MP Assignment I

1. **Explain the flag register associated with 8086. Assume that you have two 8-bit binary numbers in the memory locations. Perform the multiplication of them assuming the numbers as unsigned and signed numbers. The products obtained in either case needed to be logically ORed and then 2's complement of ORed result has to be found and to be stored in memory. Write an ALP to meet the above specification. Show your calculation by taking suitable examples as it is done by the processor.**

The flag register in 8086 is a 16-bit register with each bit corresponding to a flip-flop. It changes its status according to the output stored in the Accumulator (AX).

There are 9 flags active in the register, rest are undefined.

Out of the 9 active flags, 6 are conditional flags, and 3 are status flags.

Conditional Flags - Zero Flag, Carry Flag, Parity Bit Flag, Auxiliary Carry Flag, Sign Flag, and Overflow Flag.

Control Flags - Trap Flag, Interrupt Flag, and Directional Flag.

```
data segment

     a db 10010101b

     b db 11010011b

     c dw ?

data ends

code segment

assume ds: data, cs: code

start:     mov ax, data

           mov ds, ax
```

```
            mov al, a
            mov bl, b
            mul bl;                 unsigned multiplication
            mov dx, ax;             store unsigned result in dx
            mov al, a
            imul bl;                signed multiplication
            or ax, dx;              ORing signed and unsigned result
            neg ax;                 2's compliment of result
            mov c, ax;              storing result in memory
            mov ah, 4ch
            int 21h
        code ends
end start
```

Numbers taken, a = 1001 0101, and b = 1101 0011

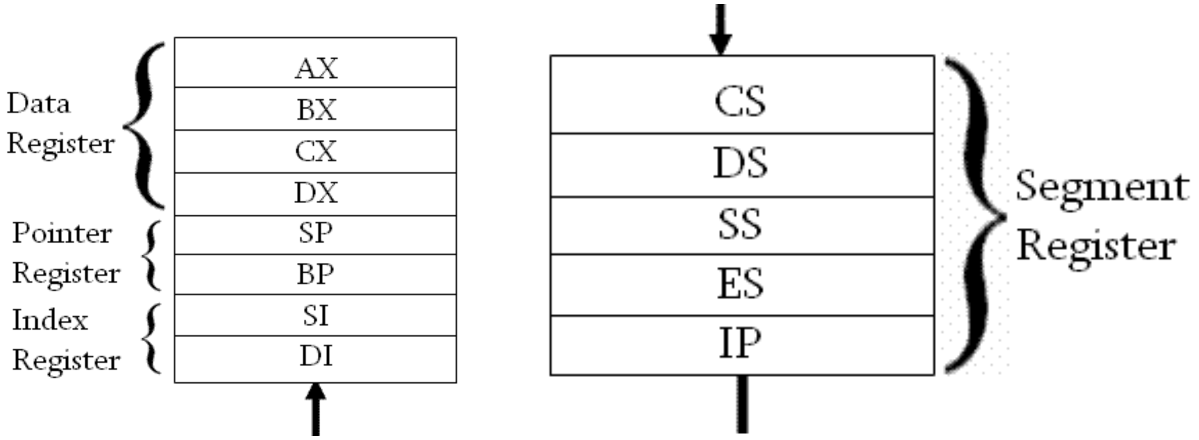
a ∗ b (unsigned) =   0111 1010 1100 1111 (7ACFh) (Stored as CF 7A)

a ∗ b (signed)   =   0001 0010 1100 1111 (12CFh) (Stored as CF 12)

unsigned OR signed = 0111 1010 1100 1111 (7ACFh)

neg above result =   1000 0101 0011 0001 (8531h) (Stored as 31 85)

2. **Extract the blocks of Registers, Instruction Queue and the flags from the internal architecture of 8086. Draw them with complete details on it. Explain the importance of them in the 8086 programming.**



The 8086 has 8, 16-Bit general purpose registers (GPRs).

4 out of the 8 (AX, BX, CX, and DX) are in the data register file.

The other 4 are Pointer (SP, and BP) and Index (SI, and DI) registers.

