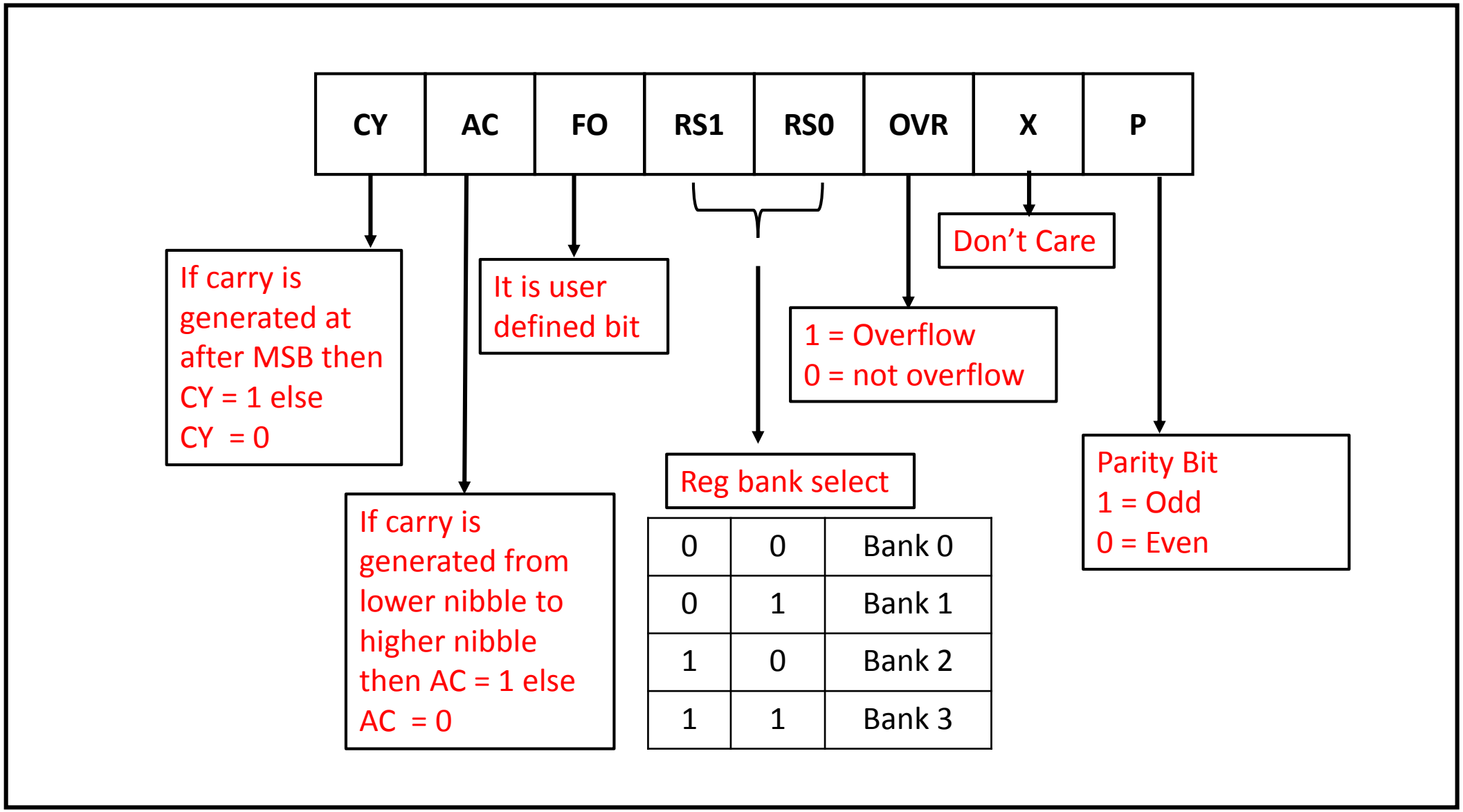
PSW of 8051

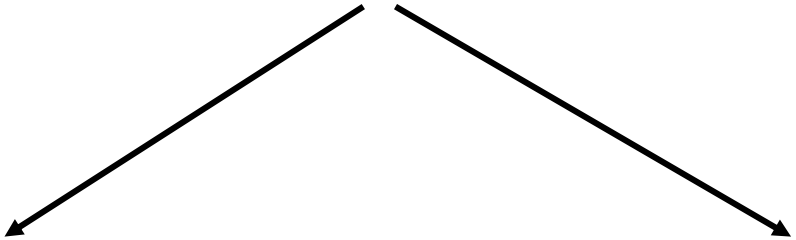
PSW (8 bit) : (Program Status WORD) :

1. PSW contains the status of the result after each operation performed by ALU.
2. Working of PSW is similar with flag reg of processor.
3. PSW can changed by ALU
4. In case of 8051 PSW can also modified by programmer(i.e. mov psw, 10h)

PSW (8 bit)


CY
AC
FO
RS1
RS0
OVR
X
P
If carry is generated at after MSB then CY = 1 else CY = 0
It is user defined bit
Don't Care
1 = Overflow
0 = not overflow
Parity Bit
1 = Odd
0 = Even
If carry is generated from lower nibble to higher nibble then AC = 1 else AC = 0
Reg bank select
0 0 Bank 0
0 1 Bank 1
1 0 Bank 2
1 1 Bank 3

