EE 457 Unit 2

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

1	0	0	1	0	0	1	1	= +147
128	64	32	16	8	4	2	1	

- Signed variables use the 2's complement system (Neg. MSB weight) to represent numbers

1	0	0	1	0	0	1	1	= -109
-128	64	32	16	8	4	2	1	

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 => +7 or +2 => -2), requires ***taking the complement***
  - 2's complement of a # is found by flipping bits and adding 1

1001	x = -7
0110	Bit flip (1's comp.)
+ 1	Add 1
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):

pos. 011010 = 00011010

Sign bit is just repeated as many times as necessary

neg. 110011 = 11110011

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):

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

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?
  - Not if A=5, B=(-2)…5 + (-2) = 5 – 2 = 3
  - What if A=(2), B=(-5)?
  - Can't perform 2-5
  - Flip operands and keep sign of larger
    - 5 – 2 = 3 => Apply sign of larger mag. operand => -3
- Human add/sub algorithm depends on sign!!

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:

		If unsigned	If signed
	1 1		
	1001	(9)	(-7)
+	0011	(3)	(3)
	1100	(12)	(-4)

Unsigned and Signed Subtraction

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

		If unsigned	If signed
	1100	(12)	(-4)
–	0010	(2)	(2)

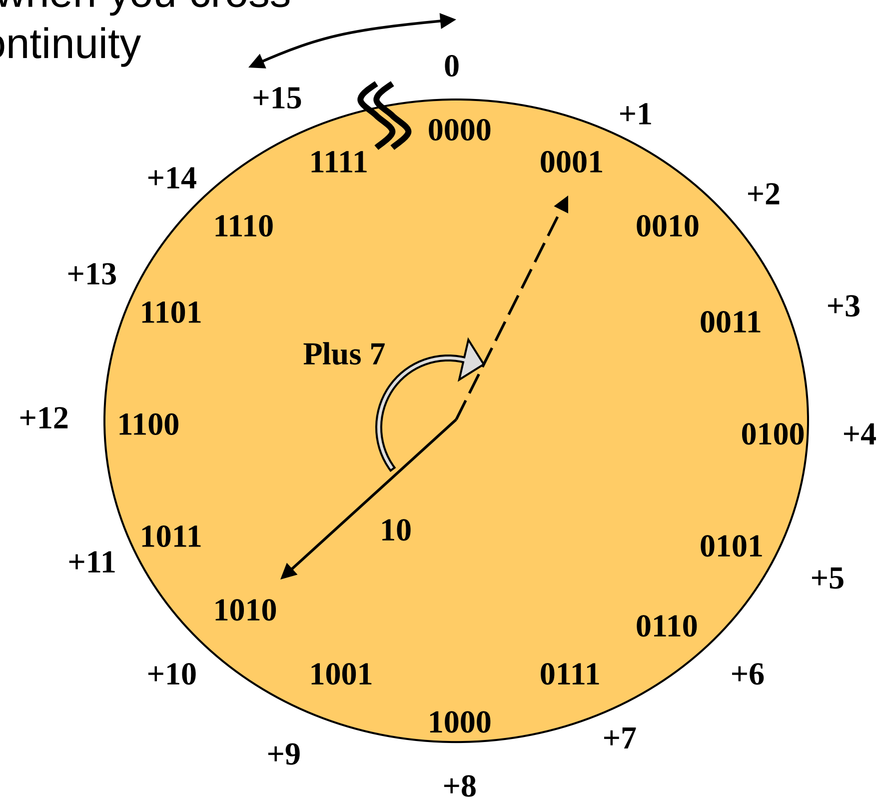
	11_1_		
	1100	A	
	1101	1's comp. of B	
+	1	Add 1	
	1010	(10)	(-6)
		If unsigned	If signed

Overflow

- Overflow occurs when the result of an arithmetic operation is too large to be represented with the given number of bits
  - Unsigned overflow (**C**) occurs when adding or subtracting unsigned numbers
  - Signed (2's complement overflow) overflow (**V**) occurs when adding or subtracting 2's complement numbers

