

# EE 457 Unit 2a

Fixed Point Systems and Arithmetic

Unsigned

2's Complement

Sign and Zero Extension

Hexadecimal Representation

# **SIGNED AND UNSIGNED SYSTEMS**

# Signed Systems

- Several systems have been used
  - 2's complement system
  - 1's complement system
  - Sign and magnitude

# Unsigned and Signed Variables

- Unsigned variables use unsigned binary (normal power-of-2 place values) to represent numbers

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \end{array}$$

- Signed variables use the \_\_\_\_\_ system (\_\_\_\_\_ weight) to represent numbers

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline & & & & & & & \end{array}$$

# 2's Complement System

- MSB has negative weight
- MSB determines sign of the number
  - 1 = negative
  - 0 = positive
- To take the negative of a number (e.g.  $-7 \Rightarrow +7$  or  $+2 \Rightarrow -2$ ), requires \_\_\_\_\_  
\_\_\_\_\_
- This is accomplished by \_\_\_\_\_

1001	$x = -7$
0110	Bit flip (1's comp.)
+ 1	Add 1
<hr style="width: 100%; border: 0.5px solid black; margin: 0;"/> 0111	$-x = -(-7) = +7$

# Zero and Sign Extension

- Extension is the process of increasing the number of bits used to represent a number without changing its value

Unsigned = Zero Extension (Always add leading 0's):

$$111011 = 00111011$$

↑  
Increase a 6-bit number to 8-bit number by zero extending

2's complement = Sign Extension (Replicate sign bit):

$$\text{pos. } 011010 = 00011010$$

$$\text{neg. } 110011 = 11110011$$

Sign bit is just repeated as many times as necessary

# Zero and Sign Truncation

- Truncation is the process of decreasing the number of bits used to represent a number without changing its value

Unsigned = Zero Truncation (Remove leading 0's):

$$\cancel{00}111011 = 111011$$

Decrease an 8-bit number to 6-bit number by truncating 0's. Can't remove a '1' because value is changed

2's complement = Sign Truncation (Remove copies of sign bit):

$$\text{pos. } \cancel{00}011010 = 011010$$

$$\text{neg. } \cancel{111}10011 = 10011$$

Any copies of the MSB can be removed without changing the numbers value. Be careful not to change the sign by cutting off ALL the sign bits.

# Arithmetic & Sign

- You learned the addition (carry-method) and subtraction (borrow-method) algorithms in grade school
- Consider  $A + B$ ...do you definitely use the addition algorithm?
  - What if  $A=(2)$ ,  $B=(-5)$ ?
- Human add/sub algorithm depends on \_\_\_\_\_!!

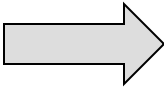
# Unsigned and Signed Arithmetic

- Addition/subtraction process is the same for both unsigned and signed numbers
  - Add columns right to left
  - Drop any final carry out
- This is the KEY reason we use 2's complement system to represent signed numbers
- Examples:

$$\begin{array}{r} \phantom{+} 11 \quad \text{If unsigned} \quad \text{If signed} \\ 1001 \\ + 0011 \\ \hline 1100 \end{array}$$

# Unsigned and Signed Subtraction

- Subtraction process is the same for both unsigned and signed numbers
  - Convert  $A - B$  to  $A + \text{Comp. of } B$
  - Drop any final carry out
- Examples:

	<u>If unsigned</u>	<u>If signed</u>	
1100	(12)	(-4)	
- 0010	(2)	(2)	

