

Number Representation and Binary Arithmetic

Unsigned Addition

Binary Addition

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ and carry } 1 \text{ to the next column}$$

Examples:

$$\begin{array}{r} 0101 \\ + 0010 \\ \hline 0111 \end{array} \quad \begin{array}{l} (5_{10}) \\ (2_{10}) \\ (7_{10}) \end{array}$$

$$\begin{array}{r} \overset{1 \ 1 \ 1}{0101} \\ + 0011 \\ \hline 1000 \end{array} \quad \begin{array}{l} (5_{10}) \\ (3_{10}) \\ (8_{10}) \end{array} \quad \begin{array}{c} \longleftarrow \text{Carries} \end{array}$$

Binary Addition w/Overflow

Add 6-bit numbers 45_{10} and 44_{10} in binary

$$\begin{array}{r} \begin{array}{ccccccc} & & 1 & & 1 & 1 & \\ & & \leftarrow & & & & \end{array} \\ \begin{array}{r} 101101 \\ + 101100 \\ \hline 1011001 \end{array} \end{array} \quad \begin{array}{l} (45_{10}) \\ (44_{10}) \\ (89_{10}) \end{array} \quad \begin{array}{l} \text{Carries} \end{array}$$

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

Adding 2 k -bit numbers $\Leftrightarrow k+1$ bit result

More Binary Addition w/Overflow

$$\begin{array}{r} \\ 1 1 1 1 \\ + 0 0 0 1 \\ \hline 0 0 0 0 \end{array} \quad \begin{array}{l} (15_{10}) \\ (1_{10}) \\ (0_{10}) \end{array}$$

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

Here, 4 bits 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

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

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

Signs are the same, just add the magnitudes

Another Sign-Magnitude Example

Signs are different:
determine which has
larger magnitude

$$\begin{array}{r} 0 \ 0101 \quad (+5_{10}) \\ + \ 1 \ 0011 \quad (-3_{10}) \\ \hline \end{array}$$

Put larger magnitude
number on top

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

Subtract

Result has sign of larger
magnitude number...

Yet Another Sign-Magnitude Example

Signs are different:
determine which has
larger magnitude

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

Put larger magnitude
number on top

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

Subtract

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

$$\begin{array}{r}
 000101 \\
 + 010100 \\
 \hline
 011001
 \end{array}
 \begin{array}{l}
 (5_{10}) \\
 (20_{10}) \\
 (25_{10})
 \end{array}$$

$$\begin{array}{r}
 \boxed{1100} \\
 0101 \\
 + 1100 \\
 \hline
 0001 \leftarrow + \\
 \hline
 0010
 \end{array}
 \begin{array}{l}
 (5_{10}) \\
 (-3_{10}) \\
 (+2_{10})
 \end{array}$$

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 (unlike sign-magnitude)
- 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

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

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

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

More Sign-Extension

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

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

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

Works for both positive and negative numbers

Negating a 2's Complement Number

1. Invert all the bits
2. Add 1

$$+2_{10} = 0010_{2C} \rightarrow 1101 + 1 = 1110_{2C} = -2_{10}$$

$$-2_{10} = 1110_{2C} \rightarrow 0001 + 1 = 0010_{2C} = +2_{10}$$

Two's Complement Addition

$$\begin{array}{r} 001 \\ 0101 \quad (+5_{10}) \\ + 0001 \quad (+1_{10}) \\ \hline 0110 \quad (+6_{10}) \end{array}$$

$$\begin{array}{r} 111 \\ 1011 \quad (-5_{10}) \\ + 1111 \quad (-1_{10}) \\ \hline 1010 \quad (-6_{10}) \end{array}$$

$$\begin{array}{r} 110 \\ 0110 \quad (+6_{10}) \\ + 1111 \quad (-1_{10}) \\ \hline 0101 \quad (+5_{10}) \end{array}$$

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
- Negation somewhat complicated
- The representation of choice

Two's Complement Overflow

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

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

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?

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

Works here...

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

Does NOT work here...