Unsigned Overflow

Overflow occurs when you cross
this discontinuity



10 + 7 = 17

4 - 6 = 14

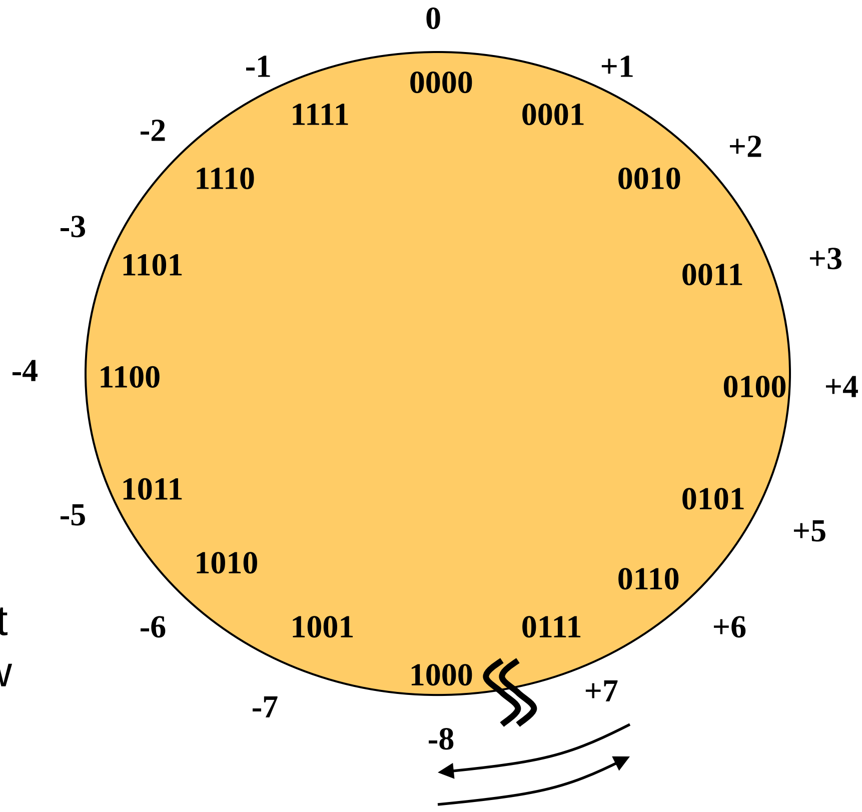
With 4-bit *unsigned* numbers we can only represent 0 – 15. Thus, we say overflow has occurred.

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 carry-out of final position equals ‘1’
- Signed (2’s complement) overflow (**V**) test
  - Only occurs if two positives are added and result is negative or two negatives are added and result is positive
  - Alternate test: See following slides

Alternate Signed Overflow Test

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

- Check if Cin & Cout of MSB column are different

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 Cout = 1
  - Signed: if p + p = n   or   n + n = p

	If unsigned	If signed
1 1		
1101	(13)	(-3)
+ 0100	(4)	(4)
0001	(17)	(+1)
	Overflow Cout = 1	No Overflow n + p

	If unsigned	If signed
0 1		
0110	(6)	(6)
+ 0101	(5)	(5)
1011	(11)	(-5)
	No Overflow Cout = 0	Overflow p + p = n

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 Cout = 0
  - Signed: if addition is p + p = n or n + n = p

	If unsigned	If signed
0111	(7)	(7)
– 1000	(8)	(-8)
	(-1)	(15)

Desired Results

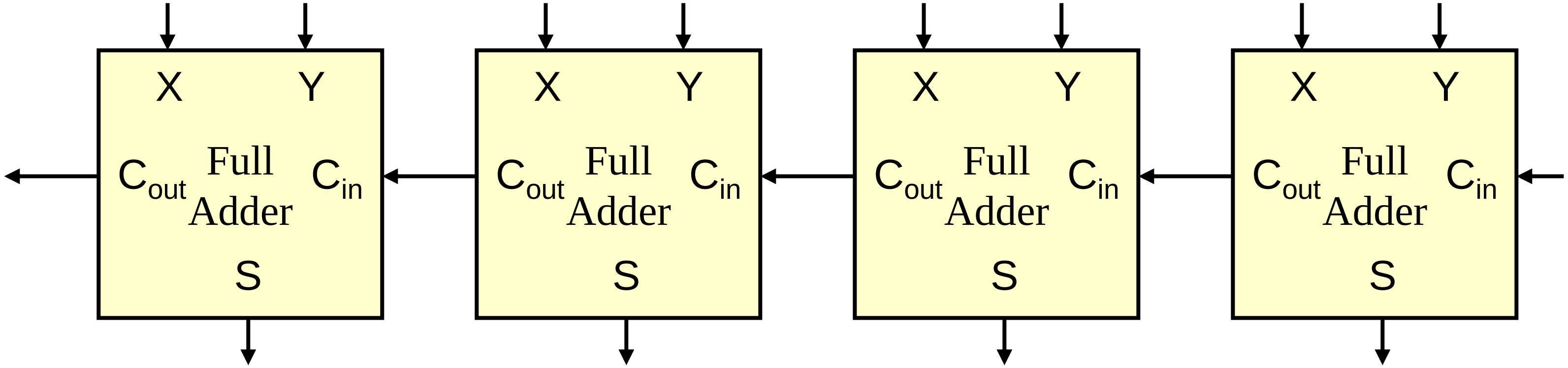
0111_			
0111	A		
0111	1's comp. of B		
+ 1	Add 1		
1111	(15)		(-1)

