



UM EECS 270 F21

Introduction to Logic Design

7. Binary Arithmetic

Binary Arithmetic



- Representation of positive numbers (old news)
- Binary addition
- Representation of negative numbers
- Binary subtraction
- (Basic) Binary multiplication

Binary Addition



- How do we add two binary numbers?

Just like elementary school!

Decimal:

$$\begin{array}{r} 1 \quad 1 \leftarrow \text{This is the } \textit{carry out} \text{ of the first} \\ 4 \ 2 \ 5 \ 9 \quad \text{column, which becomes the} \\ + 1 \ 8 \ 3 \ 7 \quad \text{carry in to the second column} \\ \hline 6 \ 0 \ 9 \ 6 \end{array}$$

Binary:

$$\begin{array}{r} 1 \quad 1 \\ 1 \ 0 \ 0 \ 1 \\ + 0 \ 0 \ 1 \ 1 \\ \hline 1 \ 1 \ 0 \ 0 \end{array}$$

- If we have a fixed number of bits (which is usually the case), a carry out of the most significant column indicates that there's not enough bits to hold the sum value. This case is referred to as **overflow**.

Addition Implementation

- Addition of two 1-bit binary numbers, A and B – requires two output bits, which we'll call S (sum) and C (carry).

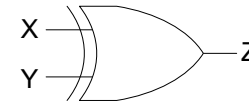
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = AB$$

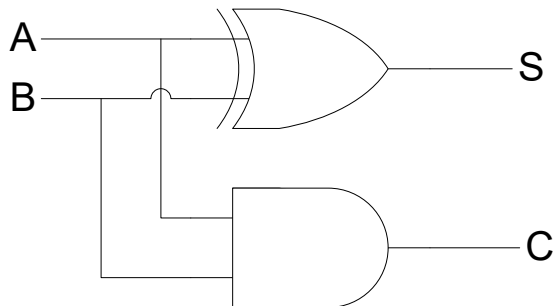
$$S = A\bar{B} + \bar{A}B = A \oplus B$$

XOR Operation

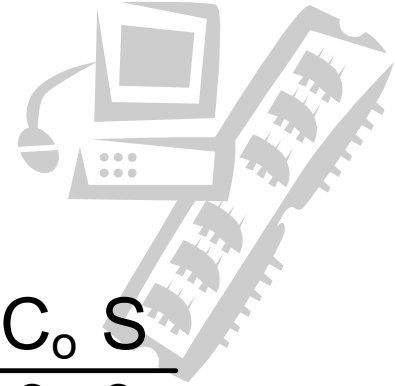
$$X \oplus Y \equiv X\bar{Y} + \bar{X}Y$$



Implementation:



This circuit is called a
Half Adder (HA)



- What if we want to add larger numbers?
 - What's missing from HA?

$$\begin{array}{r}
 \begin{array}{|c|c|} \hline C_o & C_i \\ \hline \end{array} \\
 \begin{array}{r}
 1\ 0\ 1 \\
 +\ 0\ 0\ 1 \\
 \hline
 1\ 1\ 0
 \end{array}
 \end{array}$$

Need to add carry-in (C_i) input!

A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = \sum_{A,B,C_i} (1,2,4,7)$$

$$C_o = \sum_{A,B,C_i} (3,5,6,7)$$

- Sum output:

$$S = \sum_{A,B,C_i} (1, 2, 4, 7)$$

$$S = A'B'C_i + A'BC'_i + AB'C'_i + ABC_i$$

$$S = C_i (A'B' + AB) + C'_i (A'B + AB')$$

$$S = C_i (A \oplus B)' + C'_i (A \oplus B)$$

$$S = A \oplus B \oplus C_i$$





- Carry-out output:

$$C_o = \sum_{A,B,C_i} (3,5,6,7)$$

$$C_o = A'BC_i + AB'C_i + ABC'_i + ABC_i$$

$$C_o = A'BC_i + AB'C_i + ABC'_i + \underbrace{ABC_i}_{BC_i} + \underbrace{ABC_i}_{AC_i} + \underbrace{ABC_i}_{AB}$$

Diagram illustrating the simplification of the carry-out output equation. Red arrows show the grouping of the four minterms into three pairs: ABC_i and ABC_i (labeled BC_i), ABC_i and ABC_i (labeled AC_i), and ABC_i and ABC_i (labeled AB).

$$C_o = AB + AC_i + BC_i$$

This requires 3 2-input AND gates
and 1 3-input OR gate.

Can we do any better?

- Carry-out output: A bit of cleverness ...



