Numbers and Arithmetic

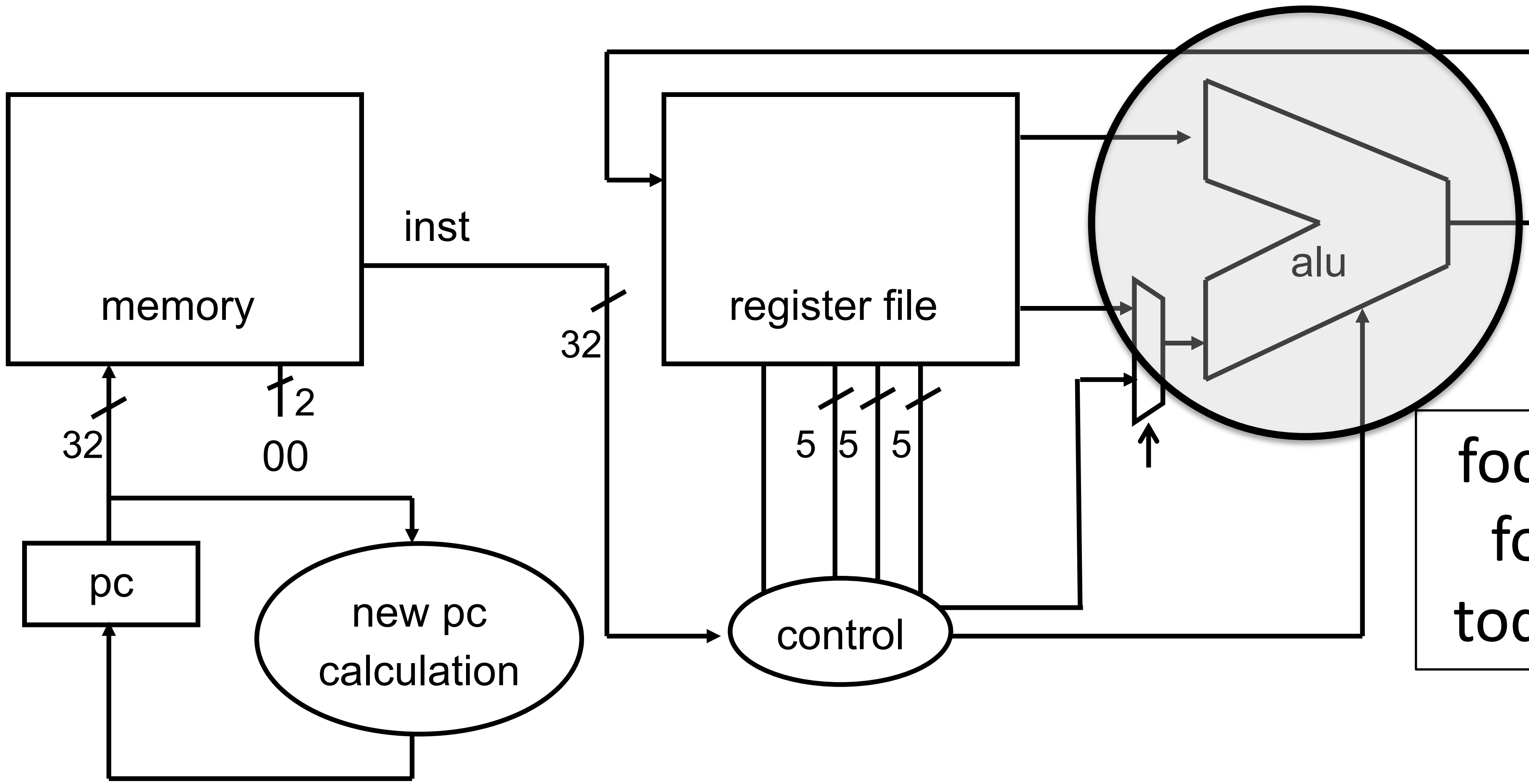
CS 3410

Computer Science

Cornell University

[K. Bala, A. Bracy, E. Sirer, and H. Weatherspoon]

Big Picture: Building a Processor



focus for today

Simplified 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)?

