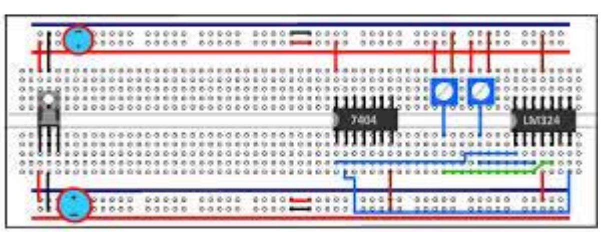
CSE 260 : **Digital Logic Design**

Number Systems and Codes

Getting ready for Lab

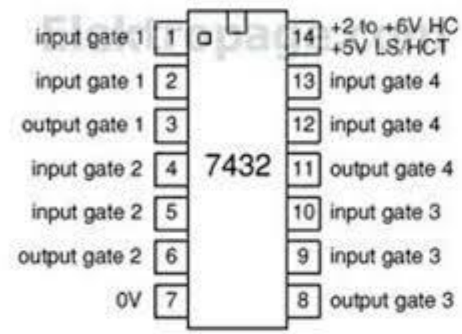
DSCH2

- Bread board

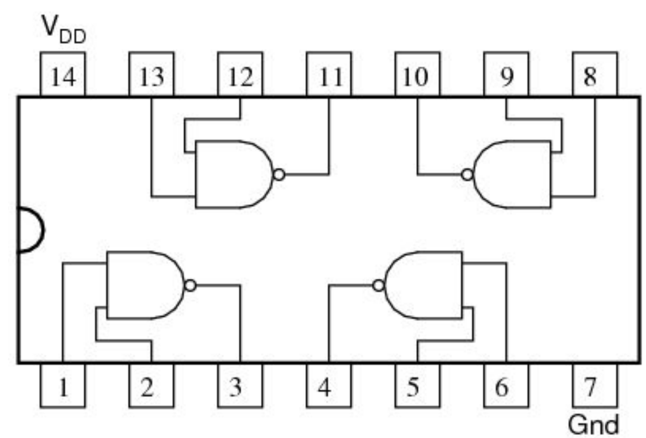


join me on
facebook.com/letslearnelectronics

- IC



"Pinout," or "connection" diagram for the 4011 quad NAND gate



Binary Coded Decimal (BCD)

- Decimal numbers are more natural to humans. Binary numbers are natural to computers. Quite expensive to convert between the two.
- If little calculation is involved, we can use some *coding schemes* for decimal numbers.
- One such scheme is BCD, also known as the *8421* code.
- Represent each decimal digit as a 4-bit binary code.

Binary Coded Decimal (BCD)

Decimal digit **BCD**	0 **0000**	1 **0001**	2 **0010**	3 **0011**	4 **0100**
Decimal digit **BCD**	5 **0101**	6 **0110**	7 **0111**	8 **1000**	9 **1001**

- Some codes are unused, eg: $(1010)_{BCD}$, $(1011)_{BCD}$, …, $(1111)_{BCD}$. These codes are considered as errors.
- Easy to convert, but arithmetic operations are more complicated.
- Suitable for interfaces such as keypad inputs and digital readouts.

Binary Coded Decimal (BCD)

Decimal digit **BCD**	0 **0000**	1 **0001**	2 **0010**	3 **0011**	4 **0100**
Decimal digit **BCD**	5 **0101**	6 **0110**	7 **0111**	8 **1000**	9 **1001**

- Examples:

  $(234)_{10} = (0010\ 0011\ 0100)_{BCD}$

  $(7093)_{10} = (0111\ 0000\ 1001\ 0011)_{BCD}$

  $(1000\ 0110)_{BCD} = (86)_{10}$

  $(1001\ 0100\ 0111\ 0010)_{BCD} = (9472)_{10}$

Notes: BCD is not equivalent to binary.

