Number Representation and Binary Arithmetic

Unsigned Addition

Binary Addition

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 and carry 1 to the next column

Examples:

$$\begin{array}{rr} & \texttt{0101} \\ + & \texttt{0010} \\ \hline & \texttt{0111} \end{array} \quad \begin{array}{l} (5_{10}) \\ (2_{10}) \\ (7_{10}) \end{array} \qquad\qquad \begin{array}{rr} & {\scriptstyle 1\,1\,1\,} \\ & \texttt{0101} \\ + & \texttt{0011} \\ \hline & \texttt{1000} \end{array} \quad \begin{array}{l} \xleftarrow{\qquad\qquad} \text{Carries} \\ (5_{10}) \\ (3_{10}) \\ (8_{10}) \end{array}$$

Binary Addition w/Overflow

Add 45_{10} and 44_{10} in binary

$$\begin{array}{rll}
111 & \longleftarrow & \text{Carries}\\
101101 & (45_{10}) & \\
+\ 101100 & (44_{10}) & \\
\hline
1011001 & (89_{10}) &
\end{array}$$

If the operands are *unsigned*, you can use the final carry-out as the MSB of the result.

Adding 2 *k* bit numbers ⇔ *k+1* bit result

More Binary Addition w/Overflow

$$\begin{array}{r l}
{\scriptstyle 1\,1\,1\,1} & \\
1111 & (15_{10}) \\
+\ 0001 & (1_{10}) \\
\hline
0000 & (0_{10})
\end{array}$$

If you don't want a 5-bit result, just keep the lower 4 bits.

This is insufficient to hold the result (16).

It *rolls over* back to 0.

Signed Numbers

Negative Binary Numbers

- Several ways of representing negative numbers
- Most obvious is to add a sign (+ or -) to the binary integer

Sign-Magnitude

0 is +
1 is -

Number	Sign	Magnitude	Full Number
+1	0	0001	00001
-1	1	0001	10001
+5	0	0101	00101
-5	1	0101	10101
+0	0	0000	00000
-0	1	0000	10000

Easy to interpret number value

Sign-Magnitude Examples

$$\begin{array}{rr} & {\scriptstyle 0\ 0\ 0\ \ } \\ & 0\ 0011 \\ + & 0\ 0100 \\ \hline & 0\ 0111 \end{array} \quad \begin{array}{l} \\ (+3_{10}) \\ (+4_{10}) \\ (+7_{10}) \end{array}$$

$$\begin{array}{rr} & {\scriptstyle 0\ 0\ 0\ \ } \\ & 1\ 0011 \\ + & 1\ 0100 \\ \hline & 1\ 0111 \end{array} \quad \begin{array}{l} \\ (-3_{10}) \\ (-4_{10}) \\ (-7_{10}) \end{array}$$

Signs are the same

Just add the magnitudes

Another Sign-Magnitude Example

Signs are different - determine which is larger magnitude

$$\begin{array}{rr} 0\ 0101 & (+5_{10}) \\ \underline{+\ 1\ 0011} & (-3_{10}) \end{array}$$

Put larger magnitude number on top

Subtract

Result has sign of larger magnitude number…

$$\begin{array}{rr} 0\ 0101 & (+5_{10}) \\ \underline{-\ 1\ 0011} & (-3_{10}) \\ 0\ 0010 & (+2_{10}) \end{array}$$

Yet Another Sign-Magnitude Example

Signs are different - determine which is larger magnitude

$$\begin{array}{rr} 0\ 0010 & (+2_{10}) \\ +\ 1\ 0101 & (-5_{10}) \\ \hline \end{array}$$

Put larger magnitude number on top

Subtract

$$\begin{array}{rr} 1\ 0101 & (-5_{10}) \\ -\ 0\ 0010 & (+2_{10}) \\ \hline 1\ 0011 & (-3_{10}) \end{array}$$

Result has sign of larger magnitude number…

Sign-Magnitude

- Addition requires two separate operations
  - addition
  - subtraction
- Several decisions:
  - Signs same or different?
  - Which operand is larger?
  - What is sign of final result?
- Two zeroes (+0, -0)

Sign-Magnitude

- Advantages
  - Easy to understand
- Disadvantages
  - Two different 0s
  - Hard to implement in logic

