Number Systems

- Readings: 3-3.3.3, 3.3.5
- Problem: Implement simple pocket calculator
- Need: Display, adders & subtractors, inputs
  - Display: Seven segment displays
  - Inputs: Switches
- Missing: Way to implement numbers in binary
- Approach: From decimal to binary numbers
  (and back)

Arithmetic Operations

Decimal:

```
   5 7 8 9 2
+  7 8 9 5 6
-----------
```

Binary:

FULL ADDER

HALF-ADDER

```
   1 0 1 0 1 1 1
+  0 1 0 0 1 0 1
----------------
   1 1 1 1 1 0 0
```

Decimal:

```
   5 7 8 9 2
-  3 2 9 4 6
-----------
```

Binary:

```
   1 0 1 0 0 1 1 0
-  0 0 1 1 0 1 1 1
------------------
   0 1 1 0 1 1 1 1
```

Arithmetic Operations (cont.)

Decimal:

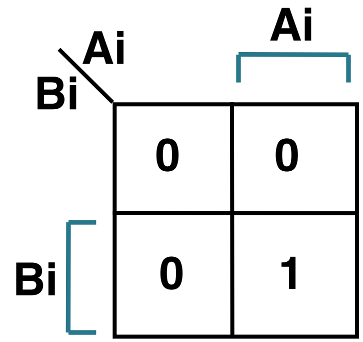
```
    2 0 1
  * 2 1 4
  -------
```

Binary:

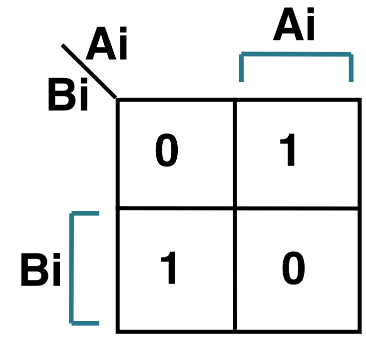
```
          1 0 0 1
        * 1 0 1 1
        ---------
          1 0 0 1
        1 0 0 1 -
      0 0 0 0 - -
    1 0 0 1 - - -
  ---------------
    1 1 0 0 0 1 1
```