If unsigned	If signed
Overflow	Overflow
Cout = 0	p + p = n

Addition – Full Adders

- Use 1 Full Adder for each column of addition

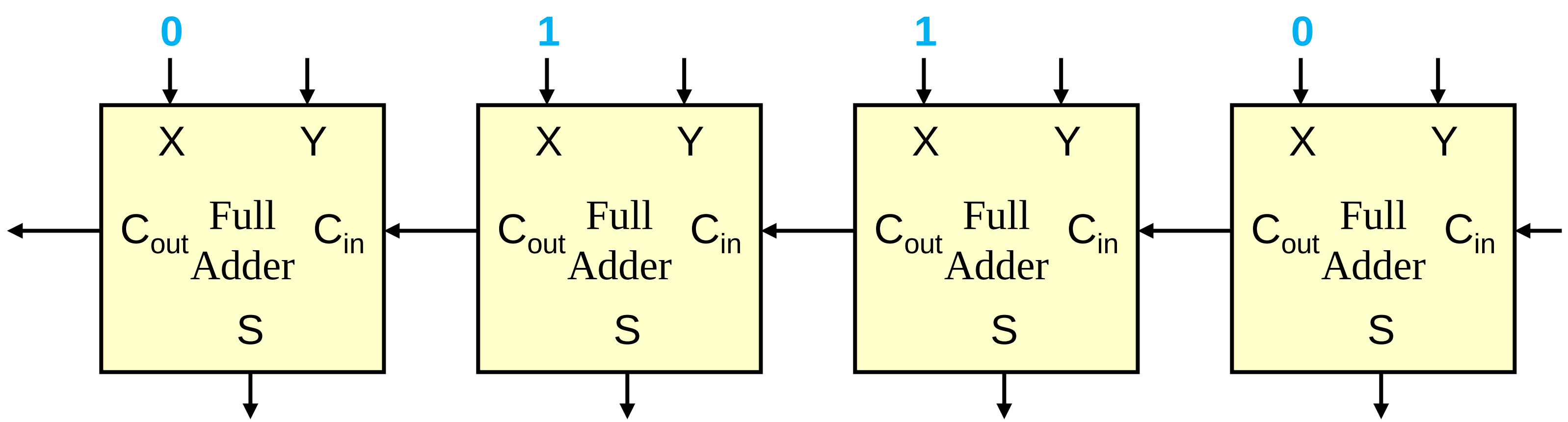
```
  0110
+ 0111
------
```



Addition – Full Adders

- Connect bits of top number to X inputs

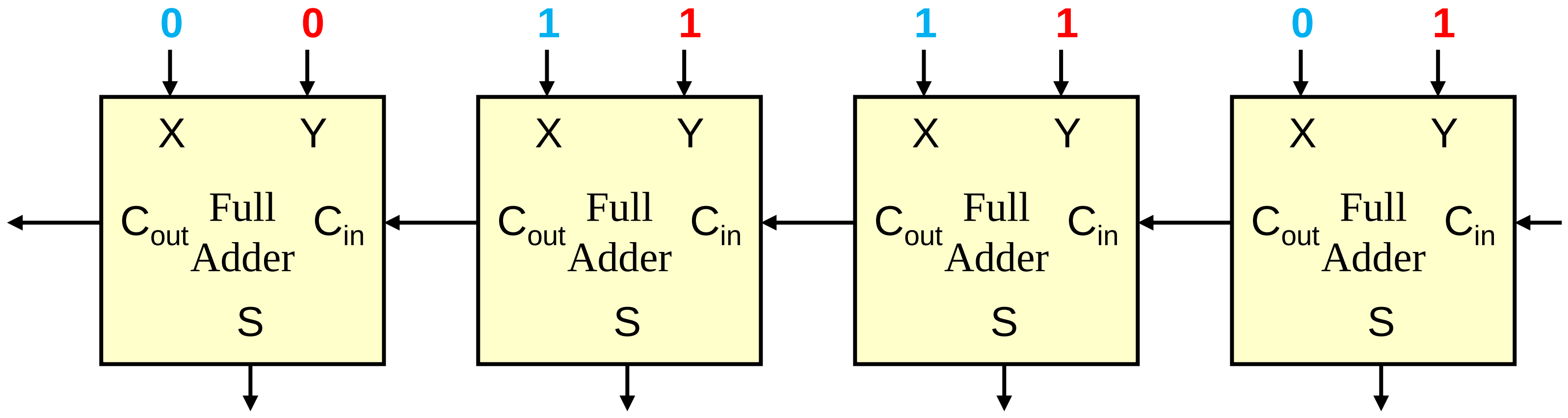
```
  0110
+ 0111
```



Addition – Full Adders

- Connect bits of bottom number to Y inputs

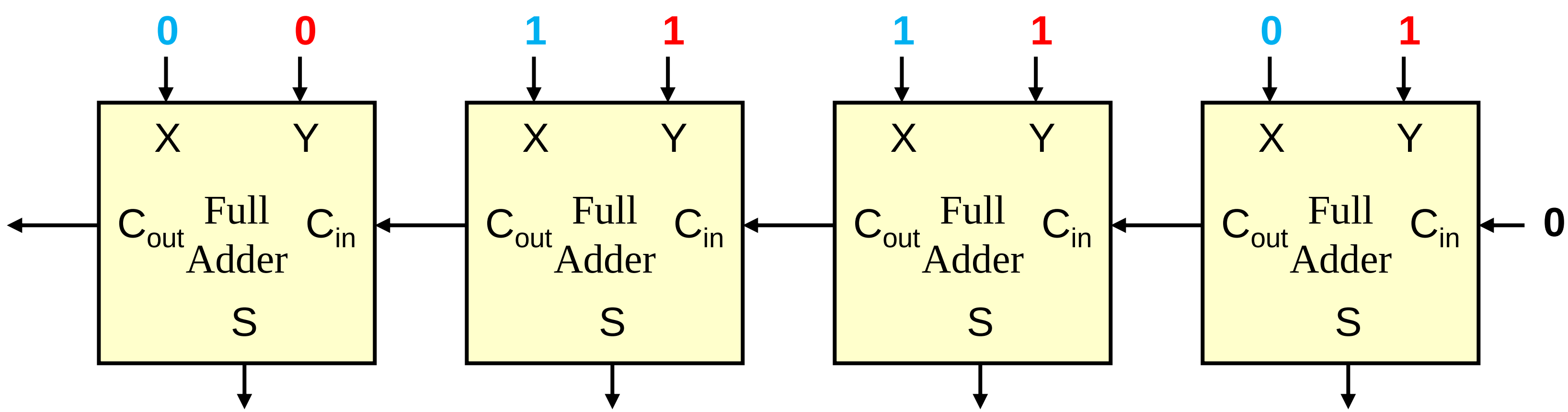
```
    0110 = X
  + 0111 = Y
  ------
```