- The answer is no, in general

Handling Overflow

1. Sign-extend the operands
2. Do the addition

$$\begin{array}{rcl} & 0 & 0 & 0 & 0 \\ & \textcolor{red}{1} & 1 & 0 & 0 & 0 & (-8_{10}) \\ + & \textcolor{red}{0} & 0 & 1 & 1 & 1 & (+7_{10}) \\ \hline & 1 & 1 & 1 & 1 & 1 & (-1_{10}) \end{array}$$

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 for handling:
 1. Keep entire $k+1$ bit sum
 2. Keep only k bits (throw away generated bit)
 - i. Detect overflow and signal an error
 - ii. Ignore the overflow
- The choice is up to the designer

Detecting Overflow

- Overflow occurs if:
 - Adding two positive numbers gives a negative result
 - Adding two negative numbers gives a positive result
- Overflow will never occur when adding a positive and a negative number together

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

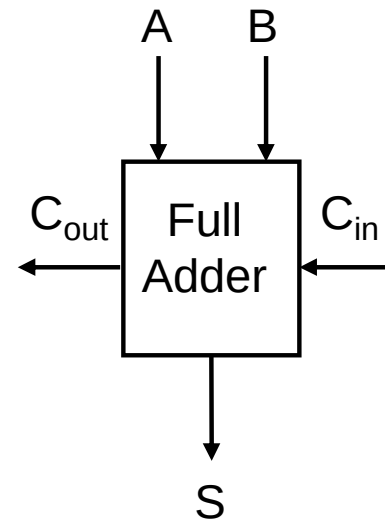
Binary Arithmetic Comparison

	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 Subtractor	Only Adder Required	Only Adder Required
	Extra Logic to Identify Larger Operand, Compute Sign, etc.	Carry Wraps Around	-
Overflow Detection	Carry from High Order Adder Bits	Sign of Both Operands is the Same and Sign of Sum is Different	Sign of Both Operands is the Same and Sign of Sum is Different

Arithmetic Circuits

Binary Adder

$$\begin{array}{rcccc} 0 & 1 & 1 & 0 \\ 100 & 10 & & \\ +00 & 111 & & \\ \hline 110 & 01 & & \end{array}$$