Number System



Signed Numbers

Positive & Negative
+ ve & - ve

Unsigned Numbers

they assume to be positive

No sign i.e. no + ve nor - ve

Unsigned Numbers

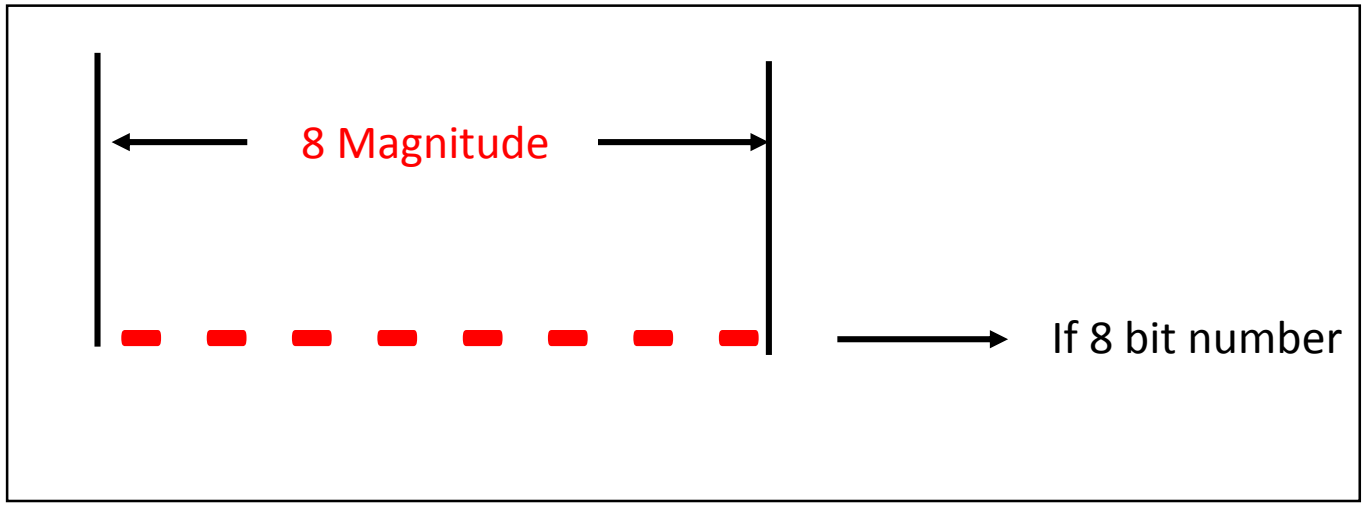
All Positive numbers

Example : Roll Number

For Unsigned numbers if magnitude is 8 then,

$2^8 = 256$

i.e. total 256 + ve numbers are used



Range for unsigned numbers

00	0000 0000
01	0000 0001
FF	1111 1111

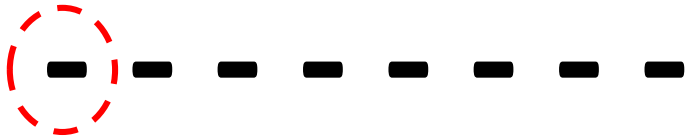
BACK

Signed Numbers

Positive & Negative
+ ve & - ve

How to find whether number is + ve or - ve?

Answers :

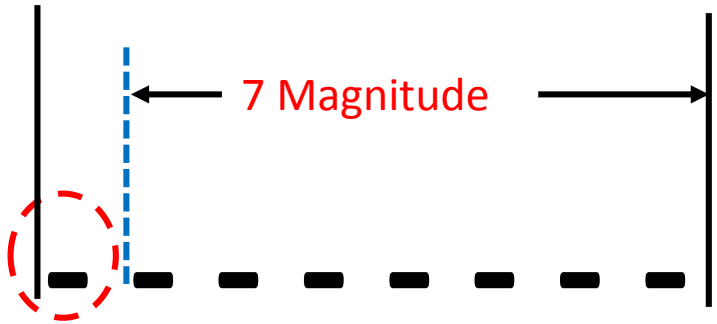


MSB LSB

If MSB of number is 0 then + ve

If MSB of number is 1 then - ve

If number is 8 bit



S	
0	+ ve
1	- ve

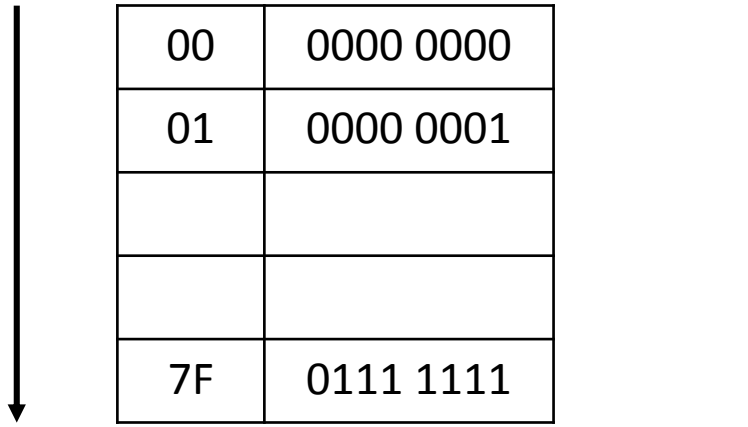
For signed numbers if magnitude is then 7 ,
$2^7 = 128$
i.e. 128 **- ve** numbers & 128 **+ ve** numbers

-128 to -1 0 to 127

Range for +ve and –ve numbers

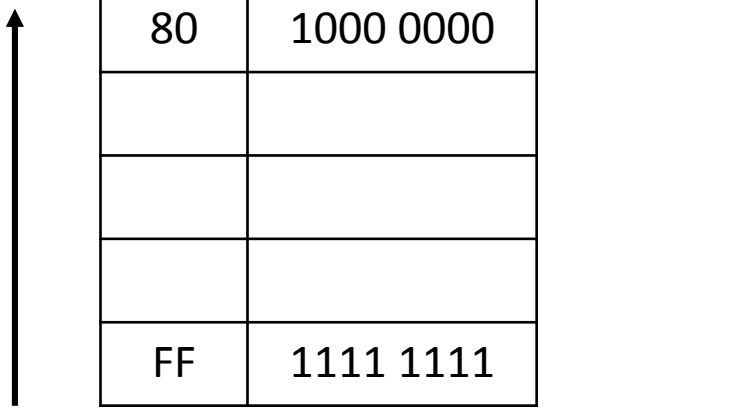
Range for +ve and –ve numbers

Range for Positive numbers (0 to 127)