Addition – Full Adders

- Be sure to connect first C_{in} to 0

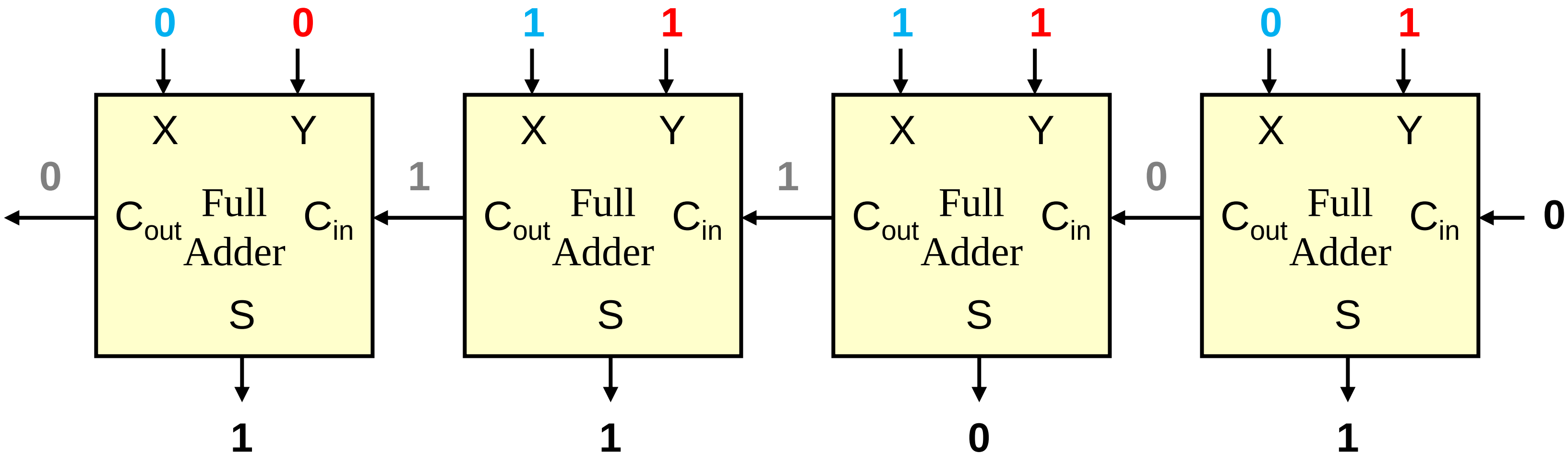
```
  0110 = X
+ 0111 = Y
```



Addition – Full Adders

- Use 1 Full Adder for each column of addition

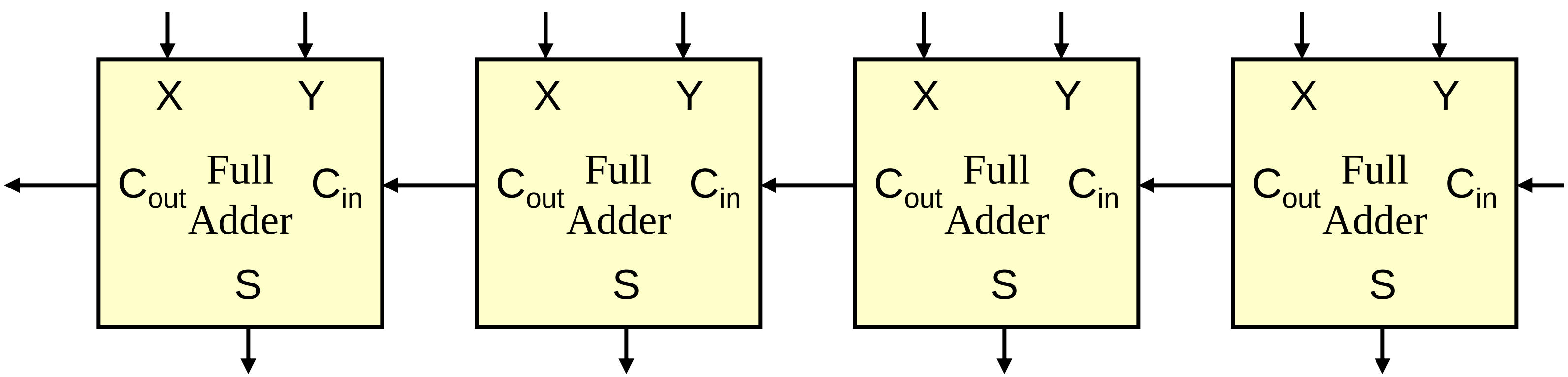
```
  01100
   0110 = X
+  0111 = Y
  -----
   1101
```



