Binary Number Representation

Ch-8.5.2 to 8.7 (Morris Mano)

Courtesy To

Lec Tasnim Ullah Shakib

Table of Contents

Signed Binary Numbers have 3 forms of representations

- Sign-Magnitude Representation
- Sign-1s Complement Representation
- Sign-2s Complement Representation

All positive numbers have the same representation in all forms.

The MSB always contain the sign bit, while the remaining bits contain the magnitude.

Sign-Magnitude Representation

Sign bit is 1for minus, 0 for plus.

The remaining bits show the magnitude.

Sign-1s complement Representation

If the number is negative, Toggle all bits including the sign bit.

Sign-2s complement Representation

If the number is negative,

Toggle all bits including the sign bit. Add 1to the result. Discard any possible carry.

Shortcut: Start from the LSB. Only toggle all the bits after the least significant 1.

Represent +9 and -9 in all 3 forms on 7 bit register

	+9	**-9**
Sign Magnitude	0 001001	1 001001
Sign 1s Complement	0 001001	1 110110
Sign 2s Complement	0 001001	1 110111

Represent +0 and -0 in all 3 forms on 7 bit register

	+0	**-0**
Sign Magnitude	0 000000	1 000000
Sign 1s Complement	0 000000	1 111111
Sign 2s Complement	0 000000	0 000000

Signed Binary Addition using Sign-1s complement

- Take 1s complement of any negative operands, excluding the sign bit.
- Add the operands, including the sign bit (MSB).
- If there is an end carry out of the MSB, add it with the remaining result bits.
- Finally, If the sign bit of the result is 0, the result is in its original form.
- But, If the sign bit of the result is 1, the result is in its complement form.

  So, re-complement the result and add a minus sign to interpret it.

Signed Binary Addition using Sign-2s complement

- Take 2s complement of any negative operands, excluding the sign bit.
- Add the operands, including the sign bit (MSB).
- If there is an end carry out of the MSB, discard it and keep the remaining result bits.
- Finally, If the sign bit of the result is 0, the result is in its original form.
- But, If the sign bit of the result is 1, the result is in its complement form.

  So, re-complement the result and add a minus sign to interpret it.

(+6) + (+9)
using sign 1s complement, on 7 bit register

No end carry

No overflow

Sign bit 0, so result is in **original** form

Try,
(+6) + (+9)
using sign 2s complement, on 7 bit register

(-6) + (+9)
using sign 1s complement, on 7 bit register

		1	1	1			1	
-6 :		1	1	1	1	0	0	1
+9 :		0	0	0	1	0	0	1
	1	0	0	0	0	0	1	0

Add end carry

0 000011

+3

End carry occured

No overflow

Sign bit 0, so result is in **original** form

Try,
(-6) + (+9)
using sign 2s complement, on 7 bit register

Try,
(+6) + (-9)
using sign 1s complement, on 7 bit register

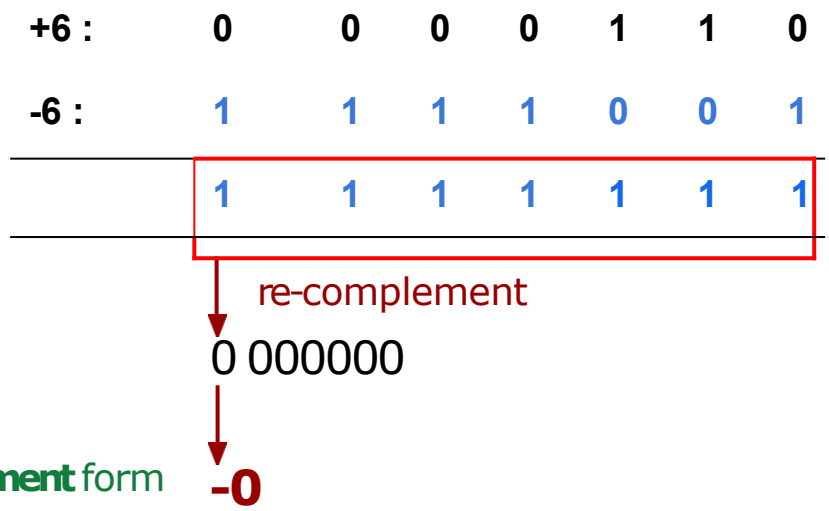
(+6) + (-9)
using sign 2s complement, on 7 bit register