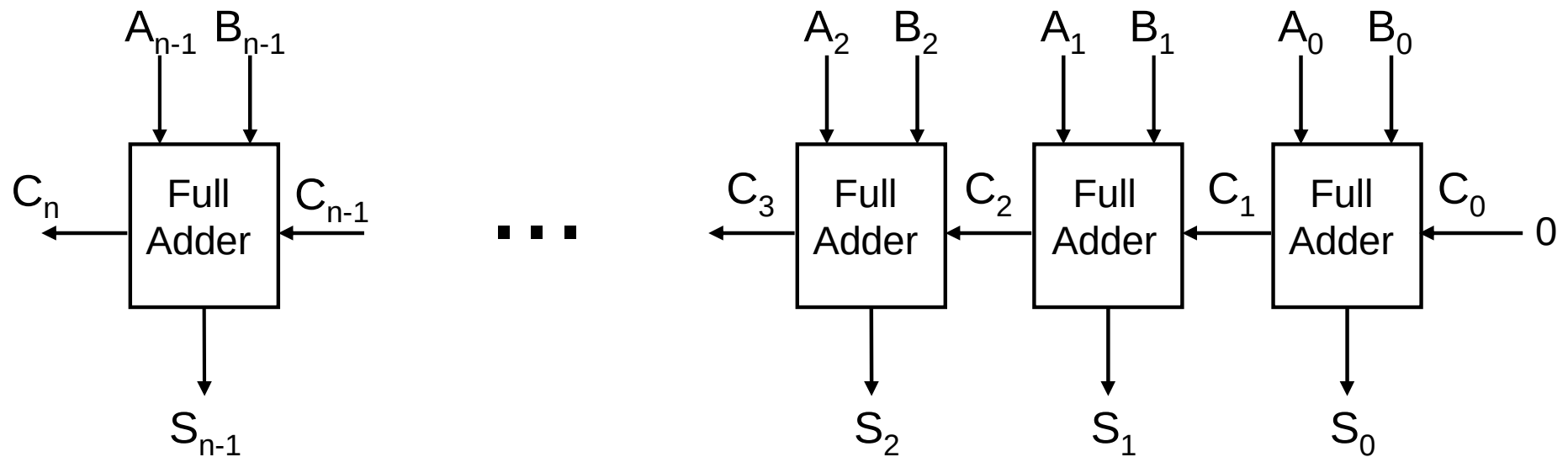


Inputs: Operand A
Operand B
Carry In

Outputs: Sum
Carry Out

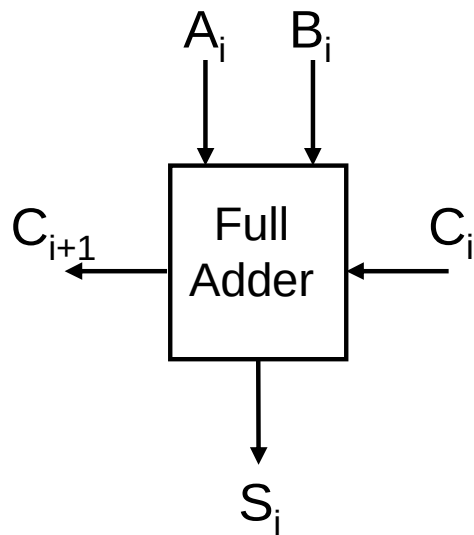
Binary Adder

- An example of an iterative network



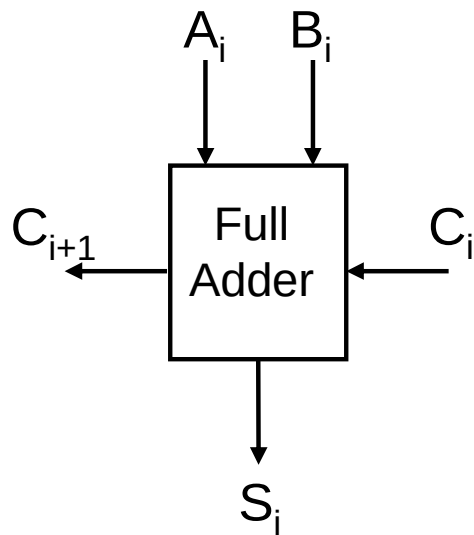
This type of adder is often called a **ripple-carry adder** because the carry ripples through from cell to cell

Full Adder Derivation



A_i	B_i	C_i	C_{i+1}	S_i
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Full Adder Derivation