First, we observe that:

$$A + B = (A \oplus B) + AB$$

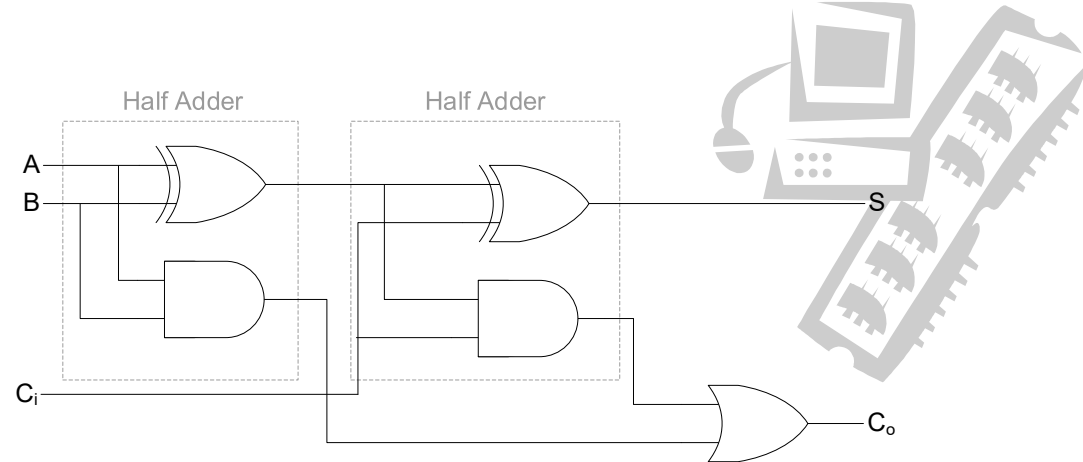
$$\begin{aligned} C_o &= AB + AC_i + BC_i \\ &= AB + C_i(A + B) \\ &= AB + C_i((A \oplus B) + AB) \\ &= AB + (A \oplus B)C_i + ABC_i \\ &= AB + (A \oplus B)C_i \end{aligned}$$

This requires 2 2-input AND gates, a 2-input OR gate, and a 2-input XOR, *but we've already implemented $A \oplus B$ for the Sum output!* So we only require 3 additional gates..

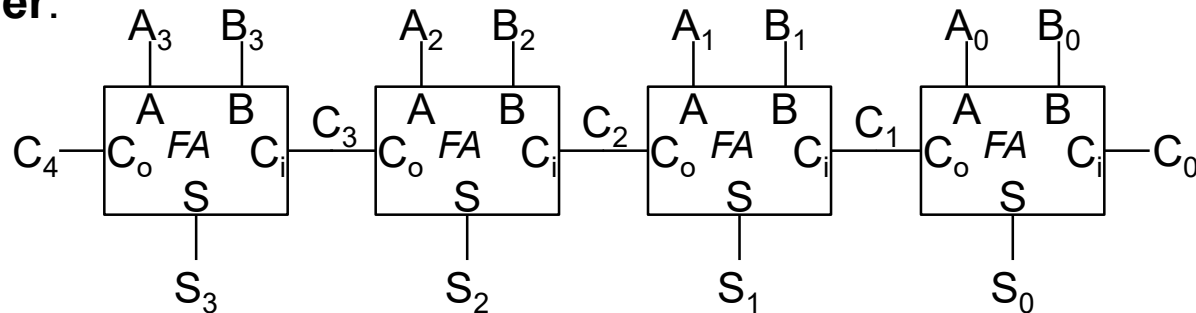
- Final Circuit:

$$S = A \oplus B \oplus C_i$$

$$C_o = AB + (A \oplus B)C_i$$



- This circuit is called a **Full Adder (FA)**
- After all that design work, we really just have 2 HAs with an OR gate
- To make an n-bit adder, simply cascade Full Adders to make a **Ripple Carry Adder**:

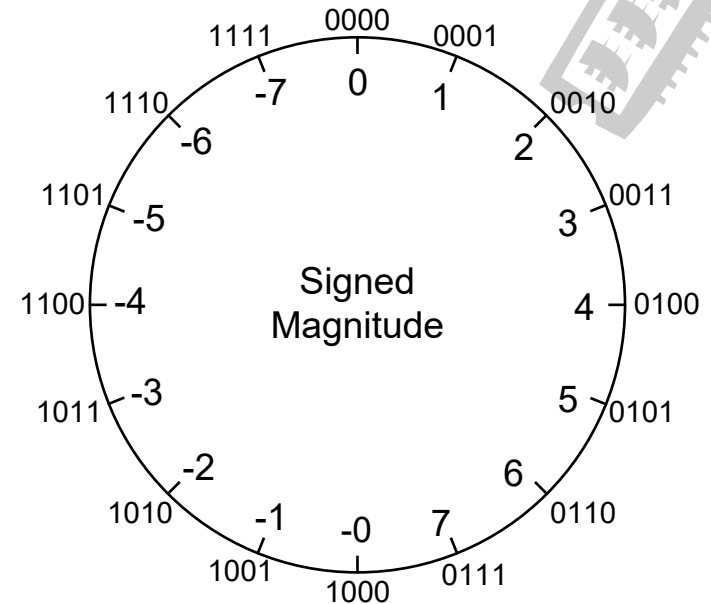


- What about subtraction hardware?
 - Could design subtractor hardware using process similar to adder design
 - Simpler way: re-use our addition hardware! $A - B = A + (-B)$