No end carry

No overflow

Sign bit 1, so result is in **complement** form

(+6) + (-6)
using sign 1s complement, on 7 bit register

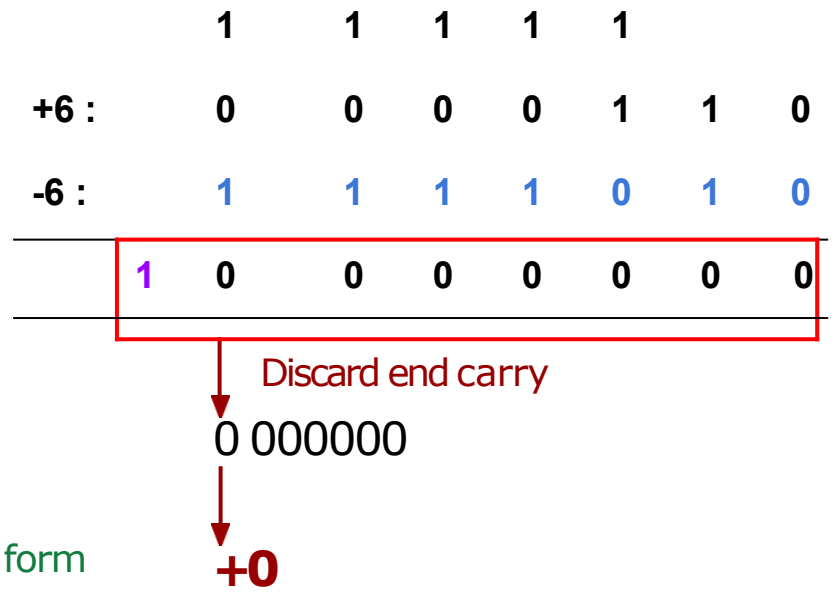


No end carry

No overflow

Sign bit 1, so result is in **complement** form

(+6) + (-6)
using sign 2s complement, on 7 bit register

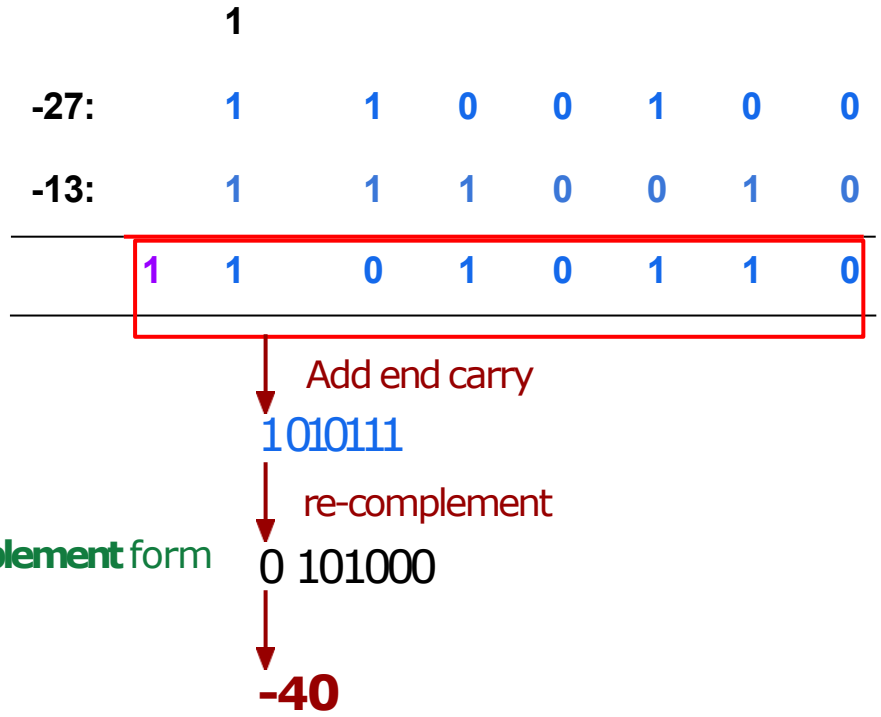


End carry occured

No overflow

Sign bit 0, so result is in **original** form

(-27) + (-13)
using sign 1s complement, on 7 bit register

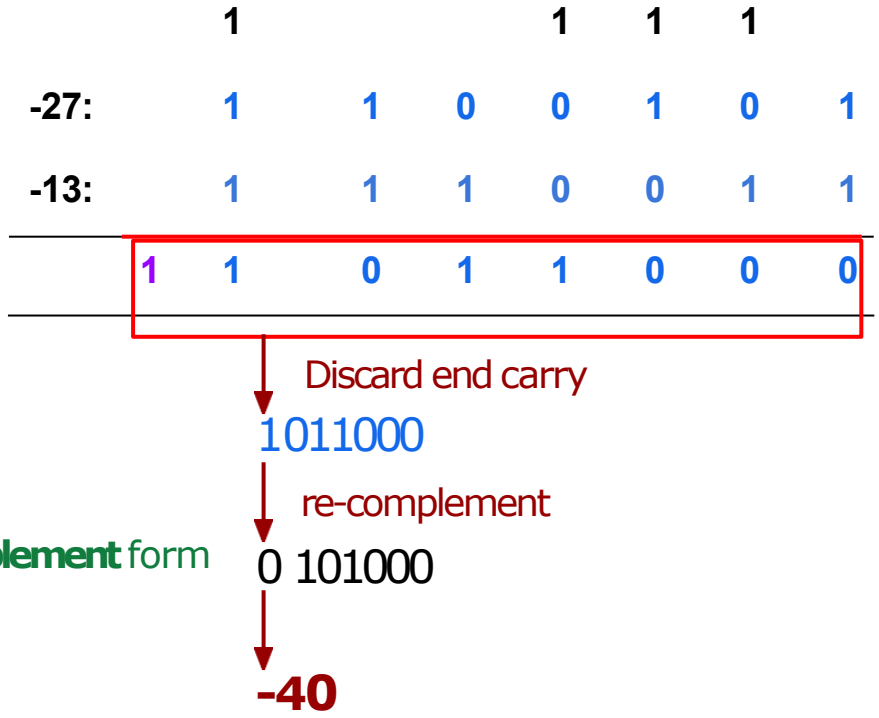


End carry occured

No overflow

Sign bit 1, so result is in **complement** form

(-27) + (-13)
using sign 2s complement, on 7 bit register



End carry occured

No overflow

Sign bit 1, so result is in **complement** form

An overflow occurs when the result of the arithmetic operation is too big to fit **within the specified number of bits.**

Until now we have not seen such cases where an overflow may occur during signed binary addition.

If the operands have a different sign, an **overflow will never occur. As the result can never be larger than the operands itself.**

(+□) + (-□) or **(-□) + (+□)**

But if the operands have the same sign, an overflow may or may not occur.

(+□) + (+□) or **(-□) + (-□)**

If the operands are of the same sign, but the result has a different sign, then it can be said that an overflow has **occurred.**

(+□) + (+□) = (-□) or (-□) + (-□) = (+□)

Another easy way to detect overflow: The Carry In at the sign bit position is different from the Carry Out at the sign bit position.

Handling overflows in signed binary addition

- At first, the end carry issue is handled. Then, the overflow situation is handled.
- If an overflow is detected, the MSB or the sign bit becomes the data bit while the carry out becomes the new sign bit.

(+35) + (+40)
using sign 2s complement, on 7 bit register

		1					
+35:	0	1	0	0	0	1	1
+40:	0	1	0	1	0	0	0
0	1	0	0	1	0	1	1

Sign bit becomes data bit. End carry is the new sign bit.

