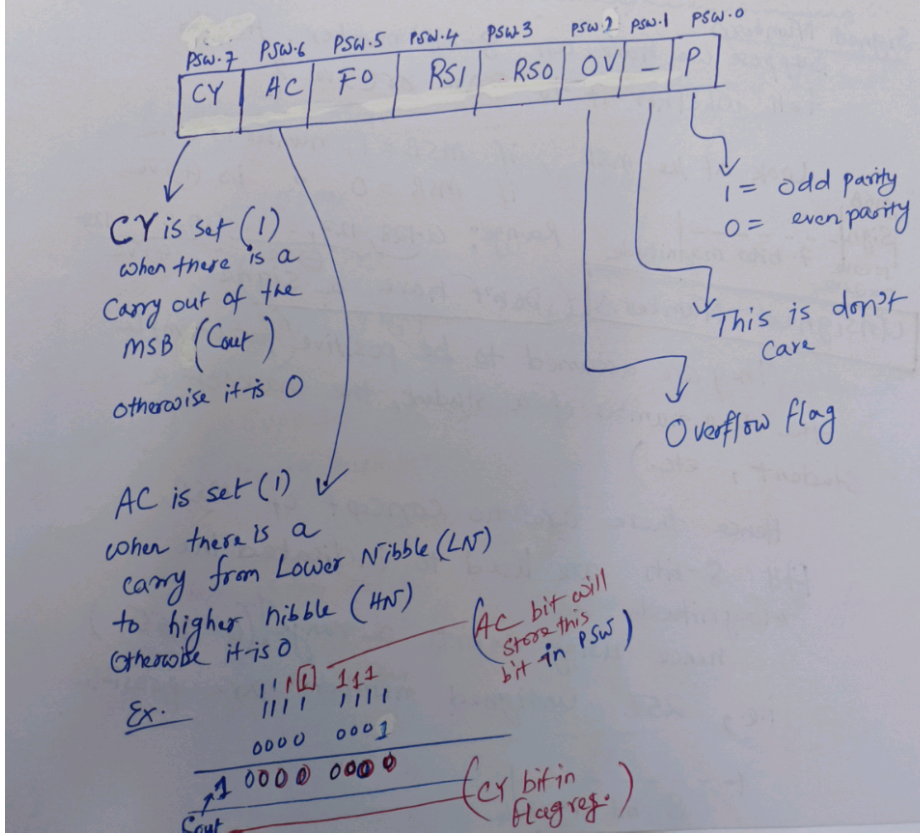
PSW - Flag Register

(Program Status word)

- Flag Reg. gives the status of the current result.
- It is an 8-bit reg.
- When ALU does an operation, it produces two things
  - (i) The result which it stores in A reg.
  - (ii) The status of the result which stores in PSW
    (whether there is a carry, " " an Aux. carry, and so on)

- PSW is Bit Addressable reg. i.e., every bit of the PSW can be changed by the programmer.
  (You don't need to alter the entire flag. reg.)



PSW.0 (Parity flag)

✓ Parity flag is used to check the parity of the result. What do you mean by parity? No. of 1's are there in the result.

⇒ Ex. Suppose we have the following result:-

P=1 ; no of 1's in the result is odd (odd parity)

P=0 ; no. of 1's in the result is even (even parity)

PSW.2 (Overflow flag)
OV

Concept of Signed and Unsigned Numbers

Signed Numbers

Suppose we have an 8-bit number. How to tell whether it is (+)ve or (−)ve?

Look at the MSB, if MSB = 1, number is (−)ve
if MSB = 0, " is (+)ve

Range: −128, −127, ... −1, 0, 1, ... 127
(−)ve (+)ve

Unsigned Numbers (Don't have a sign)

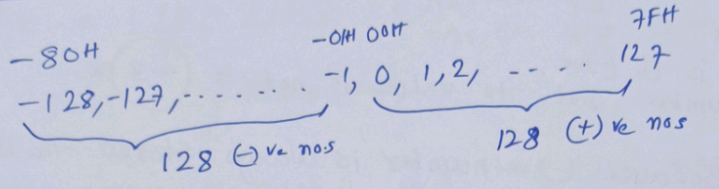
They are assumed to be positive (for example the Roll number of a student, the marks of a student, etc.)