*(-B)??? How do we represent **negative numbers**?*

Binary Representation of Negative Numbers

- **Signed Magnitude:** Numbers consist of a magnitude and a symbol indicating whether the number is positive or negative
 - We're used to this:
(-10: "-": sign, "10": magnitude)
- In binary, reserve MSB to represent the sign: 0 indicates a positive number, 1 indicates a negative number
- Sign bit position has no weight
- What is the range of numbers that can be represented with 4-bit binary signed-magnitude representation
[-7 : 7]
- What is the range of numbers that can be represented with n -bit binary signed-magnitude representation?
[-($2^{n-1}-1$) : $2^{n-1}-1$]



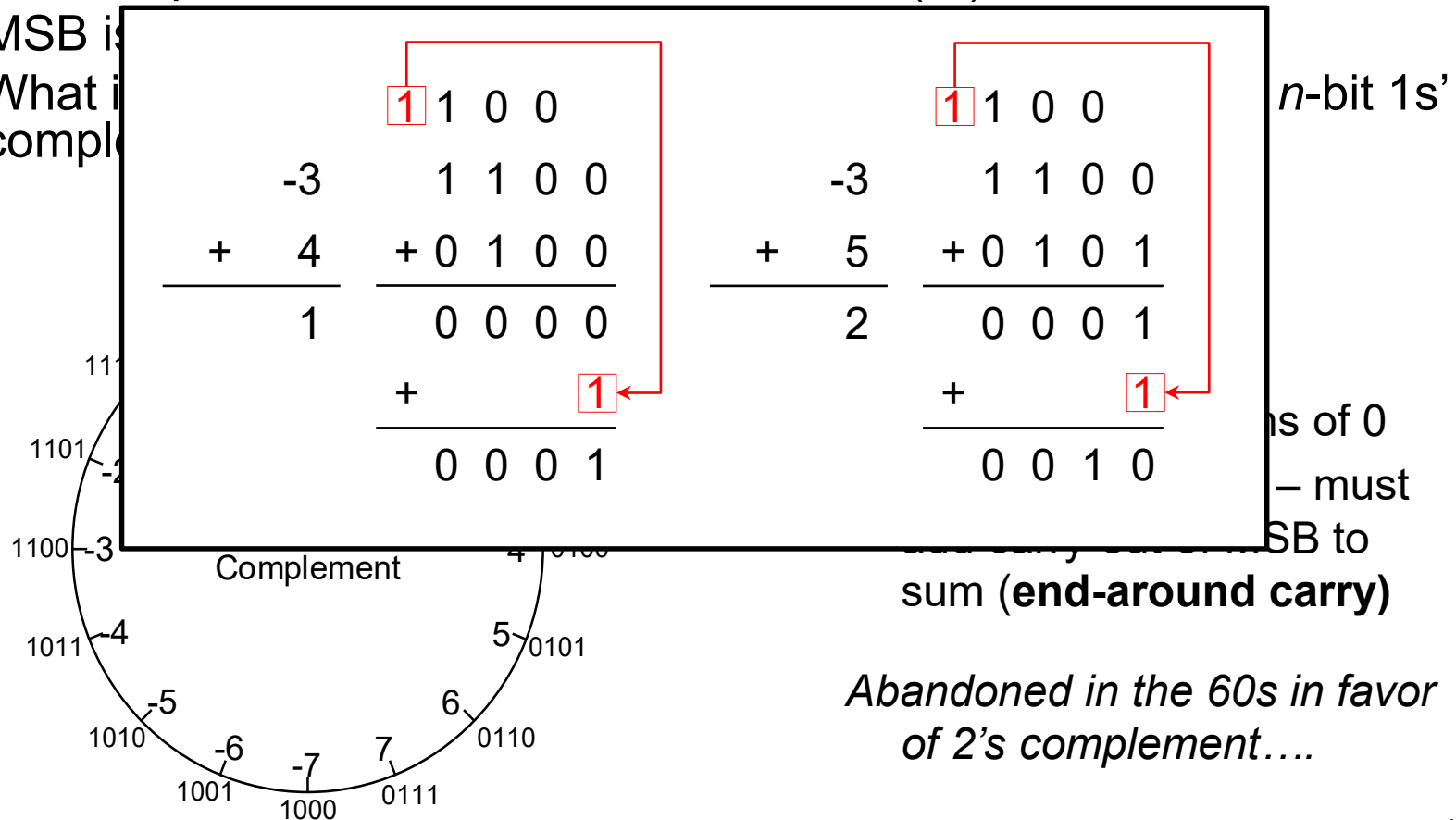
- Drawbacks
 - Two representations of 0
 - Signed magnitude arithmetic requires comparing sign bits and then performing addition or subtraction – leads to complex hardware