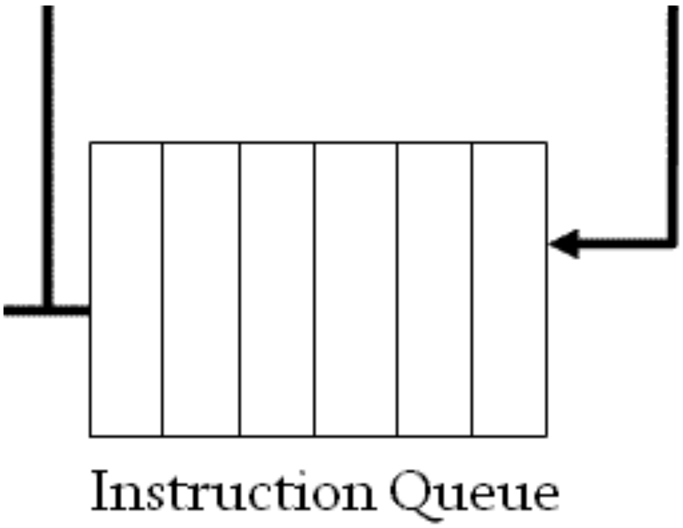
The data registers are used for temporary storage for faster access and some special operations.

The Pointer registers are used to point to program stack (SP) and to data in stack segment (BP).

The Index registers are used to hold the index of memory locations SI for source and DI for data in indexed, base indexed, indirect addressing and some string operations.

There are 4 segment registers (Code, Data, Stack, and Extra) that point to the respective segments in the memory.

There is also a special register IP (Instruction Pointer) that points to the next instruction to be executed.



The instruction queue of 8086 is a set of six 8-Bit registers that contain address of the next instruction(s) to be fetched. The execution unit directly fetches the instructions from the instruction queue, speeding up the execution.

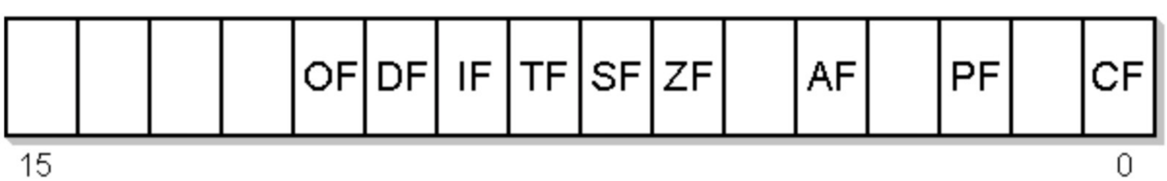
The flag register in 8086 is a 16-bit register with each bit corresponding to a flip-flop. It changes its status according to the output stored in the Accumulator (AX).

There are 9 flags active in the register, rest are undefined.

Out of the 9 active flags, 6 are conditional flags, and 3 are status flags.

Conditional Flags - Zero Flag (ZF), Carry Flag (CF), Parity Bit Flag (PF), Auxiliary Carry Flag (AF), Sign Flag (SF), and Overflow Flag (OF).

Control Flags - Trap Flag (TF), Interrupt Flag (IF), and Directional Flag (DF).



3. Discuss all the instructions associated with flags of 8086. It is required to perform a division of signed 32-bit number by a signed byte. Write an ALP to meet the above division. Also write the input and the expected output how it appears in memory location according to your program.

```
CLC – Clears carry flag. (CF ← 0)

STC – Sets carry flag. (CF ← 1)

CMC – Compliments carry flag. (CF ← ~CF)

CLD – Clears directional flag. (DF ← 0)

STD – Sets directional flag. (DF ← 1)

CLI – Clears interrupt flag. (IF ← 0)

STI – Sets interrupt flag. (IF ← 1)
```

```
data segment

     a0 dw ?; lower word of number

     a1 dw ?; higher word of number

     b db ?; byte

     quo dw ?

     rem dw ?

data ends


code segment

assume ds: data, cs: code

start:  mov ax, data

          mov ds, ax
```

```
        mov ax, 0000h;        clear ax
        mov al, b;            move the byte to al
        cbw;                  convert byte to word

        mov bx, ax;           copy newly formed word to bx
        mov ax, a0;           copy the lower word of dividend to ax
        mov dx, a1;           copy the higher word of dividend to dx
        xor dx, dx;           prevents overflow (maybe?)

        idiv bx;              signed divide dx:ax by bx;
        mov quo, ax;          copy quotient to quo
        mov rem, dx;          copy remainder to rem

        mov ah, 4ch
        int 21h
    code ends
end start
```

For example, let 'a1:a0' be 00023FC9h and 'b' be 69h.