Half Adder

Ai	Bi	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

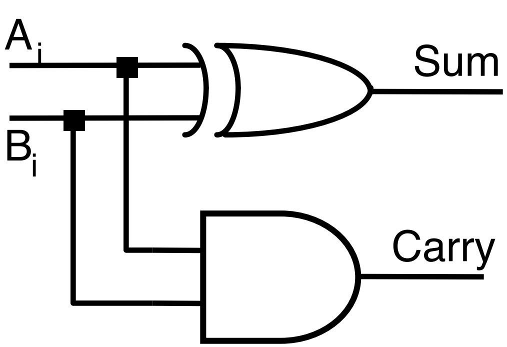


Carry = Ai Bi



Sum = $\overline{Ai}\,Bi + Ai\,\overline{Bi}$

= $Ai \oplus Bi$



Half-adder Schematic

Full Adder

A	B	CI	CO	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

$$C_{OUT} = AB + AC_{IN} + BC_{IN}$$

$$S = A \oplus B \oplus C$$

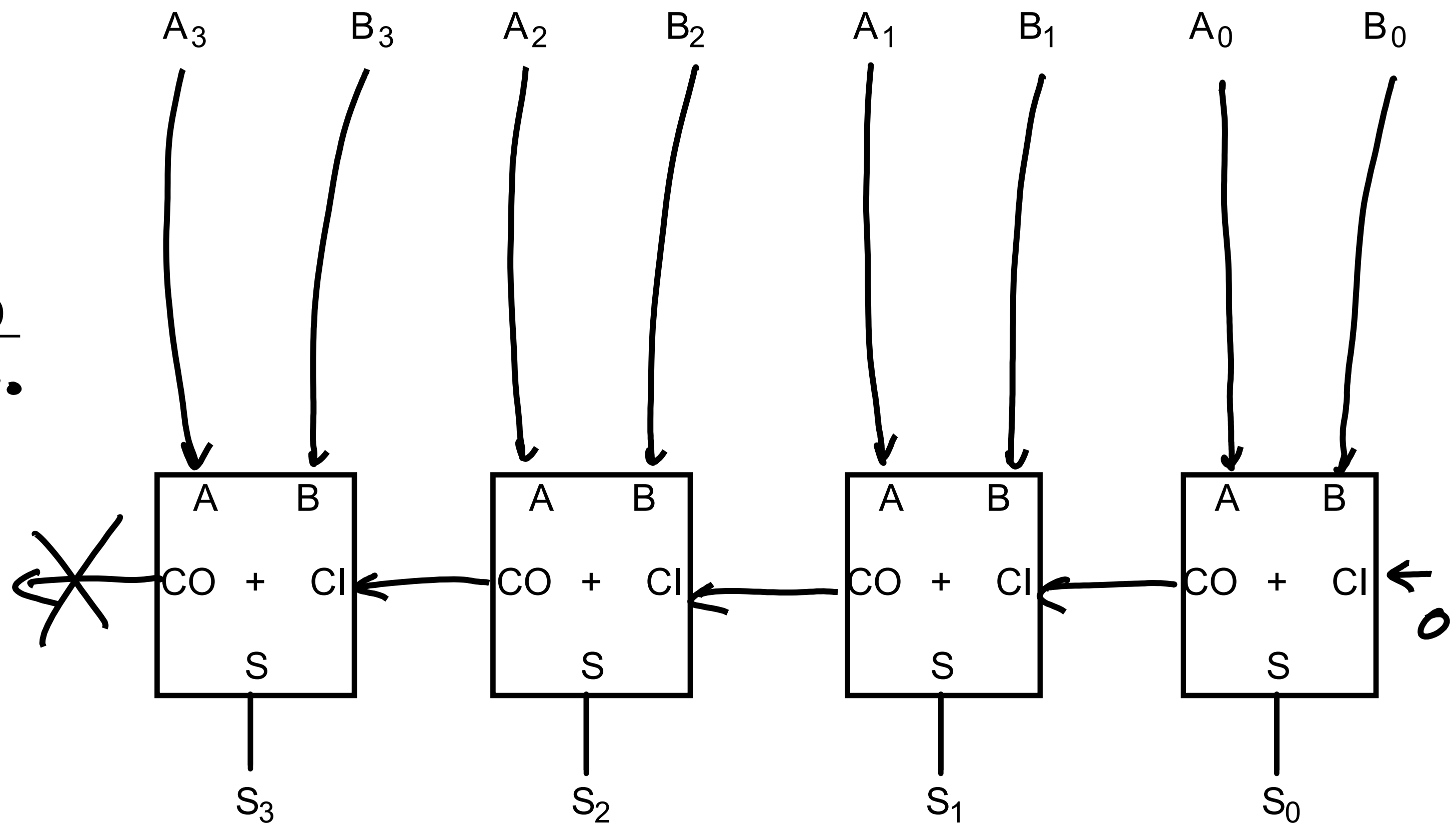
Full Adder Implementation

Multi-Bit Addition

"RIPPLE-CARRY ADDER"