Complement Representations

- **1s' Complement Representation:** MSB has weight of $-(2^{n-1}-1)$
- To get the representation of a negative number, write the positive representation and complement all bits

Example: -3: 3 = 0011, -3 = 1100 = $1*(-7) + 1*4 + 0*2 + 0*1$

- MSB is
- What is the complement





- **2's Complement Representation:** MSB has weight of -2^{n-1}
- To get the negative representation of a number, write the positive representation, complement all bits, and add 1
(*ignoring any carry-out of the most significant column*)

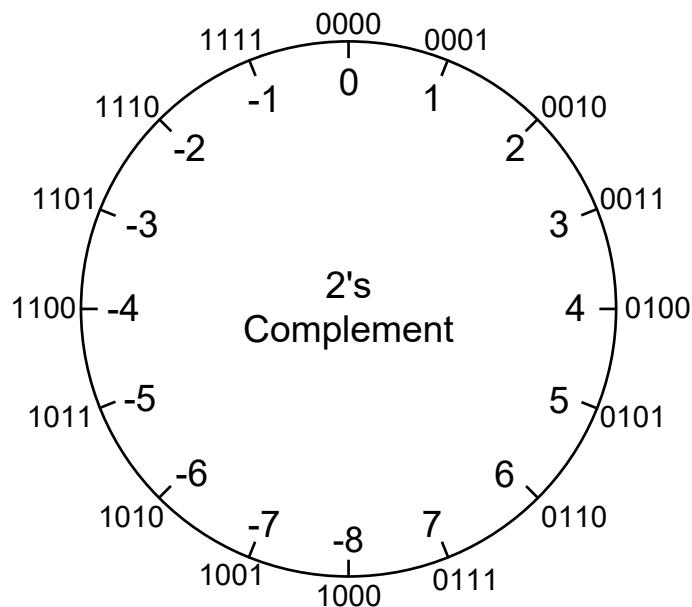
Example: -4: $4 = 0100$

$$1011 + 1 = 1100$$

$$= -8 * 1 + 4 * 1 + 0 * 2 + 0 * 1 = -4$$

- MSB is 0 if number is positive, 1 if number is negative
- What is the range of numbers that can be represented with n -bit 2's complement representation?

$$[-(2^{n-1}) : 2^{n-1}-1]$$



Complement of -8 (1000):

$$0111 + 1 = 1000$$

Complement of 0 (0000):

$$1111 + 1 = 0000$$

2's Complement Arithmetic



- To add two numbers (positive or negative), use normal binary addition and *ignore carry out of most significant column*
- Addends and sum should always have the same number of bits

$$\begin{array}{r} \\ (-3) + 4 \\ 1 0 \\ + 0 0 0 \\ \hline 0 0 0 \end{array}$$

$$\begin{array}{r} \\ 5 + (-7) \\ 0 0 \\ + 1 0 0 \\ \hline 1 1 0 \end{array}$$

$$\begin{array}{r} \\ (-2) + (-3) \\ 1 1 0 \\ + 1 1 0 \\ \hline 1 0 1 \end{array}$$

- To subtract, take complement of subtrahend and add to minuend:

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

- Step 1: Generate $(-B)$ – how?
- Step 2: Add - use ripple-carry adder!

Subtraction Using RCA



- Step 1: Generate $(-B)$
 - Complement bits of B
 - Add 1
- Step 2: Add A to $(-B)$ with RCA - *use carry-in of LSB!*

Subtract 2 from 6 (6-2):

$$\begin{array}{r}
 +2: 0010 \\
 \quad 1111 \text{ (complement)} \\
 + 1101 \text{ (+1)} \\
 \hline
 0100
 \end{array}$$

