UM EECS 270 F22
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:

$$\begin{array}{rcccc} & 1 & & 1 & \\ & 4 & 2 & 5 & 9 \\ + & 1 & 8 & 3 & 7 \\ \hline & 6 & 0 & 9 & 6 \end{array}$$

This is the *carry out* of the first column, which becomes the *carry in* to the second column

Binary:

$$\begin{array}{rcccc} & & 1 & 1 & \\ & 1 & 0 & 0 & 1 \\ + & 0 & 0 & 1 & 1 \\ \hline & 1 & 1 & 0 & 0 \end{array}$$

- 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).

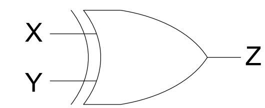
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = AB$$

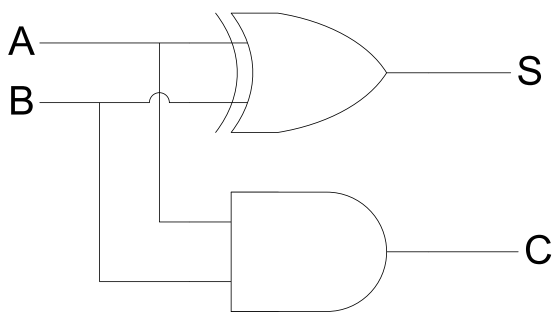
$$S = A\overline{B} + \overline{A}B = A \oplus B$$

XOR Operation

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

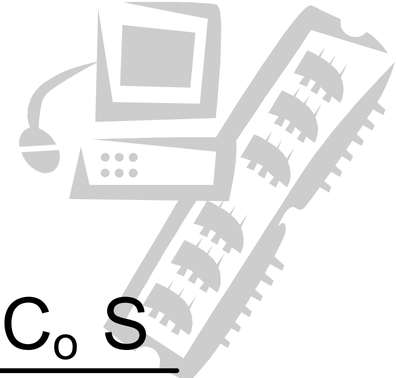


Implementation:



This circuit is called a **Half Adder (HA)**

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



		C_o 1	C_i 1	
	1	0	0 / A	1
+	0	0	1 / B	1
	1	1	0 / S	0

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:

$$S = \sum\nolimits_{A,B,C_i} \left(1, 2, 4, 7\right)$$

$$S = A'B'C_i + A'BC_i' + AB'C_i' + ABC_i$$

$$S = C_i\left(A'B' + AB\right) + C_i'\left(A'B + AB'\right)$$

$$S = C_i\left(A \oplus B\right)' + C_i'\left(A \oplus B\right)$$

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

- Carry-out output:

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

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

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

$$BC_i \qquad AC_i \qquad AB$$

$$C_o = AB + AC_i + BC_i$$

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:

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

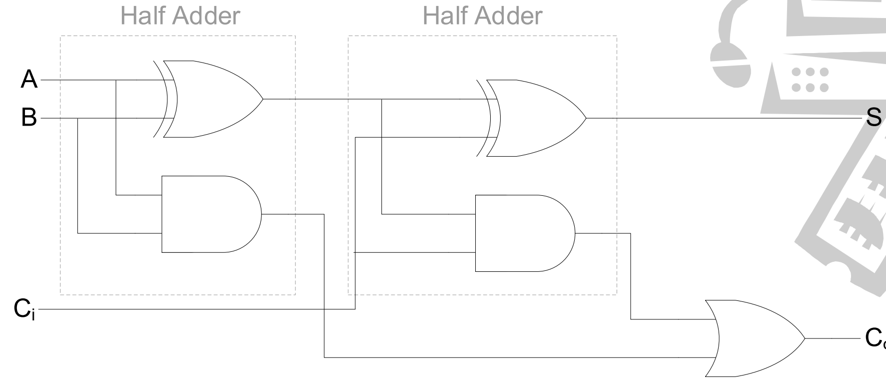
$$\begin{aligned} C_o &= AB + AC_i + BC_i \\ &= AB + C_i(A + B) \\ &= AB + C_i\left(\left(A \oplus B\right) + AB\right) \\ &= AB + (A \oplus B)C_i + ABC_i \\ &= AB + (A \oplus B)C_i \end{aligned}$$

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..

