COMPUTER ORGANIZATION AND ARCHITECTURE

Course Code : CSE 2151

Credits : 04

SIGN AND MAGNITUDE: ARITHMETIC OPERATIONS

- Add +5 and -2 = +3 (4-bit representation)

  0 101 +

  1 010

  111 (7) wrong answer!!

- Subtract -6 and 1 = -7 (4-bit representation)

  1 110 -

  0 001

  101 (5) wrong answer!!

- Hence, not suitable for arithmetic operations

1'S COMPLEMENT: ARITHMETIC OPERATION

- Add +1 and -1 = 0 (4-bit representation)

  0 001 +

  1 110

  111 (wrong answer!!)

  \+ 1

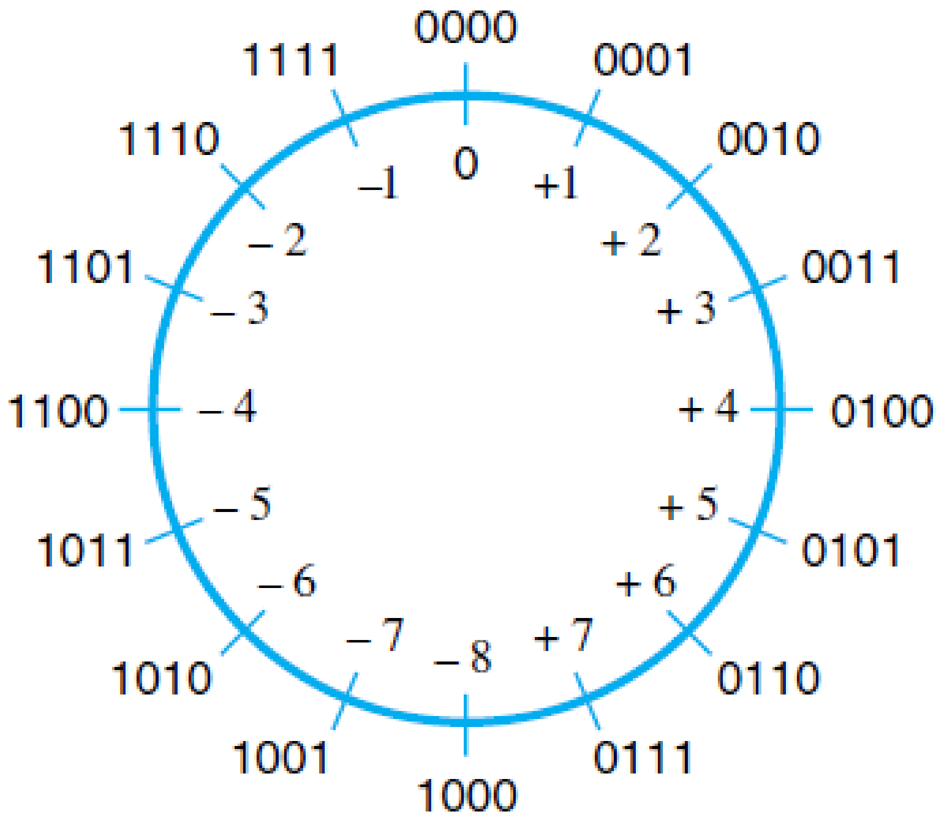
  000 (correct answer)

- Additional step (hardware) required to arrive at the correct answer.
- Hence, not suitable for arithmetic operations

2'S COMPLEMENT: ARITHMETIC OPERATION (ADDITION)

- Add +7 and -3
  - +7 is 0111 and -3 is 1101,
  - Locate 0111 in the diagram and move 1101 (13) steps in clockwise
  - Solution: 0100
  - 2's-complement representation of −3 is interpreted as an unsigned value for the number of steps to move.
- Adding bit pair wise:
  - ignore the carry-out

$$\begin{array}{ccccc} & 0 & 1 & 1 & 1 \\ + & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 \\ \uparrow & & & & \\ \text{Carry-out} & & & & \end{array}$$



2'S COMPLEMENT: ADDITION

- +2 + (+3)

  +2→ 0010 +
  +3→ 0011
  +5→ 0101

- +4 + (-6)

  +4→ 0100 +
  -6→ 1010
  -2→ 1110

- -5 + (-2)

  -5→ 1011 +
  -2→ 1110
  -7→ 1001

  Carry is ignored

- +7 + (-3)

  +7→ 0111 +
  -3→ 1101
  +4→ 0100

  Carry is ignored

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

2'S COMPLEMENT: ADDITION AND SUBTRACTION

- To add two numbers (X+Y),
  - Represent X and Y in binary format (2's complement representation)
  - add their n-bit representations,
  - ignore the carry-out bit from the most significant bit (MSB) position.
  - The sum will be the algebraically correct value in 2's-complement representation if the actual result is in the range -2^{n-1} through $+2^{n-1} - 1$.
- To subtract two numbers X and Y (X − Y),
  - Represent X and Y in binary format (2's complement representation)
  - form the 2's-complement of Y,
  - add it to X using the add rule.
  - The result will be the algebraically correct value in 2's-complement representation if the actual result is in the range -2^{n-1} through $+2^{n-1} - 1$.