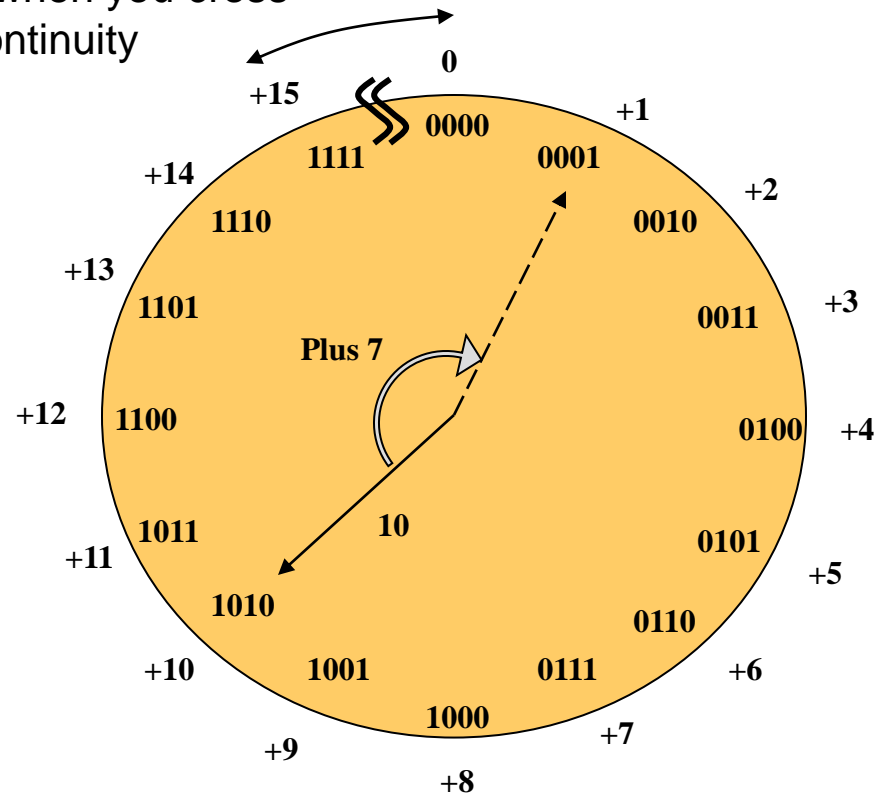
If unsigned    If signed

# Overflow

- Overflow occurs when the result of an arithmetic operation is too \_\_\_\_\_ with the given number of bits
  - Unsigned overflow (\_\_\_) occurs when adding or subtracting unsigned numbers
  - Signed (2's complement overflow) overflow (\_\_\_) occurs when adding or subtracting 2's complement numbers

# Unsigned Overflow

Overflow occurs when you cross this discontinuity



$$10 + 7 = 17$$

$$4 - 6 = 14$$

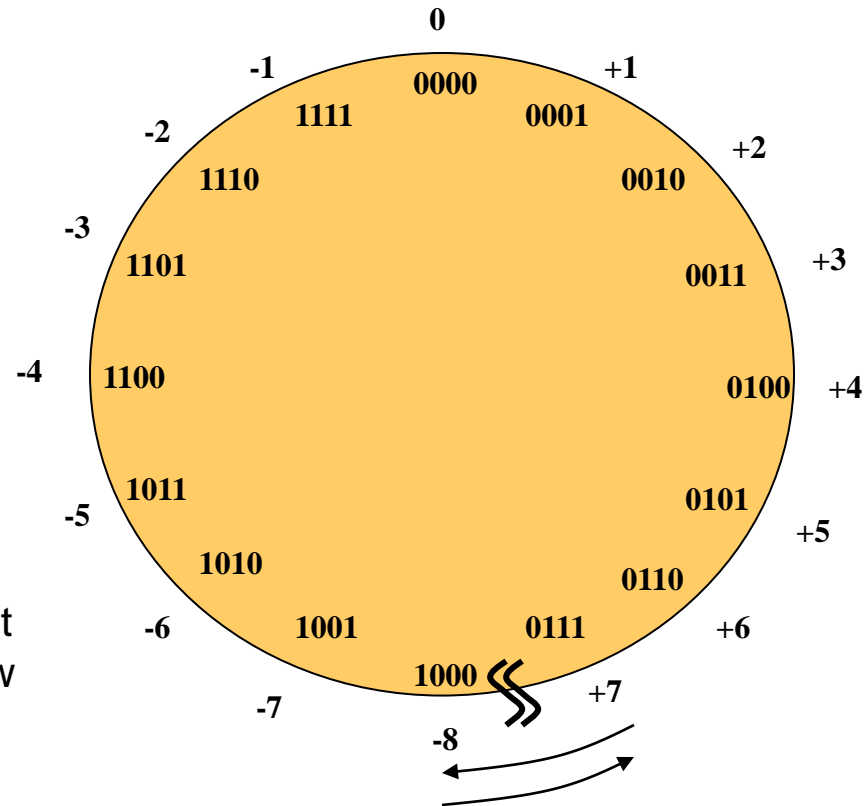
With 4-bit *unsigned* numbers we can only represent 0 – 15. Thus, we say overflow has occurred.