- We know represent numbers in Decimal (base 10).
  - E.g. $\underset{10^2}{\underline{6}}\ \underset{10^1}{\underline{3}}\ \underset{10^0}{\underline{7}}$ $\qquad 6 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 637$
- Can just as easily use other bases
  - Base 2 — Binary $\underset{2^9}{\underline{1}}\ \underset{2^8}{\underline{0}}\ \ \underset{2^7}{\underline{0}}\ \underset{2^6}{\underline{1}}\ \underset{2^5}{\underline{1}}\ \underset{2^4}{\underline{1}}\ \ \underset{2^3}{\underline{1}}\ \underset{2^2}{\underline{1}}\ \underset{2^1}{\underline{0}}\ \underset{2^0}{\underline{1}}$
  - Base 8 — Octal 0o $\underset{8^3}{\underline{1}}\ \underset{8^2}{\underline{1}}\ \underset{8^1}{\underline{7}}\ \underset{8^0}{\underline{5}}$ $\qquad 1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 = 637$
  - Base 16 — Hexadecimal 0x $\underset{16^2}{\underline{2}}\ \underset{16^1}{\underline{7}}\ \underset{16^0}{\underline{d}}$

Counting 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

0b 1111 1111 = 255

0b 1 0000 0000 = 256

0o 77 = 63

0o 100 = 64

0x ff = 255

0x 100 = 256

Converting between bases (10→8)

Base conversion via repetitive division

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

$637 \div 8 = 79$	remainder	5	lsb (least significant bit)
$79 \div 8 = 9$	remainder	7	
$9 \div 8 = 1$	remainder	1	
$1 \div 8 = 0$	remainder	1	msb (most significant bit)

637 = 0o 1175

msb lsb

Convert base 10 → base 2

Base conversion via repetitive division

Divide by base, write remainder, move left with quotient

$637 \div 2 = 318$	remainder	1	lsb (least significant bit)
$318 \div 2 = 159$	remainder	0	
$159 \div 2 = 79$	remainder	1	
$79 \div 2 = 39$	remainder	1	
$39 \div 2 = 19$	remainder	1	
$19 \div 2 = 9$	remainder	1	
$9 \div 2 = 4$	remainder	1	
$4 \div 2 = 2$	remainder	0	
$2 \div 2 = 1$	remainder	0	
$1 \div 2 = 0$	remainder	1	msb (most significant bit)

637 = 10 0111 1101 (or 0b10 0111 1101)

msb lsb

Clicker Question!

Convert the number 657_{10} to base 16
What is the least significant digit of this number?

a) D
b) F
c) 0
d) 1
e) 11

Convert from Binary to other powers of 2

Binary to Octal

- Convert groups of three bits from binary to oct
- 3 bits (000—111) have values 0…7 = 1 octal digit
- E.g. 0b 1001111101

  1 1 7 5 → 0o1175

Binary to Hexadecimal

- Convert nibble (group of four bits) from binary to hex
- Nibble (0000—1111) has values 0…15 = 1 hex digit
- E.g. 0b 1001111101

  2 7 d → 0x27d

Achievement Unlocked!

There are 10 types of people in the world:

Those who understand binary

And those who do not

And those who know this joke was written in base 3

Today's Lecture

Binary Operations

- Number representations
- One-bit and four-bit adders: THIS WEEK'S LAB
- Negative numbers and two's compliment
- Addition (two's compliment)
- Detecting and handling overflow
- Subtraction (two's compliment)

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

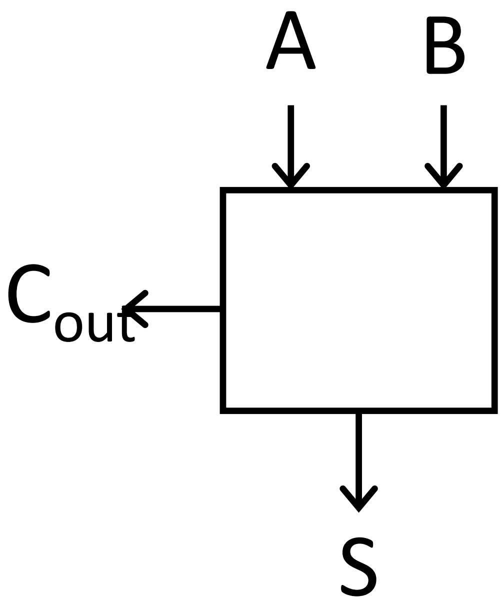
Carry-in, Carry-out

```
   111
 001110
+011100
-------
 101010
```