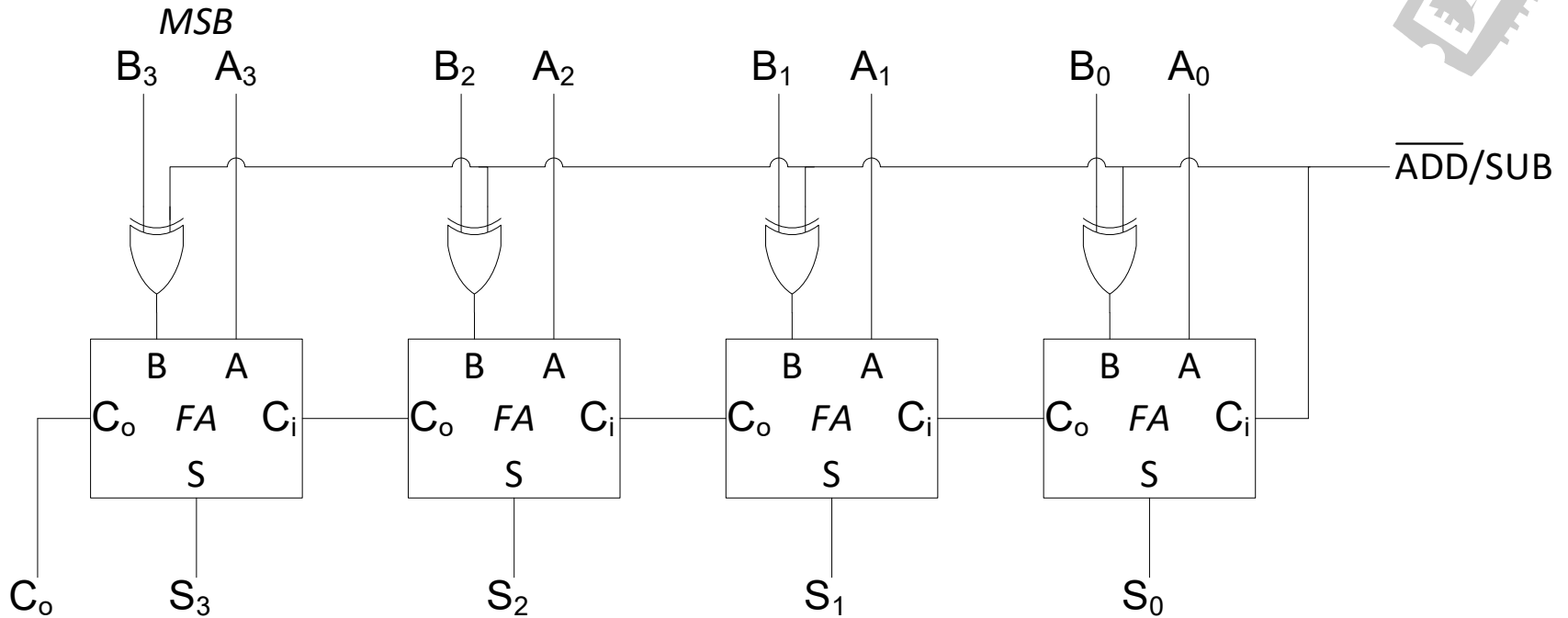
Subtract (-3) from 4 ($4 - (-3)$):

$$\begin{array}{r}
 -3: 1101 \\
 \quad 0100 \text{ (complement)} \\
 + 0010 \text{ (+1)} \\
 \hline
 0111
 \end{array}$$

- What modifications must be made to a ripple-carry adder to make a ripple-carry adder/subtractor?
 - Need a signal to determine whether we're adding or subtracting!
 - Complement bits of B (conditionally) \rightarrow XOR
 - Add 1 to B (conditionally) \rightarrow use carry-in to LSB

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0 ¹⁴

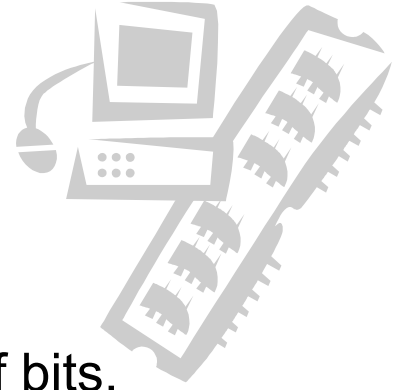
Ripple-carry Adder/Subtractor



$$\overline{\text{ADD/SUB}} = 0 \rightarrow S = A + B$$

$$\overline{\text{ADD/SUB}} = 1 \rightarrow S = A - B$$

Overflow

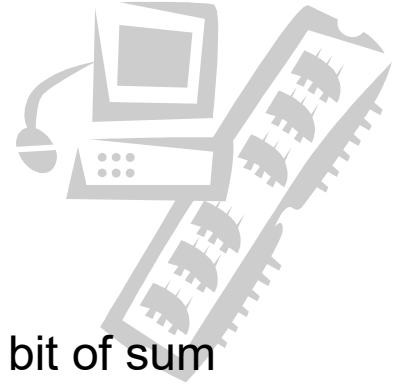


- If result of operation is too large for the current number of bits, must notify user that result is invalid

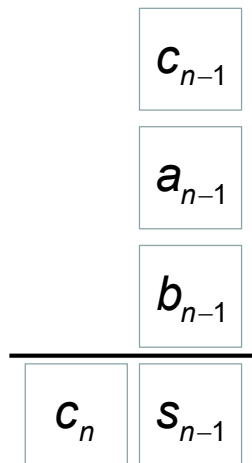
$$\begin{array}{r} 5 + 6 \\ \begin{array}{r} 1 \\ 0\ 1\ 0\ 1 \\ + 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array} = -5 \end{array} \qquad \begin{array}{r} (-3) + (-6) \\ \begin{array}{r} 1 \\ 1\ 1\ 0\ 1 \\ + 1\ 0\ 1\ 0 \\ \hline 0\ 1\ 1\ 1 \end{array} = 7 \end{array}$$