Example: $(234)_{10} = (11101010)_2$

Binary Codes

- Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi-nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

Negative Numbers Representation

- There are three common ways of representing signed numbers (positive and negative numbers) for binary numbers:
  - Sign-and-Magnitude
  - 1s Complement
  - 2s Complement

Sign-and-Magnitude

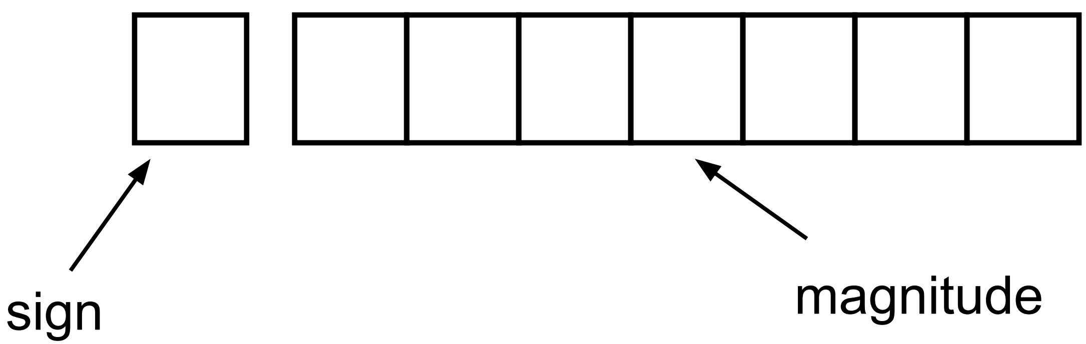
- Negative numbers are usually written by writing a minus sign in front.
  - Example:
    - $-(12)_{10}$, $-(1100)_2$
- In computer memory of fixed width, this sign is usually represented by a bit:

  0 for +

  1 for −

Sign-and-Magnitude

- Example: an 8-bit number can have 1-bit sign and 7-bits magnitude.



Signed magnitude representation

- Examples:

1101_2 = 13_{10} (a 4-bit unsigned number)
0 1101= $+13_{10}$ (a positive number in 5-bit signed magnitude)
1 1101= -13_{10} (a negative number in 5-bit signed magnitude)

0100_2 = 4_{10} (a 4-bit unsigned number)
0 0100 = $+4_{10}$ (a positive number in 5-bit signed magnitude)
1 0100 = -4_{10} (a negative number in 5-bit signed magnitude)

Sign-and-Magnitude

- Largest Positive Number: 0 1111111 $+(127)_{10}$
- Largest Negative Number: 1 1111111 $-(127)_{10}$
- Zeroes: 0 0000000 $+(0)_{10}$
  1 0000000 $-(0)_{10}$
- Range: $-(127)_{10}$ to $+(127)_{10}$
- Signed numbers needed for negative numbers.
- Representation: Sign-and-magnitude.

1s Complement

- Given a number *x* which can be expressed as an *n*-bit binary number (*i.e. integer part has n digits and fraction has m digit*), its negative value can be obtained in 1s-complement representation using:

+7	0111	-7	1000
+6	0110	-6	1001
+5	0101	-5	1010
+4	0100	-4	1011
+3	0011	-3	1100
+2	0010	-2	1101
+1	0001	-1	1110
+0	0000	-0	1111

1's complement

Approach 1

- x_{base} =>base^n – base^m– x

Example: With an 8-bit number 00001100, its negative value, expressed in 1s complement, is obtained as follows

$-(00001100)_2 = -(12)_{10}$

$\Rightarrow (2^8 - 2^0 - 12)_{10}$

$\Rightarrow (2^8 - 1 - 12)_{10}$

$= (243)_{10}$

$= \mathbf{(11110011)_{1s}}$

Approach 2

$-(00001100)_2$

inverse

$(11110011)_2$

1s Complement

- Essential technique: invert all the bits.

  Examples: 1s complement of 00000001 = $(11111110)_{1s}$

  1s complement of 01111111 = $(10000000)_{1s}$
- Largest Positive Number: 0 1111111 $+(127)_{10}$
- Largest Negative Number: 1 0000000 $-(127)_{10}$
- Zeroes: 0 0000000

  1 1111111
- Range: $-(127)_{10}$ to $+(127)_{10}$ $=-(2^{n-1}-1)$ to $+(2^{n-1}-1)$
- The most significant bit still represents the sign:

  0 = +ve; 1 = -ve

Note: Range for n bit no. is –(2^(n-1)-1) to (2^(n-1)-1)

1s Complement

- Examples (assuming 8-bit binary numbers):

  $(14)_{10} = (00001110)_2 = (00001110)_{1s}$

  $-(14)_{10} = -(00001110)_2 = (11110001)_{1s}$

  $-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{1s}$

Other (base-1) complements

Formula: $\mathbf{r^n - r^{-m} - N}$

<u>9's Complement</u>

- Example:
  - **$(52520)_{10}$ is $= 10^5 - 1 - 52520$**

    $= 99999 - 52520$

    $= 47479$
  - **$(25.639)_{10}$ is $= 10^2 - 10^{-3} - 25.639$**

    $= 99.999 - 25.639$

    $= 74.360$

2s Complement

- Given a number x which can be expressed as an n-bit (*i.e. integer part has n digits and fraction has m digit*) number , its negative number can be obtained in 2s-complement representation using:

$$- x = 2^n - x$$

Example: With an 8-bit number 00001100, its negative value in 2s complement is thus:

$$\begin{aligned} -(00001100)_2 &= -(12)_{10} \\ &= (2^8 - 12)_{10} \\ &= (244)_{10} \\ &= \mathbf{(11110100)_{2s}} \end{aligned}$$

2s Complement

- **Method 1:** Essential technique: invert all the bits and add 1.