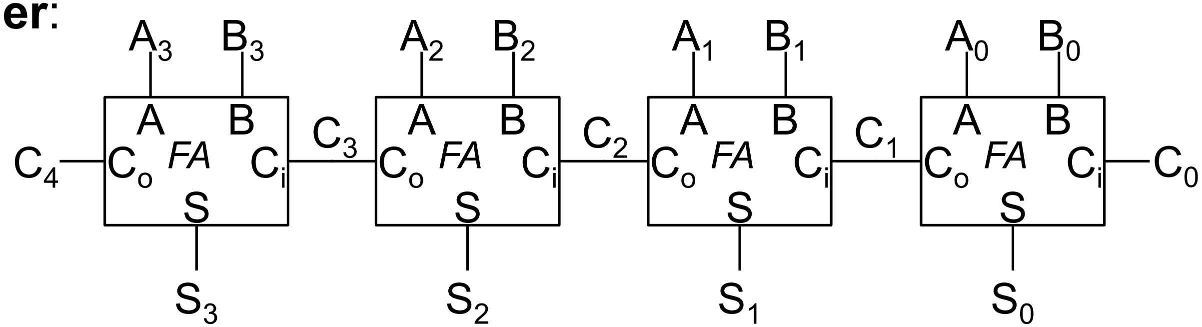
- Final Circuit:

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

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



- 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?***

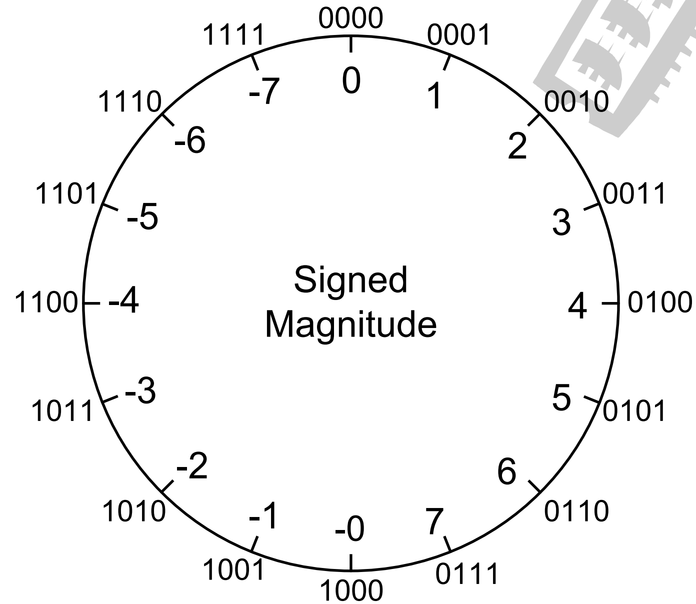
Binary Representation of Negative Numbers

- **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?

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



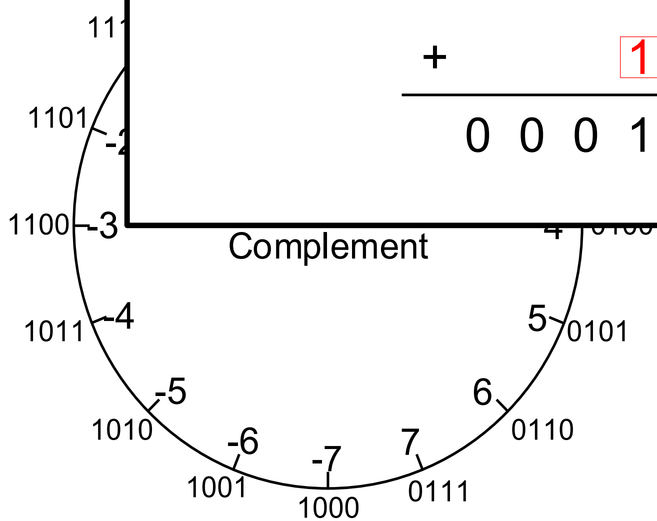
- Drawbacks
  - Two representations of 0
  - Signed magnitude arithmetic requires comparing sign bits and then performing addition or subtraction – leads to complex hardware