Dividing '00023FC9h' by '0069h',

Quotient (AX) = 057Bh

Remainder (DX) = 0056h

4. **Assume that you have read two single digit BCD numbers through the K/B and these numbers are stored in memory. Write an ALP to add and subtract these values without modifying the numbers and keep the result in unpacked BCD format (use appropriate instructions). Give an example for each at the end of the program. Also write the input and the expected output how it appears in memory location according to your program.**

```
data segment
     a db ?
     b db ?
     sum dw ?
     dif dw ?
     input1 db "  Enter bcd number 1: $"
     input2 db "  Enter bcd number 2: $"
data ends

code segment
assume ds: data, cs: code
start:  mov ax, data
           mov ds, ax

           lea dx, input1;
           mov ah, 09h;
           int 21h;               Print message for input 1
           mov ah, 01h;
           int 21h;               Take input 1

           sub al, 30h;           Convert ASCII to number
           mov a, al;             Move to memory
```

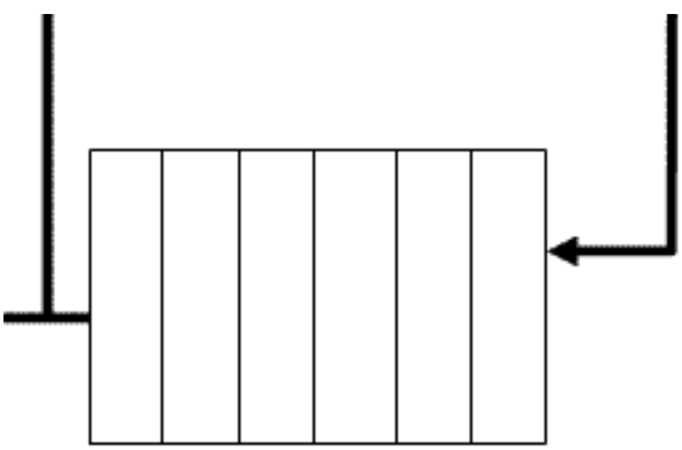
```
lea dx, input2;
mov ah, 09h;
int 21h;             Print message for input 2
mov ah, 01h;
int 21h;             Take input 2

sub al, 30h;         Convert ASCII to number
mov b, al;           Move to memory

mov ax, 0000h;       Clear ax

mov al, a;
add al, b;
aaa;                 ASCII adjust after addition

mov sum, ax;         Move the sum to memory

mov ax, 0000h;       Clear ax again...

mov al, a;
sub al, b;
aas;                 ASCII adjust after subtraction

mov dif, ax;         Move the difference to memory

mov ah, 4ch
int 21h
```

```
    code ends
end start
```

```
Enter bcd for number 1: 9  Enter bcd for number 2: 7
```

```
Dump
    ds: 0000  09  07  06  01  02  00  00  00
               a   b    sum     dif
```

5. **Explain the importance of Instruction Queue in 8086. Write an ALP to have a 16-bit number (given as 81D2H) in the memory location. This given number is the result of summation of two BCD numbers. With appropriate instructions in your program convert it to BCD and store the result in memory. Assume that AF got set and CF was reset while addition. Show your calculation as it is done by the processor.**



Instruction Queue

The instruction queue of 8086 is a set of six 8-Bit registers that contain address of the next instruction(s) to be fetched. The execution unit directly fetches the instructions from the instruction queue, speeding up the execution.

(Come on, this again!)

```
data segment
     n dw 081D2h
     a dw ?
data ends

code segment
assume ds: data, cs: code
start:  mov ax, data
          mov ds, ax
          mov ax, n;       Move the number to ax
          add al, 00h;     add zero to al
          daa;             decimal adjust al
          adc ah, 00h;     add zero with carry to ah
          mov a, ax;       store result in memory
          mov ah, 4ch
          int 21h
     code ends
end start
```

Dump: ds: 0000 D2 81 38 82 00 00 00 00

```
  8 1     D 2
        + 0 6 H      (daa | Since aux carry flag is set)
  8 1     3 8
+ 0 1 H              (adc | Since carry flag is now set)
  8 2     3 8        (Result)
```