Performing Subtraction w/ Adders

- To subtract
  - Flip bits of Y
  - Add 1

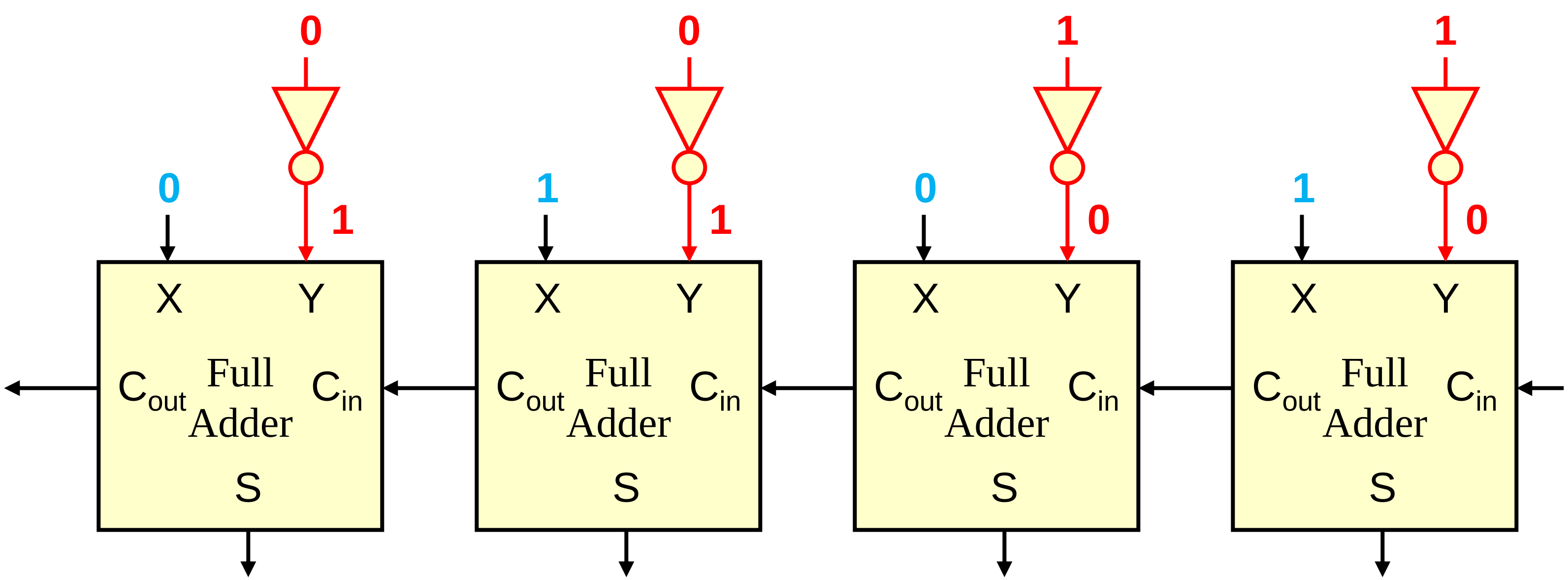
```
  0101 = X            0101
- 0011 = Y   ⇨     + 1100
------                   1
  1101             ------
                      0010
```



Performing Subtraction w/ Adders

- To subtract
  - Flip bits of Y
  - Add 1

```
  0101 = X          0101
- 0011 = Y   ⇒   + 1100
  ----               1
  1101            ----
                  0010
```