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 i
- What i … *n*-bit 1s' compl

$$
\begin{array}{rr}
 & \\
 & -3 \\
+ & 4 \\
\hline
 & 1
\end{array}
\qquad
\begin{array}{rcccc}
 & \boxed{1} & 1 & 0 & 0 \\
 & 1 & 1 & 0 & 0 \\
+ & 0 & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 0 & 0 \\
+ & & & & \boxed{1} \\
\hline
 & 0 & 0 & 0 & 1
\end{array}
\qquad\qquad
\begin{array}{rr}
 & \\
 & -3 \\
+ & 5 \\
\hline
 & 2
\end{array}
\qquad
\begin{array}{rcccc}
 & \boxed{1} & 1 & 0 & 0 \\
 & 1 & 1 & 0 & 0 \\
+ & 0 & 1 & 0 & 1 \\
\hline
 & 0 & 0 & 0 & 1 \\
+ & & & & \boxed{1} \\
\hline
 & 0 & 0 & 1 & 0
\end{array}
$$



ns of 0

– must

SB to

sum (**end-around carry)**

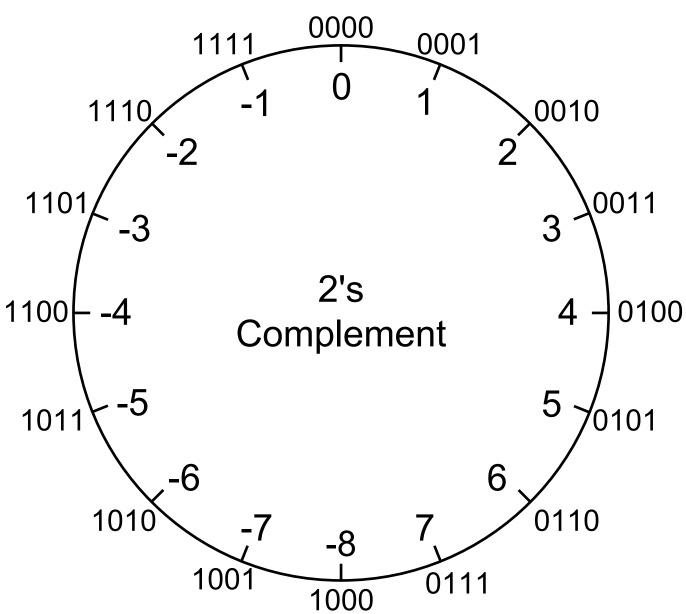
Abandoned in the 60s in favor of 2's complement….

- **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

(-3) + 4

$$\begin{array}{r} {}^{1\,1} \\ 1\;1\;0\;1 \\ +\;0\;1\;0\;0 \\ \hline 0\;0\;0\;1 \end{array}$$

5 + (-7)

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

(-2) + (-3)

$$\begin{array}{r} {}^{1\,1} \\ 1\;1\;1\;0 \\ +\;1\;1\;0\;1 \\ \hline 1\;0\;1\;1 \end{array}$$

- 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):

```
+2:  0 0 1 0
                 complement
     1 1 1 1
     0 1 1 0
   + 1 1 0 1
   ---------
     0 1 0 0
```