# 2's Complement Overflow

$$5 + 7 = +12$$

$$-6 + -4 = -10$$

With 4-bit 2's complement numbers we can only represent -8 to +7. Thus, we say overflow has occurred.



Overflow occurs when you cross this discontinuity

# Testing for Overflow

- Most fundamental test
  - Check if answer is wrong (i.e. Positive + Positive yields a negative)
- Unsigned overflow (**C**) test
  - If \_\_\_\_\_
- Signed (2's complement) overflow (**V**) test
  - Only occurs if
    - \_\_\_\_\_ ... or...
    - \_\_\_\_\_
  - Alternate test: See following slides

# Alternate Signed Overflow Test

A & B	A3	B3	S3	C3	C4	V
Both Positive	0	0	0			
			1			
One Positive & One Negative	0	1	0			
			1			
	1	0	0			
			1			
Both Negative	1	1	0			
			1			

- Check if \_\_\_\_\_

# Overflow in Addition

- Overflow occurs when the result of the addition cannot be represented with the given number of bits.
- Tests for overflow:
  - Unsigned: if  $C_{out} = 1$
  - Signed: if  $p + p = n$  or  $n + n = p$

	<u>1101</u>
+	<u>0100</u>

If unsigned    If signed

	0110
+	<u>0101</u>

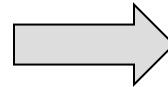
If unsigned    If signed

# Overflow in Subtraction

- Overflow occurs when the result of the subtraction cannot be represented with the given number of bits.
- Tests for overflow:
  - Unsigned: if  $C_{out} = 0$
  - Signed: if addition is  $p + p = n$  or  $n + n = p$