Unsigned binary addition is pretty easy

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

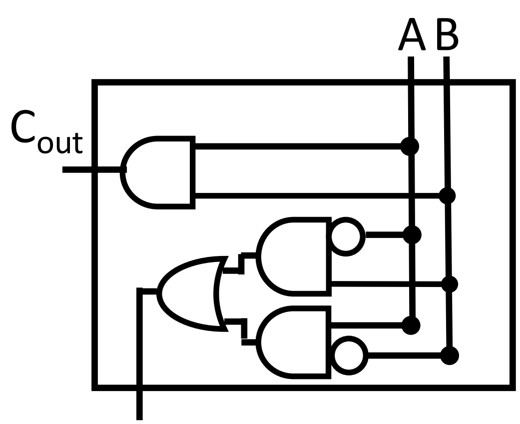
1-bit Half Adder

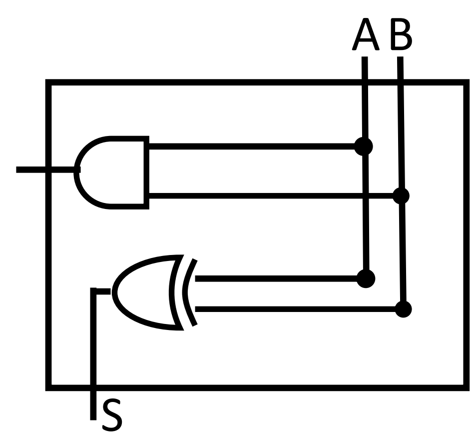


- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
- No carry-in

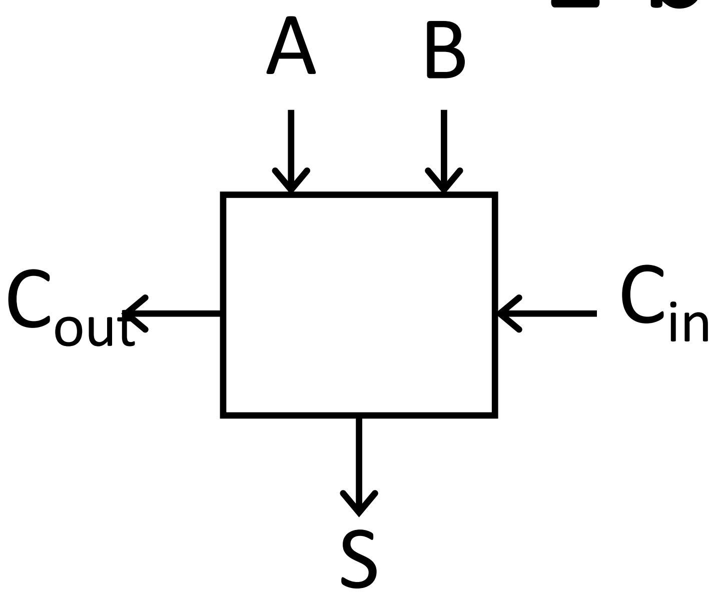
- $S = \overline{A}B + A\overline{B}$
- $C_{out} = AB$

