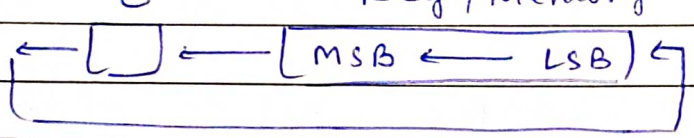
It is also used for swapping and also to test LSB.

```
ex: MOV AX, 10H
    ROR AX, 4
; AX = 01H //
```

Assignment 11

11) Explain RCL and RCR.

→ RCL (Rotate Left with Carry) :



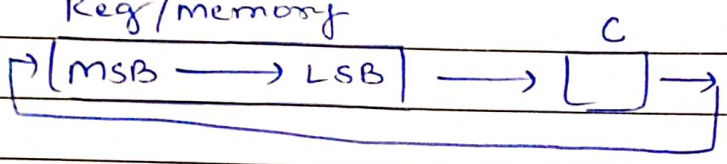
It rotates left the contents of a reg/memory by including the carry bit.

The data may be a byte or word. The rotation may be once or more as given in CL reg. After the rotation, the MSB goes to the carray flag and before that carry bit goes to the LSB. Only CF and OF flags are affected. The overflow flag is 1 if there is a change in MSB.

Ex: RCL Ax, 1
RCL Ax, CL

Ex: MOV Ax, A6H
RCL Ax, 4
; Ax = 6AH

→ RCR (Rotate right with carry):



It rotates right bit by bit the contents of reg/memory.

The data in RCR, as bits are shifted from left to right, they exit the right end (LSB) to the carry flag, and the carry flag enters the left end (MSB). In other words, in RCR the LSB is moved to CF and CF is moved to the MSB.

Ex: RCR AX, 1
RCR AX, CL

ex MOV AX, A6H
RCR AX, 4
; AX = 6AH