numbers

Positional notation for numbers

D_i is a "digit",
a symbol for a value === value(D_i).

B is a value,
the "base" of the number notation.

Given a string of digit chars,
D_n D_{n-1} D_{n-2} ... D_1 D_0

there is a rule, an algorithm,
to find the value it represents.

unsigned 3-bit binary:

--- { D_i } = { "0", "1" }

value("0") == {}
the empty set

value("1") == { {} }
the set containing the empty set

--- base, B == { {}, { {} } }
written "2"

$110 \rightarrow D_2 = 1 \quad D_1 = 1 \quad D_0 = 0$

$value(110) = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

$value(1)$

$D_i \rightarrow value(D_i) \cdot B^i$

1-bit ADDITION:

C_{in}
A
+ B

C_{out} S

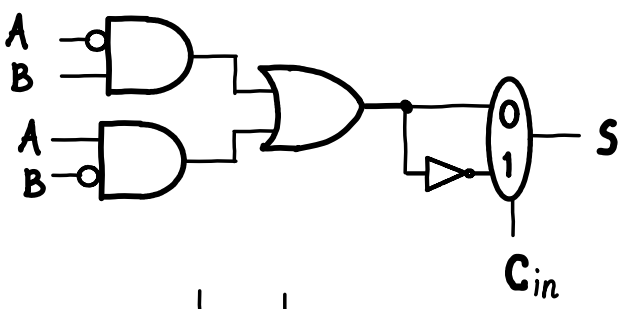
$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array}$ $\begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array}$ $\begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array}$ $\begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$ ← carry

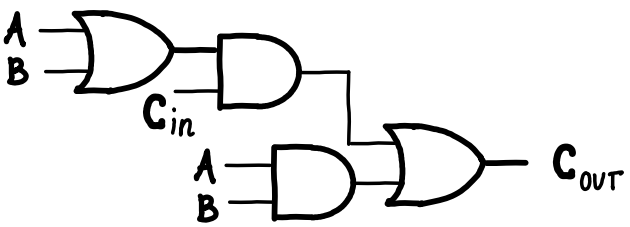
A	B	S = XOR
0	0	0
0	1	1
1	0	1
1	1	0

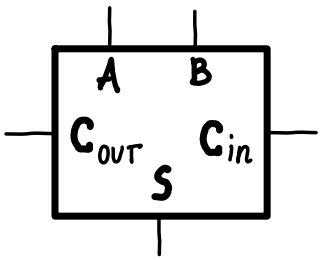
w/ Carry In:

$\begin{array}{r} 1 \\ 0 \\ +0 \\ \hline 1 \end{array}$ $\begin{array}{r} 1 \\ 0 \\ +1 \\ \hline 10 \end{array}$ $\begin{array}{r} 1 \\ 1 \\ +0 \\ \hline 10 \end{array}$ $\begin{array}{r} 1 \\ 1 \\ +1 \\ \hline 11 \end{array}$