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





1-bit (Full) Adder



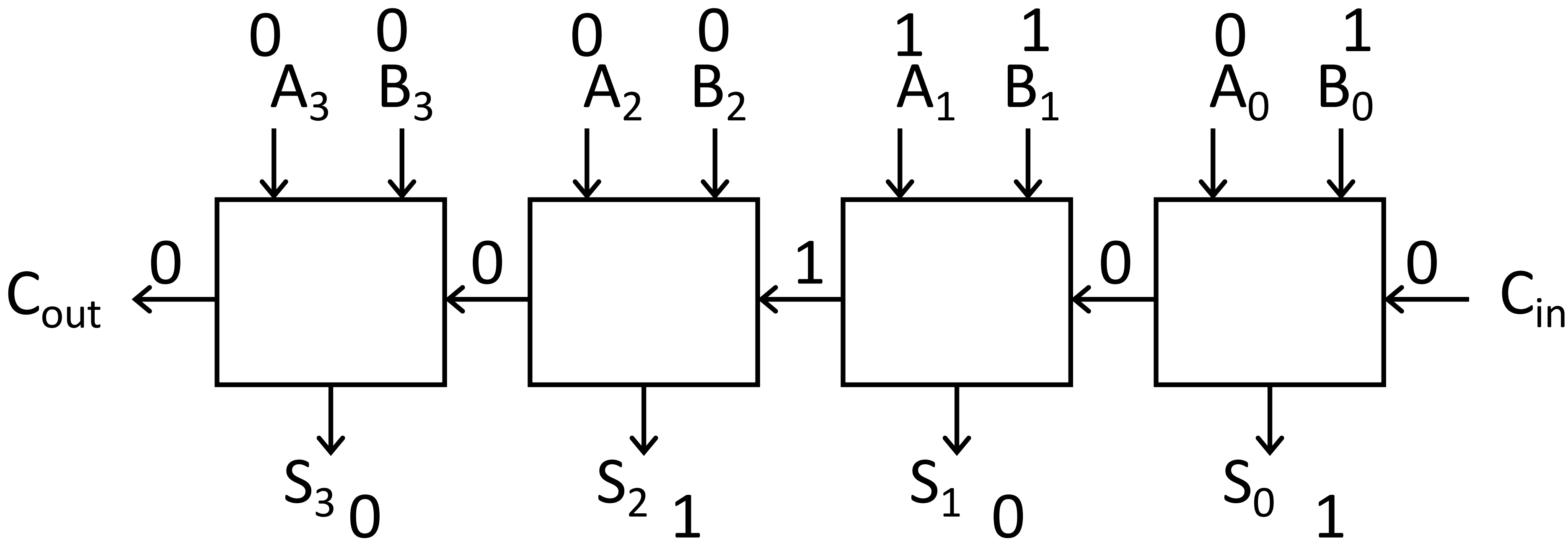
- Adds three 1-bit numbers
- Computes 1-bit result, 1-bit carry
- Can be cascaded

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		

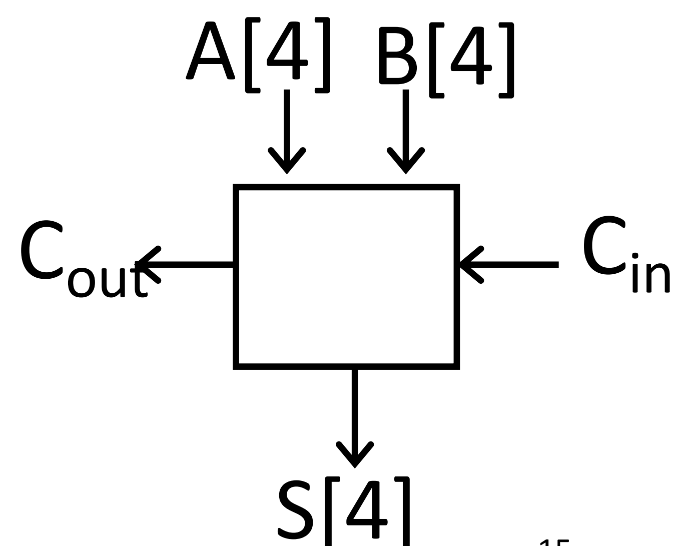
Now You Try (in Lab):

1. Fill in Truth Table
2. Create Sum-of-Product Form
3. Draw the Circuits

4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- Carry-out → result > 4 bits



Today's Lecture

Binary Operations

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

1st Try: Sign/Magnitude Representation

First Attempt: Sign/Magnitude Representation

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

<u>0</u>111 = 7

<u>1</u>111 = -7

Problem?

- 2 zero's: +0 different than -0

<u>0</u>000 = +0

<u>1</u>000 = -0

- Complicated circuits
- -2 + 1 = ???

IBM 7090, 1959:
"a second-generation transistorized version of the earlier IBM 709 vacuum tube mainframe computers"

Final Try: Two's Complement Representation

Positive 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 $\Rightarrow$ 0001 $\Rightarrow$ 1110 $\Rightarrow$ 1111
- -3: 3 $\Rightarrow$ 0011 $\Rightarrow$ 1100 $\Rightarrow$ 1101
- -8: 8 $\Rightarrow$ 1000 $\Rightarrow$ 0111 $\Rightarrow$ 1000
- -0: 0 $\Rightarrow$ 0000 $\Rightarrow$ 1111 $\Rightarrow$ 0000 (this is good, -0 = +0)

Two's Complement

Non-negatives	Negatives	
unchanged:	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 vs. Unsigned

4 bit Two’s Complement -8 … 7

4 bit Unsigned Binary 0 … 15

-1 =	1111	= 15
-2 =	1110	= 14
-3 =	1101	= 13
-4 =	1100	= 12
-5 =	1011	= 11
-6 =	1010	= 10
-7 =	1001	= 9
-8 =	1000	= 8
+7 =	0111	= 7
+6 =	0110	= 6
+5 =	0101	= 5
+4 =	0100	= 4
+3 =	0011	= 3
+2 =	0010	= 2
+1 =	0001	= 1
0 =	0000	= 0

Clicker Question!