A_i	B_i	C_i	C_{i+1}	S_i
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

Full Adder Derivation

A	B	C_{in}	C_{out}	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

BC_{in}	00	01	11	10
A				
0				
1				

C_{out}

BC_{in}	00	01	11	10
A				
0				
1				

S

Full Adder Derivation

A	B	C _{in}	C _{out}	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

A \ BC _{in}	BC _{in}			
	00	01	11	10
0	0	0	1	0
1	0	1	1	1

C_{out}

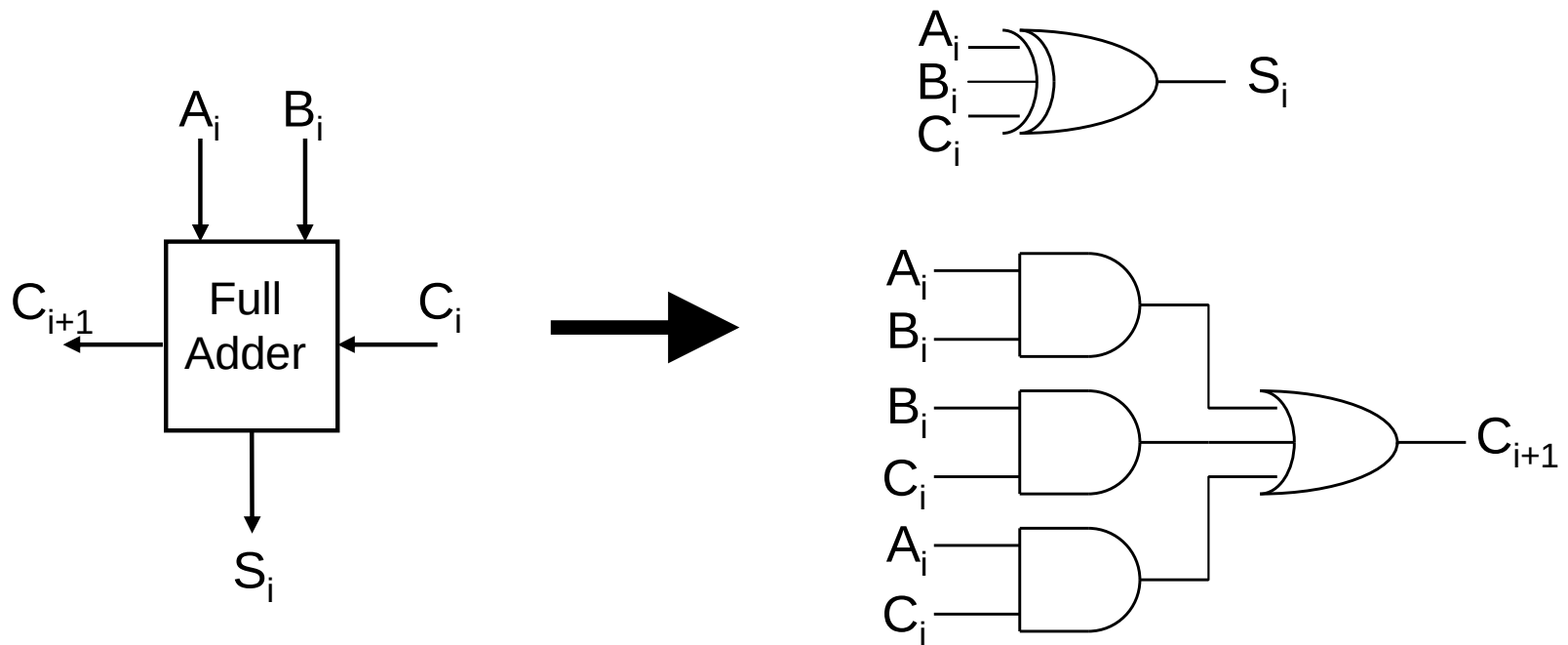
$$C_{out} = AC_{in} + AB + BC_{in}$$

A \ BC _{in}	BC _{in}			
	00	01	11	10
0	0	1	0	1
1	1	0	1	0

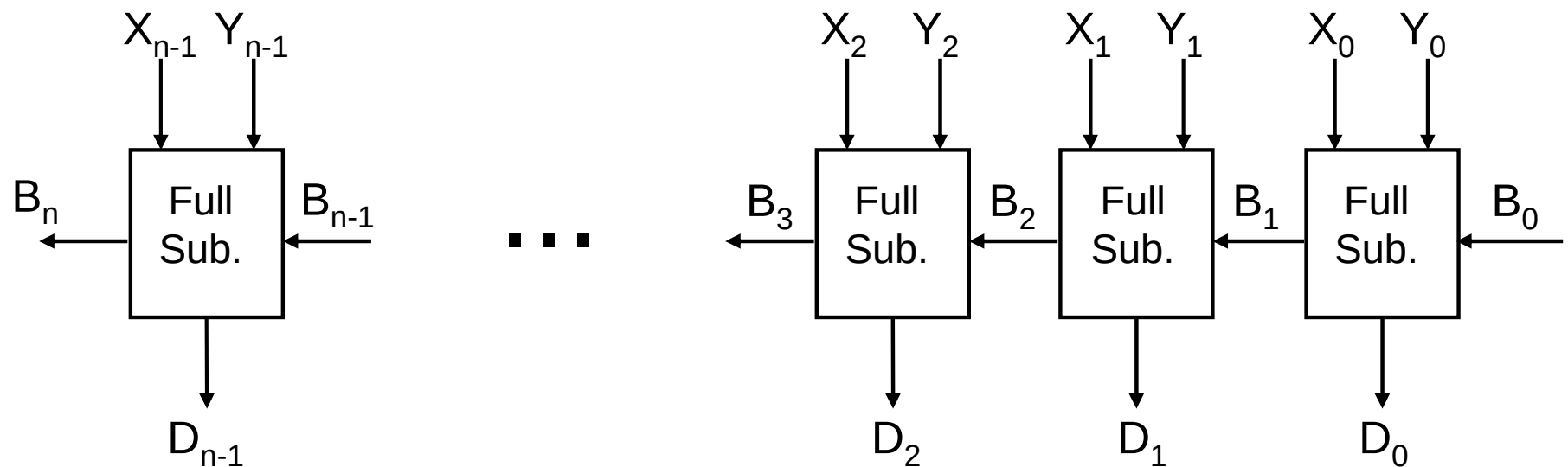
S

$$S = AB'C_{in}' + A'B'C_{in} + ABC + A'BC_{in}'$$

Full Adder Derivation



Binary Subtractor



D is the difference ($X-Y$)