A	B	$\overline{XOR}$
0	0	1
0	1	0
1	0	0
1	1	1







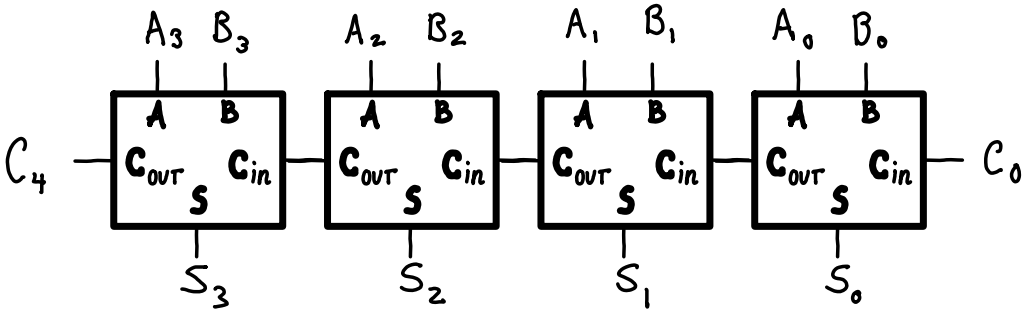
1-bit Full ADDER

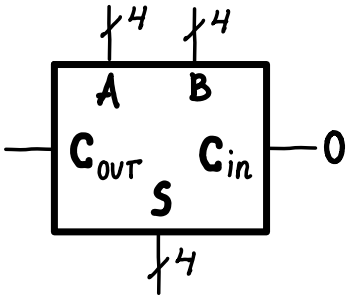
Delay: 1 per gate

7 for S; 3 for C_{out}

4-bit Full Adder:

$$\begin{array}{r} A_3\ A_2\ A_1\ A_0 \\ +\ B_3\ B_2\ B_1\ B_0 \\ \hline C_4\ S_3\ S_2\ S_1\ S_0 \end{array}$$





4-bit ripple-carry ADDER

delay = 16

SUBTRACTION:

$$\begin{array}{r} X_0 \\ -\ Y_0 \\ \hline B_1\ S_0 \end{array}$$

w/o Borrow In:

$$\begin{array}{r} 0 \\ -0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ -1 \\ \hline -11 \end{array} \quad \begin{array}{r} 1 \\ -0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ -1 \\ \hline 0 \end{array}$$

$S = XOR$

$B = \overline{X} \cdot Y$

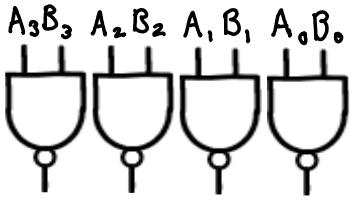
w/ Borrow In:

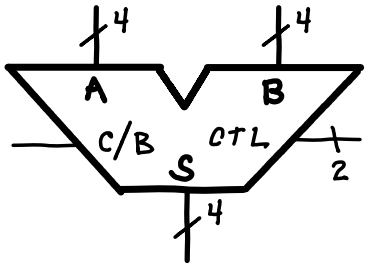
$$\begin{array}{r} -1 \\ 0 \\ -0 \\ \hline -11 \end{array} \quad \begin{array}{r} -1 \\ 0 \\ -1 \\ \hline -10 \end{array} \quad \begin{array}{r} -1 \\ 1 \\ -0 \\ \hline 0 \end{array} \quad \begin{array}{r} -1 \\ 1 \\ -1 \\ \hline -11 \end{array}$$