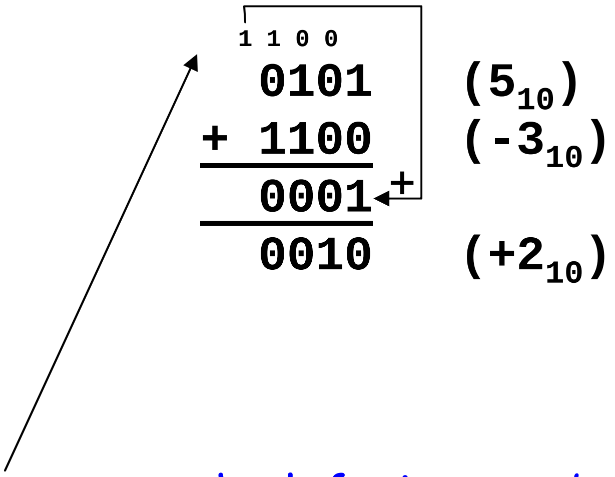
One’s Complement

- Positive numbers are the same as sign-magnitude
- -N is represented as the complement of N
  - -N = N’

Number	1's Complement
+1	0001
-1	1110
+5	0101
-5	1010
+0	0000
-0	1111

One's Complement Examples

```
   000101     (5₁₀)
+  010100     (20₁₀)
  -------
   011001     (25₁₀)
```



```
 1 1 0 0
    0101      (5₁₀)
+   1100      (-3₁₀)
  ------
    0001  +
  ------
    0010      (+2₁₀)
```

If there is a carry out on the left, it must be *wrapped around* and added back in on the right

One's Complement

- Addition complicated by end around carry
- No decisions (like SM)
- Still two zeroes (+0, -0)

One's Complement

- Advantages
  - Easy to generate –N
  - Only one addition process

- Disadvantages
  - End around carry
  - Two different 0s

Two's Complement

- Treat positional digits differently

$$0111_{2C} = -0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7_{10}$$

$$1111_{2C} = -1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = -1_{10}$$

Most significant bit (MSB) given negative weight…
Other bits same as in unsigned

Two's Complement

Number	2's Complement
+1	0001
-1	1111
+5	0101
-5	1011
+0	0000
-0	none

Sign-Extension in 2's Complement

- To make a *k*-bit number wider
  - replicate sign bit

$110_{2C} = -1 \text{ x } 2^2 + 1 \text{ x } 2^1 = -2_{10}$

$1110_{2C} = -1 \text{ x } 2^3 + 1 \text{ x } 2^2 + 1 \text{ x } 2^1 = -2_{10}$

$11110_{2C} = -1 \text{ x } 2^4 + 1 \text{ x } 2^3 + 1 \text{ x } 2^2 + 1 \text{ x } 2^1 = -2_{10}$

More Sign-Extension

$$010_{2C} = 1 \text{ x } 2^1 = 2_{10}$$

$$0010_{2C} = 1 \text{ x } 2^1 = 2_{10}$$

$$00010_{2C} = 1 \text{ x } 2^1 = 2_{10}$$

Works for both positive and negative numbers

Negating a 2's Complement Number

- Invert all the bits
- Add 1

$$+2_{10} = 0010_{2C} \xrightarrow{\text{invert}} 1101 + 1 = 1110_{2C} = -2_{10}$$

$$-2_{10} = 1110_{2C} \xrightarrow{\text{invert}} 0001 + 1 = 0010_{2C} = +2_{10}$$

Two's Complement Addition

$$\begin{array}{rl}
\scriptstyle 0\ 0\ 1\ \ & \\
0101 & (+5_{10}) \\
+\ \underline{0001} & (+1_{10}) \\
0110 & (+6_{10})
\end{array}$$

$$\begin{array}{rl}
\scriptstyle 1\ 1\ 1\ \ & \\
1011 & (-5_{10}) \\
+\ \underline{1111} & (-1_{10}) \\
1010 & (-6_{10})
\end{array}$$

$$\begin{array}{rl}
\scriptstyle 1\ 1\ 1\ \ & \\
0110 & (+6_{10}) \\
+\ \underline{1111} & (-1_{10}) \\
0101 & (+5_{10})
\end{array}$$