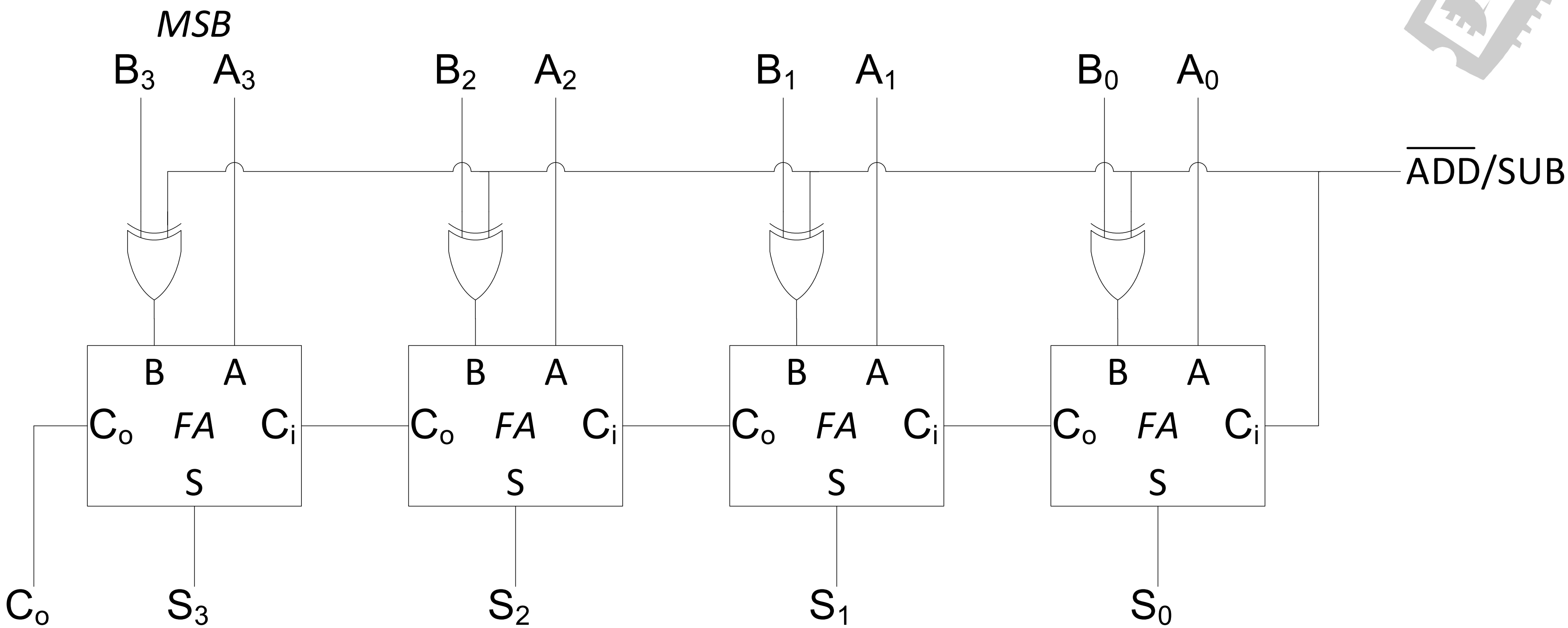
Subtract (-3) from 4 (4 – (-3)):

```
-3:  1 1 0 1
                 complement
           1
     0 1 0 0
   + 0 0 1 0
   ---------
     0 1 1 1
```

- 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) → XOR
  - Add 1 to B (conditionally) → use carry-in to LSB

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

Ripple-carry Adder/Subtractor



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

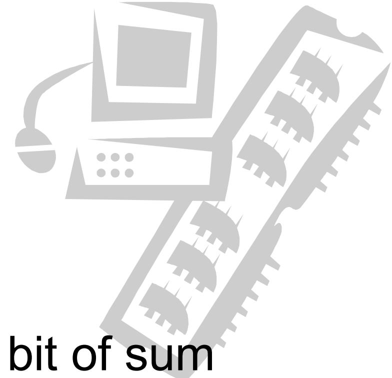
$\overline{\text{ADD}}/\text{SUB} = 1 \rightarrow S = A - B$

Overflow

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

$$
\begin{array}{c}
5 + 6 \\[4pt]
\begin{array}{r}
{}^{1} \\
0\ 1\ 0\ 1 \\
+\ \underline{0\ 1\ 1\ 0} \\
1\ 0\ 1\ 1
\end{array}
= \cancel{-5}
\end{array}
\qquad\qquad
\begin{array}{c}
(-3) + (-6) \\[4pt]
\begin{array}{r}
{}^{1} \\
1\ 1\ 0\ 1 \\
+\ \underline{1\ 0\ 1\ 0} \\
0\ 1\ 1\ 1
\end{array}
= \cancel{7}
\end{array}
$$

- 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

$$\begin{array}{cc} & c_{n-1} \\ & a_{n-1} \\ & b_{n-1} \\ \hline c_n & s_{n-1} \end{array}$$