Hence there are no concept of MSB, All 8-bits are used to indicate the magnitude

hence using 8-bits a range of (0-255)

i.e., 256 unsigned numbers are possible

Range of Signed numbers in Hexadecimal System



total = 256 nos

For (+)ve numbers

MSB
0 000 0000 = 00H
. . . . . .
0 111 1111 = 7FH

For (-)ve numbers
take a 2's c

1000 0001 = -01H
. . . . . .
1000 0000 = -80H
↓
Take a 2's c

Range of Unsigned no.s in Hexadecimal System

0000 0000 = 00H
.
.
.
1111 1111 = FFH

(*) Suppose an operation (Arithmetic) produces the following number

1000 0011

What is the value of this number in Unsigned and Signed format?

⇒ Unsigned Number

1000 0011
8 3 = (83H)

Signed Number

MSB

1 000 0011

It is a (-)ve number but its value is not = (-3) ✗

Because (-)ve number is always stored in its 2's complement form. So, to read it you again have to do 2's complement of the number.

Conv?

1000 0010 = -2

1000 0001 = -1

1000 0000 = -0

There is no -0 in the number system, we are having two 0's -0 and +0.

∴ So, we need a better method to store (-)ve nos

+5 = 0101

-5 = take 2's complement of (+5)

= 1011 (copy the digits from the LSB till you get the first 1, then you complement everything)

* So, if we see 1011 we understand it is (-)ve as MSB = 1,

Now do a 2's complement of 1011 and get 0101 which is (+5)

Hence the no is = -5

+24 = 0010 0100

-24 = 1101 1100 (This is how -24 is stored inside any computer)

↓ How to read

0010 0100 and it is (-)ve hence (-24)

2 4

0 ⇒ 0000

Take 2's complement and it gives you back 0 (not minus zero)

-0 ⇒ 0000

Both are Same there is no (-0) in 2's complement system.

Now, Find 1000 0011

↓ 2's Complement

→ 0111 1101 = -7DH

7 D

Overflow Flag: It matters only for signed numbers

To see the MSB we decide whether a no. is (+)ve (-)ve

Sometimes the MSB gives you a wrong sign when the event named Overflow occurs.

⇒ When the result is overflown out of the range of a signed number

-80, ... -01,00H, ... 7F

When your result is more than 7FH (maximum value of (+)ve no.) or less than −80H (minimum value of (−)ve no.), such an event is called an Overflow.

Your result is still of 8-bits only, but it has gone out of range.

At that point, OV flag is set.

Ex. Add two (+)ve numbers why we are getting a (−)ve number.

```
  7F
  01
 ----
  80
  /\
1000 0000
```

Because a signed no. can't be bigger than 7F, Hence you've gone out of range.

How to know a no. is (+)ve or (−)ve ?

Hence, before check the MSB, check the OV flag, if OV = 0, then MSB is giving actual sign.

if OV = 1, then MSB gives the opposite of the actual sign.

Ex.1

```
        1  1
  23H   0010 0011
+ 31H   0011 0001
 ----   ---------
  54H   0101 0100
        ---- ----
         5    4
```

No C_6 and no C_7

Hence

$OV = C_7 \oplus C_6$

$= 0 \oplus 0$

$= 0$

CY = 0

AC = 0

OV = 0 (No overflow was 54H is in the range −80H to 7FH) MSB gives the correct sign

P = 1 (odd parity)

```
  1
  27        0010 0111
+ 39        0011 1001
  60        0110 0000
              6    0
```

Here, $C_6 = 0$
$C_7 = 0$
Hence $OV = 0 \oplus 0 = 0$

CY=0
AC=1
OV=0, msB gives the correct sign.
P=0 (Even parity)

```
  -23H        1101   1101   (2's com. of +23)
+ -31H        1100   1111   (2's of 54 (0101 0100))
  -54H      1 1010   1100
```

Here, $C_6 = 1$, $C_7 = 1$
Hence $OV = 1 \oplus 1 = 0$

CY=1
AC=1
OV=0 as -54 is bigger than -80H (within the range of -80H to 7FH) the answer is supposed to be negative and it is negative hence, there is no overflow

P=0 (Even parity, there are 4 1's)

```
  -27       1101 1001
+ -39       1100 0111
  -60     1 1010 0000
```

$C_6 = 1$, $C_7 = 1$
$OV = 1 \oplus 1 = 0$

take 2's complement of it

0110 0000
6 0

hence, -60H

CY=1
AC=1
OV=0 (in range)

P=0 (2 1's)

```
  42H      0100 | 0010
+ 43H      0100 | 0011
  85H      1000 | 0101
```

$C_6 = 1$, $C_7 = 0$

$OV = 1 \oplus 0 = 1$ ✓

Out of range

MSB = 1, but the number is not negative

✓ Check OV flag which is 1 indicating an overflow occurred.

Hence, the sign bit is wrong

Hence, the no. is actually a (+)ve no.