0 1001011

+75

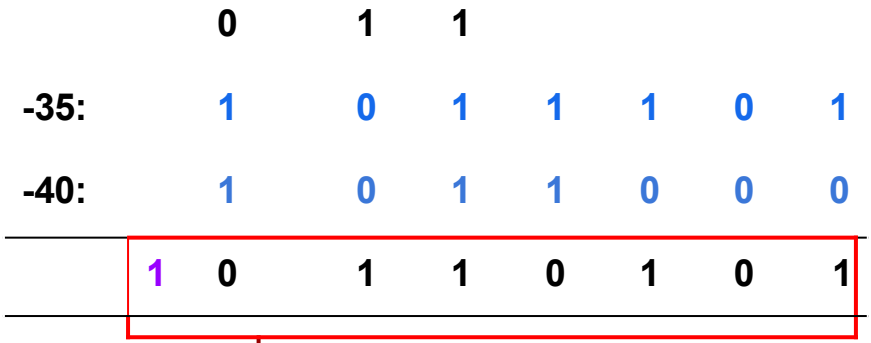
No end carry

Overflow occurred

New Sign bit 0, so result is in **original** form

Try,
(+35) + (+40)
using sign 1s complement, on 7 bit register

(-35) + (-40)
using sign 2s complement, on 7 bit register



End carry occured

Overflow occurred

New Sign bit 1, so result in **complement** form

Discard end carry

0 110101

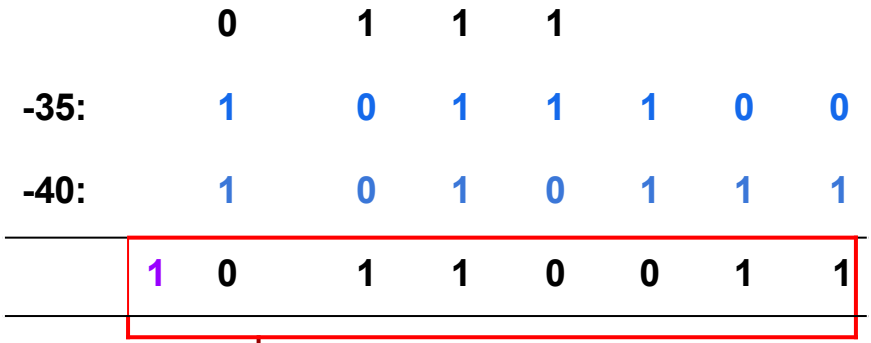
Sign bit becomes data bit. End carry is the new sign bit.

1 0 110101

re-complement

0 1001011 → **-75**

(-35) + (-40)
using sign 1s complement, on 7 bit register



End carry occured

Overflow occurred

New Sign bit 1, so result in **complement** form

Add end carry

0 110100

Sign bit becomes data bit. End carry is the new sign bit.

1 0 110100

re-complement

0 1001011 → **-75**

Try,
(-6) + (-9)
using sign 1s complement, on 5 bit register

Signed Binary Subtraction

Subtraction of two signed binary numbers can be easily done by converting them to addition operation.