00	0000 0000
01	0000 0001
7F	0111 1111

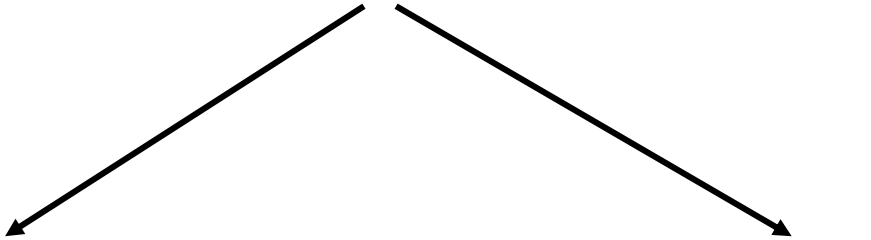
Range for Negative numbers (- 80 to -01)



80	1000 0000
FF	1111 1111

What is unsigned and signed number following binary ?

1 0 0 0 0 0 1 1



Unsigned **Signed**

83 H **03 H** **7D H**

- ve number means 2's complement of given number

Shortcut for 2's complement : copy number as it is from right side till gets first 1 after 1 complement all numbers

example 1 :

+ 24 h	**0010 0100**
2's complement of 24 i.e. -24	**1101 1100**

example 1 :

+ 5 h	**0101**
2's complement of 05 i.e. -5	**1011**

NEXT

Why 2's complement is used ?

Example Without using 2's complement just put 1 for negative

1 0 0 0 0 0 1 1

7 bit magnitude then **-03 H**

If we consider 1 for –ve number

1 0 0 0 0 0 0 1

7 bit magnitude then **-01 H**

If we consider 1 for –ve number

1 0 0 0 0 0 0 0

7 bit magnitude then **- 00 H**

If we consider 1 for –ve number

Which is not possible

To avoid above problem 2's complement is used

NEXT

Why 2’s complement is used ?

1. It is universal method to store –ve number
2. Anything is 2’s complement of anything (any number)

0 0 0 0 0 0 0 0

2’s complement of 0 is :

0 0 0 0 0 0 0 0

Overflow Flag :

Overflow flag matters only for signed numbers

Range for Positive numbers (0 to 127)

00	0000 0000
01	0000 0001
7F	0111 1111

Range for Negative numbers (- 80 to -01)

80	1000 0000
FF	1111 1111

01	0000 0001
-01	1111 1111
80	1000 0000
-80	1000 0000

Range for Positive numbers : 00 to 7F

Range for Positive numbers : FF to 80

After addition if result is going beyond above ranges then overflow flag is set i.e. OVR = 1

Example for overflow flag:

Overflow flag matters only for signed numbers

23 h	0010 0011
+ 31 h	0011 0001
54 h	0101 0100

CY	AC	OVR	P
0	0	0	1

-23 h	1101 1101
+ -31 h	0011 0001
- 54 h	1010 1100

CY	AC	OVR	P
1	1	0	0

27 h	0010 0111
+ 39 h	0011 1001
60 h	0110 0000

CY	AC	OVR	P
0	1	0	0

-27 h	1101 1001
+ - 39 h	1100 0111
- 60 h	1010 0000

CY	AC	OVR	P
1	1	0	1

42 h	0100 0010
+ 43 h	0100 0011
85 h	1000 0101

CY	AC	OVR	P
0	0	1	1

-42 h	1011 1110
+ -43 h	1011 1101
-85 h	0111 1011

CY	AC	OVR	P
1	1	1	0

1. Using MSB bit we can identify whether number is +ve or –ve
2. If MSB is 1 it means number is – ve
3. But sometimes it will give wrong sign bit
4. In such cases checking only MSB is not sufficient
5. We have to check range of both numbers
6. If number cross range it means there is overflow problem