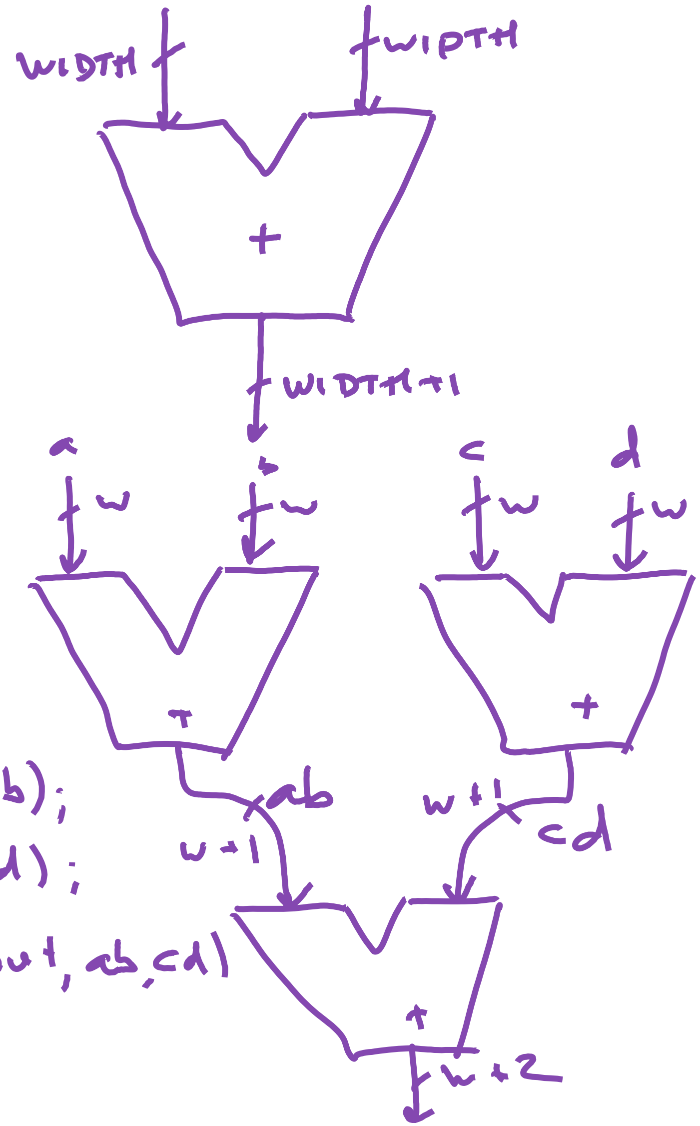
$$\begin{array}{r} \not{c_3}\; A_3\, A_2\, A_1\, A_0 \\ +\; B_3\, B_2\, B_1\, B_0 \\ \hline s_3\; s_2\; s_1\; s_0 \end{array}$$

c_2 c_1 c_0



Multi-Bit Addition in Verilog, Parameters

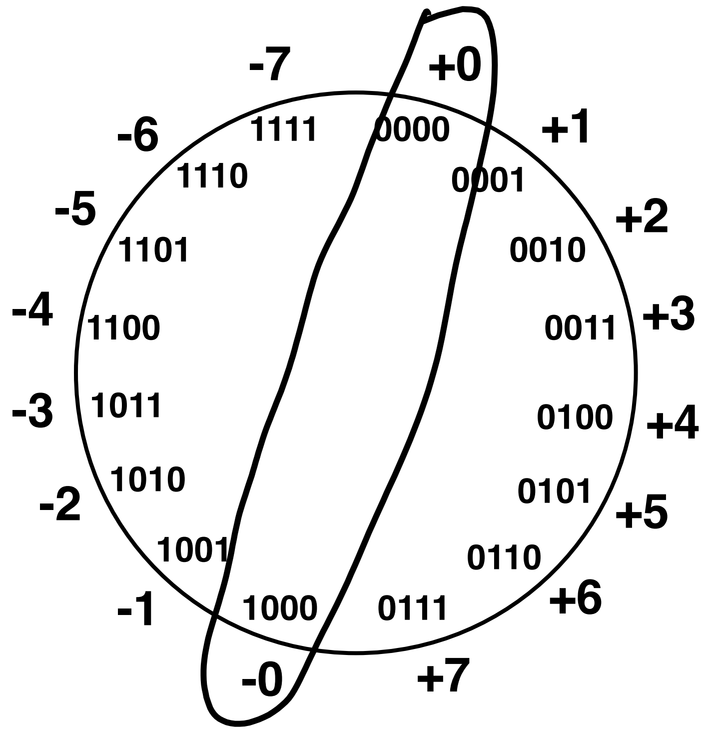
```
module uadd #(parameter WIDTH=8)
   (out, a, b);
   output reg [WIDTH:0] out;
   input      [WIDTH-1:0] a, b;

   always @(*) begin
     out = a + b;
   end
endmodule

module add4 #(parameter W=22)
   (out, a, b, c, d);
   output [W+1:0] out;
   input  [W-1:0] a, b, c, d;
   wire [w:0] ab, cd;

   uadd #(.WIDTH(w)) u_ab(ab, a, b);
   uadd #(.WIDTH(w)) u_cd(cd, c, d);
   uadd #(.WIDTH(w+1)) u_abcd(out, ab, cd)

endmodule
```