CY = 0

AC = 0

OV = 1

P = 1

(*) Add two (-)ve numbers.

```
  -42H      1 0 1 1 | 1 1 1 0
  -43H      1 0 1 1 | 1 1 0 1
 -85H     1  0 1 1 1 | 1 0 1 1
```

which is a (+)ve no.

which is impossible as we have added two (-)ve no.s.

Out of the range ↓ CY ↓ MSB

CY = 1

AC = 1

OV = 1 (as the result is out of the range)

P = 0

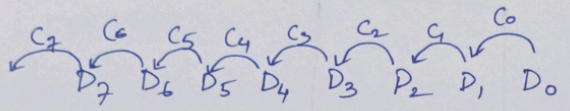
✓ $C_6 = 0$

✓ $C_7 = 1$

$OV = 1 \oplus 0 = 1$

(*) How does the processor determine that there is an Overflow?

⇒ Determined by looking at the CY (C_7) bit and the carry that comes into MSB (C_6)



- C_0: the first carry
- C_6: the carry that comes into MSB
- C_7: the carry that goes out of MSB (which is the Carry flag)

✓ Overflow flag (OV) is mathematically determined as follows:

$$OV = C_7 \oplus C_6$$

Ex-ORing

C_7 X	C_6 Y	O/P
0	0	0
0	1	1
1	0	1
1	1	0

i.e., to generate OV C_6 and C_7 should be opposite of each other.

✓ F0 : User-defined flag

✓ processor does not read/change this flag.

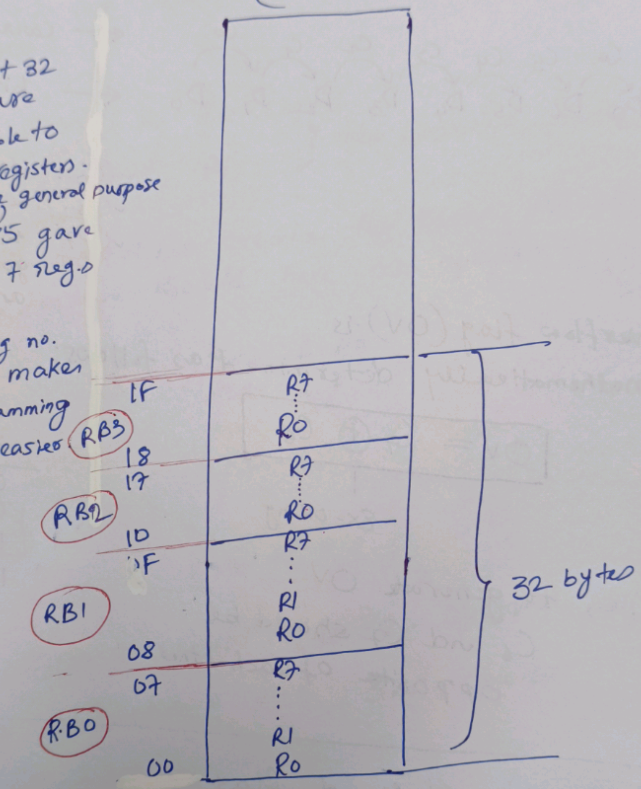
```
SETB  PSW.5 ; Set F0 bit
CLR   PSW.5 ; Clear F0 bit
```

RS1, RS0 : register bank selectors

There are 4 reg. banks which are a part of the INT. RAM

INT. RAM has a size of 128 bytes, i.e., 128 locations. (to store the data)

- The first 32 bytes are available to us as registers. (They are general purpose reg.s)
- In 8085 gave us only 7 reg.s
- Increasing no. of reg.s makes the programming faster and easier.



MOV A, R0 ; A gets the value of R0 (of RB 0)

This bank is active by default, at a time only one bank can be active.

To activate bank 1, we should write:

SETB	PSW-3
CLR	PSW-4

PSW-4 PSW-3

RS1	RS0	RB
0	0	0
0	1	1
1	0	2
1	1	3

(*) Why have they created Reg. Banks?

⇒ Why not number the registers as follows:

R0, R1, R2, . . . R31

(*) R.B.'s are used to reduce the number of opcodes.

⇒ MOV A, R0
MOV A, R1
} two different opcodes

✓ if we would not create Reg. Banks, then for one MOV instruction we would have 32 opcodes

MOV A, R0
MOV A, R1
MOV A, R2
MOV A, R3
:
MOV A, R31
} 32

✓ For ADD we would require 32 opcodes
✓ For subtraction " " 32 opcodes.

✓ You can't afford 32 opcodes for every instruction.

Let's name them

R0 to R7

instead of

R0 to R31