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

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

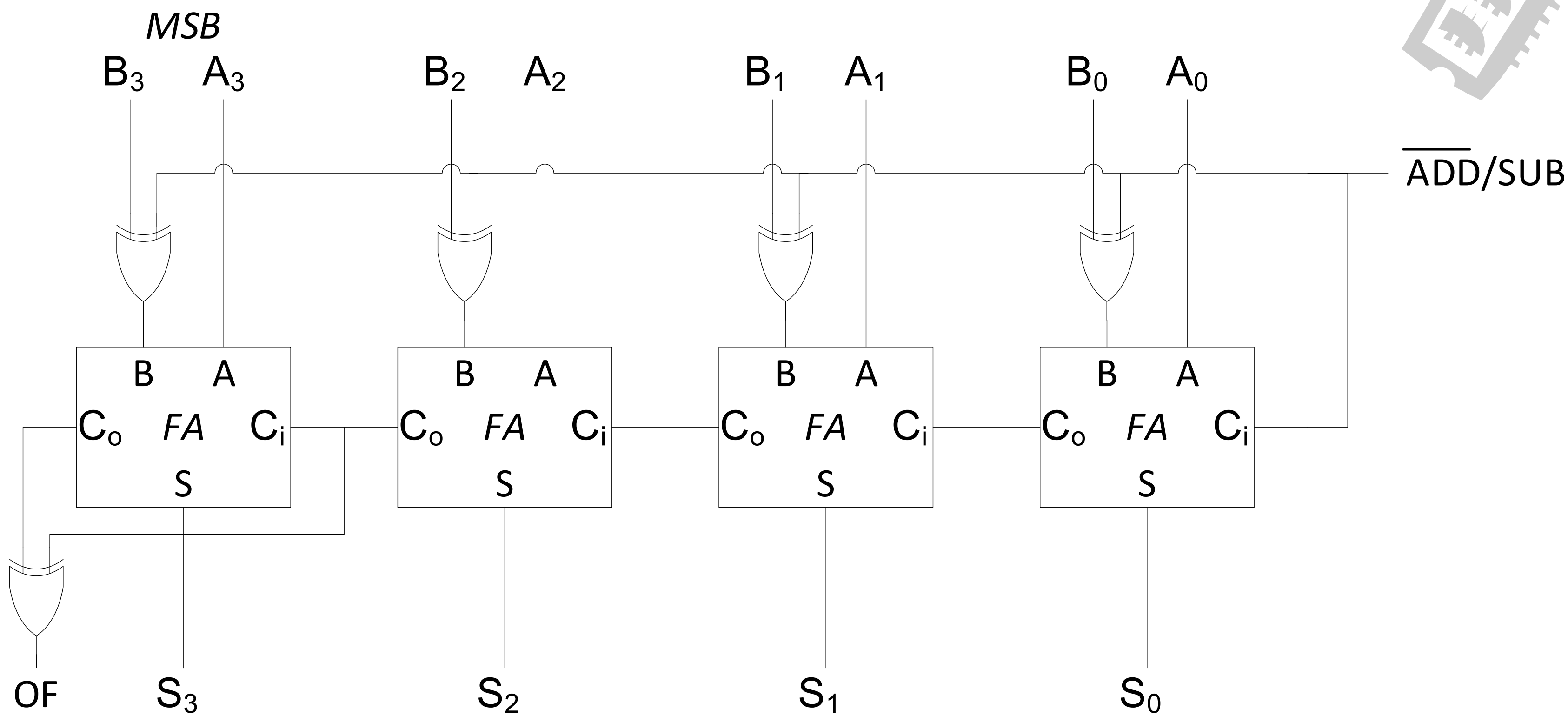
$$\Big\Downarrow \; b_{n-1} = a_{n-1}$$

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

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

$$\begin{aligned} ovf &= a_{n-1} \oplus s_{n-1} \\ &= c_n \oplus c_{n-1} \end{aligned}$$

Ripple-carry Adder/Subtractor with Overflow Detection



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

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

$OF = 1 \rightarrow$ Overflow

Ones' v. Two's Complement

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

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

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

$$\begin{array}{rcccc} & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{1} \\ - & \boxed{x_3} & \boxed{x_2} & \boxed{x_1} & \boxed{x_0} \\ \hline -X_{1C} = & \boxed{x_3'} & \boxed{x_2'} & \boxed{x_1'} & \boxed{x_0'} \end{array}$$

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

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

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

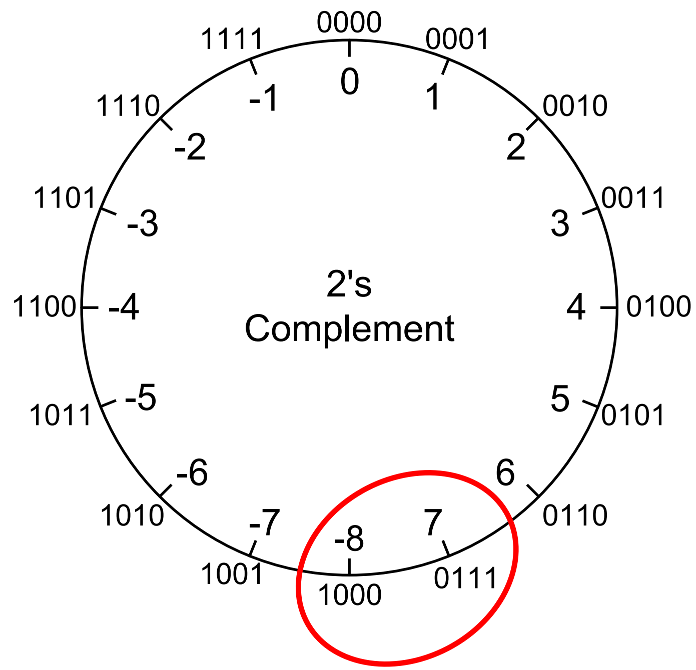
$$-X_{2C} = x_3' x_2' x_1' x_0' + 0001$$