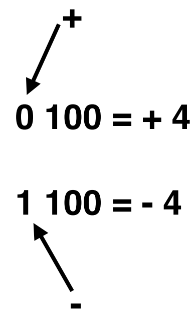


Negative Numbers

- Need an efficient way to represent negative numbers in binary
  - Both positive & negative numbers will be strings of bits
  - Use fixed-width formats (4-bit, 16-bit, etc.)
- Must provide efficient mathematical operations
  - Addition & subtraction with potentially mixed signs
  - Negation (multiply by -1)

Sign/Magnitude Representation





High order bit is sign: 0 = positive (or zero), 1 = negative

Three low order bits is the magnitude: 0 (000) thru 7 (111)

Number range for n bits = $+/-2^{n-1}$ -1

Representations for 0:

Sign/Magnitude Addition

SIGNS ARE SAME: ADD MAGNITUDE, KEEP SIGN

DIFFERENT: SUBTRACT SMALLER FROM BIGGER MAG., KEEP SIGN OF BIGGER #

```
    0 | 0 1 0  (+2)
+   0 | 1 0 0  (+4)
-------------------
    0 | 1 1 0  (+6)
```

```
    1 | 0 1 0  (-2)
+   1 | 1 0 0  (-4)
-------------------
    1 | 1 1 0  -6
```

```
    0 | 0 1 0  (+2)     100
+   1 | 1 0 0  ( -4)   -010
-------------------    ----
    1 | 0 1 0           010
```

```
    1 | 0 1 0  ( -2)    100
+   0 | 1 0 0  (+4)    -010
-------------------    ----
    0 | 0 1 0           010
```

Bottom line: Basic mathematics are too complex in Sign/Magnitude

Idea: Pick negatives so that addition works

- Let -1 = 0 - (+1):

```
    0 0 0 0  ( 0)
  - 0 0 0 1  (+1)
  ---------
    1 1 1 1
```

- Does addition work?

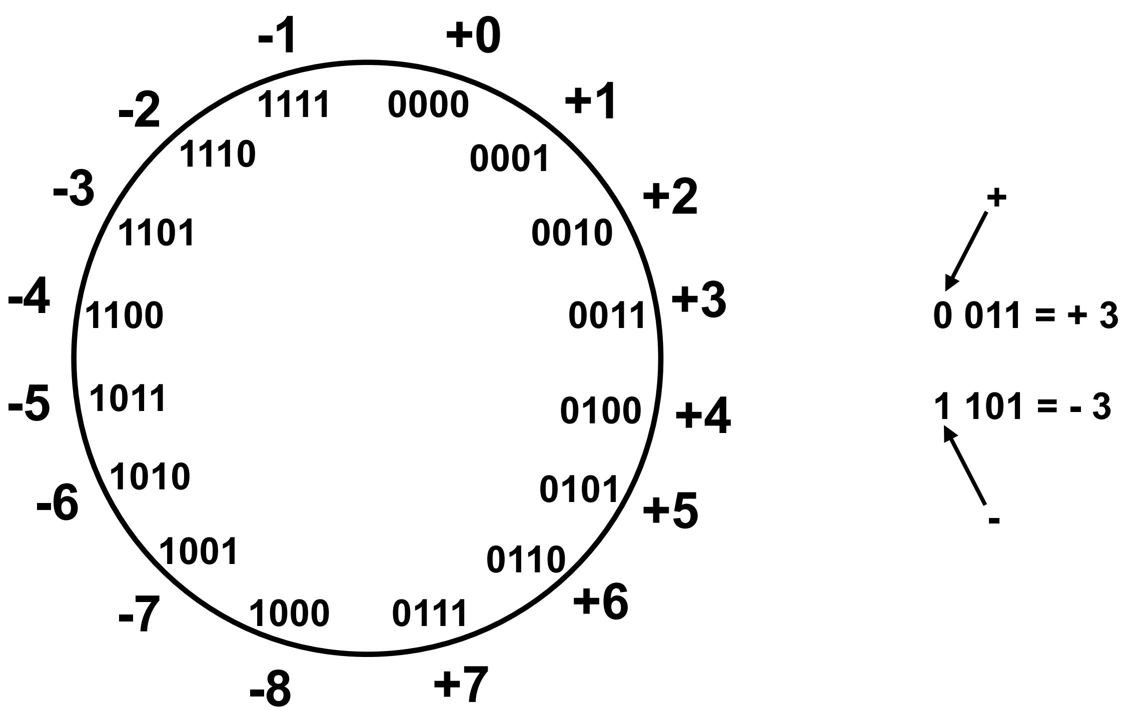
```
    0 0 1 0  (+2)
  + 1 1 1 1  ( -1)
  ---------
    0 0 0 1
```