2'S COMPLEMENT: SUBTRACTION

- -7- (-5)

-7→	1001 -		-7→	1001 +
-5→	1011	Becomes:	5→	0101
			-2→	1110

- -7- (+1)

-7→	1001 -		-7→	1001 +
+1→	0001	Becomes:	-1→	1111
			-8→	1000

- +2- (-3)

+2→	0010 -		+2→	0010 +
-3→	1101	Becomes:	+3→	0011
			+5→	0101

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

2'S COMPLEMENT: SUBTRACTION

- -3- (-7)

-3→	1101 -		-3→	1101 +
-7→	1001	Becomes:	7→	0111
			+4→	0100

- +2- (+4)

+2→	0010 -		+2→	0010 +
+4→	0100	Becomes:	-4→	1100
			-2→	1110

- +6- (+3)

+6→	0110 -		+6→	0110 +
+3→	0011	Becomes:	-3→	1101
			+3→	0011

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

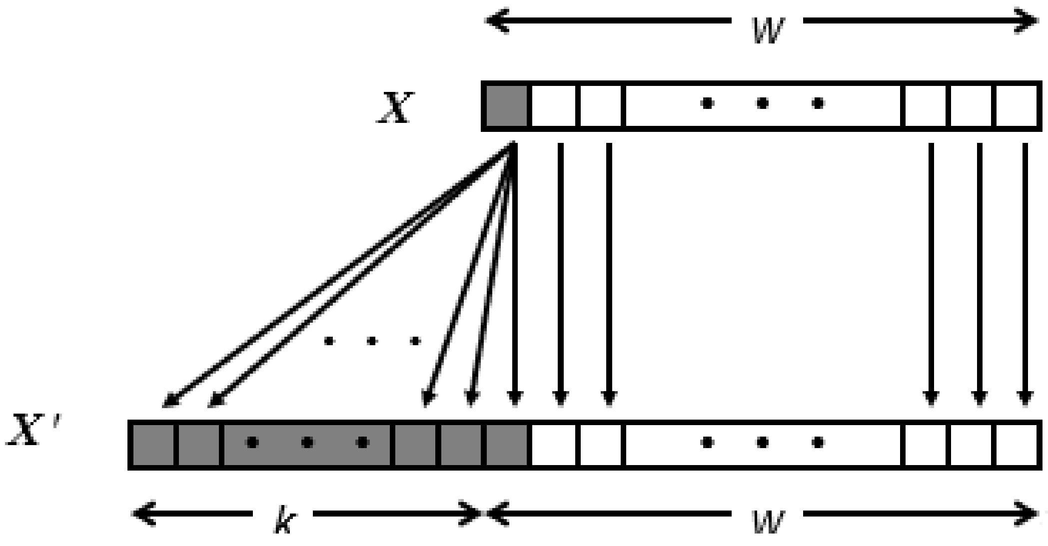
SIGNED INTEGERS: ADDITION AND SUBTRACTION

- Sign and Magnitude:
  - Undesired results
- 1's Complement:
  - The results are not always correct
- 2's Complement:
  - simplicity of adding and subtracting signed numbers
  - used in modern computers

SIGN EXTENSION

- Represent a value given in a certain number of bits by using a larger number of bits
- Positive numbers: zeroes are added to the left
- Negative numbers: ones are added to the left

- To convert a given (w)-bit, signed integer x to (w+k)-bit integer with same value
  - Make k copies of sign bit:
  - $X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$



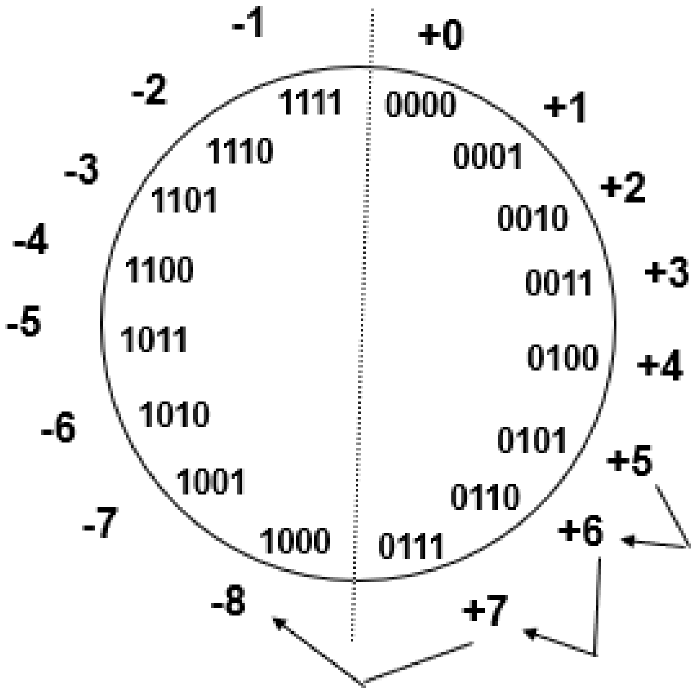
SIGN EXTENSION: EXAMPLE

```
short int x =  15213;
int       ix = (int) x;
short int y = -15213;
int       iy = (int) y;
```

