Numbers and Arithmetic

Prof. Hakim Weatherspoon

CS 3410, Spring 2015

Computer Science

Cornell University

See: P&H Chapter 2.4, 3.2, B.2, B.5, B.6

Announcements

Make sure you are

- Registered for class, can access CMS
- Have a Section you can go to.
- *Lab Sections are required.*
    - “Make up” lab sections ***only*** **8:40am Wed, Thur, or Fri**
    - Bring laptop to Labs
- Project partners are required for projects.
    - Have project partner in same Lab Section, if possible

HW1 will be out soon out

- Do problem with lecture
- Work alone
- But, use your resources
    - Lab Section, Piazza.com, Office Hours, Homework Help Session,
    - Class notes, book, Sections, CSUGLab

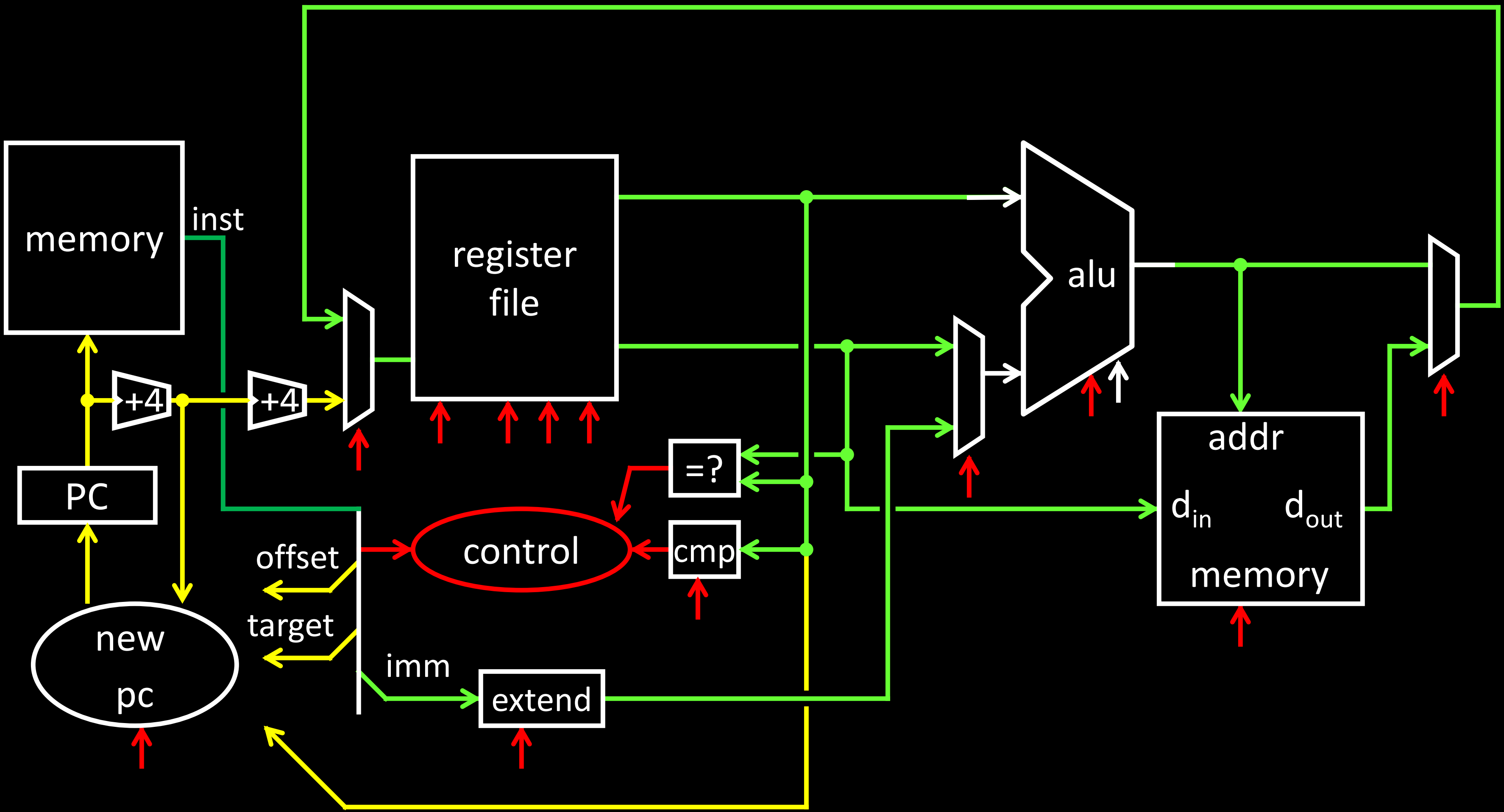
Annnouncements

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2015sp/schedule.html
- Slides and Reading for lectures
- Office Hours
- ***Pictures of all TAs***
- Homework and Programming Assignments
- Dates to keep in Mind
    - Prelims: Tue Mar 3rd and Thur April 30th
    - Lab 1: Due Fri Feb 13th before Winter break
    - Proj2: Due Thur Mar 26th before Spring break
    - Final Project: Due when final would be (not known until Feb 14t

Schedule is subject to change

Big Picture: Building a Processor



A Single cycle processor

Goals for Today

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)

Number Representations

Recall: Binary

- Two symbols (base 2): true and false; 1 and 0
- Basis of Logic Circuits and all digital computers

So, how do we represent numbers in *Binary* (base 2)?

Number Representations

Recall: Binary

- Two symbols (base 2): true and false; 1 and 0
- Basis of Logic Circuits and all digital computers

So, how do we represent numbers in *Binary* (base 2)?

- We can represent numbers in Decimal (base 10).
  - E.g. $\underset{10^2}{\underline{6}}\ \underset{10^1}{\underline{3}}\ \underset{10^0}{\underline{7}}$
- Can just as easily use other bases
  - Base 2 — Binary $\underset{2^9}{\underline{1}}\ \underset{2^8}{\underline{0}}\quad \underset{2^7}{\underline{0}}\ \underset{2^6}{\underline{1}}\ \underset{2^5}{\underline{1}}\ \underset{2^4}{\underline{1}}\quad \underset{2^3}{\underline{1}}\ \underset{2^2}{\underline{1}}\ \underset{2^1}{\underline{0}}\ \underset{2^0}{\underline{1}}$
  - Base 8 — Octal $0o\ \underset{8^3}{\underline{1}}\ \underset{8^2}{\underline{1}}\ \underset{8^1}{\underline{7}}\ \underset{8^0}{\underline{5}}$
  - Base 16 — Hexadecimal $0x\ \underset{16^2}{\underline{2}}\ \underset{16^1}{\underline{7}}\ \underset{16^0}{\underline{d}}$

Number Representations

Recall: Binary

- Two symbols (base 2): true and false; 1 and 0
- Basis of Logic Circuits and all digital computers

So, how do we represent numbers in *Binary* (base 2)?

- We can represent numbers in Decimal (base 10).
  - E.g. $\underset{10^2}{\underline{6}}\ \underset{10^1}{\underline{3}}\ \underset{10^0}{\underline{7}}$ $6 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 637$

- Can just as easily use other bases
  - Base 2 — Binary $1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 637$
  - Base 8 — Octal $1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 637$
  - Base 16 — Hexadecimal $2 \cdot 16^2 + 7 \cdot 16^1 + \boxed{d} \cdot 16^0 = 637$
$2 \cdot 16^2 + 7 \cdot 16^1 + \boxed{13} \cdot 16^0 = 637$

Number Representations: Activity #1 Counting

How do we count in different bases?

Dec (base 10)	Bin (base 2)	Oct (base 8)	Hex (base 16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	1 0000	20	10
17	1 0001	21	11
18	1 0010	22	12
⋮	⋮	⋮	⋮
99			
100			

0b 1111 1111 = ?

0b 1 0000 0000 = ?

0o 77 = ?

0o 100 = ?

0x ff = ?

0x 100 = ?

Number Representations: Activity #1 Counting

How do we count in different bases?

Dec (base 10)	Bin (base 2)	Oct (base 8)	Hex (base 16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	1 0000	20	10
17	1 0001	21	11
18	1 0010	22	12
.	.	.	.
.	.	.	.
99			
100			

0b 1111 1111 = 255

0b 1 0000 0000 = 256

0o 77 = 63

0o 100 = 64

0x ff = 255

0x 100 = 256

Number Representations

How to convert a number between different bases?

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient

- 637 ÷ 8 = 79 remainder 5 lsb (least significant bit)
- 79 ÷ 8 = 9 remainder 7
- 9 ÷ 8 = 1 remainder 1
- 1 ÷ 8 = 0 remainder 1 msb (most significant bit)

637 = 0o $\underset{\text{msb}}{1}17\underset{\text{lsb}}{5}$

Number Representations

Convert a base 10 number to a base 2 number

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient
- 637 ÷ 2 = 318 remainder 1 lsb (least significant bit)
- 318 ÷ 2 = 159 remainder 0
- 159 ÷ 2 = 79 remainder 1
- 79 ÷ 2 = 39 remainder 1
- 39 ÷ 2 = 19 remainder 1
- 19 ÷ 2 = 9 remainder 1
- 9 ÷ 2 = 4 remainder 1
- 4 ÷ 2 = 2 remainder 0
- 2 ÷ 2 = 1 remainder 0
- 1 ÷ 2 = 0 remainder 1 msb (most significant bit)

637 = 10 0111 1101 (can also be written as 0b10 0111 1101)

msb lsb

Number Representations

Convert a base 10 number to a base 16 number

Base conversion via repetitive division

- Divide by base, write remainder, move left with quotient
- 637 ÷ 16 = 39 remainder 13 lsb
- 39 ÷ 16 = 2 remainder 7
- 2 ÷ 16 = 0 remainder 2 msb

dec	hex	bin
10	0xa	1010
11	0xb	1011
12	0xc	1100
13	0xd	1101
14	0xe	1110
15	0xf	1111

637 = 0x 2 7 13 = ?