If unsigned   If signed

$$\begin{array}{r} 0111 \\ - 1000 \\ \hline \end{array}$$

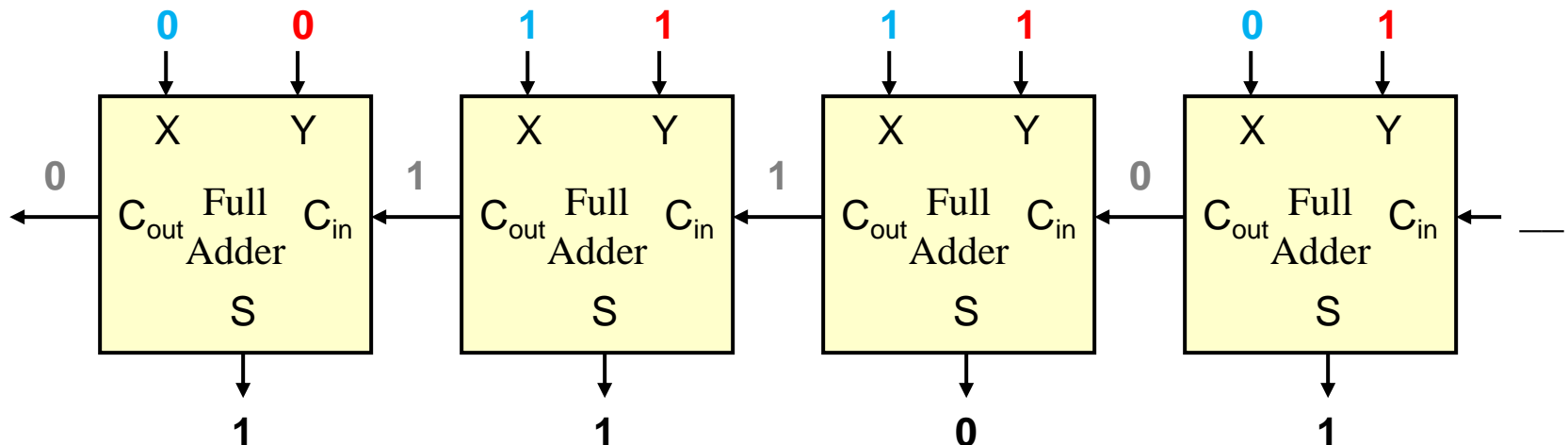


If unsigned   If signed

# Addition – Full Adders

- Perform the following addition using 1 Full Adder for each column of addition

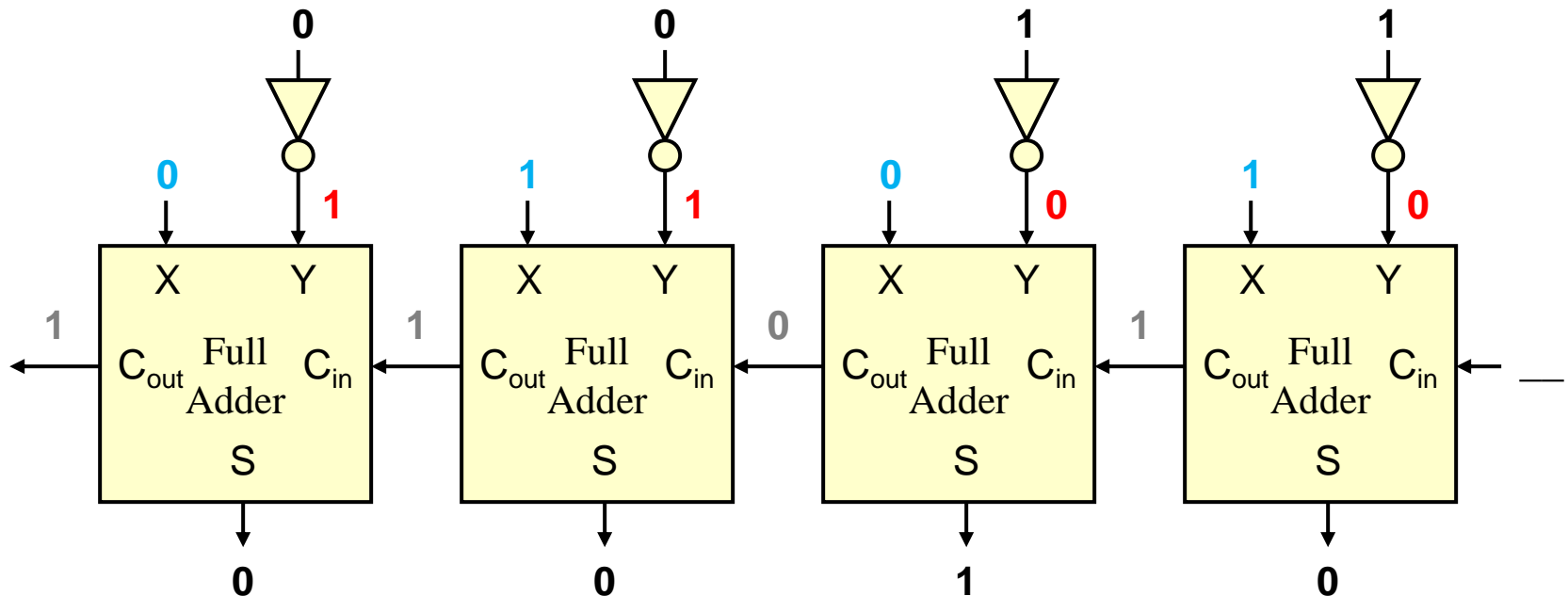
$$\begin{array}{r}
 01100 \\
 0110 = X \\
 + 0111 = Y \\
 \hline
 1101
 \end{array}$$



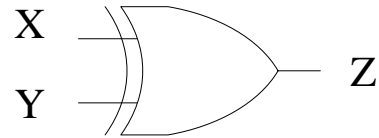
# Performing Subtraction w/ Adders

- To subtract
  - Flip bits of Y
  - Add 1

$$\begin{array}{r}
 0101 = X \\
 - 0011 = Y \\
 \hline
 1101
 \end{array}
 \Rightarrow
 \begin{array}{r}
 0101 \\
 + 1100 \\
 \phantom{+} 1 \\
 \hline
 0010
 \end{array}$$