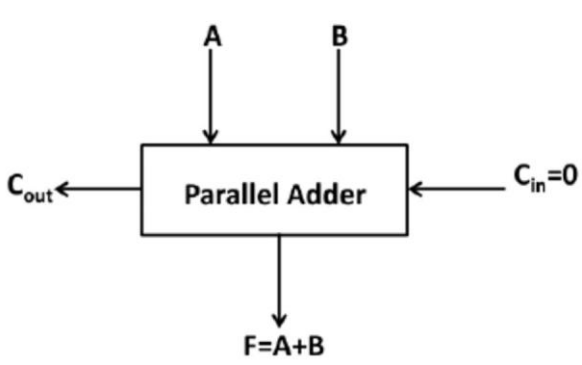
$$(\pm A) - (-B) = (\pm A) + (+B)$$

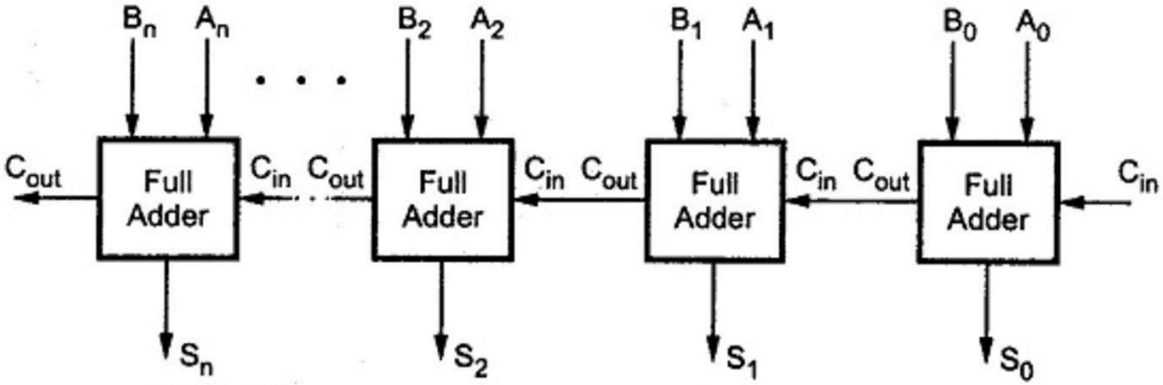
$$(\pm A) - (+B) = (\pm A) + (-B)$$

Try, (-35)-(+40) using sign-2s complement on a 7 bit register

Adder Circuits and Flags

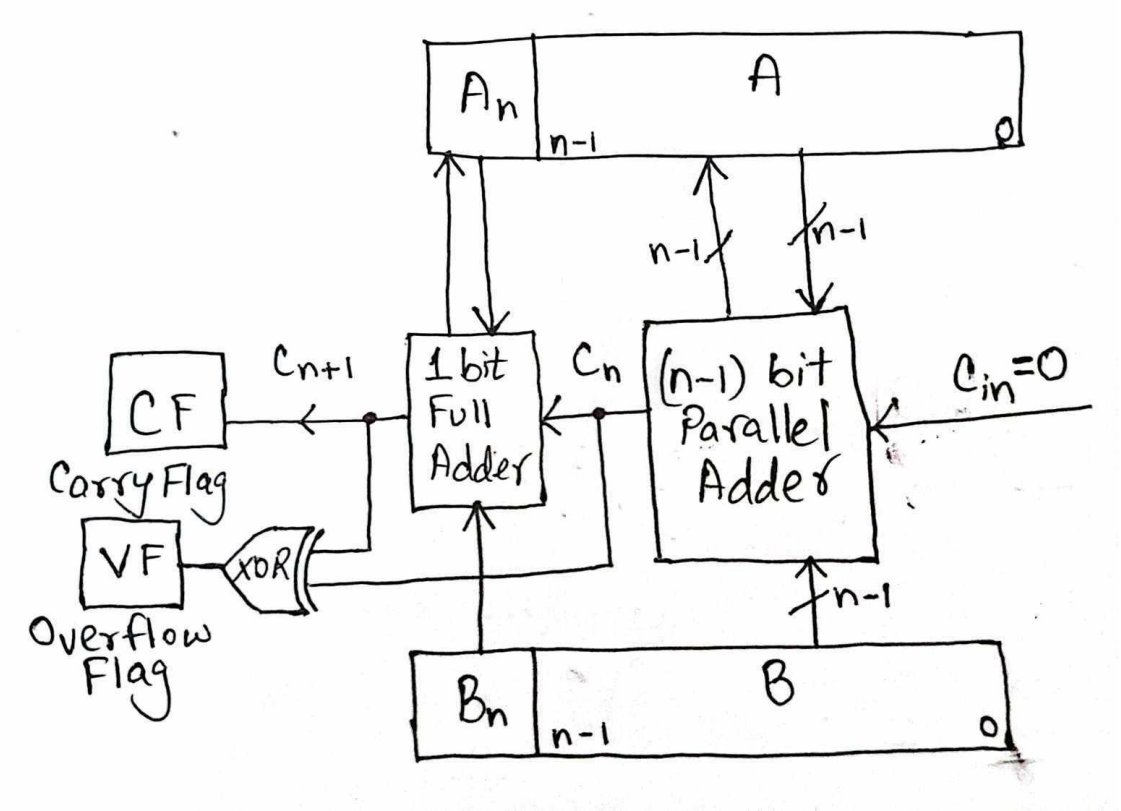
- The addition operations can be done through **Parallel Adders**.
- The inputs to a parallel adder are both of the operands (A and B) and the Carry In (C_{in}). C_{in}=0 in case of addition.
- While the outputs are the Summation result and the Carry Out, C_{out}.





N-bit **parallel adder** made up of N*1 bit **full adders**

Adder Circuits and Flags



- Carry Flag, **CF = C_{out} = C_{n+1}**
- Overflow Flag, **VF = $C_n \oplus C_{n+1}$**
- VF needs access to the last two carries, C_{n+1} and C_n.
- But the parallel adder does not provide C_n as output.

Adder Circuits and Flags

Suppose we are adding two different signed binary numbers +A and -B.

Now,

- If Carry Flag = 1, then A>B. So, (+A) + (-B) would give a positive result.
- If Carry Flag = 0, then B>A. So, (+A) + (-B) would give a negative result.

Suppose we are adding two same signed binary numbers +A and +B, or -A and -B.

Now,

- If Overflow Flag = 1, a signed overflow has occurred.
- If Overflow Flag = 0, a signed overflow did not occur.

Range of Signed Binary Numbers

If a number is represented in a register that can accomodate n bits, then n=k+1. Here, k is the number of data bits, with one sign bit at the msb.

For sign magnitude representation, $+(2^k-1)$ to $-(2^k-1)$

For sign 1s complement representation, $+(2^k-1)$ to $-(2^k-1)$

For sign 2s complement representation, $+(2^k-1)$ to -2^k

For a 3 bit number, n=3. So, k=2.