Thus, 637 = 0x27d

Number Representations

Convert a base 2 number to base 8 (oct) or 16 (hex)

Binary to Hexadecimal

- Convert each nibble (group of four bits) from binary to hex
- A nibble (four bits) ranges in value from 0…15, which is one hex digit
  - Range: 0000…1111 (binary) => 0x0 …0xF (hex) => 0…15 (decimal)
- E.g. 0b10 0111 1101
  - 0b10 = 0x2
  - 0b0111 = 0x7
  - 0b1101 = 0xd

  - Thus, 637 = 0x27d = 0b10 0111 1101

Binary to Octal

- Convert each group of three bits from binary to oct
- Three bits range in value from 0…7, which is one octal digit
  - Range: 0000…1111 (binary) => 0x0 …0xF (hex) => 0…15 (decimal)
- E.g. 0b1 001 111 101
  - 0b1 = 0x1
  - 0b001 = 0x1
  - 0b111 = 0x7
  - 0b101 = 0x5
  - Thus, 637 = 0o1175 = 0b10 0111 1101

Number Representations Summary

We can represent any number in any base

- Base 10 – Decimal

$\underset{10^2}{\underline{6}}\ \underset{10^1}{\underline{3}}\ \underset{10^0}{\underline{7}}$

$6 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 637$

- Base 2 — Binary

$\underset{2^9}{\underline{1}}\ \underset{2^8}{\underline{0}}\quad \underset{2^7}{\underline{0}}\ \underset{2^6}{\underline{1}}\ \underset{2^5}{\underline{1}}\ \underset{2^4}{\underline{1}}\quad \underset{2^3}{\underline{1}}\ \underset{2^2}{\underline{1}}\ \underset{2^1}{\underline{0}}\ \underset{2^0}{\underline{1}}$

$1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 637$

- Base 8 — Octal

$0o\ \underset{8^3}{\underline{1}}\ \underset{8^2}{\underline{1}}\ \underset{8^1}{\underline{7}}\ \underset{8^0}{\underline{5}}$

$1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 637$

- Base 16 — Hexadecimal

$0x\ \underset{16^2}{\underline{2}}\ \underset{16^1}{\underline{7}}\ \underset{16^0}{\underline{d}}$

$$2 \cdot 16^2 + 7 \cdot 16^1 + \enclose{circle}{d} \cdot 16^0 = 637$$
$$2 \cdot 16^2 + 7 \cdot 16^1 + \enclose{circle}{13} \cdot 16^0 = 637$$

Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what the computer is doing!).

Today's Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two's compliment
- Addition (two's compliment)
- Subtraction (two's compliment)

Next Goal

Binary Arithmetic: Add and Subtract two binary numbers

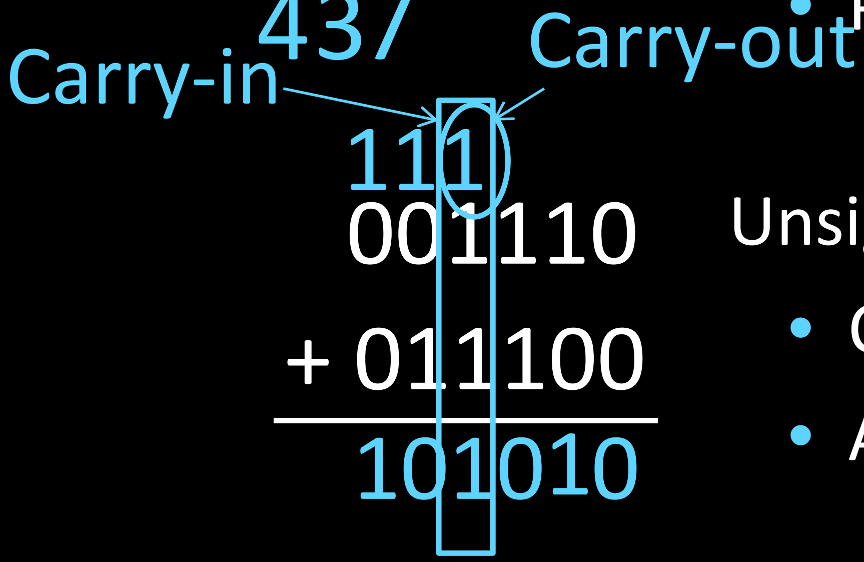
Binary Addition

How do we do arithmetic in binary?

1
183
+ 254
437

Addition works the same way regardless of base

- Add the digits in each position
- Propagate the carry



111
001110
+ 011100
101010

Unsigned binary addition is pretty easy

- Combine two bits at a time
- Along with a carry

Binary Addition

How do we do arithmetic in binary?

$$\begin{array}{r} 1 \\ 183 \\ +\ 254 \\ \hline 437 \end{array}$$

Addition works the same way regardless of base

- Add the digits in each position
- Propagate the carry

$$\begin{array}{r} 111 \\ 001110 \\ +\ 011100 \\ \hline 101010 \end{array}$$

Unsigned binary addition is pretty easy

- Combine two bits at a time
- Along with a carry

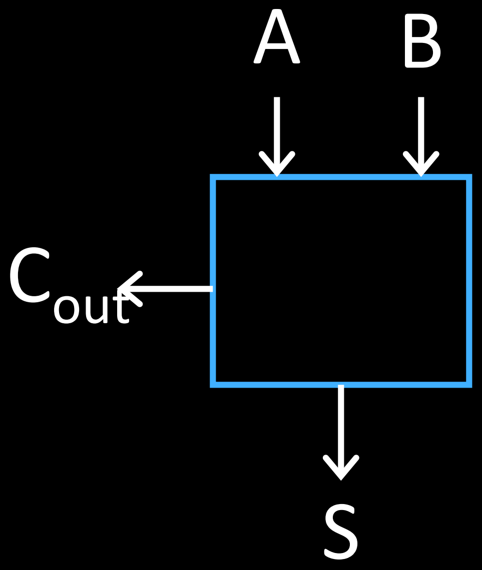
Binary Addition

Binary addition requires

- Add of ***two bits*** PLUS ***carry-in***
- Also, ***carry-out*** if necessary

1-bit Adder

Half Adder

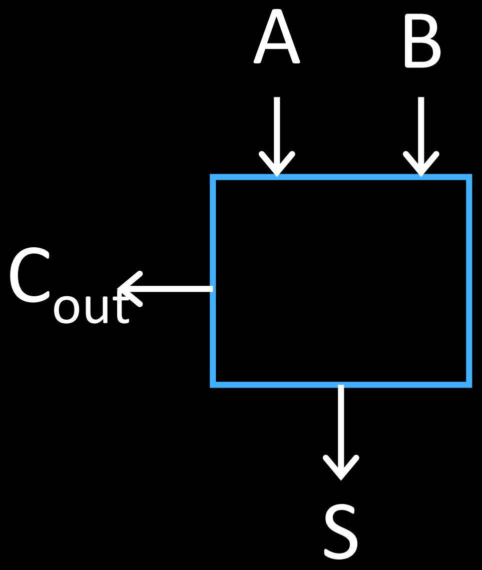


- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

A	**B**	**C_{out}**	**S**
0	0		
0	1		
1	0		
1	1		

1-bit Adder

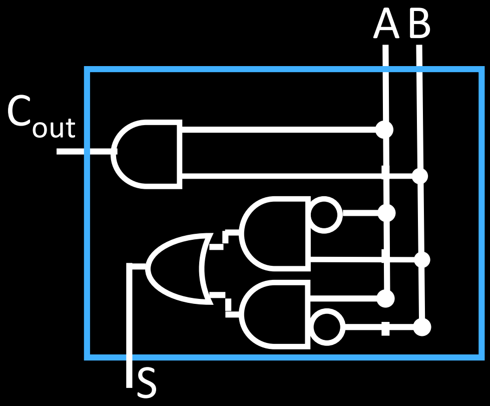
Half Adder



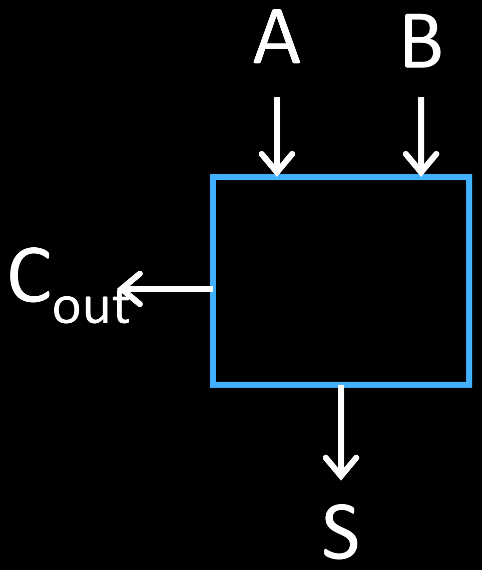
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $S = \overline{A}B + A\overline{B}$
- $C_{out} = AB$



1-bit Adder

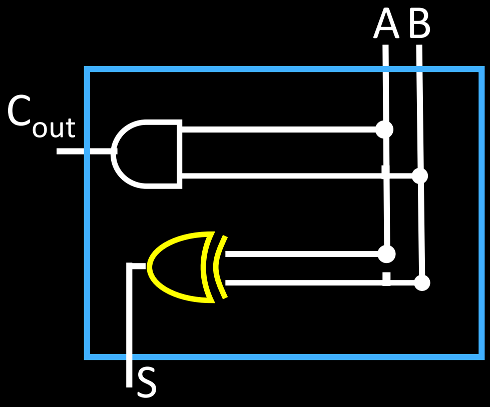


Half Adder

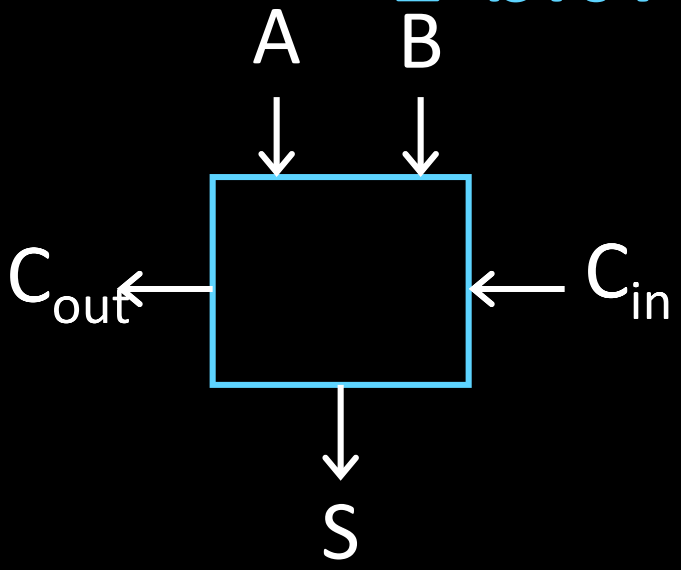
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $S = \overline{A}B + A\overline{B} = A \oplus B$
- $C_{out} = AB$



1-bit Adder with Carry



Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carr
- Can be cascaded

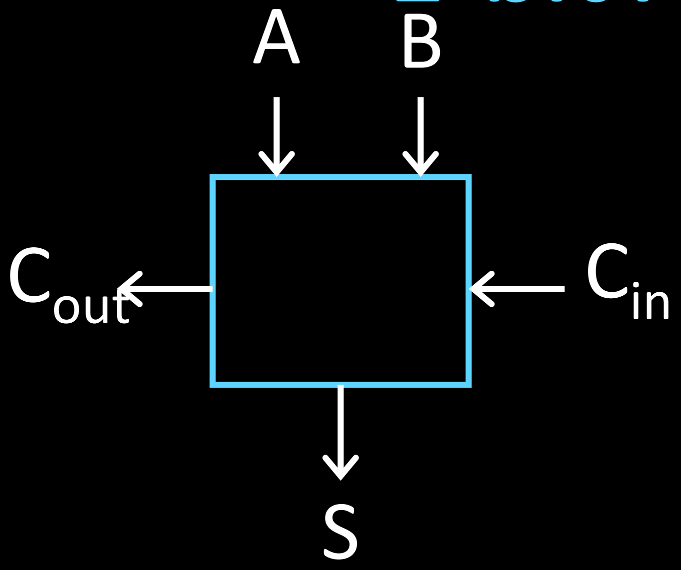
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Activity: Truth Table and Sum-of-Product.