$$\begin{array}{rccccc} & \boxed{1} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} \\ - & & \boxed{x_3} & \boxed{x_2} & \boxed{x_1} & \boxed{x_0} \\ \hline -X_{2C} = & & \boxed{x_3'} & \boxed{x_2'} & \boxed{x_1'} & \boxed{x_0'} \end{array} + 0001$$

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

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

2’s Complement Overflow

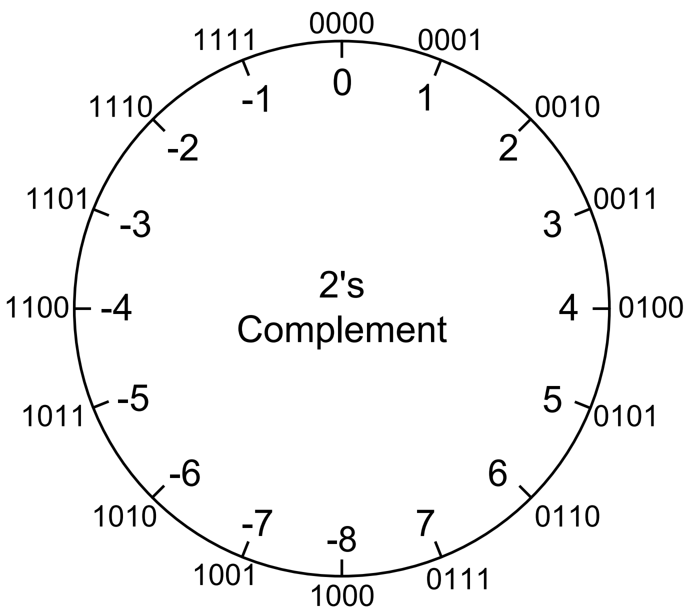


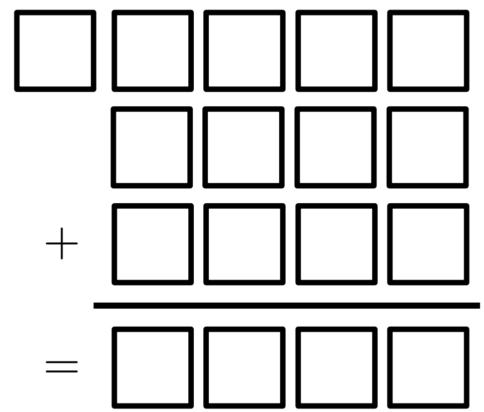
No Overflow: $A \geq 0, B < 0, A + B \in \left[-8, 7\right]$

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

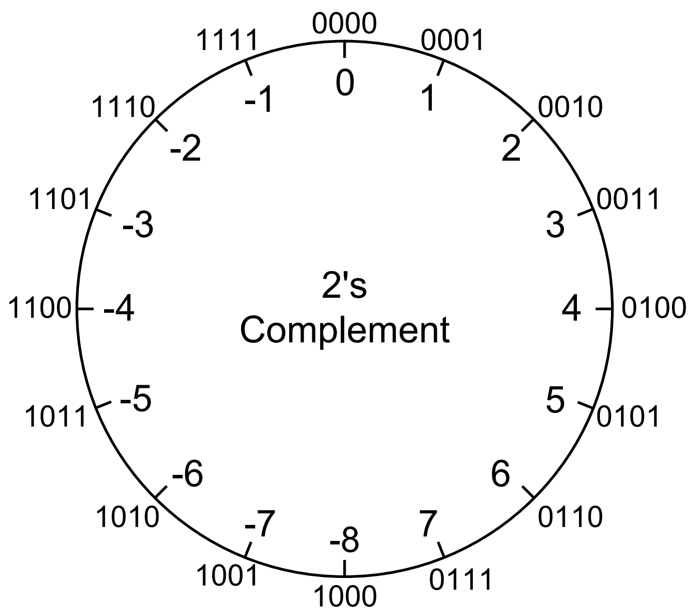
Negative Overflow: $A, B < 0, A + B < -8$