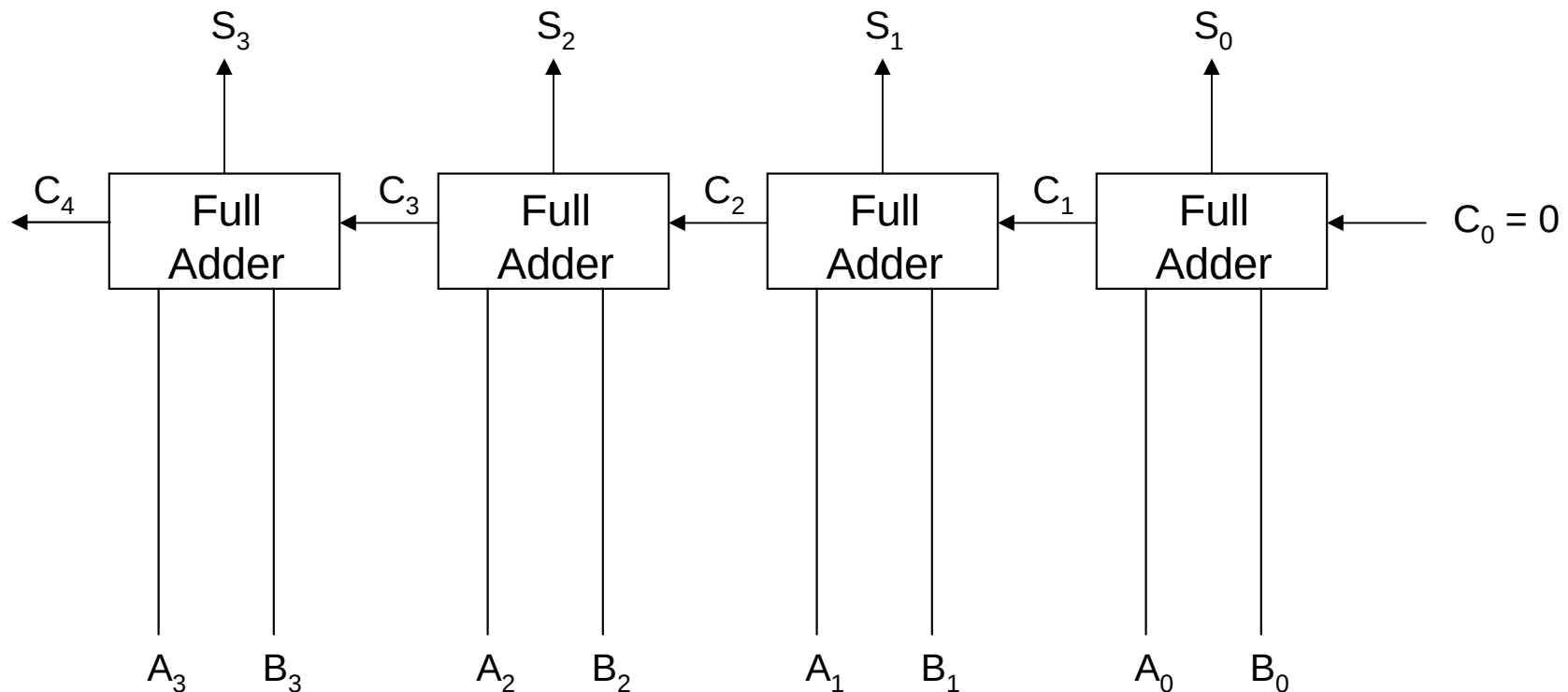
B is the borrow signal

Equations generated
similarly to full adder

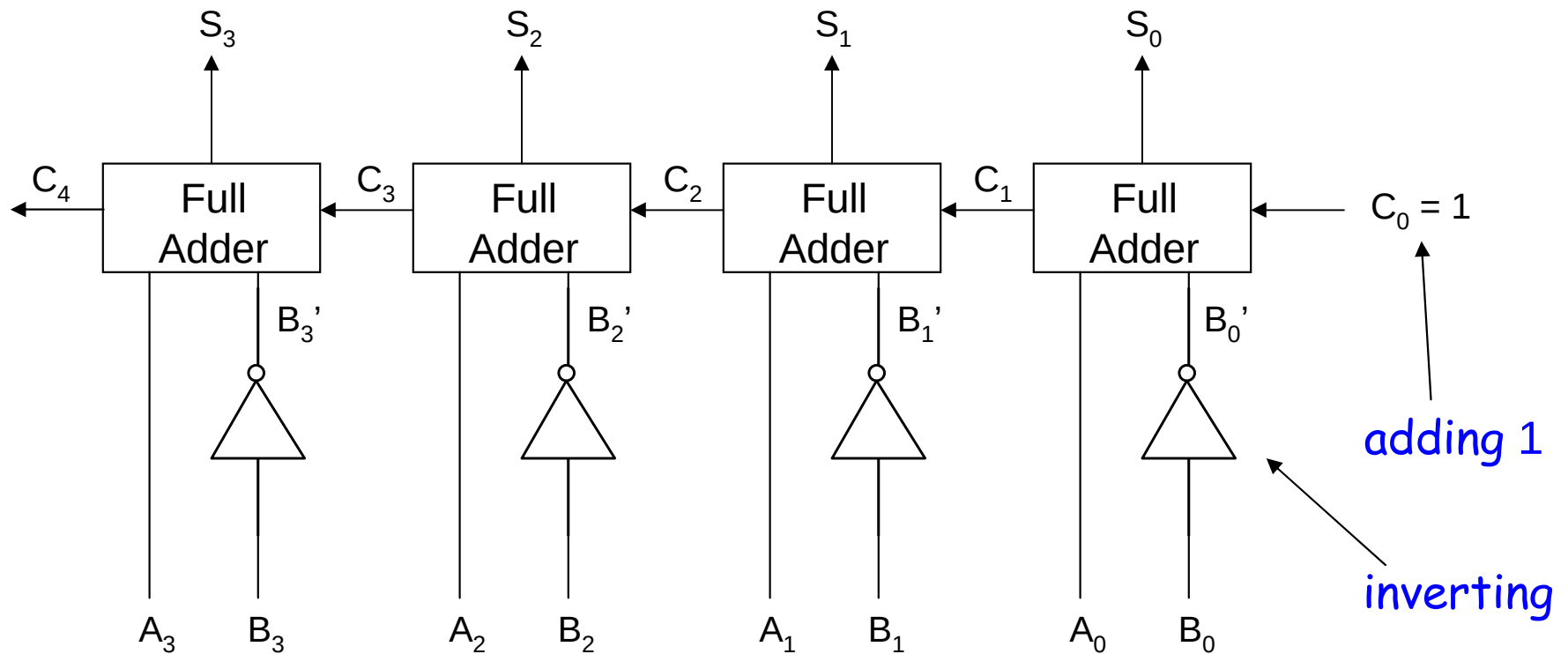
Binary Subtractor Using Adder

- Recall that $A - B$ is equivalent to $A + (-B)$
- We can negate an operand by inverting the bits and adding 1
- A subtracter can be implemented by a negater followed by an adder