Logic minimization via Karnaugh Maps and algebraic minimization.

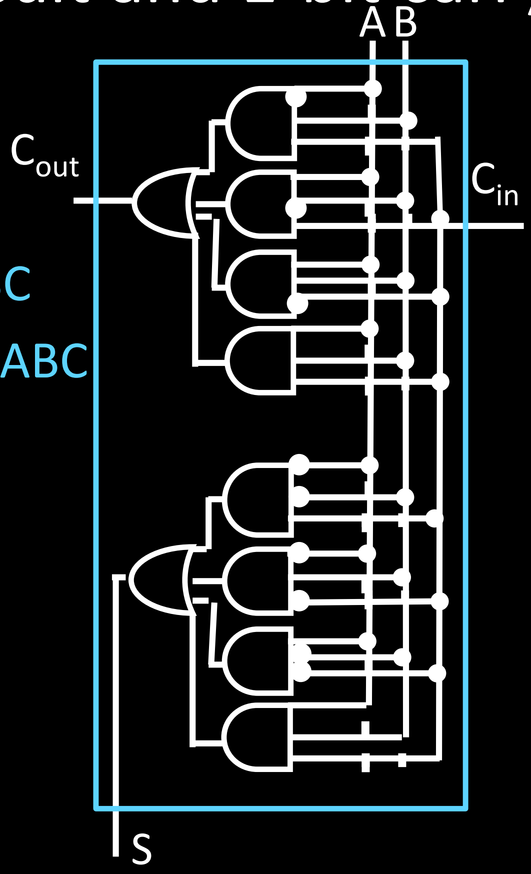
Draw Logic Circuits

1-bit Adder with Carry



Full Adder

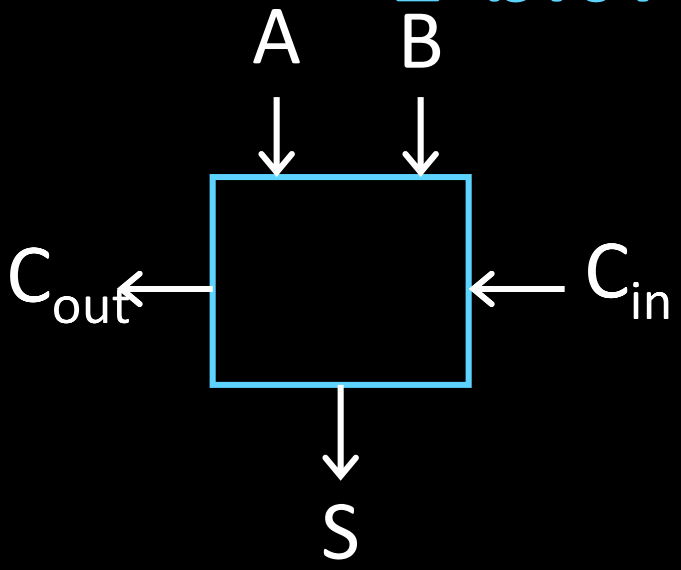
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded



- $S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$
- $C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

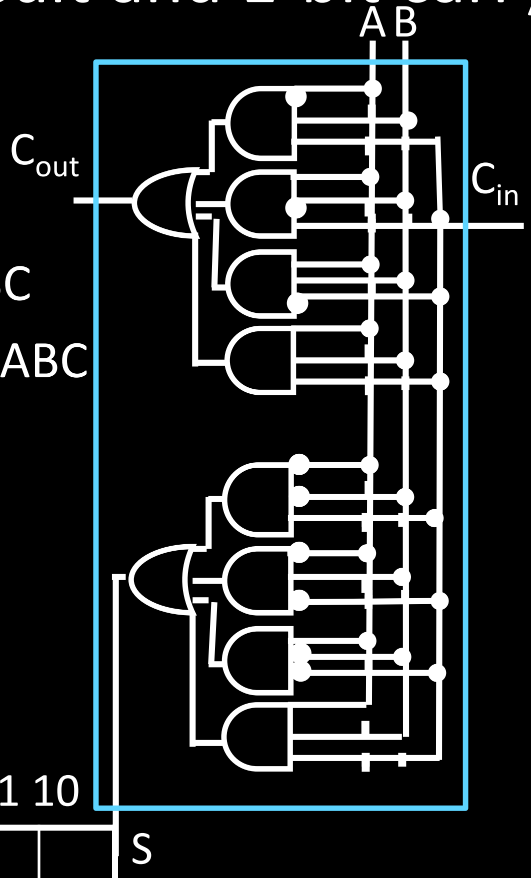
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

1-bit Adder with Carry



Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded



- $S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$
- $C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

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

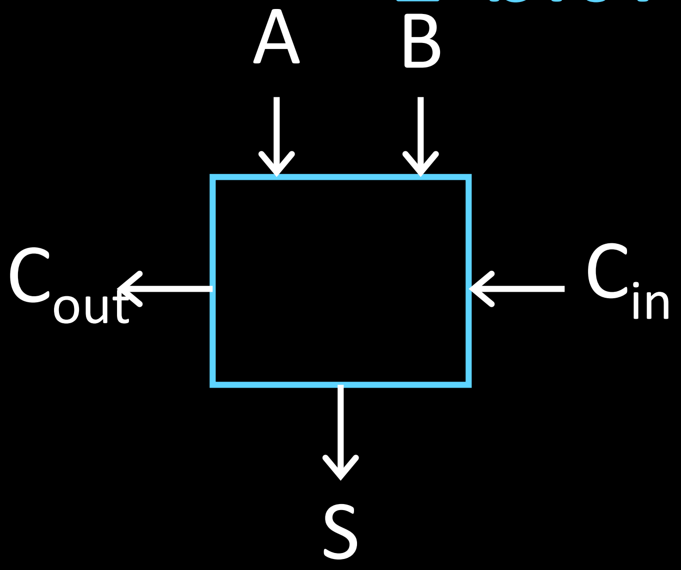
S

C_{in} \ AB	00	01	11	10
0				
1				

C_{out}

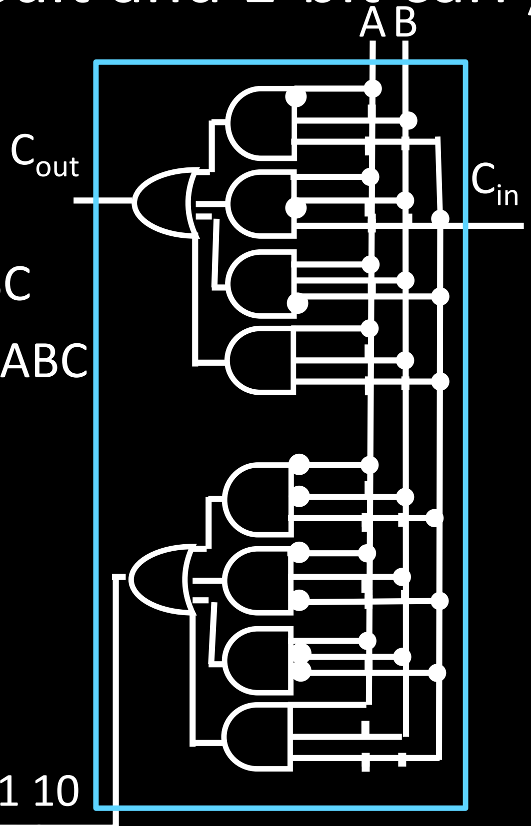
C_{in} \ AB	00	01	11	10
0				
1				

1-bit Adder with Carry



Full Adder

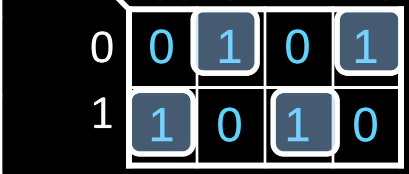
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded



- $S = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- $C_{out} = AB + AC + BC$

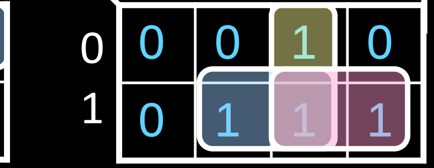
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S



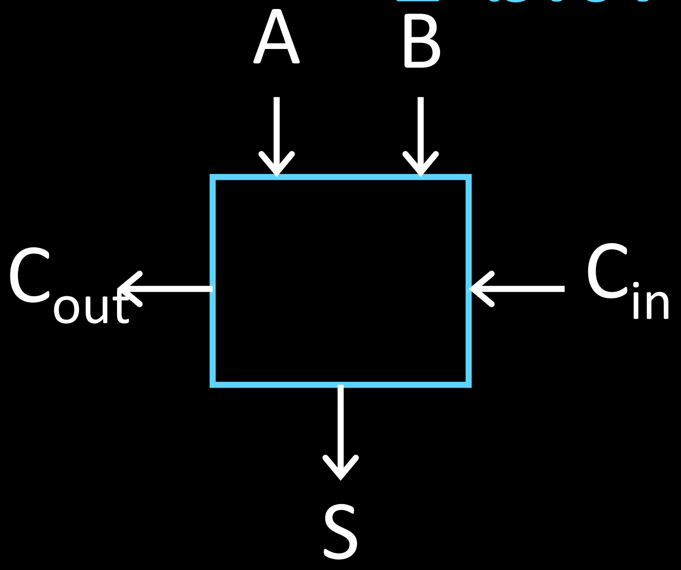
C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_{out}



C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

1-bit Adder with Carry



Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

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

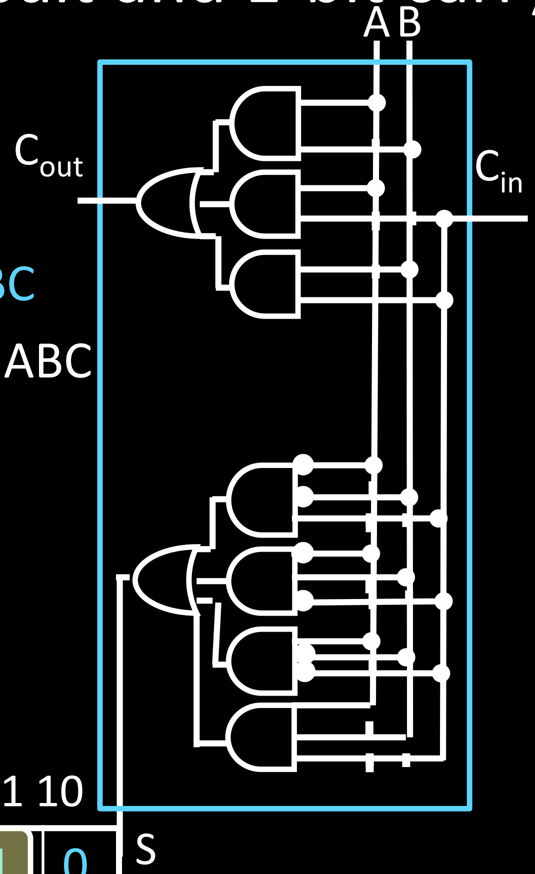
- $S = \overline{AB}C + \overline{A}B\overline{C} + A\overline{BC} + ABC$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- $C_{out} = AB + AC + BC$