$S = \overline{XOR}$

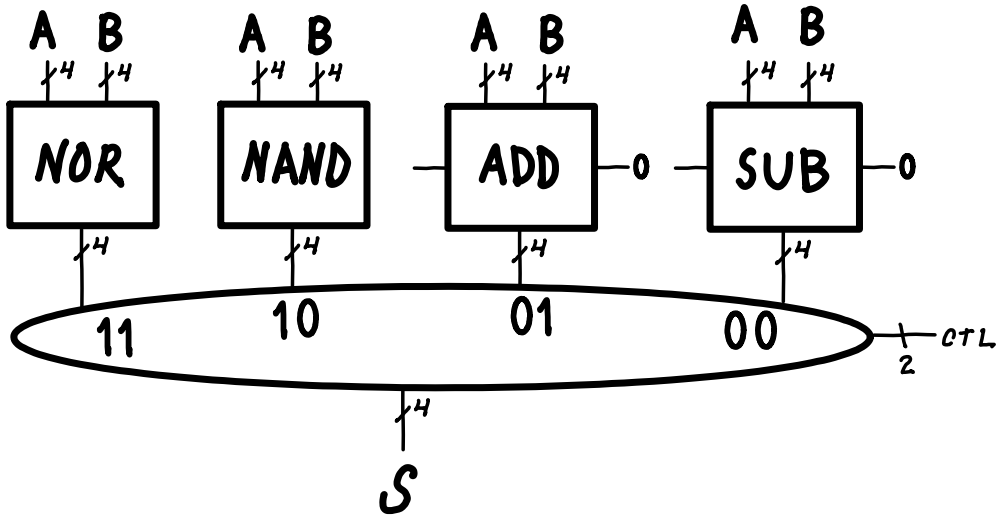
$B = \overline{X \cdot \overline{Y}}$

4-bit NAND





4-bit ALU



Unsigned Arith. Errors

3-bit numbers

A + B > 7 ===> Overflow Error

S == (A+B) mod 8 < 8 ← carry is not in S

A - B < 0 ===> Overflow Error

S == (A-B) mod 8 > -1 ← borrow is not in S

} Error if c/b = 1

Other Number Encodings

Shannon Coding? How many bits do we need to represent something?

w/ n bits we have 2^n different n-bit patterns

→ We can choose interpretation

What other number values are we interested in? **Are other encodings useful?**

3-bit Unsigned Scale Integers

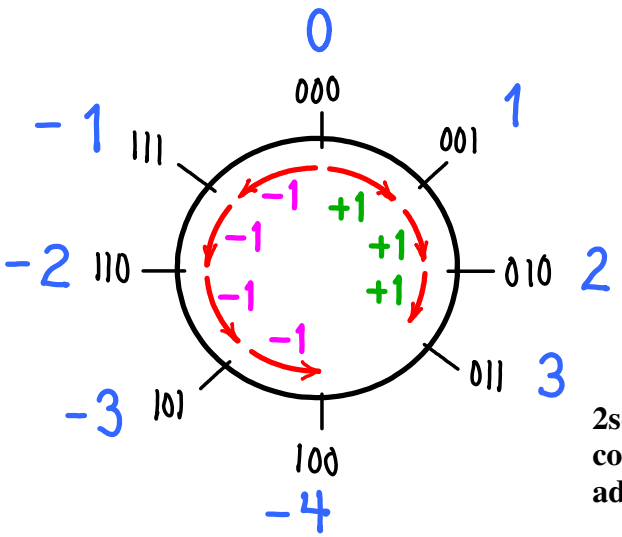
CODE	VALUE interpretation
000	0
001	8
010	16
011	24
100	32
101	40
110	48
111	56

3-bit Sign-magnitude Integers

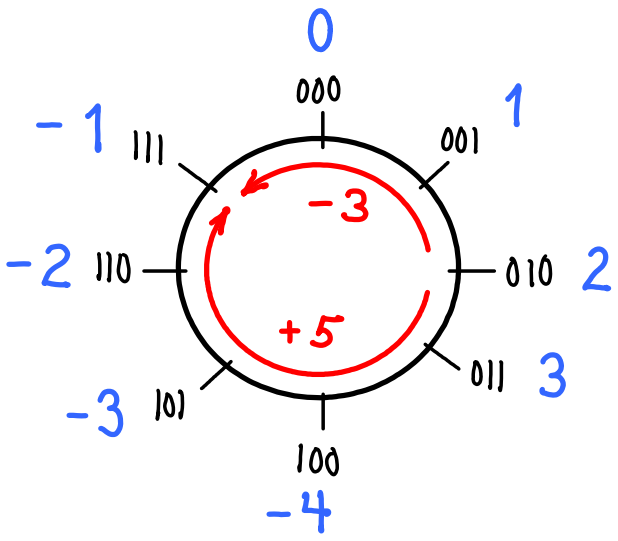
CODE	VALUE interpretation
000	+0
001	+1
010	+2
011	+3
100	- 0
101	- 1
110	- 2
111	- 3

3-bit 2s-Complement Integers

CODE	VALUE interpretation
000	0
001	+1
010	+2
011	+3
100	- 4
101	- 3
110	- 2
111	- 1