- Result: Two's Complement Numbers

FOR $0 \le b \le 2^n - 1$ $-b$ IS REPRESENTED BY $2^n - b$

Two's Complement

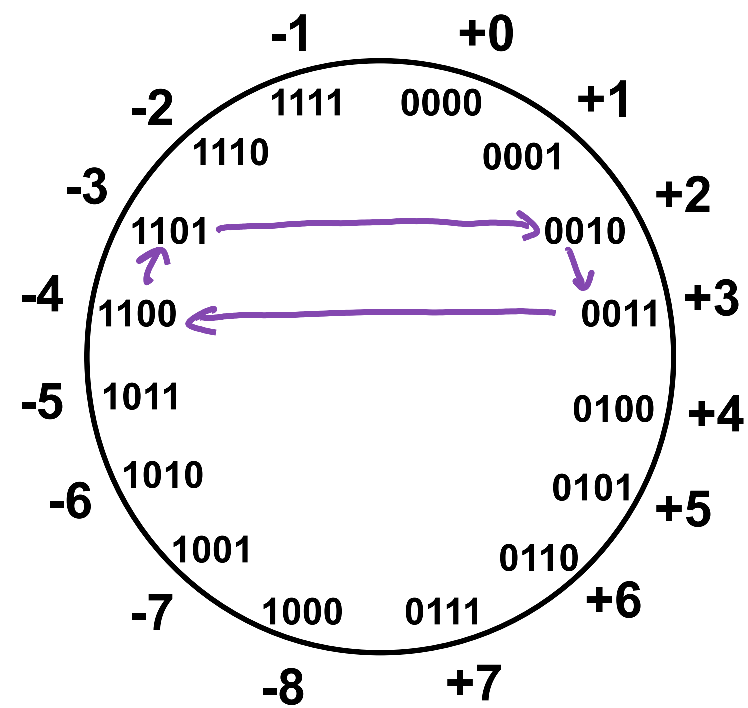
- Only one representation for 0
- One more negative number than positive number
- Fixed width format for both pos. & neg. numbers



Negating in Two’s Complement

- Flip bits & Add 1
- Negate $(0010)_2$ (+2)

-2 = -0010 = 1101 + 1 = 1110



- Negate $(1110)_2$ (-2)

-1110 = 0001+1 = 0010

Addition in Two's Complement

$$\begin{array}{r} 0\ 0\ 1\ 0 \quad (+2) \\ +\ 0\ 1\ 0\ 0 \quad (+4) \\ \hline 0\ 1\ 1\ 0 \quad +6 \end{array}$$

$$\begin{array}{r} \times\ \ 1\ 1\ 1\ 0 \quad (-2) \\ +\ 1\ 1\ 0\ 0 \quad (-4) \\ \hline 1\ 0\ 1\ 0 \end{array}$$