S

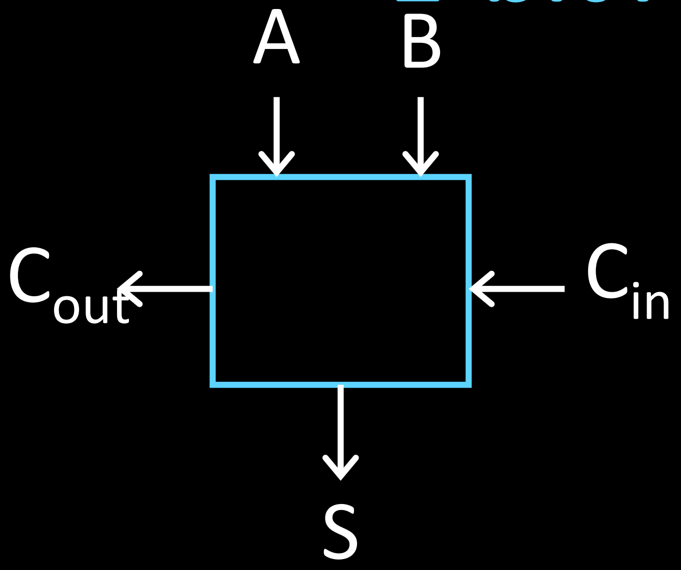
C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_{out}

C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1



1-bit Adder with Carry



Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

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

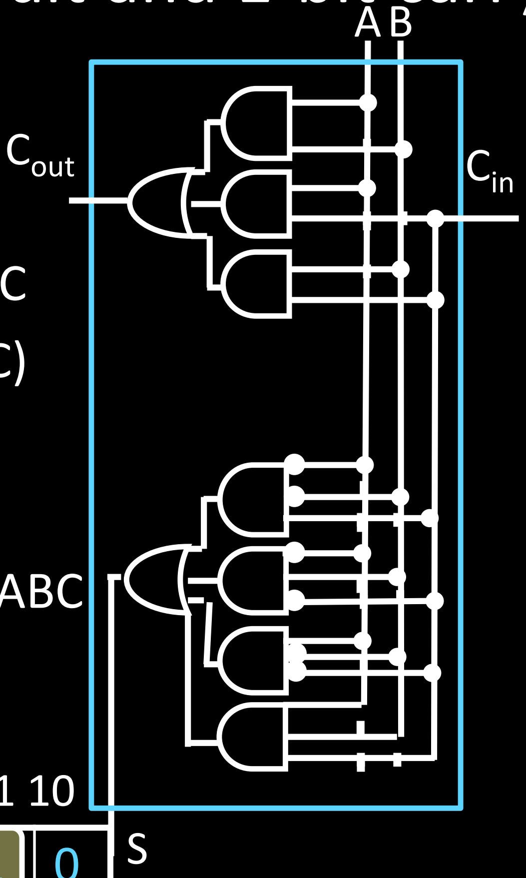
- $S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$
- $S = \overline{A}(\overline{B}C + B\overline{C}) + A(\overline{B}\,\overline{C} + BC)$
- $S = \overline{A}(B \oplus C) + A(\overline{B \oplus C})$
- $S = A \oplus (B \oplus C)$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- $C_{out} = AB + AC + BC$

S

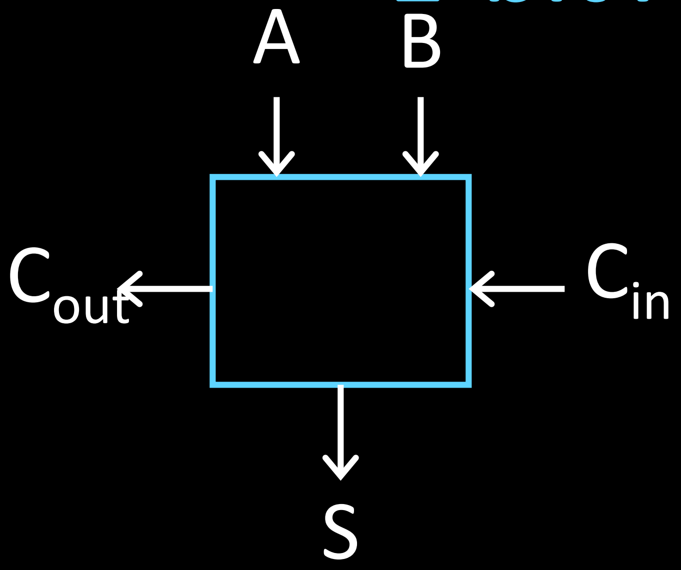
C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_{out}

C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

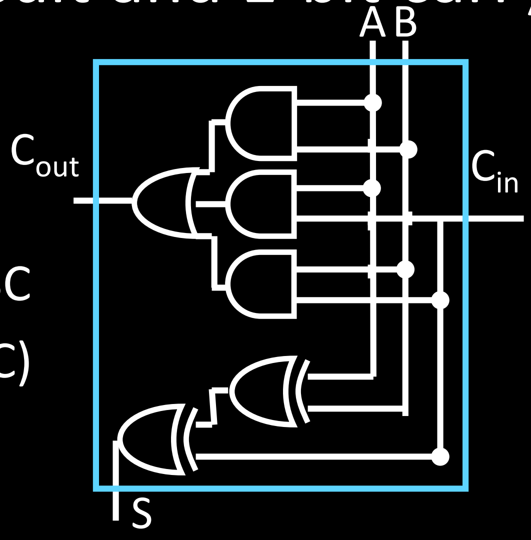


1-bit Adder with Carry



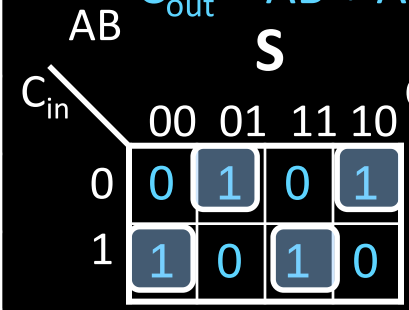
Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded



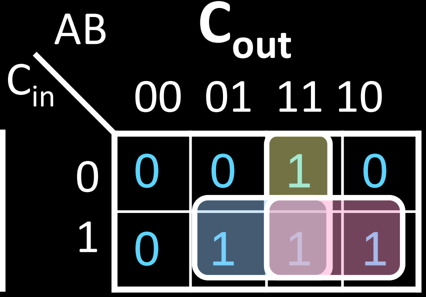
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- $S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\overline{B}\overline{C} + ABC$
- $S = \bar{A}(\bar{B}C + B\bar{C}) + A(\overline{B}\overline{C} + BC)$
- $S = \bar{A}(B \oplus C) + A(\overline{B \oplus C})$
- $S = A \oplus (B \oplus C)$
- $C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$
- $C_{out} = AB + AC + BC$



S

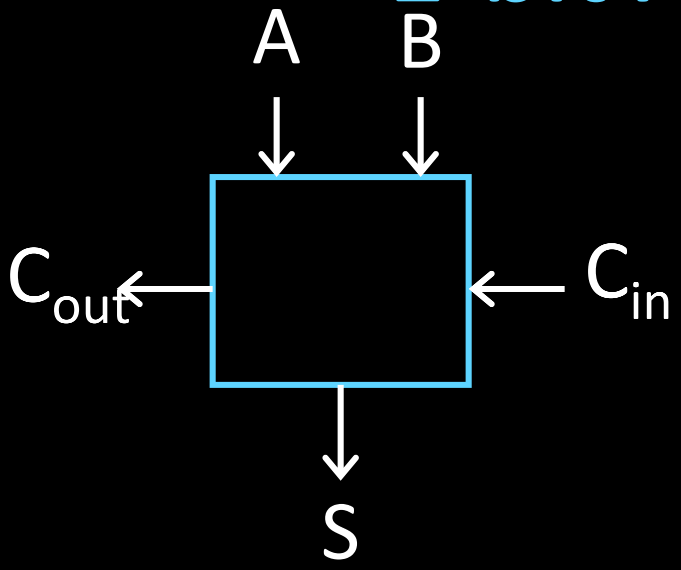
C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0



C_{out}

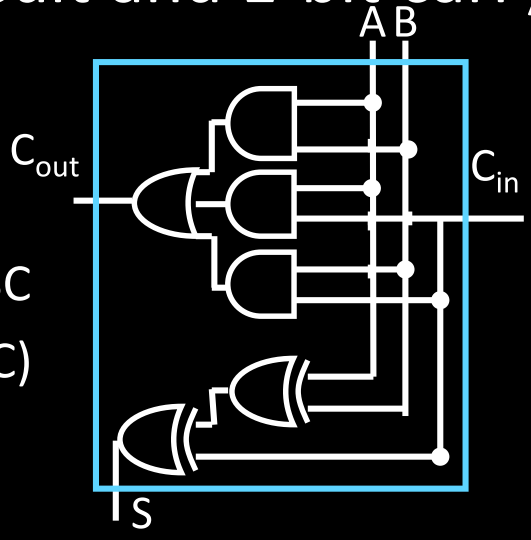
C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

1-bit Adder with Carry



Full Adder

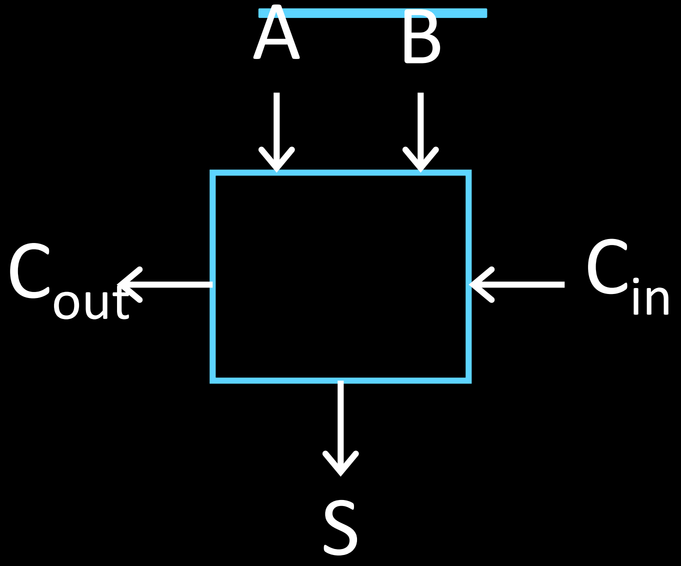
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