2s-Complement convenient for addition/subtraction



3-bit unsigned arithmetic, ignoring Carry/Borrow

====> MOD 8 arithmetic

Moving -3 == Moving +5

Subtract 3 == Adding +5

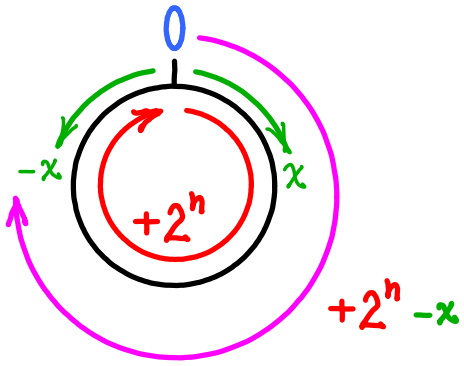
010 + 101 == 111
+2 -3 -1

We can represent negative numbers.
We can do Add and Sub using only an adder.

ADD/SUb w/ signed-magnitude? How?

n-bit 2's Complement

$$x \geq 0 \longrightarrow -x \underset{\text{mod } 2^n}{=} 2^n - x$$



All the way around is + 2^n

Stopping short by x is

2^n - x

Sanity check $-(-x)$?

$-x \longrightarrow (2^n - x)$

$-(-x) \longrightarrow 2^n - (2^n - x)$

$= x$

$-(-X) = X$

in 2's comp.

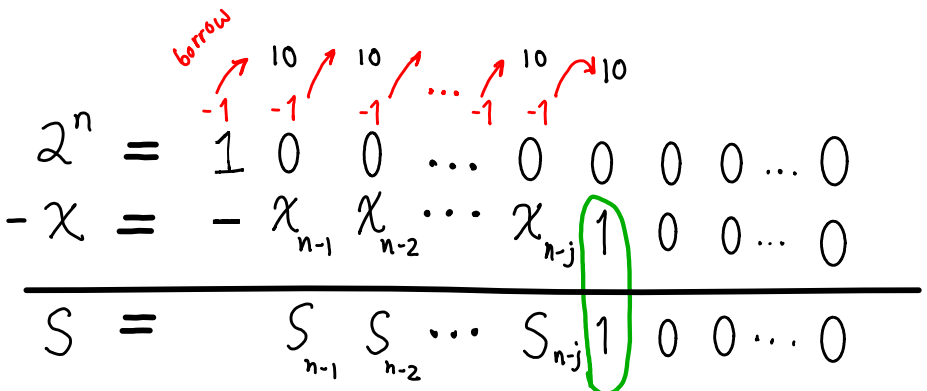
Try $(-(-3))$ in $n=3$ 2's comp : $2^n = 2^3 = 8$

$(-3)_{2\text{'s comp}} \longrightarrow 2^n - 3 = 8 - 3 = 5$

$\left(-(-3)_{2\text{'s comp}}\right)_{2\text{'s comp}} \longrightarrow 2^n - 5 = 8 - 5 = +3$

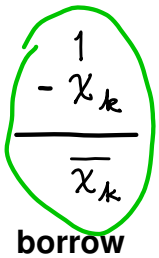
Converting 2's Comp ?