- Addition of two numbers with different signs can *never* overflow
- If sign bits of addends are the same and *different* from the sign bit of the result, then overflow has occurred
- Equivalently, if the carry-in to the most-significant column is different from the carry-out of the most-significant column, overflow has occurred
 - Easily to implement with an XOR gate

Overflow Detection



ovf = sign bits of addends are the same and *different* from the sign bit of sum



$$s_{n-1} = a_{n-1} \oplus b_{n-1} \oplus c_{n-1}$$

$$c_n = a_{n-1}b_{n-1} + c_{n-1}(a_{n-1} \oplus b_{n-1})$$



$$b_{n-1} = a_{n-1}$$

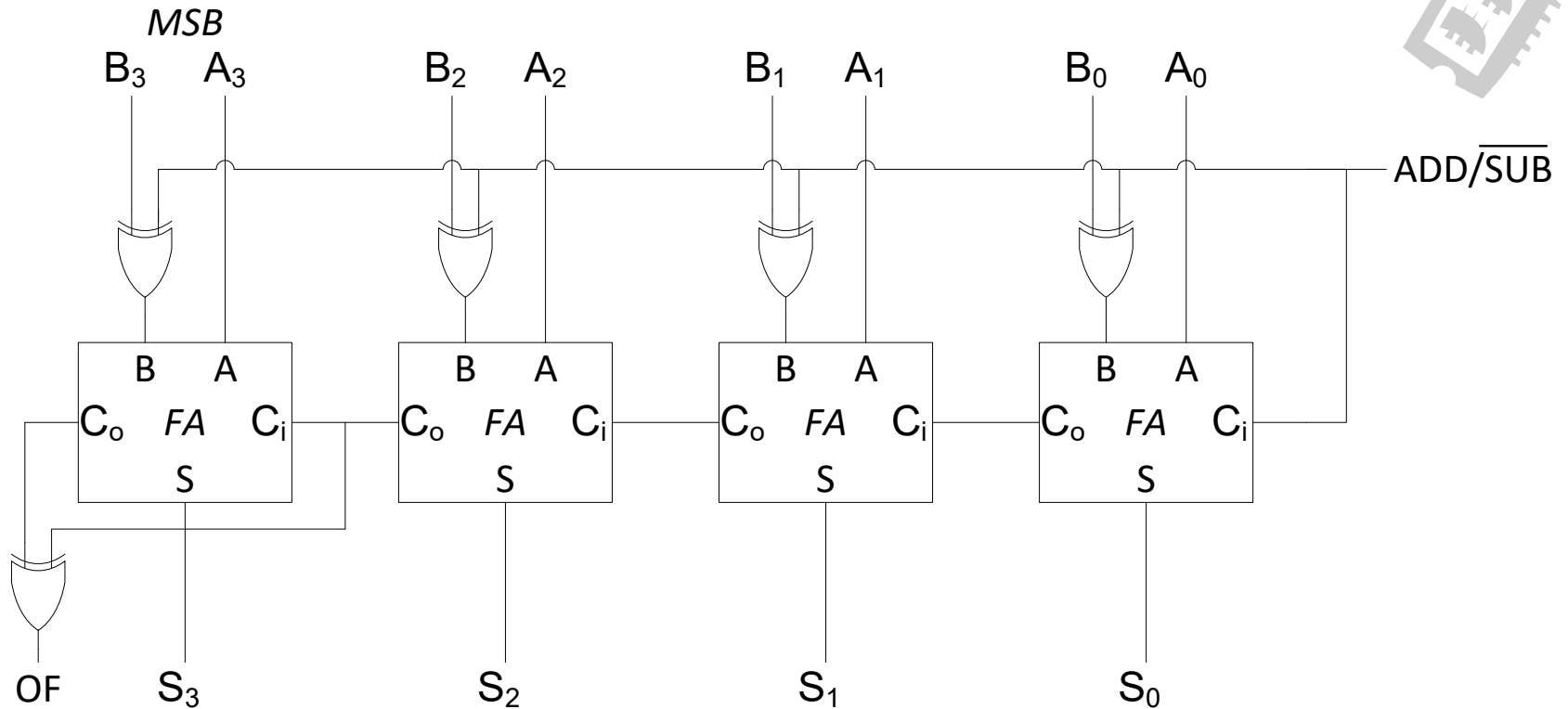
$$s_{n-1} = a_{n-1} \oplus a_{n-1} \oplus c_{n-1} = 0 \oplus c_{n-1} = c_{n-1}$$

$$c_n = a_{n-1}a_{n-1} + c_{n-1}(a_{n-1} \oplus a_{n-1}) = a_{n-1} + c_{n-1} \cdot 0 = a_{n-1}$$