What is the value of the 2s complement number 11010

a) 26
b) 6
c) -6
d) -10
e) -26

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 $\Rightarrow$ 0…255
- signed (two's complement): -(2^{N-1})…(2^{N-1} - 1)
  - E.g.: 8 bits $\Rightarrow$ (1000 0000) … (0111 1111)
  - -128 … 127

Sign Extension & Truncation

Extending to larger size (1st case on slide 23-24)

- 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 as usual. Ignore the sign. It just works!

Examples

1 + -1 =

-3 + -1 =

-7 + 3 =

7 + (-3) =

-1 =	1111	= 15
-2 =	1110	= 14
-3 =	1101	= 13
-4 =	1100	= 12
-5 =	1011	= 11
-6 =	1010	= 10
-7 =	1001	= 9
-8 =	1000	= 8
+7 =	0111	= 7
+6 =	0110	= 6
+5 =	0101	= 5
+4 =	0100	= 4
+3 =	0011	= 3
+2 =	0010	= 2
+1 =	0001	= 1
0 =	0000	= 0

Clicker Question

Which of the following has problems?

a) 7 + 1
b) -7 + -3
c) -7 + -1
d) Only A & B have problems
e) They all have problems.

Overflow

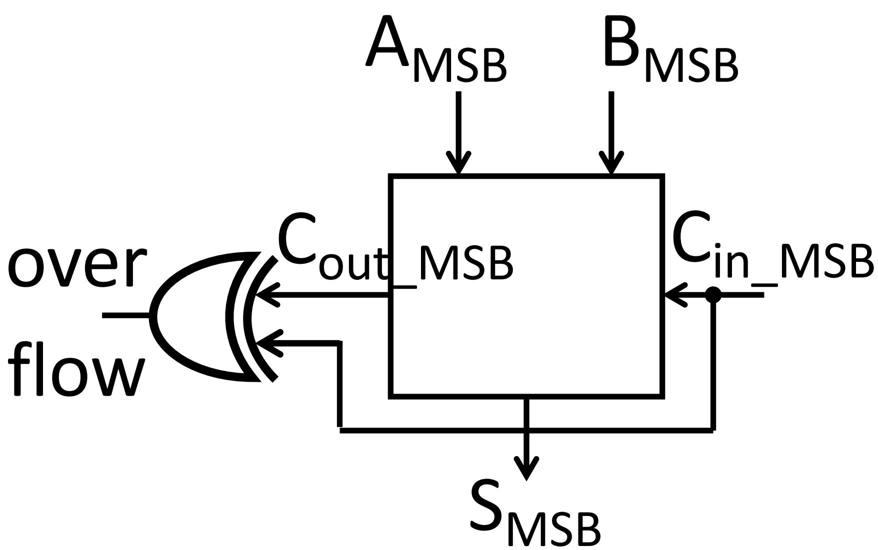
When can overflow occur?

- adding a negative and a positive?
  - Overflow *cannot occur* (Why?)
- adding two positives?
  - Overflow *can occur* (Why?)
- adding two negatives?
  - Overflow *can occur* (Why?)

-1 =	1111	= 15
-2 =	1110	= 14
-3 =	1101	= 13
-4 =	1100	= 12
-5 =	1011	= 11
-6 =	1010	= 10
-7 =	1001	= 9
-8 =	1000	= 8
+7 =	0111	= 7
+6 =	0110	= 6
+5 =	0101	= 5
+4 =	0100	= 4
+3 =	0011	= 3
+2 =	0010	= 2
+1 =	0001	= 1
0 =	0000	= 0

Overflow

When can overflow occur?