n-bit

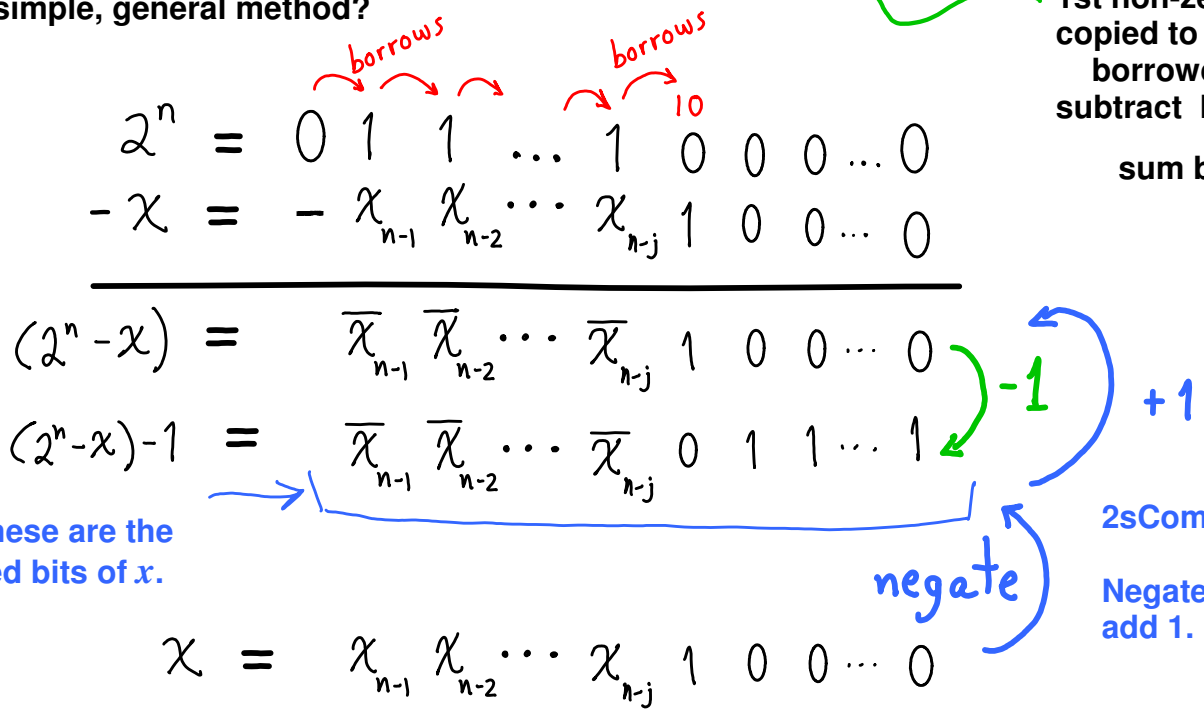


Is there a simple, general method?

1st non-zero bit copied to S:
borrowed = 10
subtract bit = - 1
sum bit = $\overline{1}$



borrow = bit flip.



NB--These are the negated bits of x.

2sComp(x):

Negate bits, add 1.

Does (2^n - x) work all the time?

Does "flip bits and add 1" work all the time?

zero 0000 ==> 1111+1 ==> 0000

most positive 0111 ==> 1000+1 ==> 1001
in between 0110 ==> 1001+1 ==> 1010
least positive 0001 ==> 1110+1 ==> 1111

OK! Carry doesn't propagate to make result positive.

least negative 1111 ==> 0000+1 ==> 0001
in between 1011 ==> 0100+1 ==> 0101
most negative 1000 ==> 0111+1 ==> 1000 !

Oops. Carry propagates, makes result negative.