Range for Positive numbers : 00 to 7F (0 to 127)

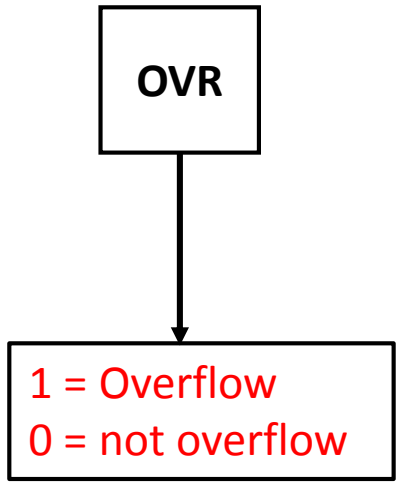
Range for Positive numbers : FF to 80 (-80 to -01)

Example :

7F h	0111 1111
+ 01 h	0000 0001
80 h	1000 0000

Result is positive and answer gives sign bit negative.

How processor will identify the overflow because range concept is not applicable for processor



Answers : Processor will come to know the overflow flag using carry

	C7	C6	C5	C4	C3	C2	C1	C0
23 h	0	0	1	1	0	0	0	1
+ 31 h	0	1	0	1	0	1	0	0
54 h	0	1	0	1	0	1	0	0

Formula used by processor is :

$$OF = C7 \oplus C6$$

X-OR table

0	0	0
0	1	1
1	0	1
1	1	0

Example for overflow flag:

Overflow flag matters only for signed numbers

23 h	0010 0011
+ 31 h	0011 0001
54 h	0101 0100

CY	AC	OVR	P
0	0	0	1

-23 h	1101 1101
+ -31 h	0011 0001
- 54 h	1010 1100

CY	AC	OVR	P
1	1	0	0

27 h	0010 0111
+ 39 h	0011 1001
60 h	0110 0000

CY	AC	OVR	P
0	1	0	0

-27 h	1101 1001
+ - 39 h	1100 0111
- 60 h	1010 0000

CY	AC	OVR	P
1	1	0	1

42 h	0100 0010
+ 43 h	0100 0011
85 h	1000 0101

CY	AC	OVR	P
0	0	1	1

-42 h	1011 1110
+ -43 h	1011 1101
-85 h	0111 1011

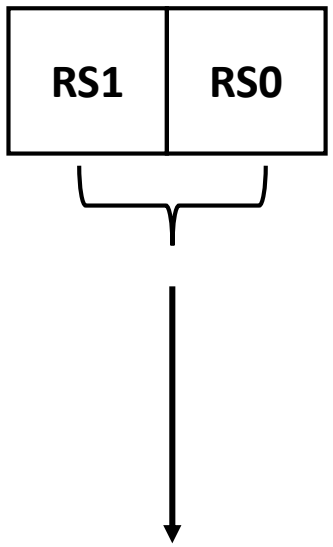
CY	AC	OVR	P
1	1	1	0

X-OR table

C7	C6	OF
0	0	0
0	1	1
1	0	1
1	1	0

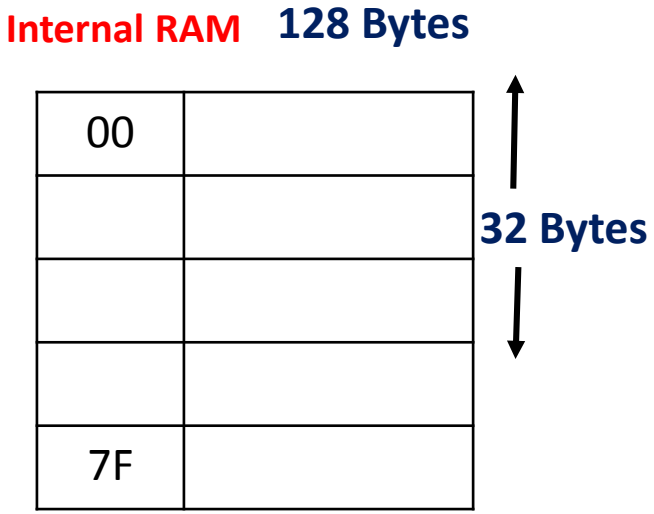
$$OF = C7 \oplus C6$$

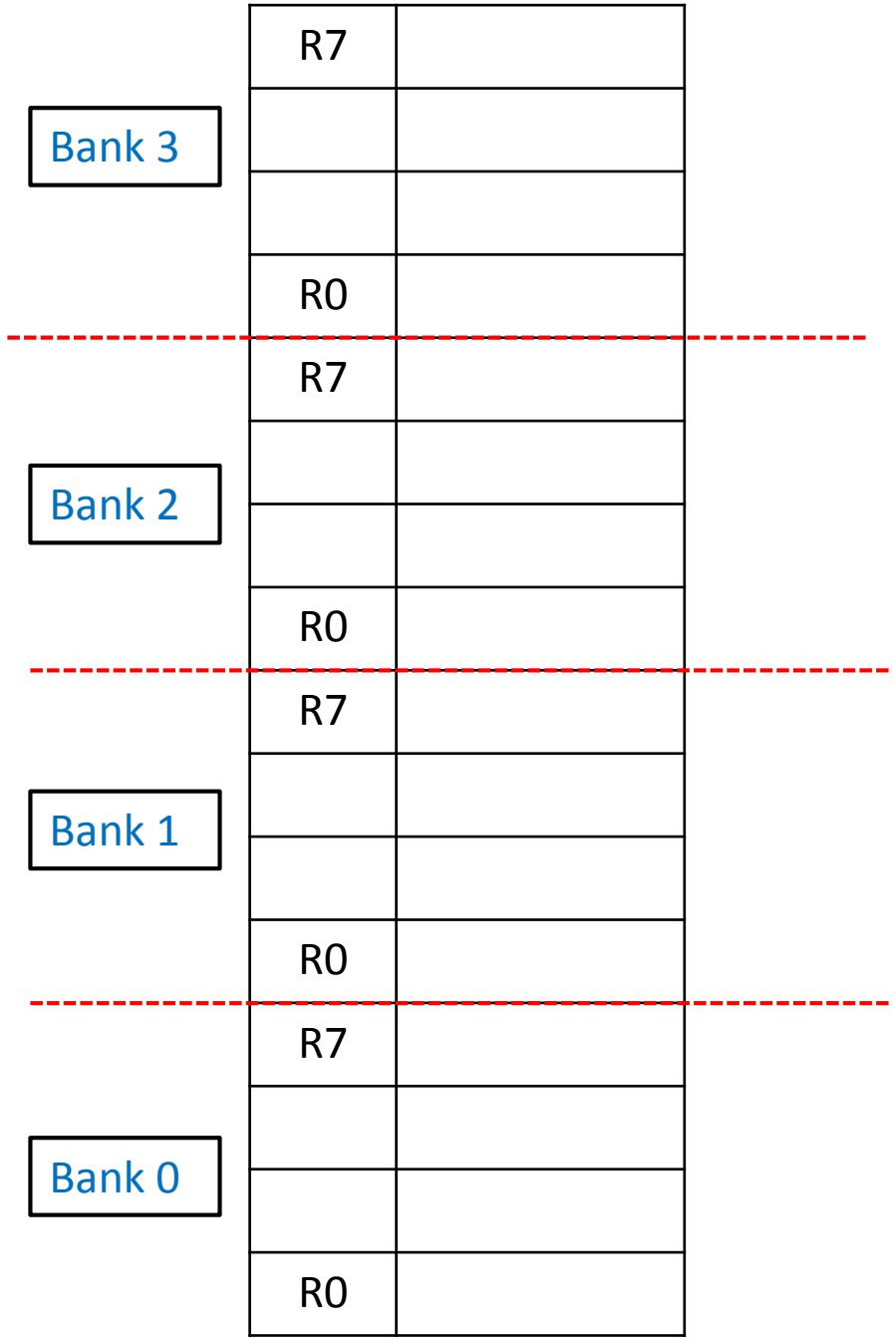
From internal RAM first 32 bytes are used for registers