MSB

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

Wrong Sign (row 0 0 1 0 1)

Wrong Sign (row 1 1 0 1 0)

Rule of thumb:

- Overflow happened iff msb's carry in != carry out
- Intuition behind this rule??

Today’s Lecture

Binary Operations

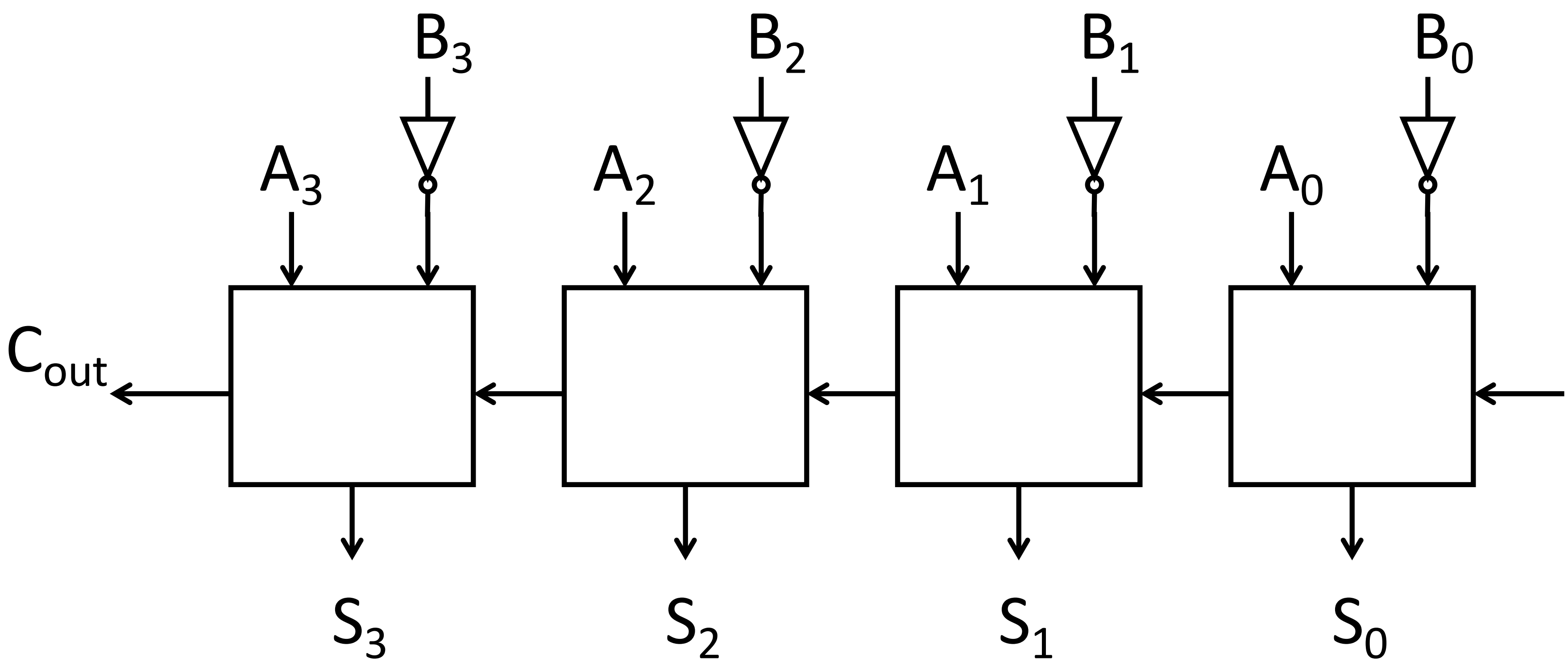
- Number representations
- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Detecting and handling overflow
- Subtraction (two’s compliment)
  - Why create a new circuit?
  - Just use addition using two’s complement math How?