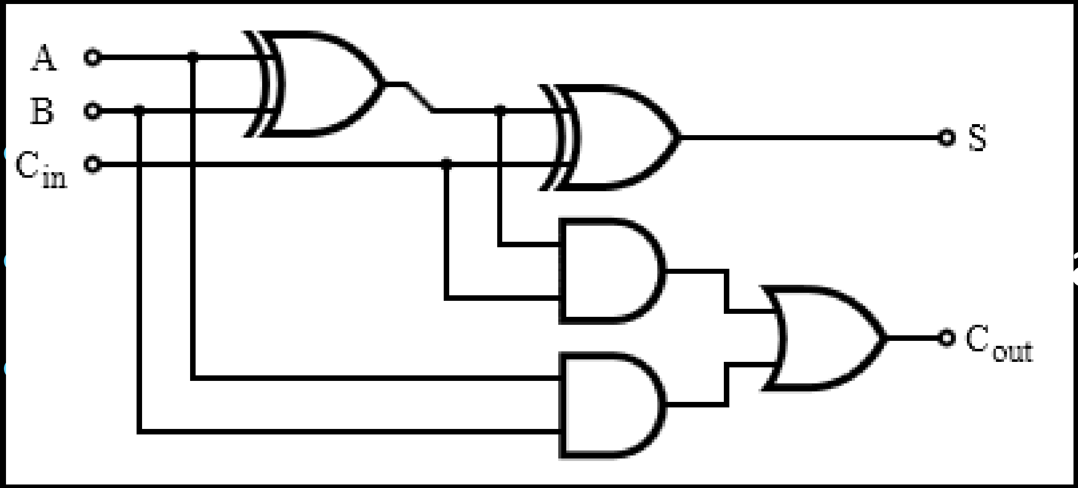


A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- $S = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$
- $S = \overline{A}(\overline{B}C + B\overline{C}) + A(\overline{B}\overline{C} + BC)$
- $S = \overline{A}(B \oplus C) + A(\overline{B \oplus C})$
- $S = A \oplus (B \oplus C)$
- $S = A \oplus B \oplus C$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C)$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB$
- $C_{out} = (\overline{A}B + A\overline{B})C + AB$
- $C_{out} = (A \oplus B)C + AB$

Lab0 1-bit Adder with Carry

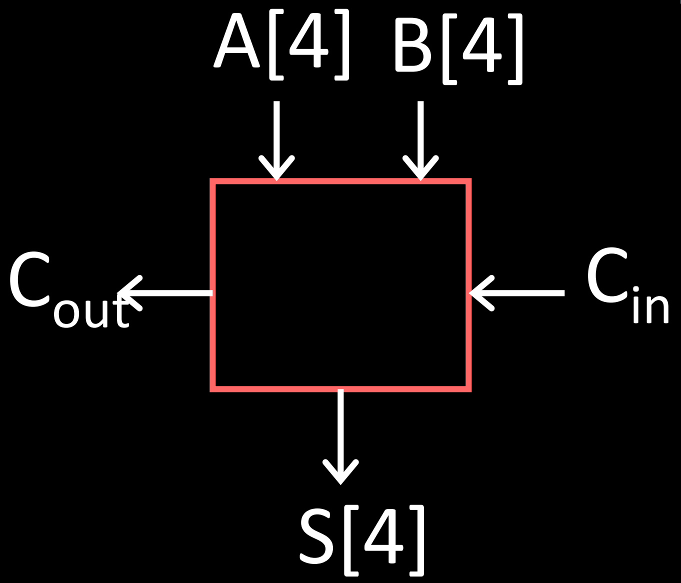




A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- $S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$
- $S = \overline{A}(\overline{B}C + B\overline{C}) + A(\overline{B}\,\overline{C} + BC)$
- $S = \overline{A}(B \oplus C) + A(\overline{B \oplus C})$
- $S = A \oplus (B \oplus C)$
- $S = A \oplus B \oplus C$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C)$
- $C_{out} = \overline{A}BC + A\overline{B}C + AB$
- $C_{out} = (\overline{A}B + A\overline{B})C + AB$
- $C_{out} = (A \oplus B)C + AB$

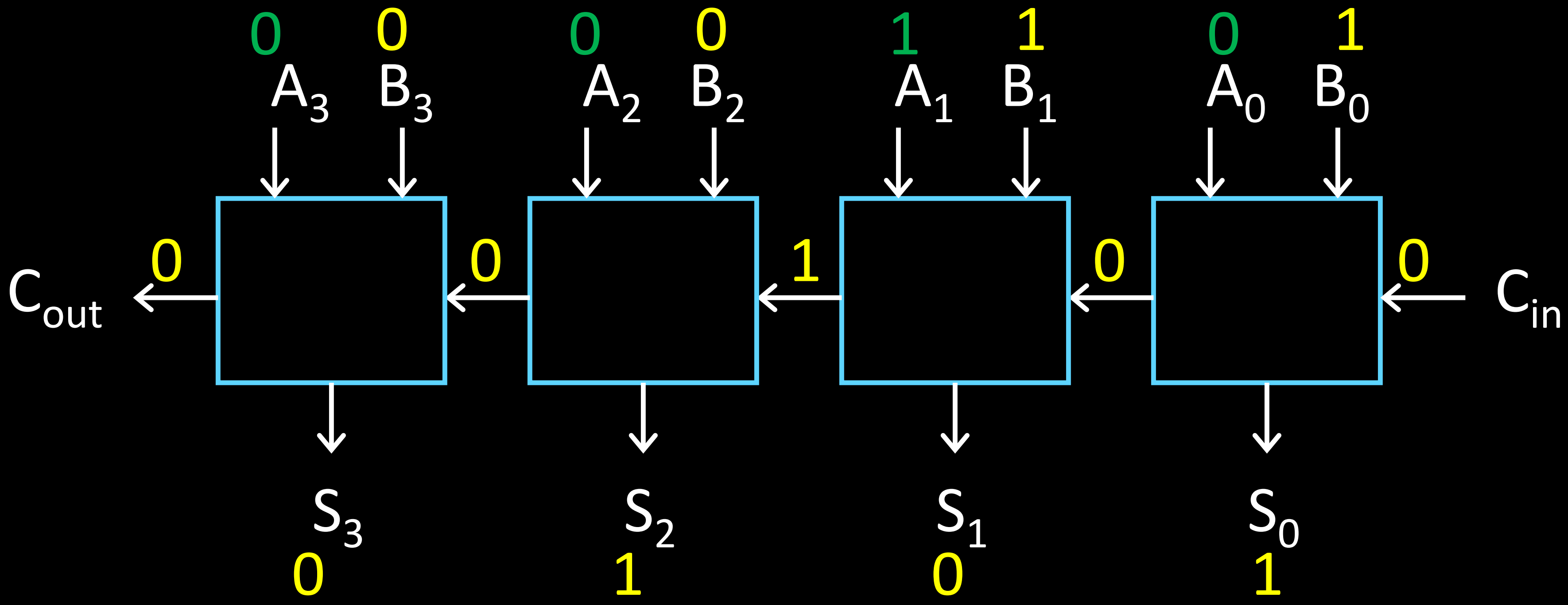
4-bit Adder



4-Bit Full Adder

- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded

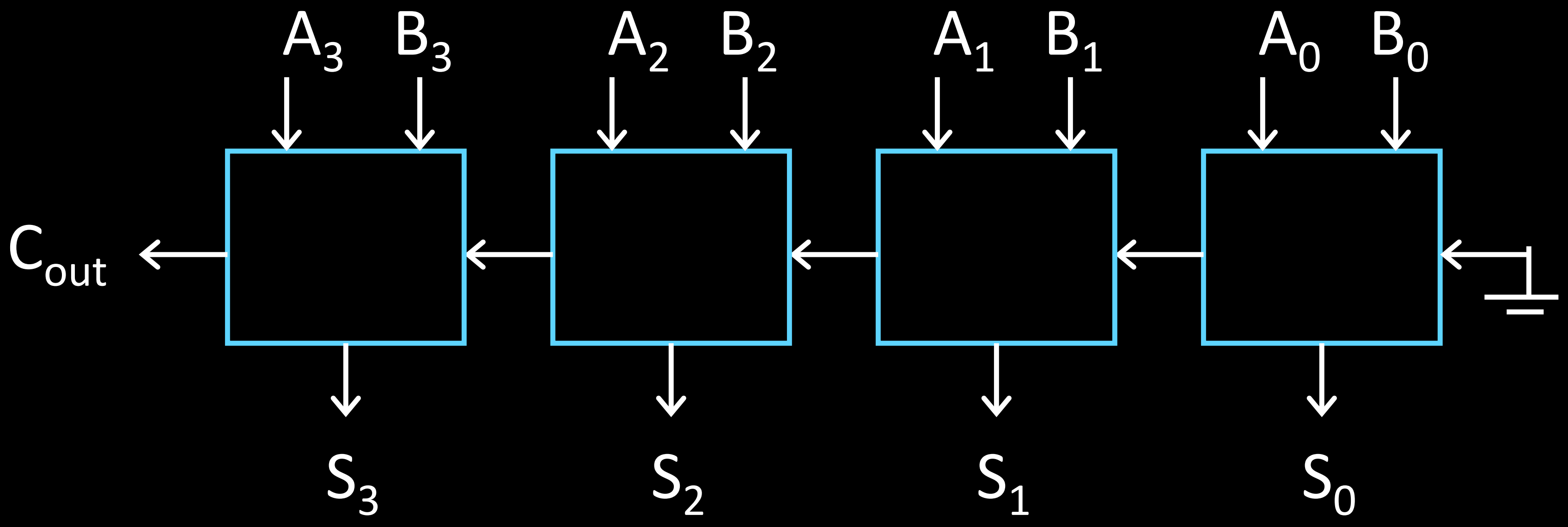
4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out

- Carry-out = overflow indicates result does not fit in 4 bits

4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out

- Carry-out = overflow indicates result does not fit in 4 bits

Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what computer is doing!).

Adding two 1-bit numbers generalizes to adding two numbers of any size since 1-bit full adders can be cascaded.

Today’s Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)

Next Goal

How do we subtract two binary numbers?

Equivalent to adding with a negative number

How do we represent negative numbers?

irst Attempt: Sign/Magnitude Representatio

First Attempt: Sign/Magnitude Representation

- 1 bit for sign (0=positive, 1=negative)
- N-1 bits for magnitude

0111 = 7
1111 = -7

Problem?

- Two zero's: +0 different than -0
- Complicated circuits

0000 = +0
1000 = -0

IBM 7090

Second Attempt: One's complement

Second Attempt: One's complement

- Leading 0's for positive and 1's for negative
- Negative numbers: complement the positive number

0111 = 7

1000 = -7

Problem?