$$ovf = a_{n-1} \oplus s_{n-1}$$

$$= c_n \oplus c_{n-1}$$

Ripple-carry Adder/Subtractor with Overflow Detection

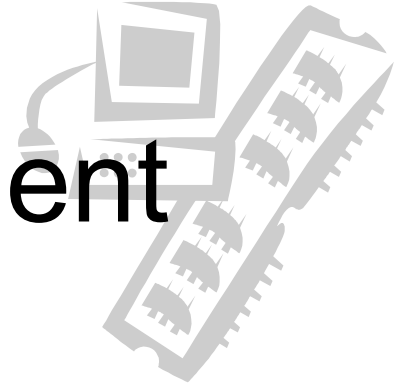


$$\overline{\text{ADD/SUB}} = 0 \rightarrow S = A + B$$

$$\overline{\text{ADD/SUB}} = 1 \rightarrow S = A - B$$

$$\text{OF} = 1 \rightarrow \text{Overflow}$$

Ones' v. Two's Complement



$$X_{1C} = x_3 x_2 x_1 x_0$$

$$-X_{1C} = x'_3 x'_2 x'_1 x'_0$$

x	$1 - x = x'$
0	$1 - 0 = 1$
1	$1 - 1 = 0$

$$\begin{array}{r}
 \boxed{1} \boxed{1} \boxed{1} \boxed{1} \\
 - \quad \boxed{x_3} \boxed{x_2} \boxed{x_1} \boxed{x_0} \\
 \hline
 -X_{1C} = \boxed{x'_3} \boxed{x'_2} \boxed{x'_1} \boxed{x'_0}
 \end{array}$$

$$-X_{1C} = 15 - X_{1C}$$

$$-X_{1C} = (2^n - 1) - X_{1C}$$

$$X_{2C} = x_3 x_2 x_1 x_0$$

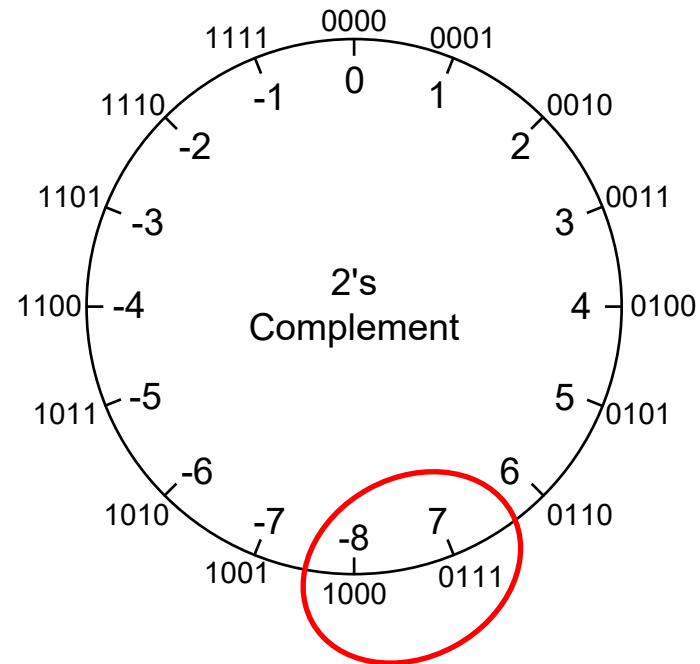
$$-X_{2C} = x'_3 x'_2 x'_1 x'_0 + 0001$$

$$\begin{array}{r}
 \boxed{1} \boxed{0} \boxed{0} \boxed{0} \boxed{0} \\
 - \quad \boxed{x_3} \boxed{x_2} \boxed{x_1} \boxed{x_0} \\
 \hline
 -X_{2C} = \boxed{x'_3} \boxed{x'_2} \boxed{x'_1} \boxed{x'_0} + 0001
 \end{array}$$