$-(-1010) = -(0101+1)$

$= -0110 = -6$

$$\begin{array}{r} 0\ 0\ 1\ 0 \quad (+2) \\ +\ 1\ 1\ 0\ 0 \quad (-4) \\ \hline 1\ 1\ 1\ 0 \end{array}$$

$-(-1110) = -(0001+1)$

$= -(0010)$

$= -2$

$$\begin{array}{r} 1\ 1\ 1\ 0 \quad (-2) \\ +\ 0\ 1\ 0\ 0 \quad (+4) \\ \hline 0\ 0\ 1\ 0 \end{array}$$

Subtraction in Two's Complement

- A - B = A + (-B) = A + $\overline{B}$ + 1

- 0010 - 0110

  0010 + (-0110) = 0010 + (1001 + 1) = 0010 + 1010

  $$\begin{array}{r} 0010 \\ +\,1010 \\ \hline 1100 \end{array}$$

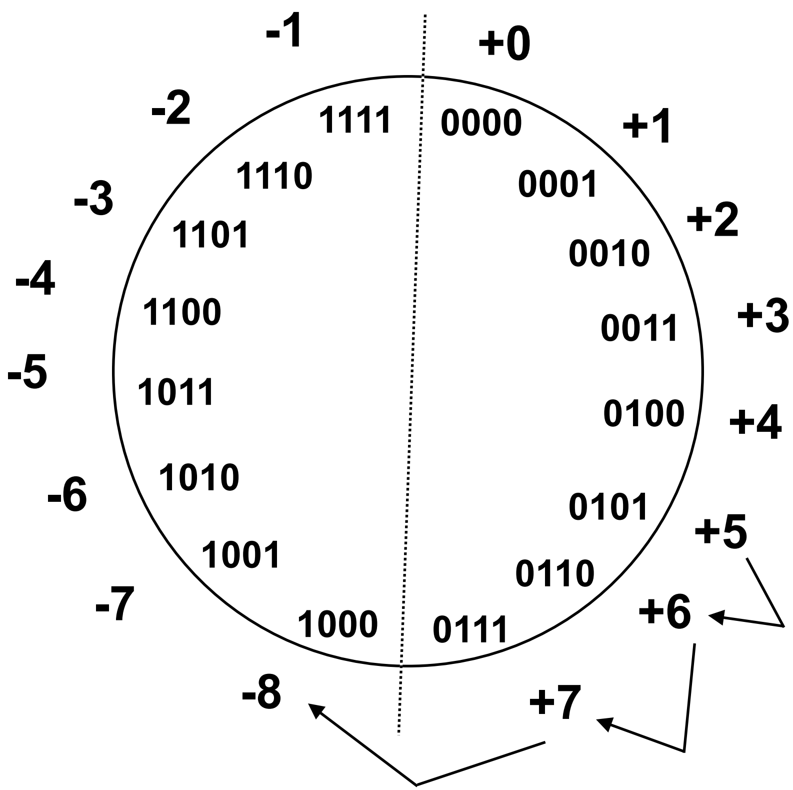
- 1011 - 1001

- 1011 - 0001

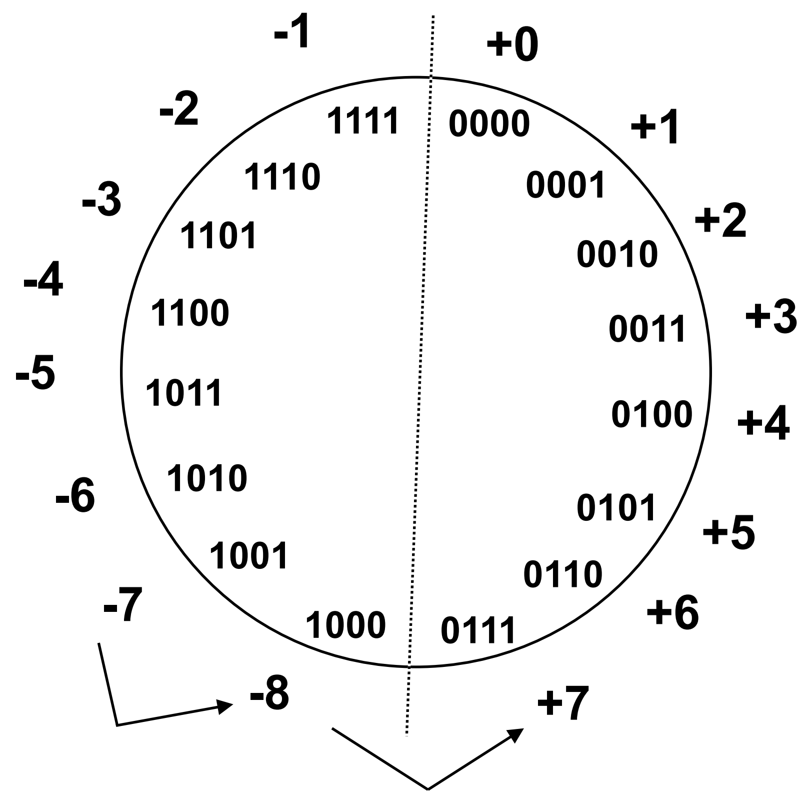
Overflows in Two's Complement

Add two positive numbers but get a negative number

or two negative numbers but get a positive number



5 + 3 = -8



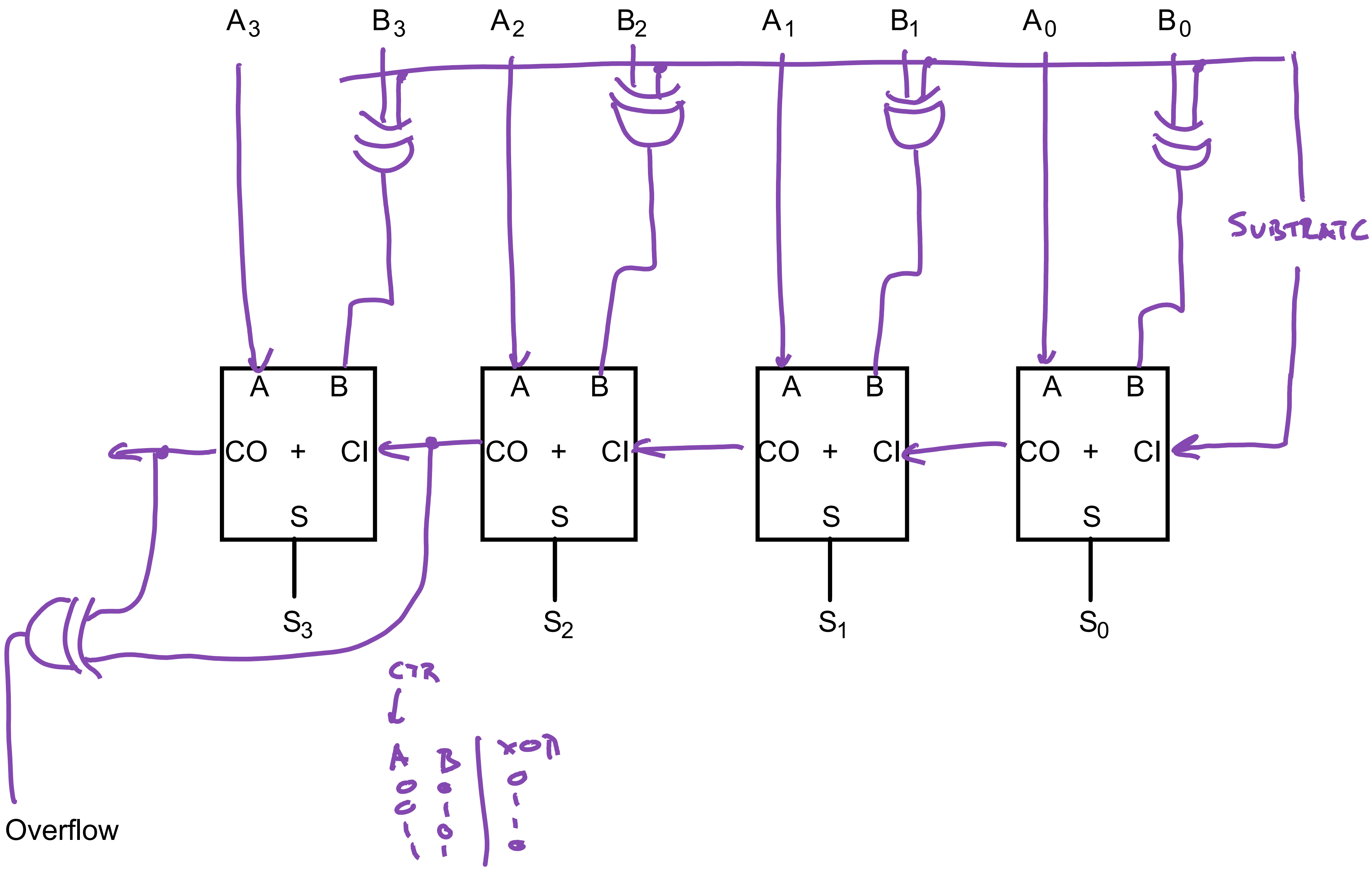
-7 - 2 = +7

Overflow Detection in Two's Complement

5	0 1 0 1			-7	1 0 0 1