  Examples:

  Official method!

  2s complement of

  $(00000001)_{2s} = (11111110)_{1s}$ (invert i.e 1's complement)

  $= (11111111)_{2s}$ (add 1)

  2s complement of

  $(01111110)_{2s} = (10000001)_{1s}$ (invert i.e 1's complement)

  $= (10000010)_{2s}$ (add 1)

- **Method 2:** Keep unchanged till 1st occurrence of 1 from LSB and invert remaining 1's into 0's and 0's into 1's till MSB

  $(01111110)_{2s} = (10000010)_{2s}$

  Unchanged

  Inverted

  Unofficial method!

2s Complement

- Largest Positive Number: 0 1111111
  $+(127)_{10}$
- Largest Negative Number: 1 0000000
  $-(128)_{10}$
- Zero: 0 0000000
- Range: $-(128)_{10}$ to $+(127)_{10} = -(2^{n-1})$ to $+(2^{n-1}-1)$
- The most significant bit still represents the sign:
  0 = +ve; 1 = -ve.

Note: Range for n bit no. is -2^(n-1) to 2^(n-1)-1

2s Complement

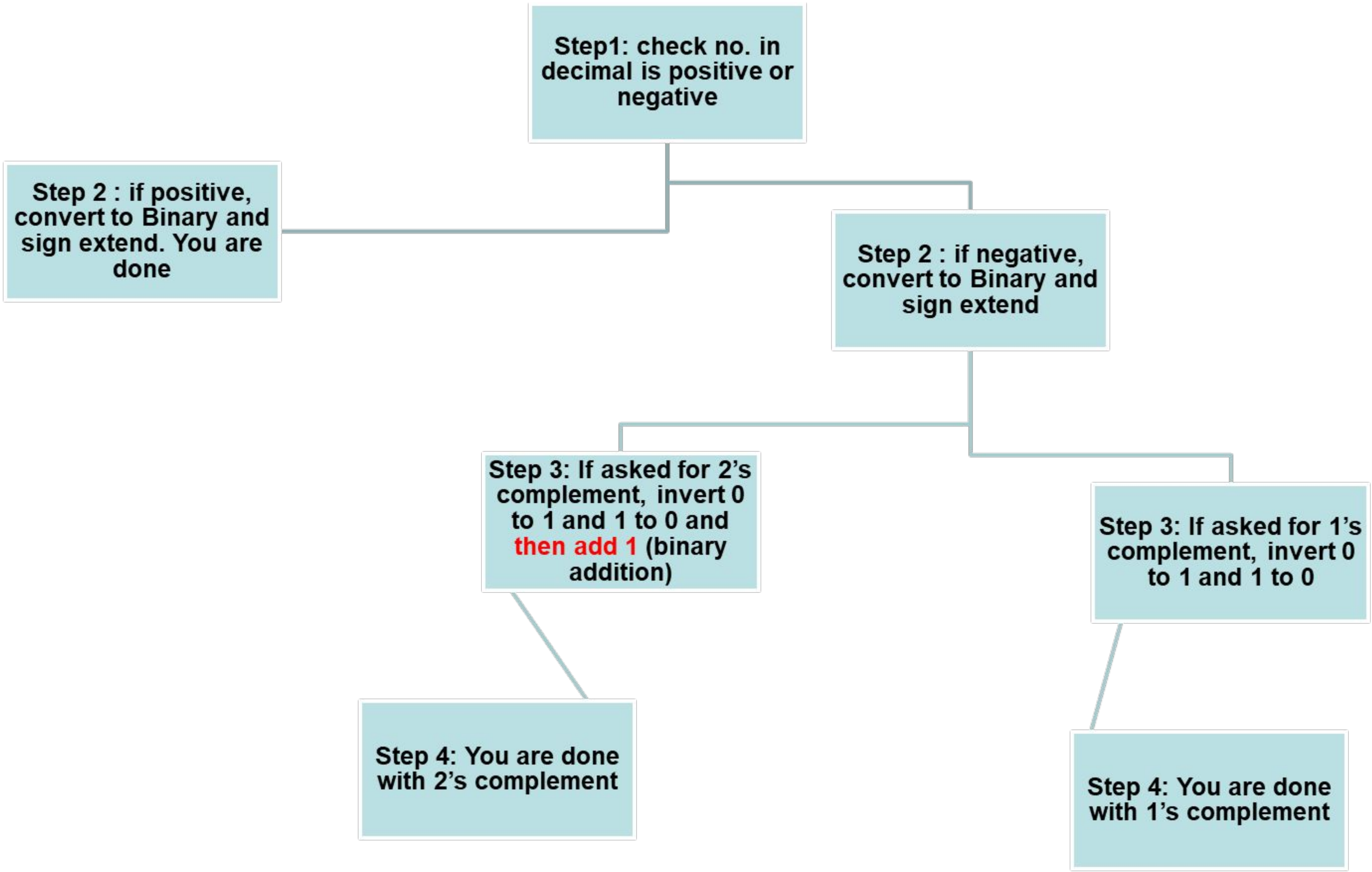
- Examples (assuming 8-bit binary numbers):