# XOR Gate Review



XOR

$$Z = X \oplus Y$$

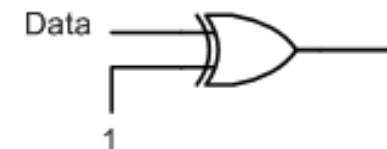
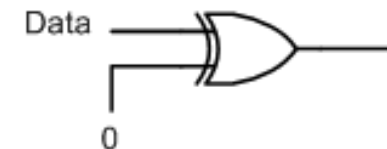
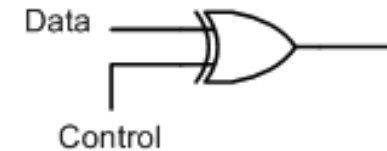
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

True if an **odd** # of inputs are true  
2 input case: True if inputs are different

# XOR Conditional Inverter

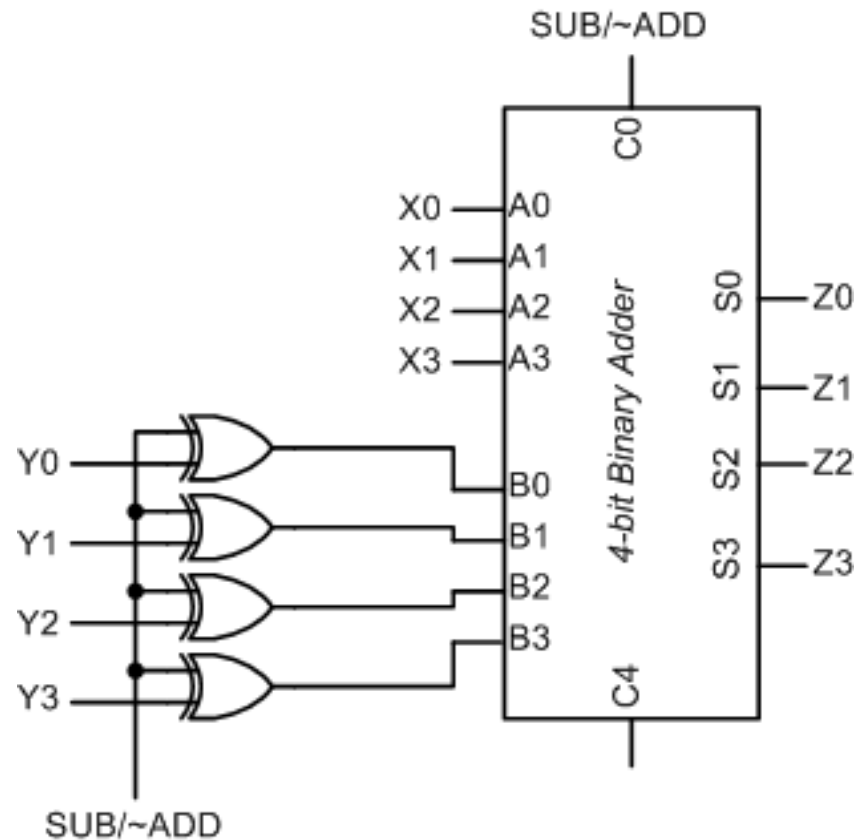
- If one input (say, X) to an XOR gate is 0, then  $Z = \underline{\hspace{2cm}}$
- If one input (say, X) to an XOR gate is 1, then  $Z = \underline{\hspace{2cm}}$
- Use one input as a control input which can conditionally pass or invert the other input