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



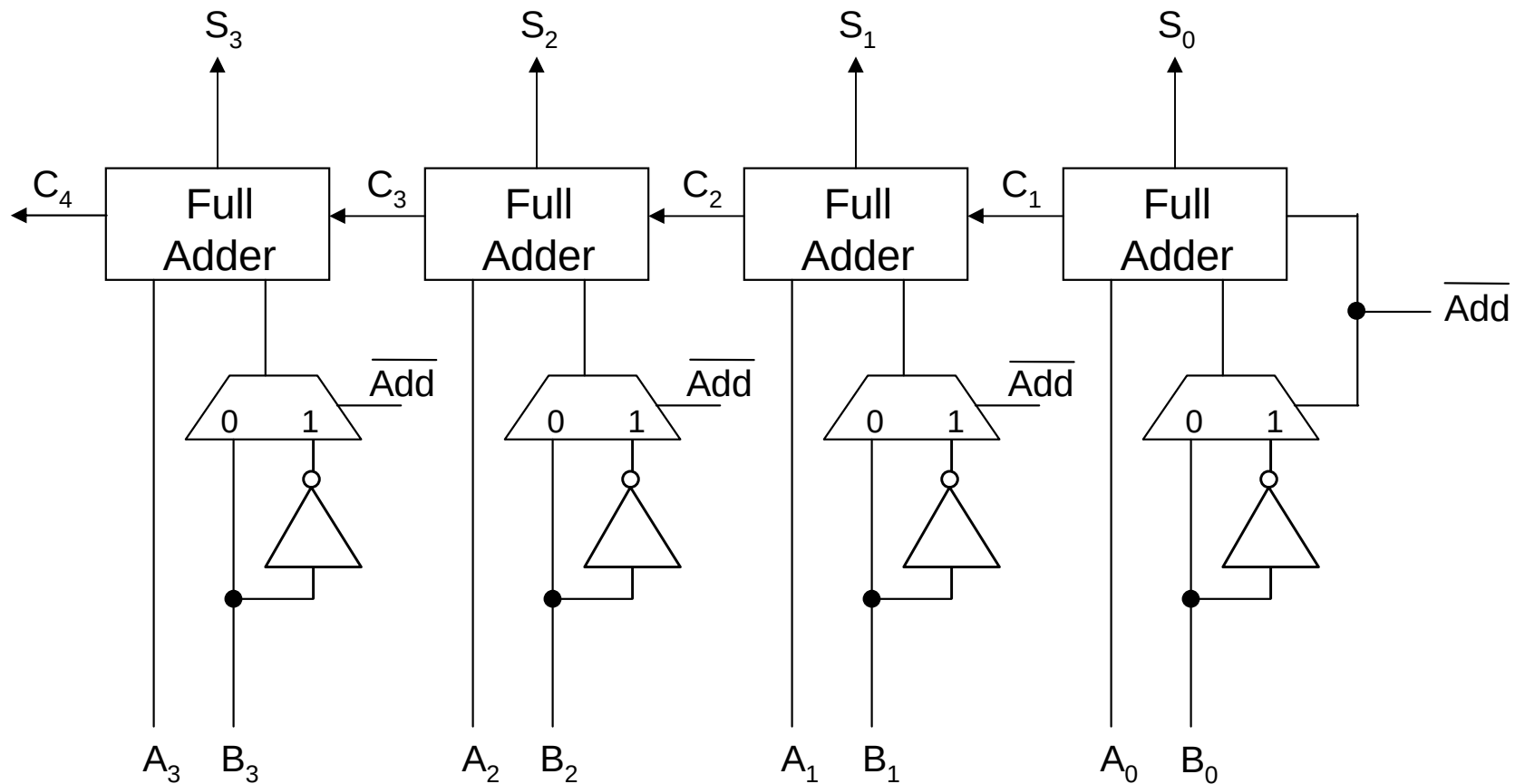
Another Way to Make a Subtractor (2's complement)



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

-B is generated by inverting and adding 1

Binary Adder/Subtractor (2's complement)



The bar over the Add input means that the circuit will add when $\text{Add} = 0$. The notation $\text{Add}\#$ is also common.

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