3	0 0 1 1			-2	1 1 1 0
-8				7	
Overflow				**Overflow**	
5	0 1 0 1			-3	1 1 0 1
2	0 0 1 0			-5	1 0 1 1
7				-8	
No overflow				**No overflow**	

OVERFLOW = $C_{IN} \oplus C_{OUT}$ OF HIGHEST ORDER BIT

Adder/Subtractor



$$A - B = A + (-B) = A + \overline{B} + 1$$

Converting Decimal to Two's Complement

- Convert absolute value to unsigned binary, then fixed width, then negate if necessary

- Convert $(-9)_{10}$ to 6-bit Two's Complement

- Convert $(9)_{10}$ to 6-bit Two's Complement

Converting Two's Complement to Decimal

- If Positive, convert as normal;
  If Negative, negate then convert.

- Convert $(11010)_2$ to Decimal

- Convert $(01101)_2$ to Decimal

Sign Extension

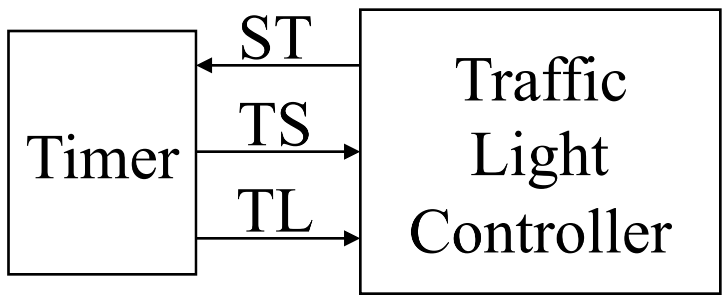
- To convert from N-bit to M-bit Two's Complement (N<M), simply duplicate sign bit:

- Convert $(0010)_2$ to 8-bit Two's Complement

- Convert $(1011)_2$ to 8-bit Two's Complement

Solving Complex Problems

- Many problems too complex to build as one system
  - Replace with communicating sub-circuits



- Design process:
  - Understand the problem
  - Break problem into subsystems, identifying connections
  - Design individual subsystems.

Complex Problem Example

- Design a digital clock, which can
  - Display the seconds, minutes and hours
  - Have three inputs
    - Increment hour
    - Increment minute
    - Reset seconds

Complex Problem Example (cont.)

Complex Problem Example (cont.)

Complex Problem Example (cont.)

Complex Problem Example

- Break into pieces:
  - Display, counter
  - Displayer becomes 6x 7-segment displays
  - Counter becomes three counters
    - Minutes, hours, seconds.
  - Need reset on seconds, override on increment on hours, minutes.
  - Break counters into digits, except hours.
  - Communicate increment to higher