Binary Subtraction

Two's Complement Subtraction

- Subtraction is addition with a negated operand
  - Negation is done by inverting all bits and adding one

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

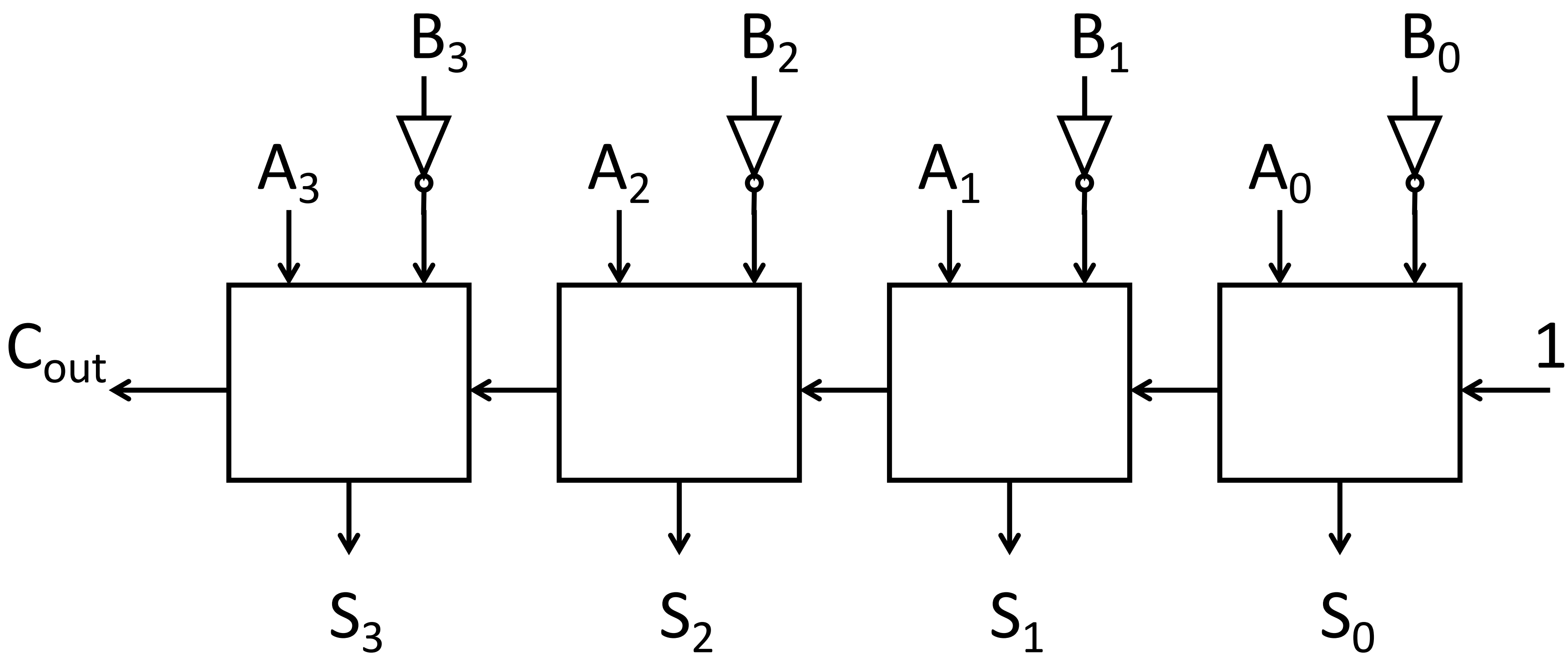


Binary Subtraction

Two's Complement Subtraction