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0



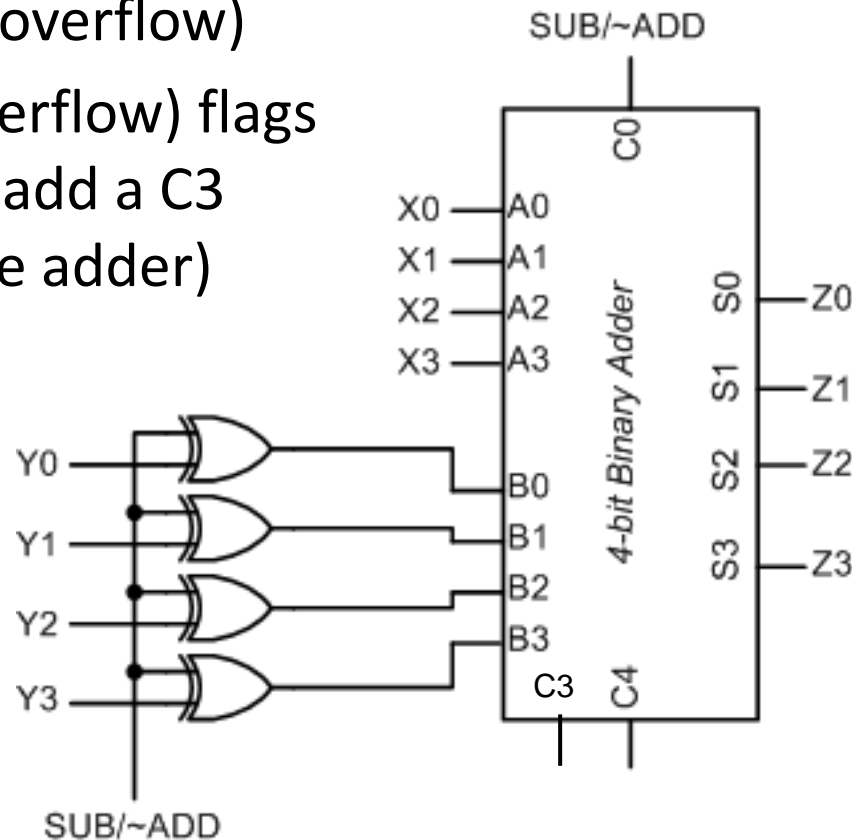
# Adder/Subtractor

- Using XOR gates before one set of adder inputs we can
  - Selectively pass or invert Y
  - Add an extra '1' via the Carry-in
- If  $\text{SUB}/\sim\text{ADD}=0$ ,
  - $Z = X+Y$
- If  $\text{SUB}/\sim\text{ADD}=1$ ,
  - $Z = X-Y$



# Adder/Subtractor

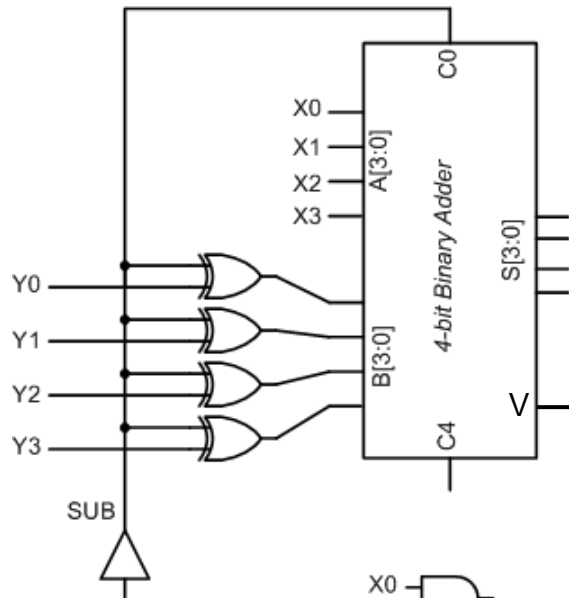
- Exercise: Add appropriate logic to produce
  - **C** (unsigned overflow)
  - **V** (signed overflow) flags (assume we add a C3 output to the adder)



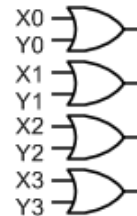
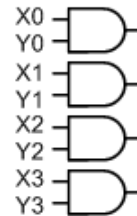
# ALU Design

Complete the ALU design given the function table below

OP[2:0]	Z
000	X+Y
001	X-Y
011	SLT: Z=1, if X<Y Z=0, other
100	AND
110	OR
Others	Z = und.



OP[2:0]  
|||



**NON-REQUIRED MATERIAL**

# Hexadecimal Representation

- Since values in modern computers are many bits, we use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
  - 11010010 = D2 hex
  - 0111011011001011 = 76CB hex
- To interpret the value of a hex number, you must know what underlying binary system is assumed (unsigned, 2's comp. etc.)