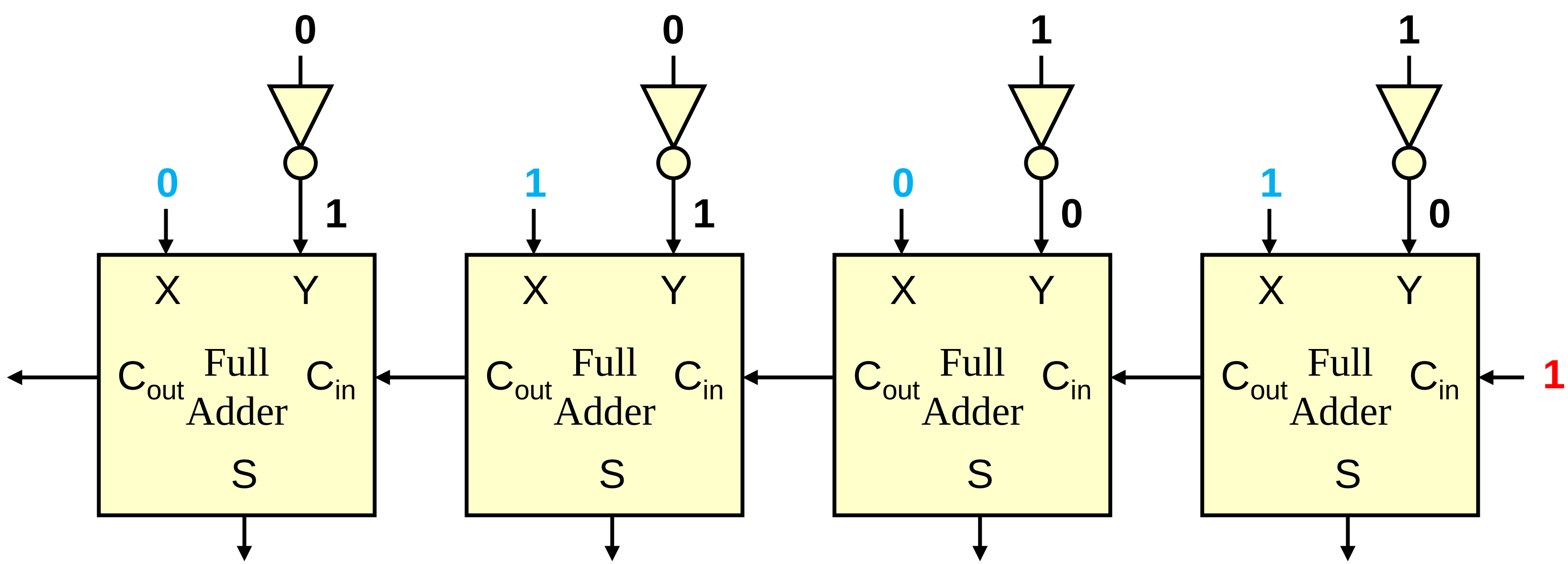


Performing Subtraction w/ Adders

- To subtract
  - Flip bits of Y
  - Add 1

```
  0101 = X
- 0011 = Y
  ----
  1101
```

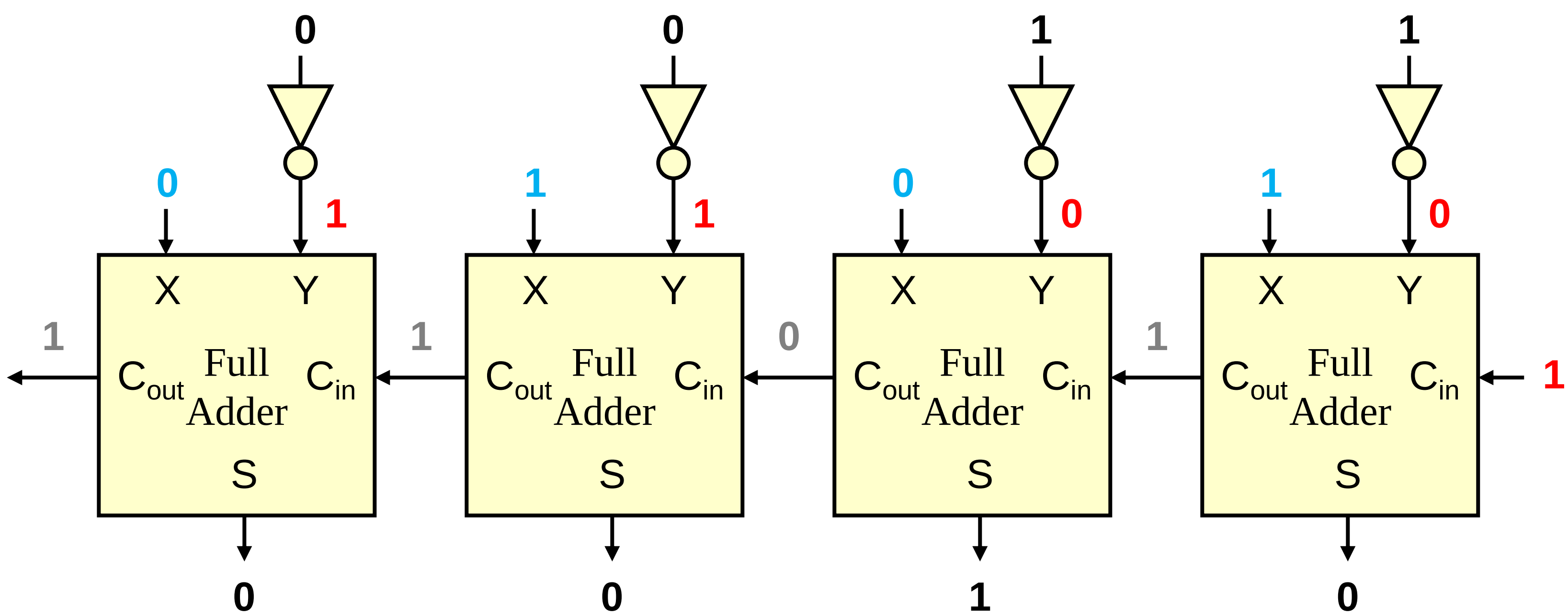
```
  0101
+ 1100
     1
  ----
  0010
```



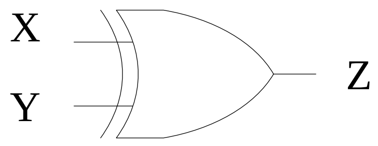
Performing Subtraction w/ Adders

- To subtract
  - Flip bits of Y
  - Add 1

```
  0101 = X          0101
- 0011 = Y   ⇨   + 1100
  ----                 1
  1101              ----
                    0010
```



XOR Gate Review



XOR

$$Z = X \oplus Y$$

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

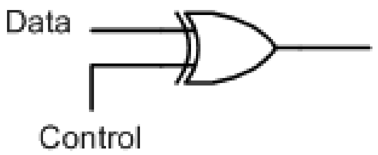
True if an odd # of inputs are true

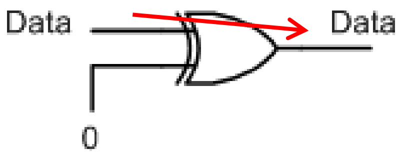
2 input case: True if inputs are different