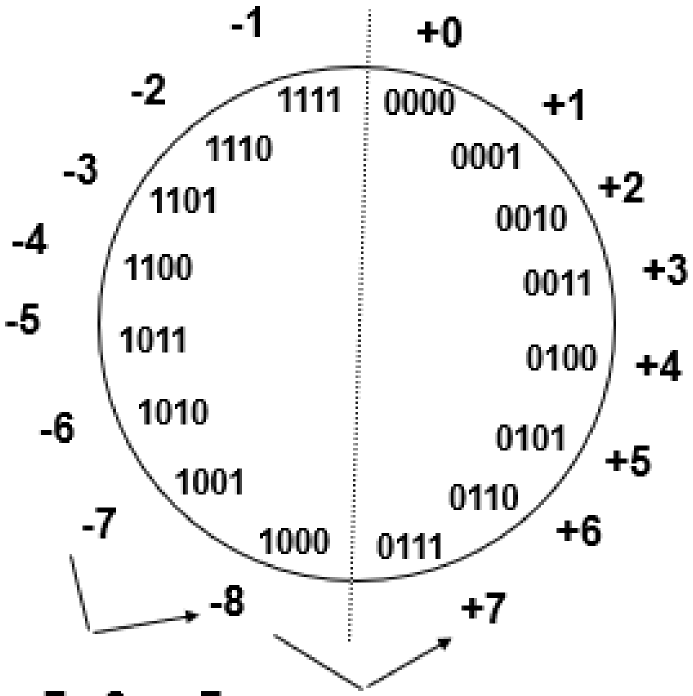
	Decimal	Binary
x	15213	00111011 01101101
ix	15213	00000000 00000000 00111011 01101101
y	-15213	11000100 10010011
iy	-15213	11111111 11111111 11000100 10010011

OVERFLOW IN INTEGER ARITHMETIC

- Arithmetic overflow:
  - The actual result of an arithmetic operation is outside the representable range
- Overflow occurs when
  - two positive numbers are added which results in a negative number or
  - two negative numbers are added which results in a positive number



5 + 3 = -8



-7 - 2 = +7

OVERFLOW IN INTEGER ARITHMETIC

- +7 + (+4)

```
carry:   0100
+7→       0111 +
+4→       0100
-----------------
 -5→       1011
```

- -4 + (-6)

```
carry:   1000
-4→       1100 +
-6→       1010
-----------------
+6→       0110
```

- The value of the carry-out bit from the sign-bit position is not an indicator of overflow
- Overflow occurs when carry-in to the high-order bit does not equal carry out

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

OVERFLOW IN INTEGER ARITHMETIC

- +5 + (+3)

carry:	0111
+5→	0101 +
+3→	0011
-8→	1000

Overflow

- -7 + (-2)

carry:	1000
-7→	1001 +
-2→	1110
+7→	0111

Overflow

- +5 + (+2)

carry:	0000
+5→	0101 +
+2→	0010
+7→	0111

No overflow

- -3 + (-5)

carry:	1111
-3→	1101 +
-5→	1011
-8→	1000

No overflow

$b_3b_2b_1b_0$	Value in decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

CHARACTER REPRESENTATION

- The most common encoding scheme for characters is ASCII
- Alphanumeric characters, operators, punctuation symbols, and control characters represented using 7-bit codes
- 8-bit byte is used to represent and store a character
- The code occupies the low-order seven bits
- The high-order bit is usually set to 0

FLOATING-POINT NUMBERS

- The basic IEEE format is a 32-bit representation that comprises of
  - a sign bit,
  - 23 significant bits, and
  - 8 bits for a signed exponent of the scale factor
- IEEE standard also defines a 64-bit representation to accommodate
  - more significant bits, and
  - more bits for the signed exponent, resulting in much higher precision and a much larger range of values
- In general, a binary floating-point number can be represented by (2008 version of IEEE Standard 754):
  - a sign for the number
  - some significant bits
  - a signed scale factor exponent for an implied base of 2

TOPICS COVERED FROM

- Textbook 1:
  - Chapter 1: 1.4.1, 1.4.2, 1.5