# Translating Hexadecimal

- Hex place values ( $16^2$ ,  $16^1$ ,  $16^0$ ) can ONLY be used if the number is positive.
- If hex represents unsigned binary simply apply hex place values
  - B2 hex =  $11 * 16^1 + 2 * 16^0 = 178_{10}$
- If hex represents signed value (2's comp.)
  - First determine the sign to be pos. or neg.
    - Convert the MS-hex digit to binary to determine the MSB (e.g. for B2 hex, B=1011 so since the MSB=1, B2 is neg.)
    - In general, hex values starting 0-7 = pos. / 8-F = neg.
  - If pos., apply hex place values (as if it were unsigned)
  - If neg., take the **16's complement** and apply hex place values to find the neg. number's magnitude

# Taking the 16's Complement

- Taking the 2's complement of a binary number yields its negative and is accomplished by finding the 1's complement (bit flip) and adding 1
- Taking the 16's complement of a hex number yields its negative and is accomplished by finding the 15's complement and adding 1
  - 15's complement is found by subtracting each digit of the hex number from  $F_{16}$

Original value B2:	FF	
	<u>- B2</u>	Subtract each digit from F
	4D	15's comp. of B2
	<u>+ 1</u>	Add 1
16's comp. of B2:	4E	16's comp. of B2

# Translating Hexadecimal

- Given 6C hex
  - If it is unsigned, apply hex place values
    - $6C \text{ hex} = 6 * 16^1 + 12 * 16^0 = 108_{10}$
  - If it is signed...
    - Determine the sign by looking at MSD
      - 0-7 hex has a 0 in the MSB [i.e. positive]
      - 8-F hex has a 1 in the MSB [i.e. negative]
      - Thus, 6C (start with 6 which has a 0 in the MSB is positive)
    - Since it is positive, apply hex place values
      - $6C \text{ hex} = 6 * 16^1 + 12 * 16^0 = 108_{10}$

# Translating Hexadecimal

- Given FE hex
  - If it is unsigned, apply hex place values
    - FE hex =  $15 * 16^1 + 14 * 16^0 = 254_{10}$
  - If it is signed...
    - Determine sign => Negative
    - Since it is negative, take 16's complement and then apply place values
      - 16's complement of FE =  $01 + 1 = 02$  and apply place values = 2
      - Add in sign => -2 = FE hex

# Finding the Value of Hex Numbers

- B2 hex representing a signed (2's comp.) value
  - Step 1: Determine the sign: Neg.
  - Step 2: Take the 16's comp. to find magnitude
$$FF - B2 + 1 = 4E \text{ hex}$$
  - Step 3: Apply hex place values ( $4E_{16} = +78_{10}$ )
  - Step 4: Final value: B2 hex =  $-78_{10}$
- 7C hex representing a signed (2's comp.) value
  - Step 1: Determine the sign: Pos.
  - Step 2: Apply hex place values ( $7C_{16} = +124_{10}$ )
- 82 hex representing an unsigned value
  - Step 1: Apply hex place values ( $82_{16} = +130_{10}$ )



# Hex Subtraction and Overflow

- Same rules as in binary
  - Convert  $A - B$  to  $A + \text{Comp. of } B$
  - Drop any final carry out
- Same subtraction overflow rules
  - Unsigned: Check if final Cout = 0
  - Signed: Check signs of addition inputs and result

$\begin{array}{r} \text{B1ED} \\ - 76\text{FE} \\ \hline \end{array}$	$\rightarrow$	$\begin{array}{r} \overset{1}{\text{B1ED}} \\ + \quad \quad 1 \\ \hline \text{3AEF} \end{array}$	<p><u>If unsigned</u>                  No Overflow                  Cout = 1</p> <p><u>If signed</u>                  Overflow  <math>n + n = p</math></p>
---	---------------	--	--

$\begin{array}{r} 0001 \\ - 0002 \\ \hline \end{array}$	$\rightarrow$	$\begin{array}{r} \overset{0}{0001} \\ + \quad \quad 1 \\ \hline \text{FFFD} \end{array}$	<p><u>If unsigned</u>                  Overflow                  Cout = 0</p> <p><u>If signed</u>                  No Overflow  <math>p + n</math></p>
---	---------------	---	--

# Credits

- These slides were derived from Gandhi Puvvada's EE 457 Class Notes