- Two zero's still: +0 different than -0
- -1 if offset from two's complement
- Complicated circuits
  - Carry is difficult

0000 = +0

1111 = -0

PDP 1

Two’s Complement Representation

What is used: Two’s Complement Representation

Nonnegative numbers are represented as usual

- 0 = 0000, 1 = 0001, 3 = 0011, 7 = 0111

Leading 1’s for negative numbers

To negate any number:

- complement *all* the bits (i.e. flip all the bits)
- then add 1
- -1: 1 ⇒ 0001 ⇒ 1110 ⇒ 1111
- -3: 3 ⇒ 0011 ⇒ 1100 ⇒ 1101
- -7: 7 ⇒ 0111 ⇒ 1000 ⇒ 1001
- -8: 8 ⇒ 1000 ⇒ 0111 ⇒ 1000
- -0: 0 ⇒ 0000 ⇒ 1111 ⇒ 0000 (this is good, -0 = +0)

Two’s Complement Representation

Is there only one zero!

$$\begin{array}{r} 0 = 0000 \\ \overline{0} = 1111 \\ +1 \\ \hline 0 = 0000 \end{array}$$

One more example. How do we represent -20?

$$\begin{array}{r} 20 = 0001\ \ 0100 \\ \overline{20} = 1110\ \ 1011 \\ +1 \\ \hline -20 = 1110\ \ 1100 \end{array}$$

Two's Complement

Non-negatives (as usual):

+0 = 0000
+1 = 0001
+2 = 0010
+3 = 0011
+4 = 0100
+5 = 0101
+6 = 0110
+7 = 0111
+8 = 1000

Negatives (two's complement: flip then add 1):

Two’s Complement

Non-negatives (as usual):	Negatives (two’s complement: flip then add 1):	
+0 = 0000	$\overline{0}$ = 1111	-0 = 0000
+1 = 0001	$\overline{1}$ = 1110	-1 = 1111
+2 = 0010	$\overline{2}$ = 1101	-2 = 1110
+3 = 0011	$\overline{3}$ = 1100	-3 = 1101
+4 = 0100	$\overline{4}$ = 1011	-4 = 1100
+5 = 0101	$\overline{5}$ = 1010	-5 = 1011
+6 = 0110	$\overline{6}$ = 1001	-6 = 1010
+7 = 0111	$\overline{7}$ = 1000	-7 = 1001
+8 = 1000	$\overline{8}$ = 0111	-8 = 1000

Two’s Complement Facts

Signed two’s complement

- Negative numbers have leading 1’s
- zero is unique: +0 = - 0
- wraps from largest positive to largest negative

N bits can be used to represent

- unsigned: range 0…2^N-1
  - eg: 8 bits ⇒ 0…255
- signed (two’s complement): -(2^{N-1})…(2^{N-1} - 1)
  - E.g.: 8 bits ⇒ (1000 000) … (0111 1111)
  - -128 … 127

Sign Extension & Truncation

Extending to larger size

- 1111 = -1
- 1111 1111 = -1
- 0111 = 7
- 0000 0111 = 7

Truncate to smaller size

- 0000 1111 = 15
- BUT, ~~0000~~ 1111 = 1111 = -1

Two’s Complement Addition

Addition with two’s complement signed numbers

Perform addition as usual, regardless of sign
(it just works)

Examples

- 1 + -1 =
- -3 + -1 =
- -7 + 3 =
- 7 + (-3) =

Two’s Complement Addition

Addition with two’s complement signed numbers

Perform addition as usual, regardless of sign
(it just works)

Examples

- 1 + -1 = 0001 + 1111 =
- -3 + -1 = 1101 + 1111 =
- -7 + 3 = 1001 + 0011 =
- 7 + (-3) = 0111 + 1101 =

Two’s Complement Addition

Addition with two’s complement signed numbers

Perform addition as usual, regardless of sign
(it just works)

Examples

- 1 + -1 = 0001 + 1111 = 0000 (0)
- -3 + -1 = 1101 + 1111 = 1100 (-4)
- -7 + 3 = 1001 + 0011 = 1100 (-4)
- 7 + (-3) = 0111 + 1101 = 0100 (4)
- What is wrong with the following additions?
  - 7 + 1, -7 + -3, -7 + -1
  - 1000 overflow, 1 0110 overflow, 1000 fine

Binary Subtraction

Why create a new circuit?

Just use addition using two’s complement math

- How?

Binary Subtraction

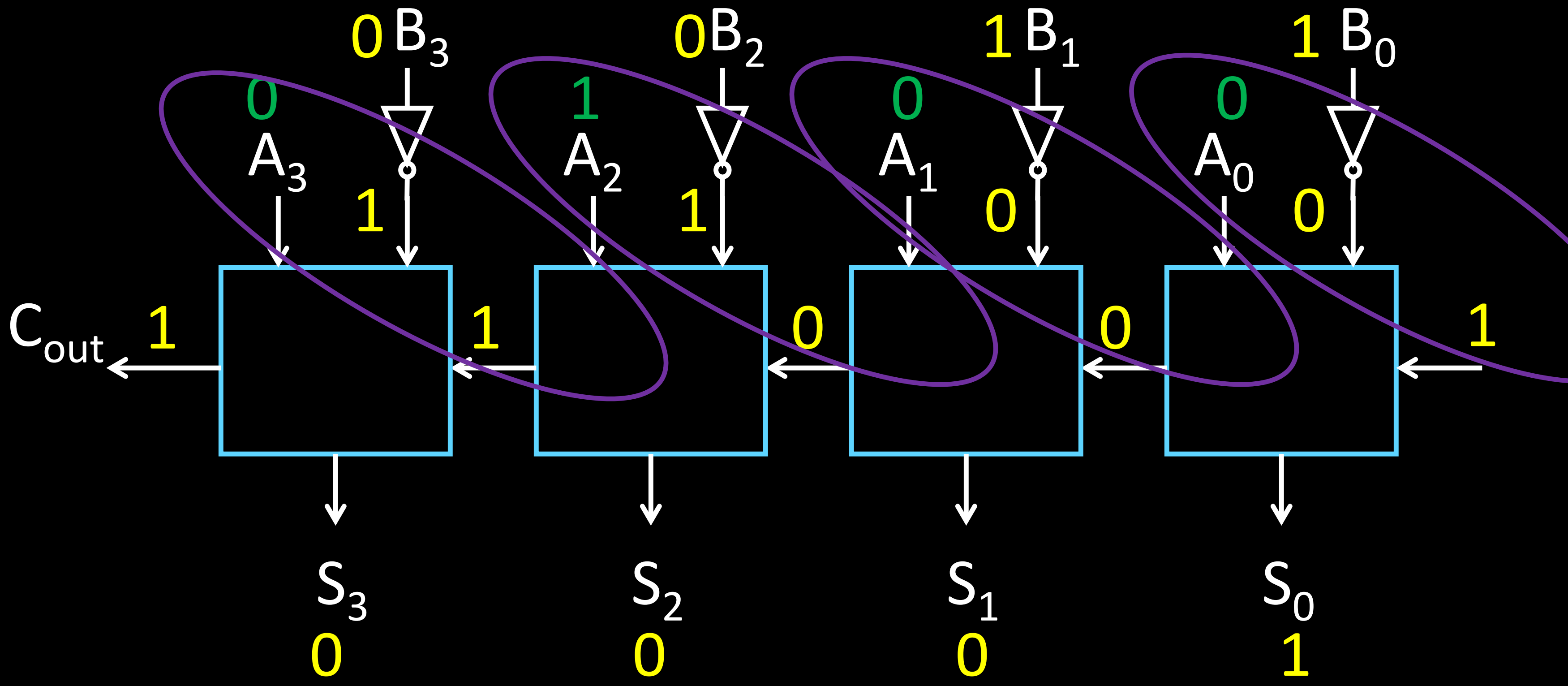
Two's Complement Subtraction

- Subtraction is simply addition,

  where one of the operands has been negated

  - Negation is done by inverting all bits and adding one

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



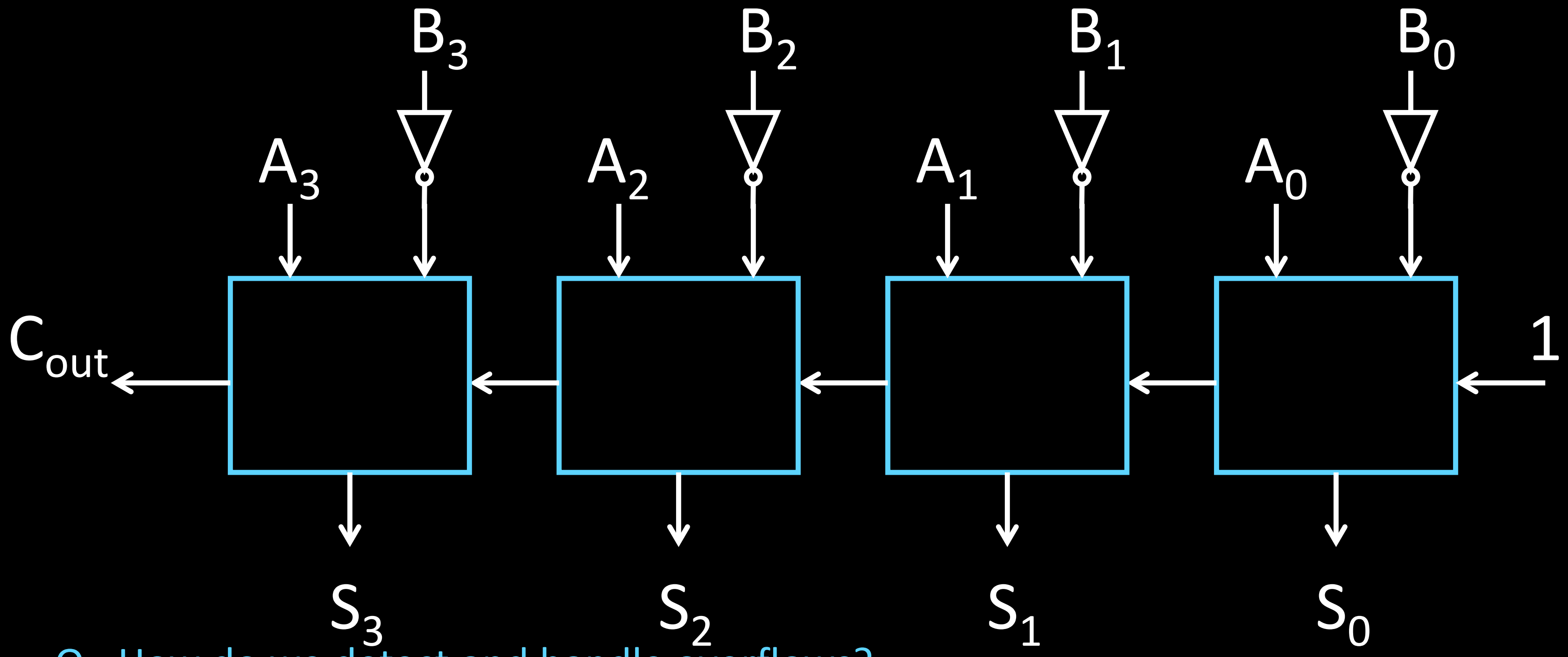
Binary Subtraction

Two's Complement Subtraction

- Subtraction is simply addition,

  where one of the operands has been negated
  - Negation is done by inverting all bits and adding one

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



Q: How do we detect and handle overflows?
Q: What if (-B) overflows?

Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what computer is doing!).

Adding two 1-bit numbers generalizes to adding two numbers of any size since 1-bit full adders can be cascaded.

Using Two's complement number representation simplifies adder Logic circuit design (0 is unique, easy to negate).

Subtraction is simply adding, where one operand is negated (two's complement; to negate just flip the bits and add 1).

.

Today’s Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)
- One more topic…

Next Goal

In general, how do we detect and handle overflow?

Overflow

When can overflow occur?

- adding a negative and a positive?

- adding two positives?

- adding two negatives?

Overflow

When can overflow occur?

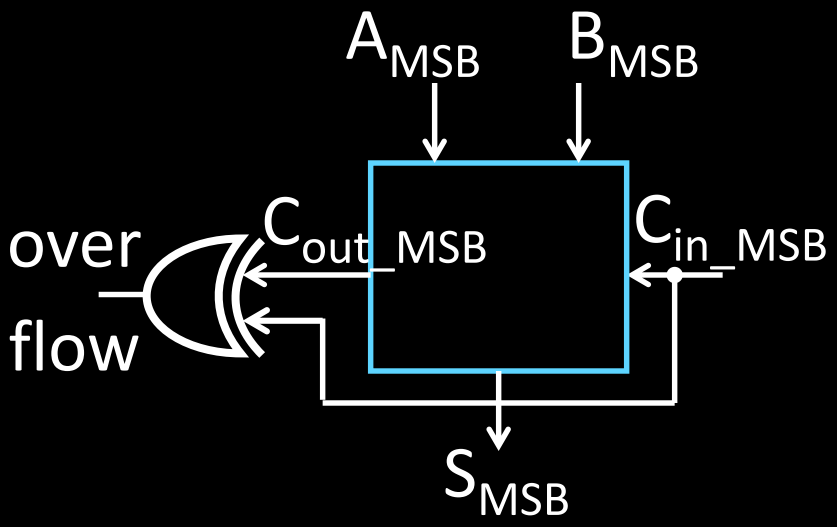
- adding a negative and a positive?
  - Overflow *cannot occur* (Why?)
  - Always subtract larger magnitude from smaller
- adding two positives?
  - Overflow *can occur* (Why?)
  - Precision: Add two positives, and get a negative number!
- adding two negatives?
  - Overflow *can occur* (Why?)
  - Precision: add two negatives, get a positive number!

Rule of thumb:

- Overflow happens iff
  carry into msb != carry out of msb

Overflow

When can overflow occur?



MSB

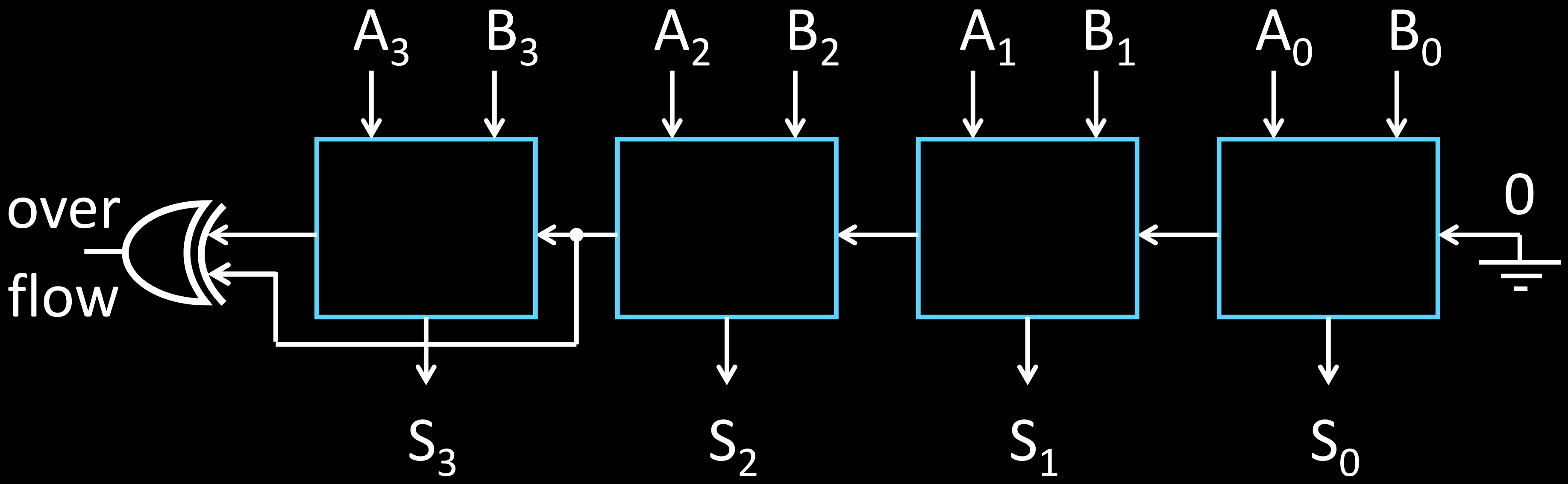
A	**B**	**C_{in}**	**C_{out}**	**S**	
0	0	0	0	0	
0	0	1	0	1	Wrong Sign
0	1	0	0	1	
0	1	1	1	0	
1	0	0	0	1	
1	0	1	1	0	
1	1	0	1	0	Wrong Sign
1	1	1	1	1	

Rule of thumb:

- Overflow happened iff
  carry into msb != carry out of msb

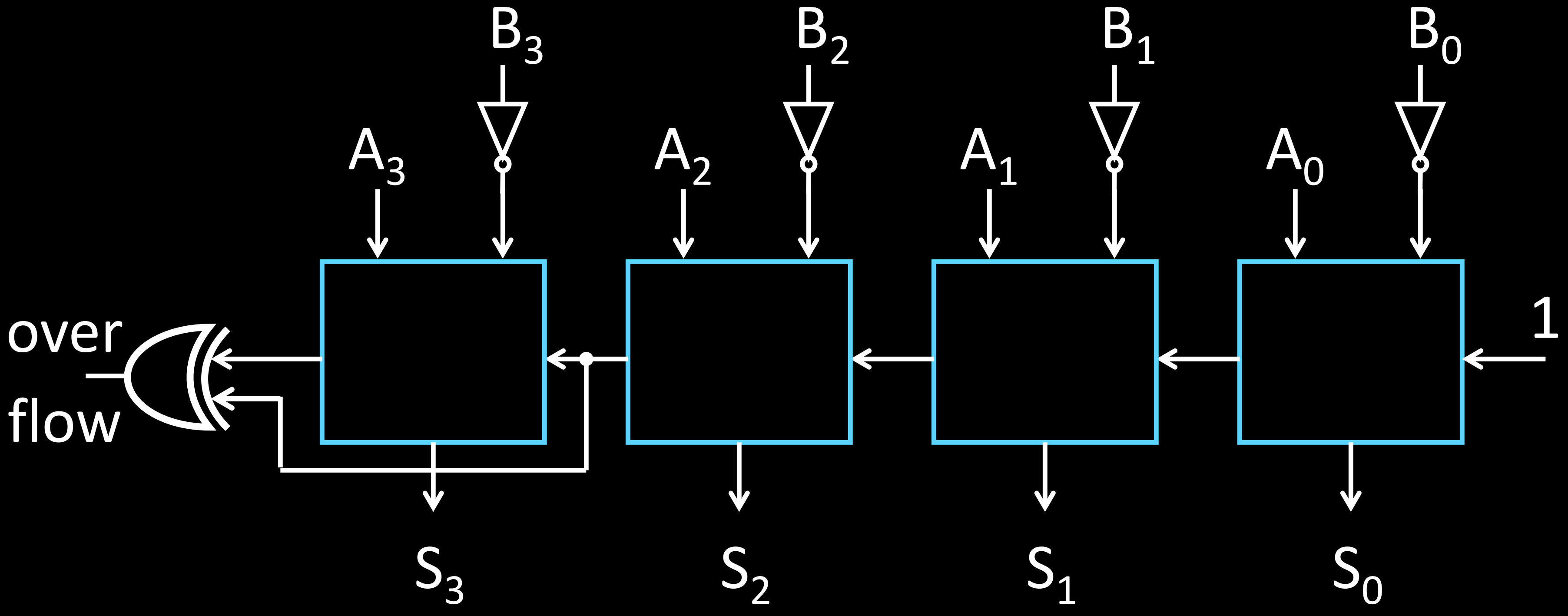
Two’s Complement Adder

Two’s Complement Adder with overflow detection



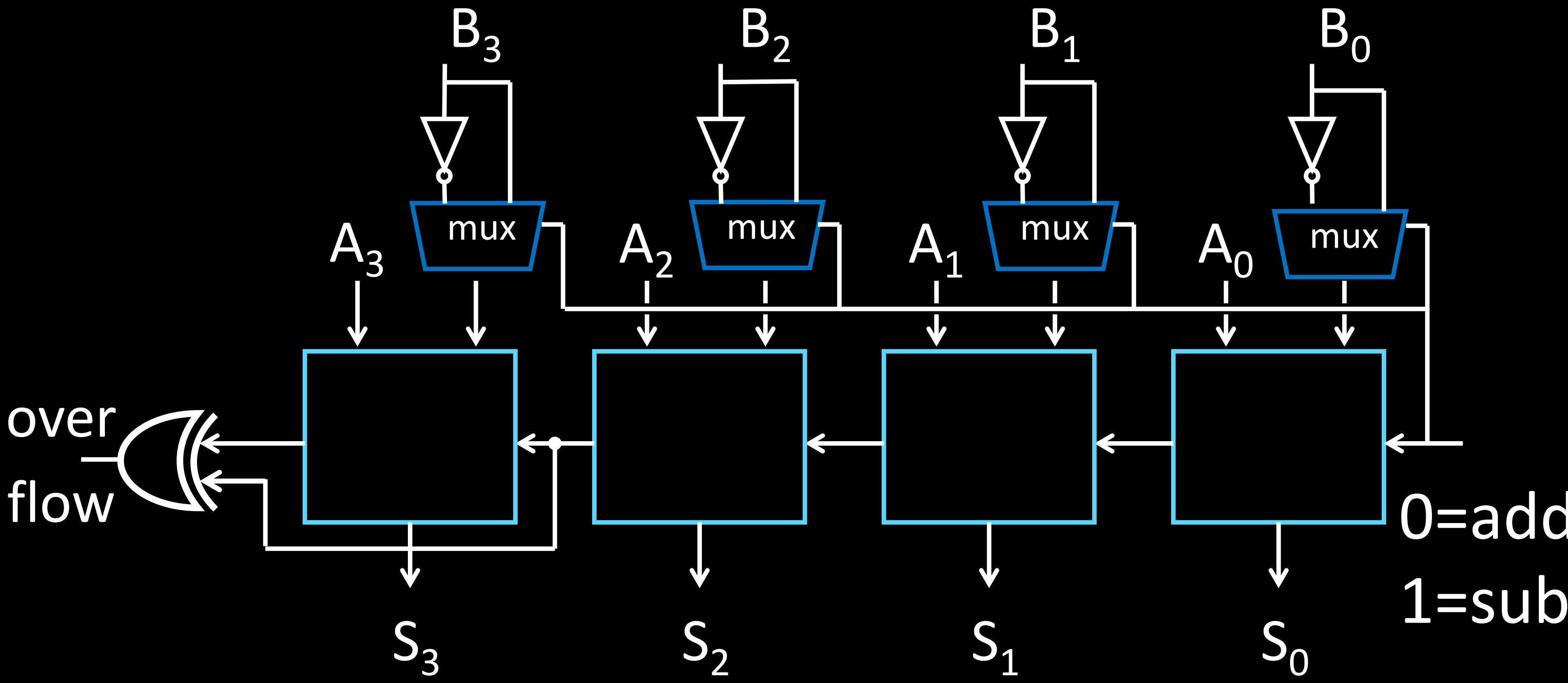
Two's Complement Adder

Two's Complement Subtraction with overflow detection



Two’s Complement Adder

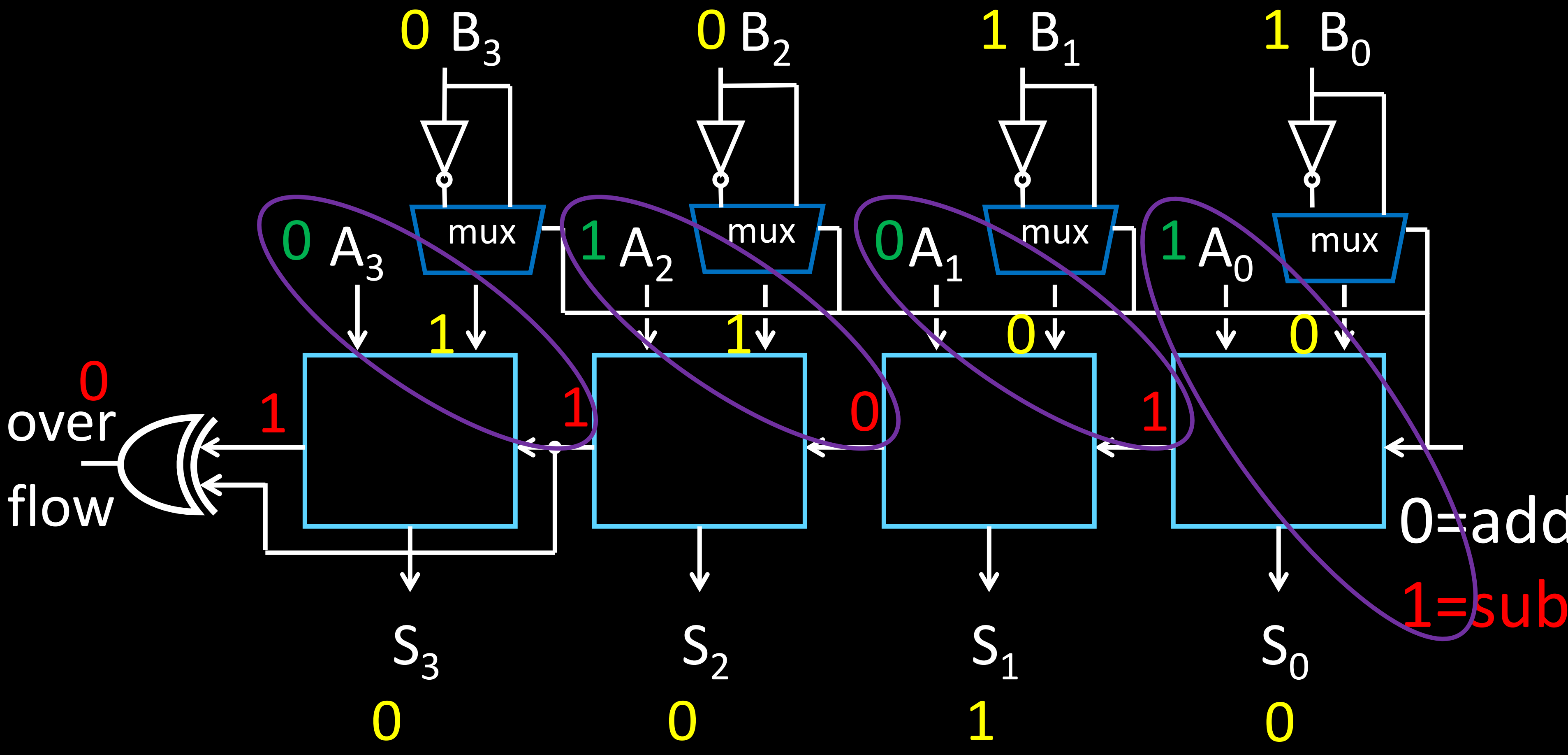
Two’s Complement Adder with overflow detection



Note: 4-bit adder is drawn for illustrative purposes and may not represent the optimal design.

Two’s Complement Adder

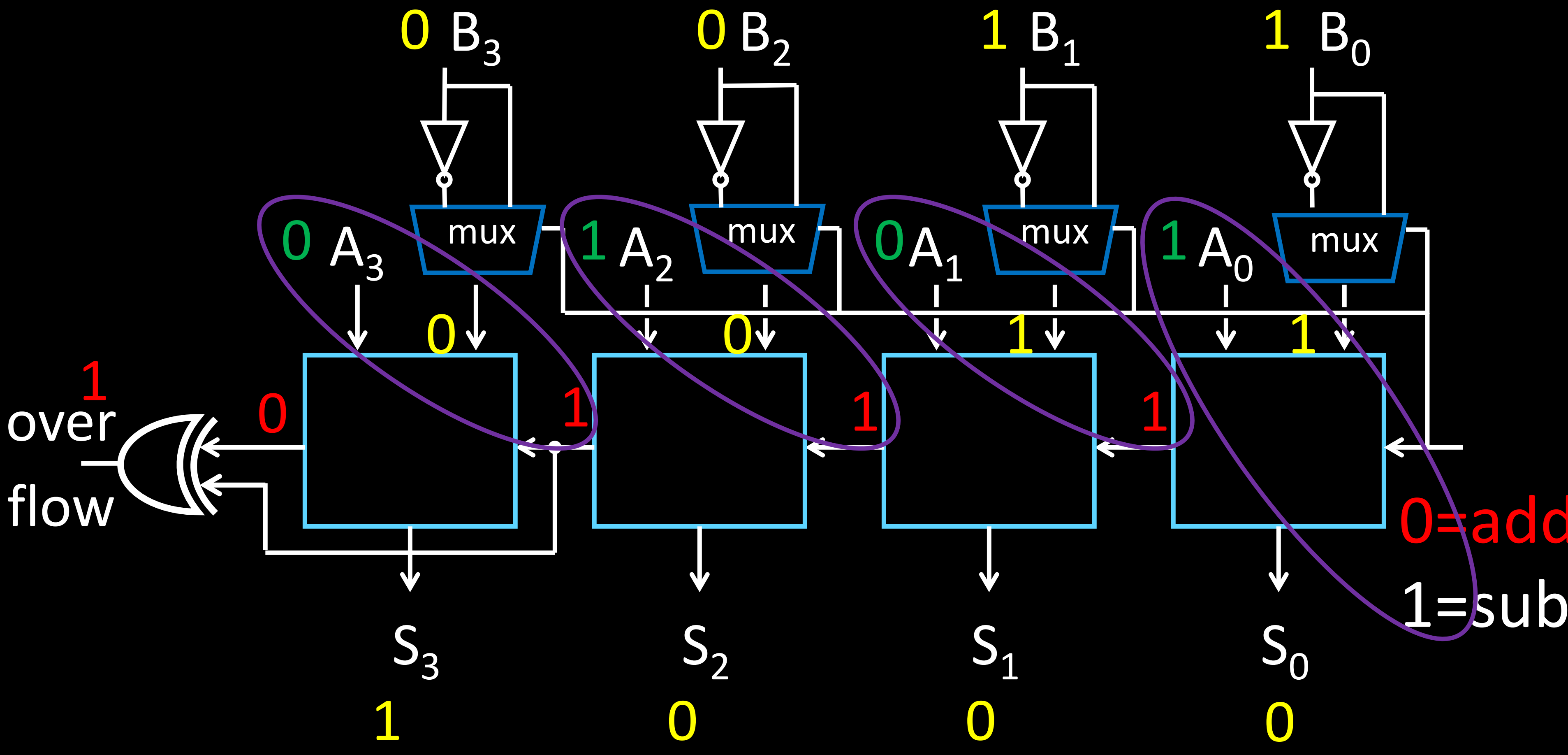
Two’s Complement Adder with overflow detection



Note: 4-bit adder is drawn for illustrative purposes and may not represent the optimal design.

Two’s Complement Adder

Two’s Complement Adder with overflow detection



Note: 4-bit adder is drawn for illustrative purposes and may not represent the optimal design.

Takeaway

Digital computers are implemented via logic circuits and thus represent *all* numbers in binary (base 2).

We (humans) often write numbers as decimal and hexadecimal for convenience, so need to be able to convert to binary and back (to understand what computer is doing!).

Adding two 1-bit numbers generalizes to adding two numbers of any size since 1-bit full adders can be cascaded.

Using Two's complement number representation simplifies adder Logic circuit design (0 is unique, easy to negate). Subtraction is simply adding, where one operand is negated (two's complement; to negate just flip the bits and add 1).

Overflow if sign of operands A and B != sign of result S.

Can detect overflow by testing C_{in} != C_{out} of the most significant bit (msb), which only occurs when previous statement is true.

Today’s Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)
- And, how to detect overflow

Summary

We can now implement combinational logic circuits

- Design each block
  - Binary encoded numbers for compactness
- Decompose large circuit into manageable blocks
  - 1-bit Half Adders, 1-bit Full Adders,
    n-bit Adders via cascaded 1-bit Full Adders, ...
- Can implement circuits using NAND or NOR gates
- Can implement gates using use PMOS and NMOS-transistors
- And can add and subtract numbers (in two's compliment)!
- Next time, state and finite state machines…