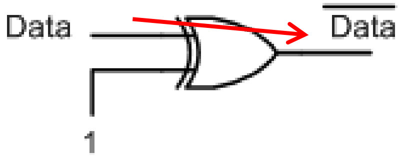
XOR Conditional Inverter

- If one input to an XOR gate is 0, the other input is passed
- If one input to an XOR gate is 1, the other input is inverted
- Use one input as a control input which can conditionally pass or invert the other input

X	Y	Z	
0	0	0	Y
0	1	1	
1	0	1	$\overline{Y}$
1	1	0	

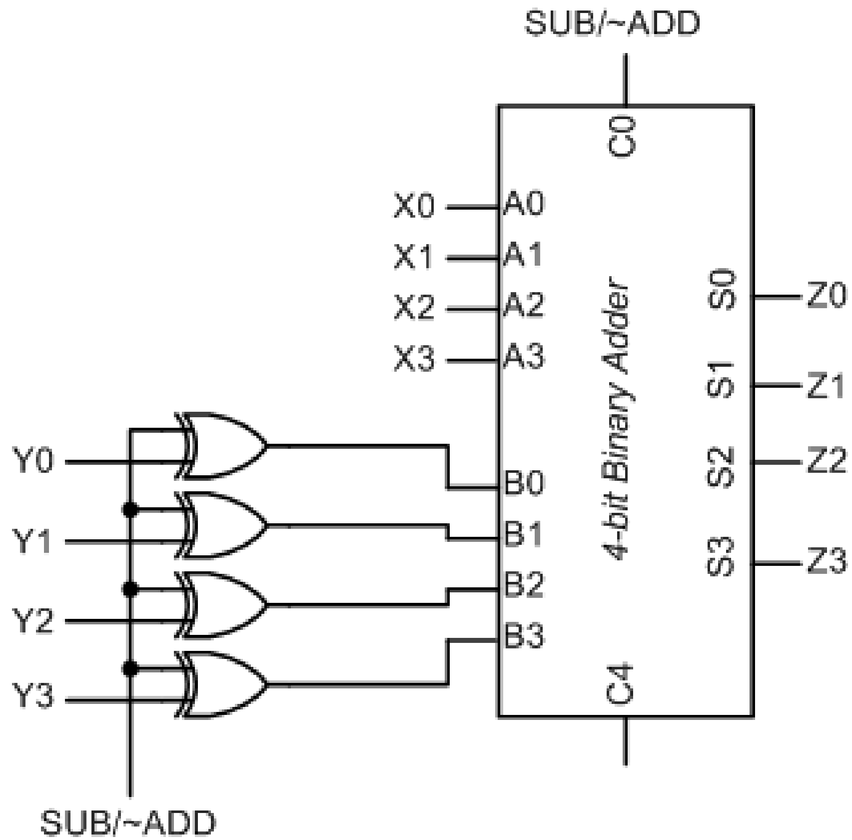






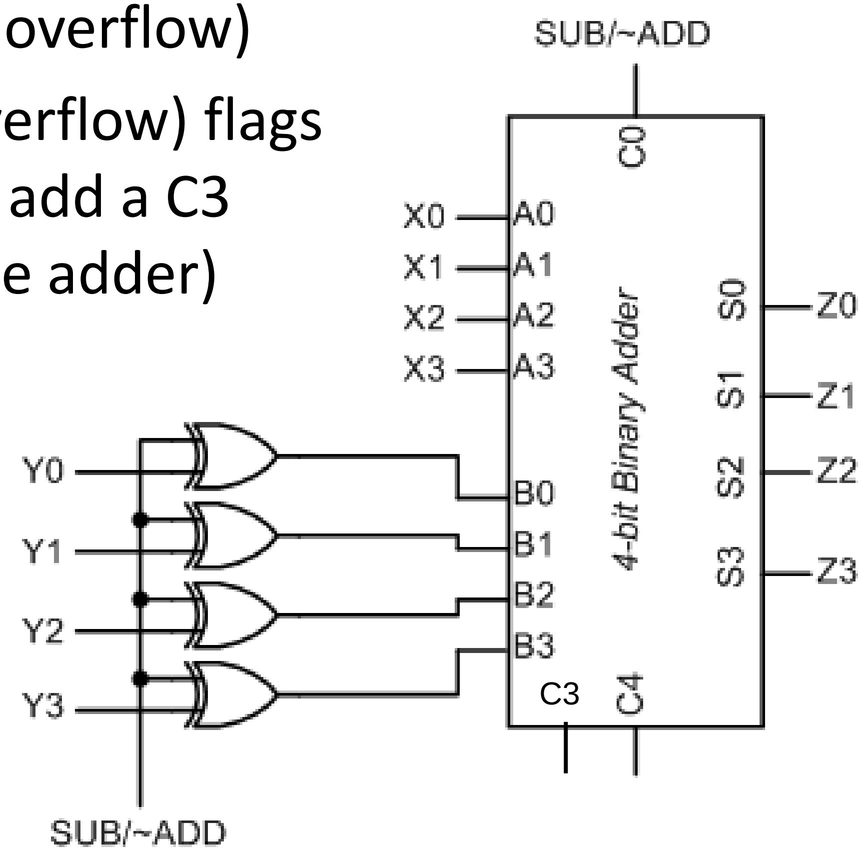
Adder/Subtractor

- 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 SUB/~ADD=0,
  - Z = X+Y
- If SUB/~ADD=1,
  - Z = X-Y