  $(14)_{10} = (00001110)_2 = (00001110)_{2s}$

  $-(14)_{10} = -(00001110)_2 = (11110010)_{2s}$

  $-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{2s}$

Summary of what we learnt so far



Comparisons of Sign-and-Magnitude and Complements

- Example: 4-bit signed number (*positive values*)

Value	Sign-and-Magnitude	1s Comp.	2s Comp.
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000

Note: Signed magnitude cannot be used for arithmetic calculations

Comparisons of Sign-and-Magnitude and Complements

- Example: 4-bit signed number (*negative values*)

Value	Sign-and-Magnitude	1s Comp.	2s Comp.
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	-	-	1000

- Note: Signed magnitude cannot be used for arithmetic calculations

Exercise:

1. For 2’s complement binary numbers, the range of values for 5-bit numbers is

   a. 0 to 31 b. -8 to +7 c. -8 to +8

   d. -15 to +15 e. -16 to +15

2. In a 6-bit 2’s complement binary number system, what is the decimal value represented by $(100100)_{2s}$?

   a. -4 b. 36 c. -36 d. -27 e. -28

Practice Time

1) Following numbers are in 1’s complement system. Turn them to their no. negative representation

A. 1010101
B. 0111000
C. 0000001
D. 00000

2) Now perform 2’s complement

Answer

1's complement

A. Already negative

B. 1000111

C. 1111110

D. 11111

2's complement

Already negative

A. 1001000

B. 1111111

C. 00000

Comparison between 1's and 2's complement

1's complement

- Easier to implement (convert between 0s and 1s)
- 2 representation of 0 (0000,1111)
- 1's complement is mostly used in *non*-arithmetic applications

2's complement

- Easier substraction (no need of adding 1 in the case of carry)
- 2's complement is mostly used in arithmetic applications

UNSIGNED NO. ARTHEMATIC OPERATION!

Binary Arithmetic Operations for Unsigned numbers

- ADDITION

- Like decimal numbers, two numbers can be added by adding each pair of digits together with carry propagation.

$$\begin{array}{r} (11011)_2 \\ +\ (10011)_2 \\ \hline (101110)_2 \\ \hline \end{array} \qquad \begin{array}{r} (647)_{10} \\ +\ (537)_{10} \\ \hline (1184)_{10} \\ \hline \end{array}$$

Binary Arithmetic Operations for Unsigned Numbers

- SUBTRACTION

- Two numbers can be subtracted by subtracting each pair of digits together with borrowing, where needed.

$$
\begin{array}{rr}
 & (11001)_2 \\
- & (10011)_2 \\
\hline
 & (00110)_2 \\
\hline
\end{array}
\qquad
\begin{array}{rr}
 & (627)_{10} \\
- & (537)_{10} \\
\hline
 & (090)_{10} \\
\hline
\end{array}
$$

Overflow!

Unsigned numbers overflow

- Carry-out can be used to detect overflow
- The largest number that we can represent with 4-bits using unsigned numbers is 15
- Suppose that we are adding 4-bit numbers: 9 (1001) and 10 (1010).

1 0 0
1 (9)
+ 1 0 1
0 (10)
1 0 0
1 1 (19)

- The value 19 cannot be represented with 4-bits
- When operating with unsigned numbers, **a carry-out** of 1 **can be used to indicate overflow**

!!Overflow in ***Signed*** Number!!