Operation is same as for unsigned
Same rules, same procedure

Interpretation of operands and results are different

Two's Complement

- Addition always the same
- Only 1 zero

- Representation of choice…

Two's Complement Overflow

- All representations can overflow
  - Focus on 2's complement here

$$\begin{array}{rl} \scriptstyle 1\ 0\ 0\ 0\ \ & \\ 1011 & (-5_{10}) \\ +\ 1100 & (-4_{10}) \\ \hline 0111 & (+7_{10})\ ?? \end{array}$$

The correct answer (-9) cannot be represented by 4 bits
The correct answer is 10111

Overflow

- Can you use leftmost carry-out as new MSB?

$$\begin{array}{r}
\scriptstyle 1\ 0\ 0\ 0\\
1011\\
+\ \ 1100\\ \hline
10111
\end{array}
\qquad
\begin{array}{l}
\\
(-5_{10})\\
(-4_{10})\\
(-9_{10})
\end{array}$$

Works here…

$$\begin{array}{r}
\scriptstyle 0\ 0\ 0\ 0\\
1000\\
+\ \ 0111\\ \hline
01111
\end{array}
\qquad
\begin{array}{l}
\\
(-8_{10})\\
(+7_{10})\\
(+15_{10})\ ??
\end{array}$$

Does NOT work here…

- The answer is no, in general

Handling Overflow

1. Sign-extend the operands

2. Do the addition

```
  0 0 0 0
  11000    (-8₁₀)
 +00111    (+7₁₀)
 ------
  11111    (-1₁₀)
```

When Can Overflow Occur?

- Adding two positive numbers - yes
- Adding two negative numbers – yes
- Adding a positive to a negative – no
- Adding a negative to a positive - no

General Handling of Overflow

- Adding two *k* bit numbers gives *k+1* bit result

  Two options include:

1. Sign extend and keep entire *k+1* bit result
2. Throw away new bit generated (keep only *k* bits)
   1. Detect overflow and signal an error
   2. Ignore the overflow (this is what most computers do)

- The choice is up to the designer

Detecting Overflow

- Overflow occurs if:
  - Adding two positive numbers gives what looks like a negative result
  - Adding two negative numbers gives what looks like a positive result

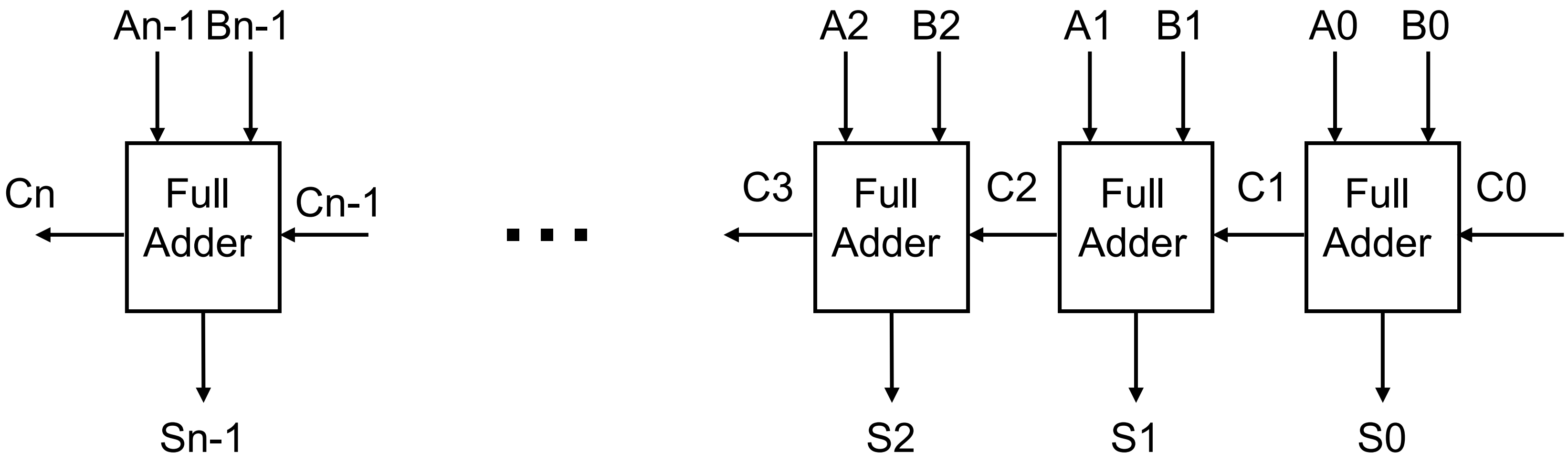
- Overflow will never occur when adding positive to negative