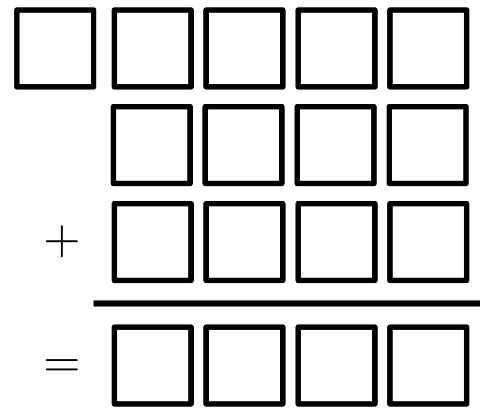
2’s Complement Overflow



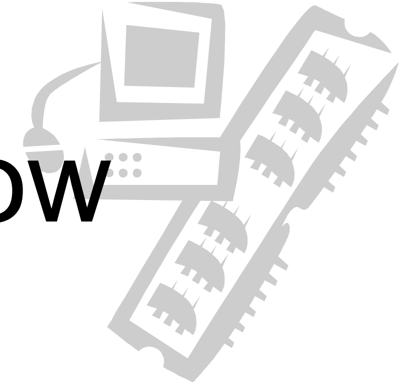


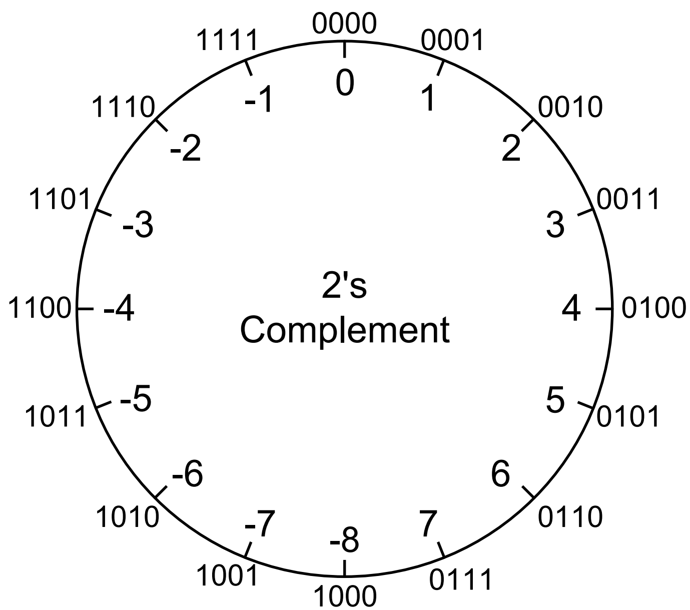
2’s Complement Overflow

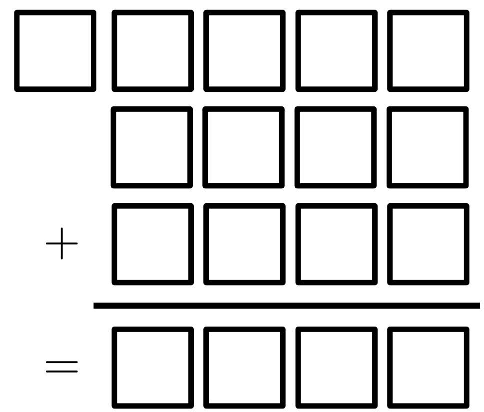




2’s Complement Overflow







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)

+7 0 1 1 1

complement

$$
\begin{array}{rcccc}
 & & 1 & 1 & \color{red}{1} \\
 & 0 & 0 & 1 & 1 \\
+ & 1 & 0 & 0 & 0 \\
\hline
 & 1 & 1 & 0 & 0
\end{array}
$$

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

Subtract -4 from 5 (5 – (-4))

-4: 1 1 0 0

complement

$$
\begin{array}{rcccc}
 & 1 & 1 & 1 & \color{red}{1} \\
 & 0 & 1 & 0 & 1 \\
+ & 0 & 0 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 1
\end{array}
$$

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: ?