- With two's complement and a 4-bit adder, for example, the largest representable decimal number is +7, and the smallest is -8.
- What if you try to compute 4 + 5, or (-4) + (-5)?

```
    0 1 0 0   (+4)          1 1 0 0   (-4)
+   0 1 0 1   (+5)      +   1 0 1 1   (-5)
-------------------     -------------------
  0 1 0 0 1   (-7)        1 0 1 1 1   (+7)
```

- We cannot just include the carry out to produce a five-digit result, as for unsigned addition. If we did, (-4) + (-5) would result in +7!
- Also, unlike the case with unsigned numbers, the carry out *cannot* be used to detect overflow.
  - In the example above, the carry out is 0 but there *is* overflow.
  - Conversely, there are situations where the carry out is 1 but there is *no* overflow.

Detecting signed overflow

- The easiest way to detect signed overflow is to look at all the sign bits.

		0	1					1	1	0	
0	0		(+4)				0		(-4)		
+		0	1			+		1	0	1	
0	1		(+5)				1		(-5)		
	0	1	0				1	0	1	1	
0	1		(-7)				1		(+7)		

- Overflow occurs only in the two situations above:
  - If you add two *positive* numbers and get a *negative* result.
  - If you add two *negative* numbers and get a *positive* result.

  (Another way of detecting overflow is Carry in of MSB ≠ Carry out of MSB)

- Overflow cannot occur if you add a positive number to a negative number. Do you see why?

Problem in signed arithmetic operation: Overflow

- Signed binary numbers are of a fixed range.
- If the result of addition/subtraction goes beyond this range, overflow occurs.
- *In case of unsigned no*, if there is a carry out in MSB, then overflow has occurred.
- *In signed number*, two conditions under which overflow can occur are:
  (i) *positive add positive* gives negative
  (ii) *negative add negative* gives positive

  OR

  Carry in of MSB ≠ Carry out of MSB

Overflow in Signed Number

- Examples: 4-bit numbers (in 2s complement)

- Range : $(1000)_{2s}$ to $(0111)_{2s}$ or (-8_{10} to 7_{10})

(i) $(0101)_{2s} + (0110)_{2s} = (1011)_{2s}$

$(5)_{10} + (6)_{10} = -(5)_{10}$?! (overflow!)

(ii) $(1001)_{2s} + (1101)_{2s} = (\underline{1}0110)_{2s}$ discard end-carry

$= (0110)_{2s}$

$(-7)_{10} + (-3)_{10} = (6)_{10}$?! (overflow!)

More examples on overflow:

Link:
http://sandbox.mc.edu/~bennet/cs110/tc/add.html

Conditions of Overflow Flag :

1. Addition of numbers with same sign,

→ $(+A) + (+B) =$ '+' then $OF = 0$

'−' " $OF = 1$

→ $(-A) + (-B) =$ '−' then $OF = 0$

$(-A) + (-B) =$ '+' " $OF = 1$

★ ২ টা same sign -এর number add করলে -ওই sign না থেকে অন্যটা আসলে OF হয় that means $OF = 1$ otherwise $OF = 0$.

2. Subtraction of numbers with same sign

২ টা same sign -এর number subtract (বিয়োগ) করলে কখনো overflow হয় না that means $OF = 0$ [$(+A) - (+B) =$ no overflow ; $(-A) - (-B) =$ no overflow

3. Addition of numbers with different sign

২ টা different sign (+ or −) number যোগ করলে $OF = 0$ হবে always মানে overflow হবে না.

$(-A) + (+B) =$ no overflow

$(+A) + (-B) =$ no overflow.

4. Subtraction of numbers with different sign

$\Rightarrow (+A) - (-B) = A + B$ -এটাকে simplify করার পর এই সূত্রের form -এ চলে আসে, তাহলে $A+B$ -এর result যদি '+' আসে তাহলে $OF = 0$ -আর যদি result '−' আসে then $OF = 1$.

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

-এটাও simplify -এর পর এই সূত্রের form -এ চলে আসে, তাহলে দুটো same sign -এর number add করলে যদি -ই sign আসে then $OF = 0$ -আর যদি opposite sign আসে তাহলে $OF = 1$.