There are other ways of detecting overflow vs. what is suggested here…

Arithmetic Circuits

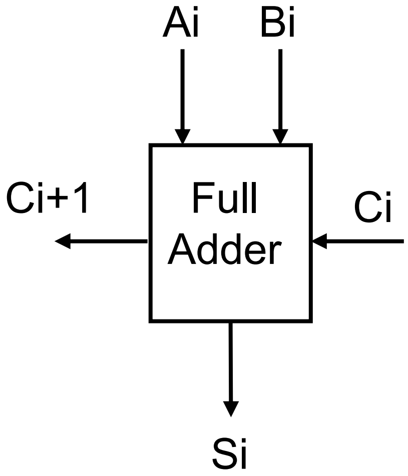
Binary Adder

- An example of an iterative network

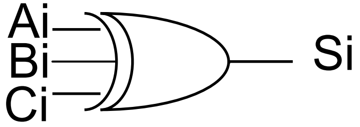


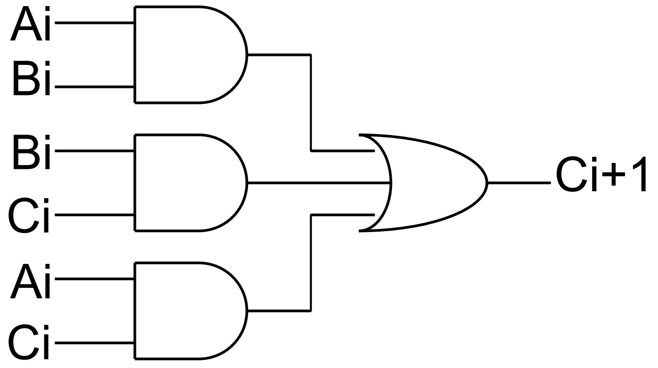
This type of adder is also called a ripple-carry adder because the carry ripples through from cell to cell