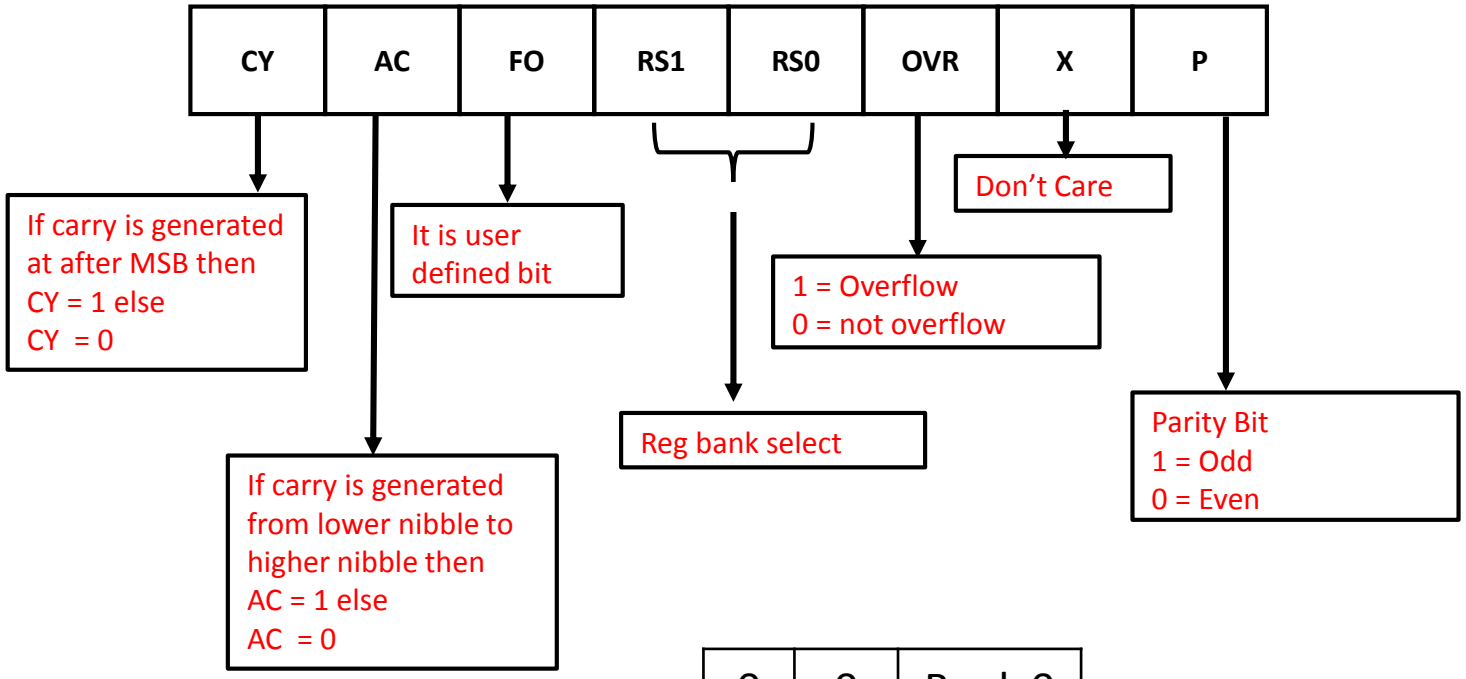


Reg bank select

0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3







0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3