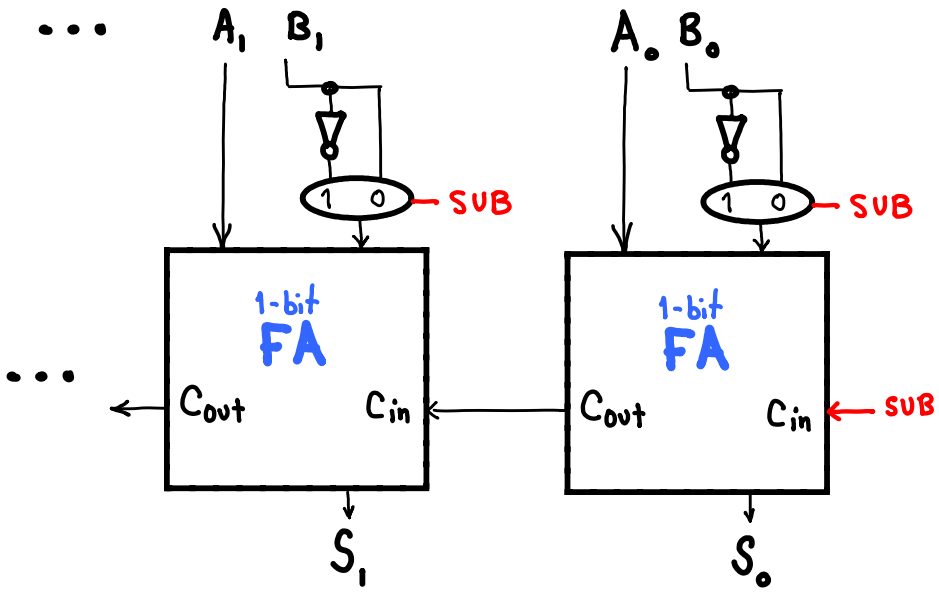
Subtraction Using an ADDER

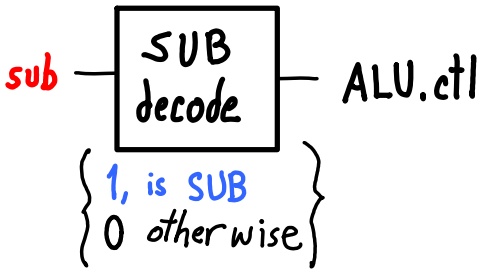
Produce - x in 2s-Complement (regardless of whether x is + or -):

Negate bits ===> NOT each bit
then add 1 ===> $C_0 = 1$

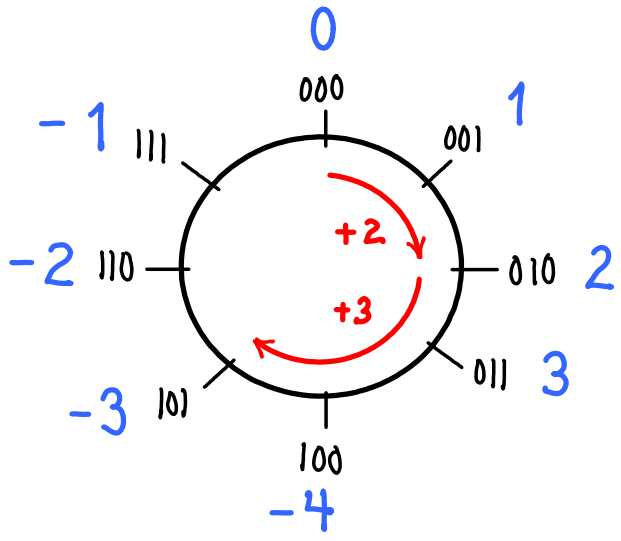
A + 2sComp(B)

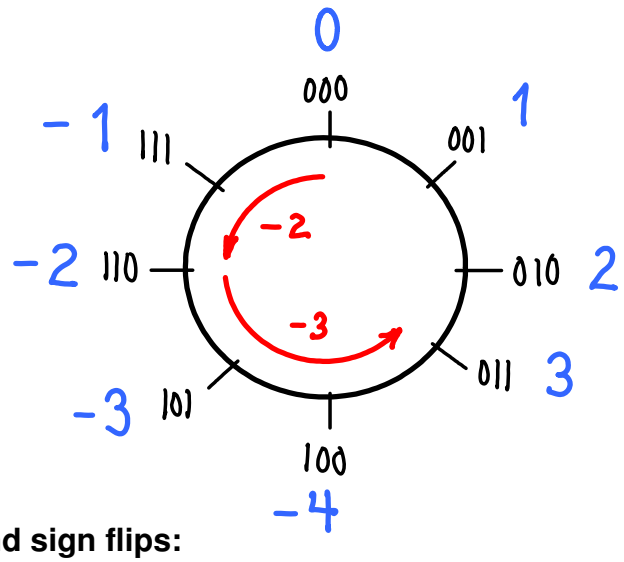
$A - B = A + (-B)$





2s-Comp. Errors: Overflow





Adding same signs, and sign flips:
Overflow ERROR