Full Adder Derivation

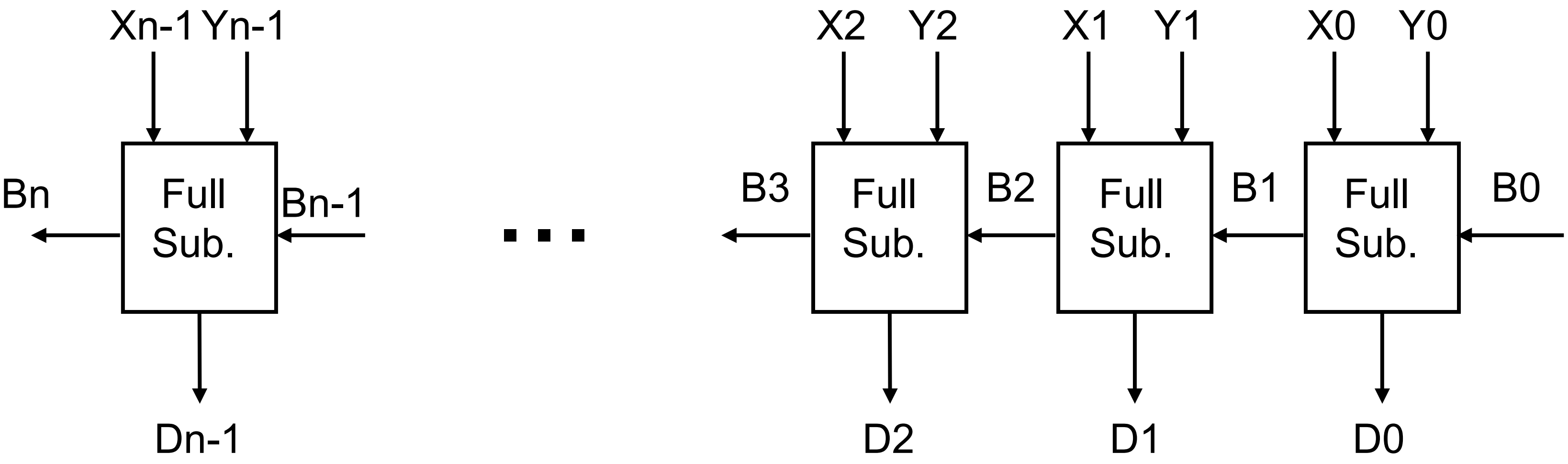


Ai	Bi	Ci	Ci+1	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





Binary Subtracter

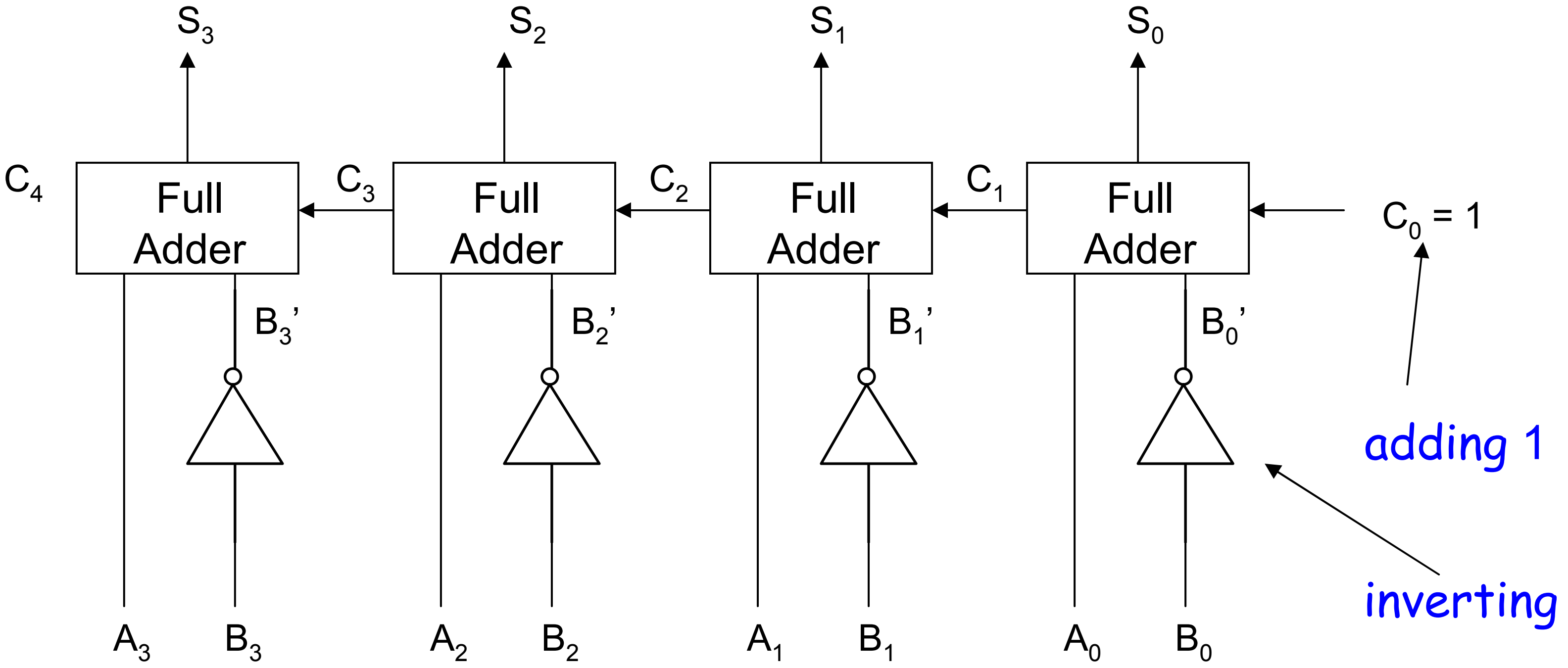


D(ifference) = **X - Y**

B is the borrow signal

Equations generated similarly to full adder

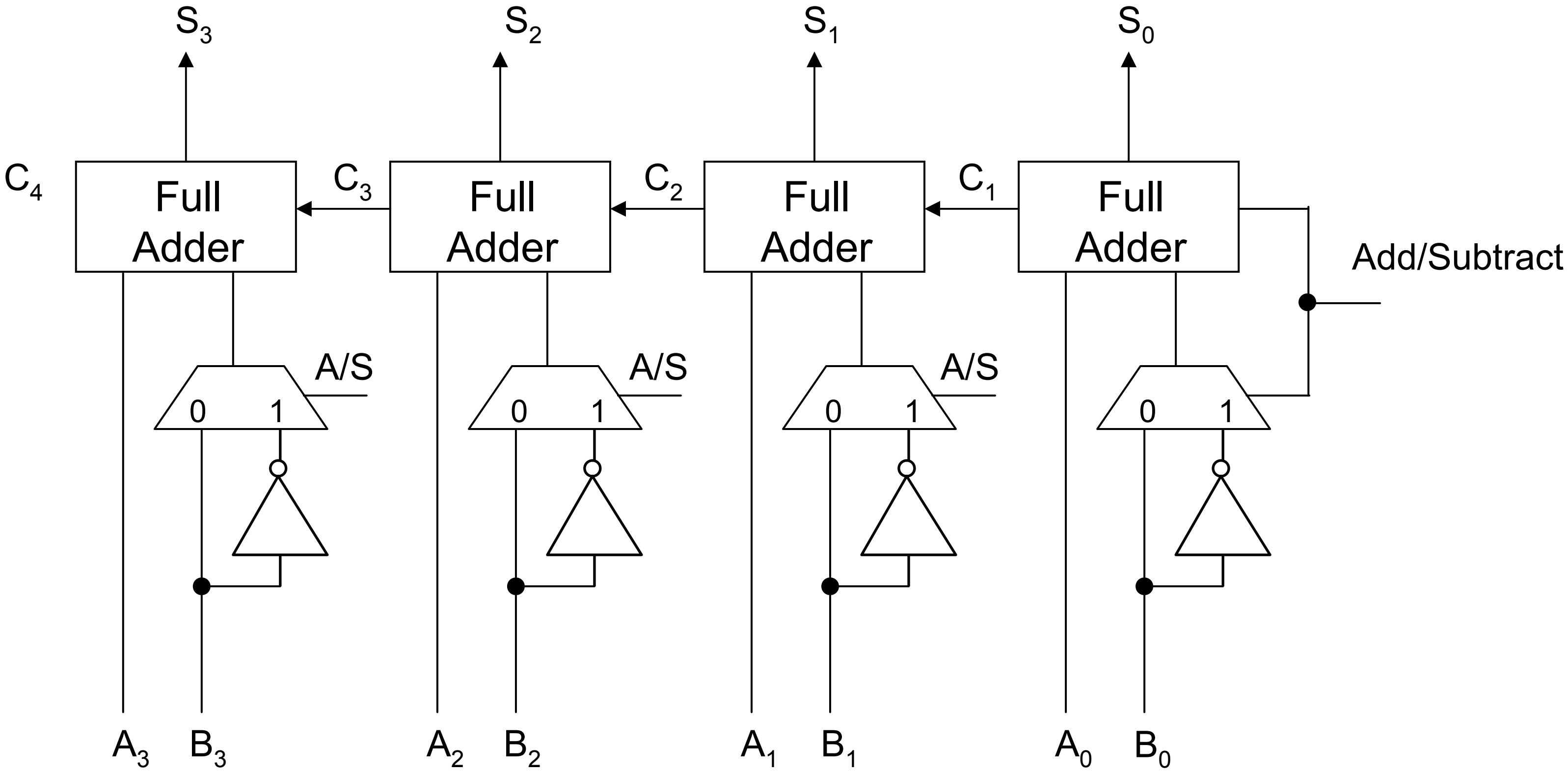
Another Way to Make a Subtracter (2's complement)



S = A + (-B)

-B is generated by inverting and adding 1

Binary Adder/Subtracter (2's complement)



Binary Arithmetic Comparison

	Sign Magnitude	One's Complement	Two's Complement
Negative Number	Easiest to Understand Simple to Compute	Easy to Compute	Hardest to Compute
Zeroes	2 Zeroes	2 Zeroes	1 Zero
Largest Number	Same number of + and - Numbers	Same number of + and - Numbers	One Extra Negative Number
Logic Required	Requires Adder and Subtracter	Only Adder Required	Only Adder Required
	Extra Logic to Identify Larger Operand, Compute Sign, etc.	Carry Wraps Around	-
Overflow Detection	Overflow: Carry from High Order Adder Bits	Overflow: Sign of Both Operands is the Same and Sign of Sum is Different	Overflow: Sign of Both Operands is the Same and Sign of Sum is Different

Summary

- Understand the three main representation of binary integers
  - Sign-Magnitude
  - One's Complement
  - Two's Complement
- Design binary adders and subtracters for each representation