Adder/Subtractor

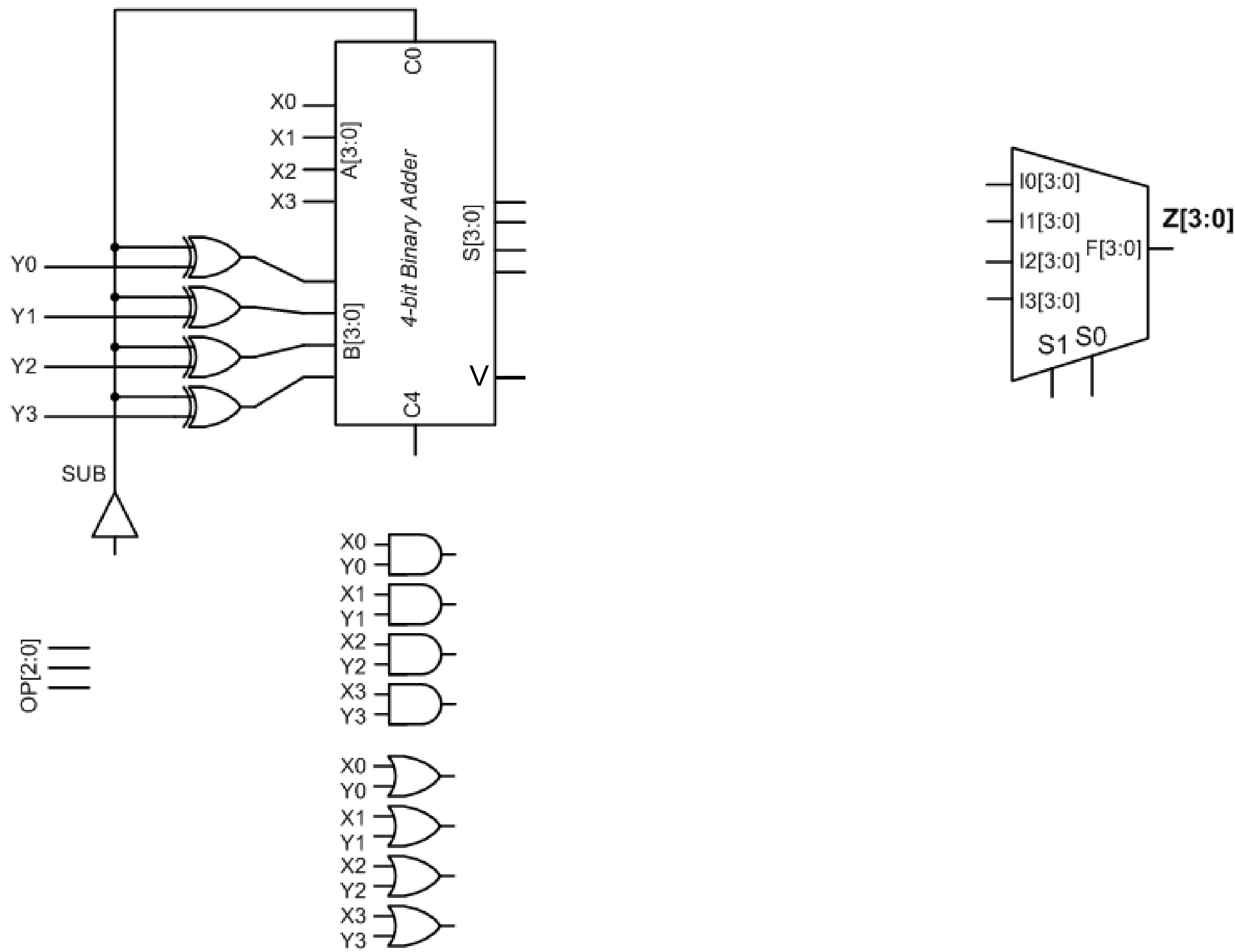
- Exercise: Add appropriate logic to produce
  - **C** (unsigned overflow)
  - **V** (signed overflow) flags (assume we add a C3 output to the adder)



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.



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	
	- B2	Subtract each digit from F
	4D	15's comp. of B2
	+ 1	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 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 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 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 Addition and Overflow

- Same rules as in binary
  - Add left to right
  - Drop any carry (carry occurs when sum > F_{16})
- Same addition overflow rules
  - Unsigned: Check if final Cout = 1
  - Signed: Check signs of inputs and result

```
 1  1
  7AC5
+ C18A
------
  3C4F
```

If unsigned

Overflow
Cout = 1

If signed

No Overflow
p + n

```
0 1  1
  6C12
+ 549F
------
  C0B1
```

If unsigned

No Overflow
Cout = 0

If signed

Overflow
p + p = n

Hex Subtraction and Overflow

- Same rules as in binary
  - Convert A – B to A + 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

```
  B1ED          1
- 76FE         B1ED
               8901
             +    1
               3AEF
```

If unsigned
No Overflow
Cout = 1

If signed
Overflow
n + n = p

```
  0001          0
- 0002         0001
               FFFD
             +    1
               FFFF
```

If unsigned
Overflow
Cout = 0

If signed
No Overflow
p + n

Credits

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