ARITHMETIC OPERATIONS ON SIGNED NUMBER

2s Complement Addition/Subtraction

- Algorithm for addition, A + B:

1. Perform binary addition on the two numbers.
2. Ignore the carry out of the MSB (most significant bit).
3. Check for overflow: Overflow occurs if the ‘carry in’ and ‘carry out’ of the MSB are different, or if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:

  A – B = A + (–B)

1. Take 2s complement of B by inverting all the bits and adding 1.
2. Add the 2s complement of B to A.

2s Complements Example

Given the two binary numbers *X* = 1010100 and *Y* = 1000011, perform the subtraction (a) X – Y and (b) *Y* − *X* by using 2's complement.

(a)	X =	1010100
	2's complement of Y =	+ $\boxed{0111101}$
	Sum =	10010001
	Discard end carry =	0010001
	Answer. X – Y =	0010001

(b)	Y =	1000011
	2's complement of X =	+ $\boxed{0101100}$
	Sum =	1101111

2s Complement Addition/Subtraction

- Examples: 4-bit binary system

```
  +3        0011
+ +4      + 0100
----      -------
  +7        0111
----      -------
```

```
  -2        1110
+ -6      + 1010
----      -------
  -8       11000
----      -------
```

```
  +6       0110
+ -3     + 1101
----      -------
  +3      10011
----      -------
```

```
  +4        0100
+ -7      + 1001
----      -------
  -3        1101
----      -------
```

- Which of the above is/are overflow(s)?

2s Complement Addition/Subtraction

- More examples: 4-bit binary system

```
   -3         1101
+ -6        + 1010
----        -------
   -9        10111
----        -------
```

```
   +5         0101
+ +6        + 0110
----        -------
  +11         1011
----        -------
```

- Which of the above is/are overflow(s)?

1s Complement Addition/Subtraction

- Algorithm for addition, A + B:

1. Perform binary addition on the two numbers.
2. If there is a carry out of the MSB, add 1 to the result.
3. Check for overflow: Overflow occurs if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:

  A – B = A + (–B)

1. Take 1s complement of B by inverting all the bits.
2. Add the 1s complement of B to A.

1s Complements Example

Repeat Previous Example, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$\begin{array}{rr} X = & 1010100 \\ \text{1's complement of } Y = & +\ 0111100 \\ \text{Sum} = & 10010000 \\ \text{End-around carry} = & \underline{+ \qquad\ 1} \\ \text{Answer. } X - Y = & 0010001 \end{array}$$

(b) $Y - X = 1000011 - 1010100$

$$\begin{array}{rr} Y = & 1000011 \\ \text{1's complement of } X = & \underline{+\,0101011} \\ \text{Sum} = & 1101110 \end{array}$$

1s Complement Addition/Subtraction

- Examples: 4-bit binary system

```
    +3         0011
  + +4       + 0100
  ----       --------
    +7         0111
  ----       --------
```

```
    +5         0101
  + -5       + 1010
  ----       --------
    -0         1111
  ----       --------
```

```
    -2        1101
  + -5   +    1010
  ----       -------
    -7       10111
  ----     +     1
             -------
              1000
```

```
    -3         1100
  + -7       + 1000
  ----       --------
   -10        10100
  ----       +    1
             --------
               0101
```

Exercise:

1. In a 4-bit twos-complement scheme, what is the result of this operation: $(1011)_{2s} + (1001)_{2s}$?

   a. 0100 b. 0010 c. 1100 d. 1001 e. overflow

2. Assuming a 6-bit system, perform subtraction with the following unsigned binary numbers by taking first the 1's complement, and then, the 2's complement, of the second value and adding it with the first value:
   (a) 011010 – 010000 (26 – 16)
   (b) 011010 – 001101 (26 – 13)
   (c) 000011 – 010000 (3 – 16)

Sign extension

- In everyday life, decimal numbers are assumed to have an infinite number of 0s in front of them. This helps in "lining up" numbers.
- To subtract 231 and 3, for instance, you can imagine:

$$\begin{array}{rr} & 231 \\ - & 003 \\ \hline & 228 \end{array}$$

- You need to be careful in extending signed binary numbers, because the leftmost bit is the *sign* and not part of the magnitude.
- If you just add 0s in front, you might accidentally change a negative number into a positive one!
- For example, going from 4-bit to 8-bit numbers:
  - 0101 (+5) should become 0000 0101 (+5).
  - But 1100 (-4) should become 1111 1100 (-4).
- The proper way to extend a signed binary number is to replicate the sign bit, so the sign is preserved.

Practice Problem

- 1-12, 1-15

(Morris Mano Chapter 1)

A-10
B-11
C-12
D-13
E-14
F-15