$$-X_{2C} = 16 - X_{2C}$$

$$-X_{2C} = 2^n - X_{2C}$$

2's Complement Overflow

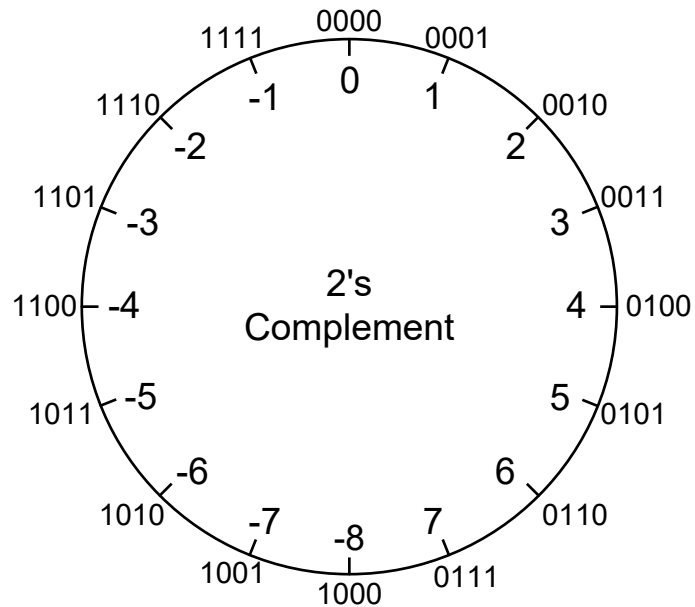


No Overflow: $A \geq 0, B < 0, A + B \in [-8, 7]$

Positive Overflow: $A, B \geq 0, A + B > 7$

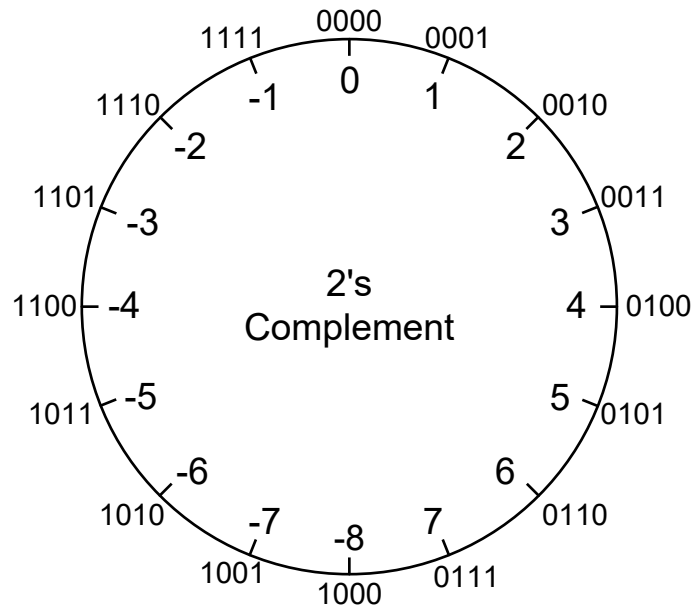
Negative Overflow: $A, B < 0, A + B < -8$

2's Complement Overflow



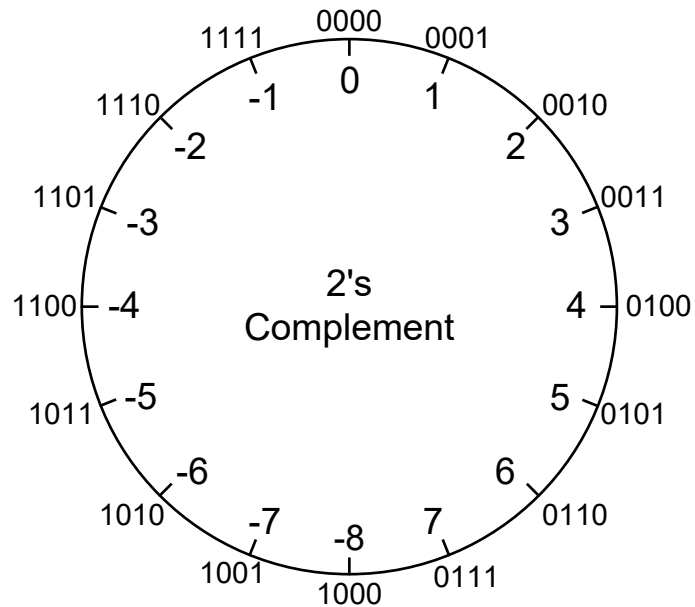
+				
<hr/>				
=				

2's Complement Overflow



+				
<hr/>				
=				

2's Complement Overflow



+				
<hr/>				
=				

In Class Exercise



Perform the following operations in the same fashion as the ripple-carry adder/subtractor. Note whether or not overflow has occurred.

Subtract 7 from 3 ($3 - 7$)

$$\begin{array}{r} +7 \quad 0 \quad 1 \quad 1 \quad 1 \\ \quad \quad 1 \quad 1 \quad \color{red}{1} \\ \quad 0 \quad 0 \quad 1 \quad 1 \\ + 1 \quad 0 \quad 0 \quad 0 \\ \hline 1 \quad 1 \quad 0 \quad 0 \end{array}$$

complement

Carry-in to most-significant column is equal to carry-out of most-significant column
→ No overflow

Subtract -4 from 5 ($5 - (-4)$)

$$\begin{array}{r} -4: \quad 1 \quad 1 \quad 0 \quad 0 \\ \quad \quad 1 \quad 1 \quad 1 \quad \color{red}{1} \\ \quad 0 \quad 1 \quad 0 \quad 1 \\ + 0 \quad 0 \quad 1 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 1 \end{array}$$

complement

Carry-in to most-significant column is not equal to carry-out of most-significant column
→ Overflow

Derivative Arithmetic Operations



- Incrementing (+1) and decrementing (-1)
 - Start with an adder and simplify the circuit
- Multiplying an unsigned by a power of two
 - Shift to the left
- Dividing an unsigned by a power of two
 - Shift to the right
- Multiplication: $A * B$
 - Same algorithm as in decimal
 - Partial products: multiply A by digits of B
 - Add up all partial products
- Division A/B : ?