Decimal Number	Sign Magnitude	Sign 1s Complement	Sign 2s Complement
+3	0 11	0 11	0 11
+2	0 10	0 10	0 10
+1	0 01	0 01	0 01
+0	0 00	0 00	0 00
-0	100	111	
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4			100

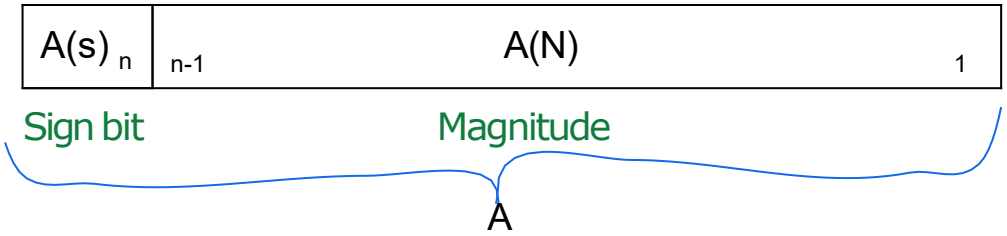
Why do we prefer 2s complement over 1s complement

2s complement requires less number of additions when end carry occurs.

2s complement has a single representation for 0, whereas 1s complement has different representations for +0 and -0.

2s complement can represent one number more than 1s complement using the same number of bits.

Signed Right Shifting



For Sign-Magnitude,

- A(N) <- SHR A(N)
- A_{n-1} <- 0
- A(s) <- A(s)

For Sign-1s complement and Sign-2s complement,

- A <- SHR A
- A(s) <- A(s)

Since right shift divides a number by 2, so an overflow will never occur after the shift operation.

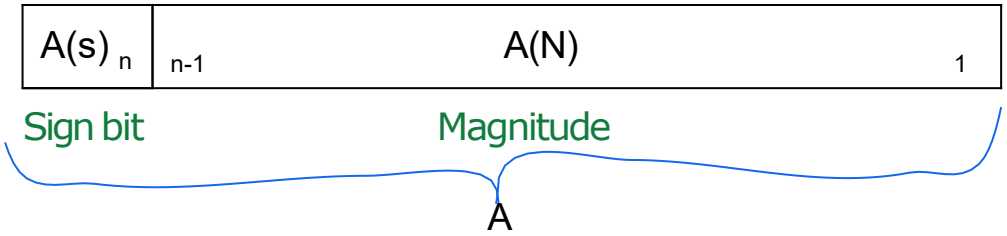
Right Shift +9 on a 6 bit register

	+9 before shifting	+4 after shifting
Sign Magnitude	0 01001	0 00100
Sign 1s Complement	0 01001	0 00100
Sign 2s Complement	0 01001	0 00100

Right Shift -12 on a 6 bit register

	-12 before shifting	-6 after shifting
Sign Magnitude	1 01100	1 00110
Sign 1s Complement	1 10011	1 11001
Sign 2s Complement	1 10100	1 11010

Signed Left Shifting



For Sign-Magnitude,

- A(N) <- SHL A(N)
- A_1 <- 0
- A(s) <- A(s)

For Sign-1s complement,

- A <- SHL A
- A_1 <- A(s)

For Sign-2s complement,

- A <- SHL A
- A_1 <- 0

Since left shift multiplies a number by 2, so an overflow may occur after the shift operation.

For sign-magnitude, overflow will occur after left shifting if $A_{n-1}=1$

For sign-1s and sign-2s complement, overflow will occur after left shifting if $A_n \oplus A_{n-1}=1$

Left Shift +9 on a 6 bit register

	+9 before shifting	+18 after shifting
Sign Magnitude	0 01001	0 10010
Sign 1s Complement	0 01001	0 10010
Sign 2s Complement	0 01001	0 10010

Left Shift -12 on a 6 bit register

	-12 before shifting	-24 after shifting
Sign Magnitude	1 01100	1 11000
Sign 1s Complement	1 10011	1 00111
Sign 2s Complement	1 10100	1 01000

Left Shift -24 on a 6 bit register

	-24 before shifting	Supposed to be -48 after shifting
Sign Magnitude	1 11000 Overflow will occur	1 10000 -16, which is wrong
Sign 1s Complement	1 00111 Overflow will occur	0 01111 +15, which is wrong
Sign 2s Complement	1 01000 Overflow will occur	0 10000 +16, which is wrong

Thank You