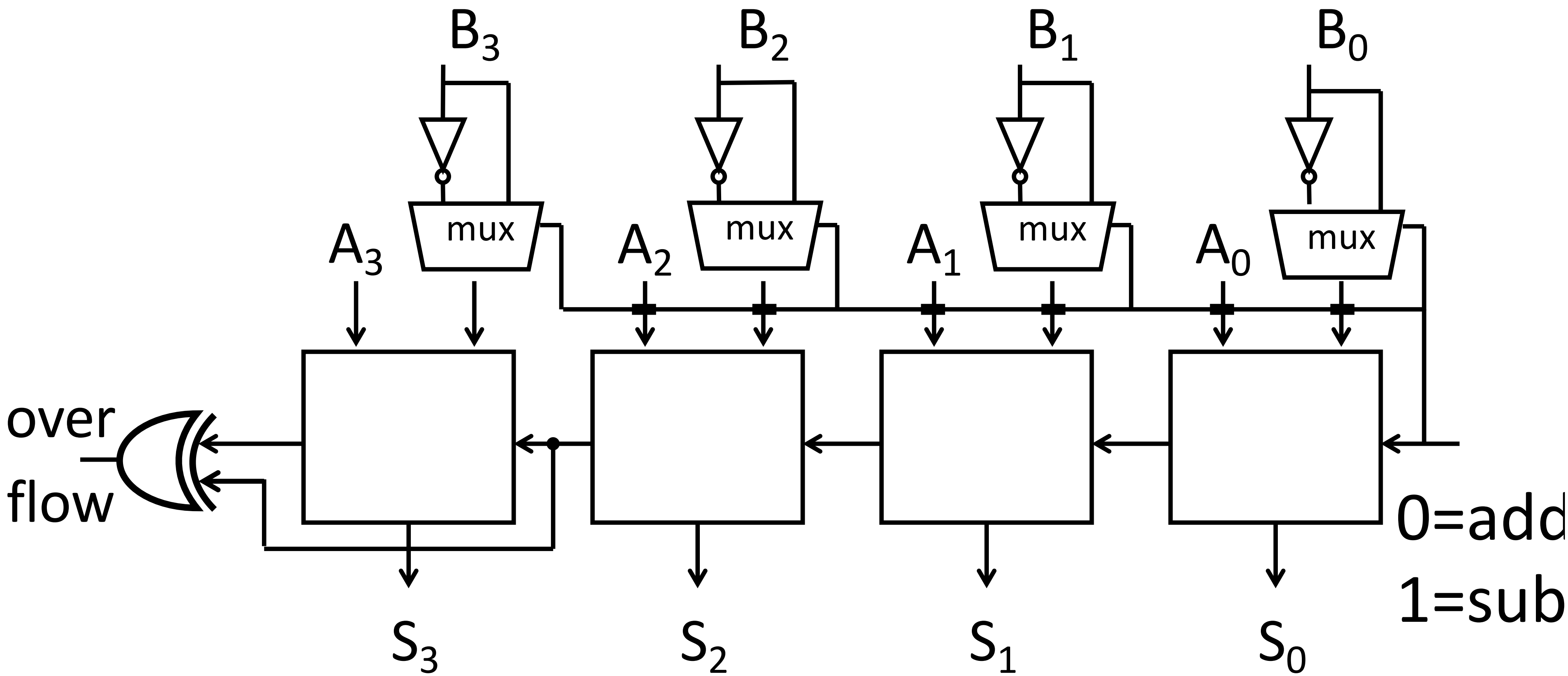
- Subtraction is addition with a negated operand
  - Negation is done by inverting all bits and adding one

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



Putting it all together

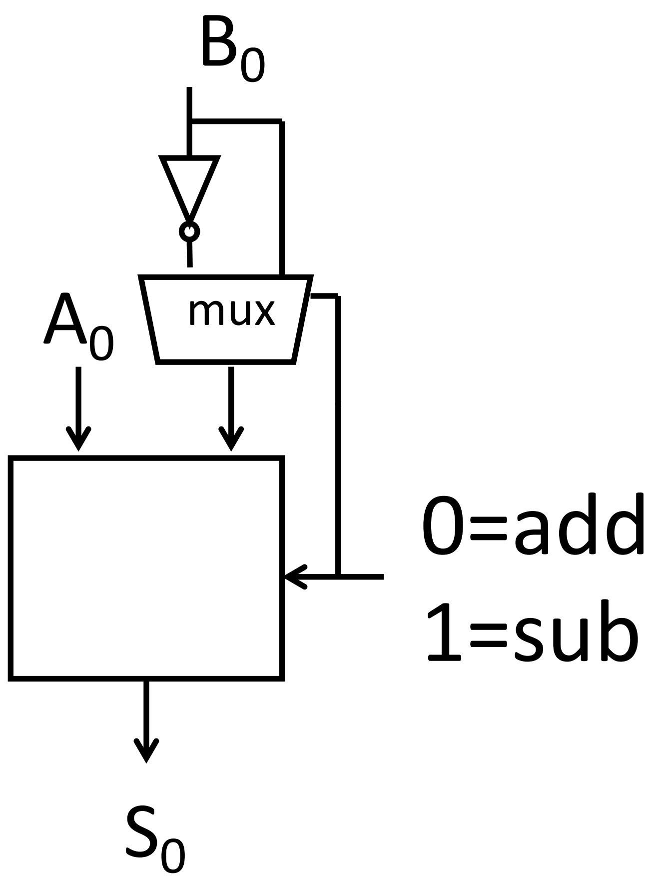
Two's Complement Adder with overflow detection

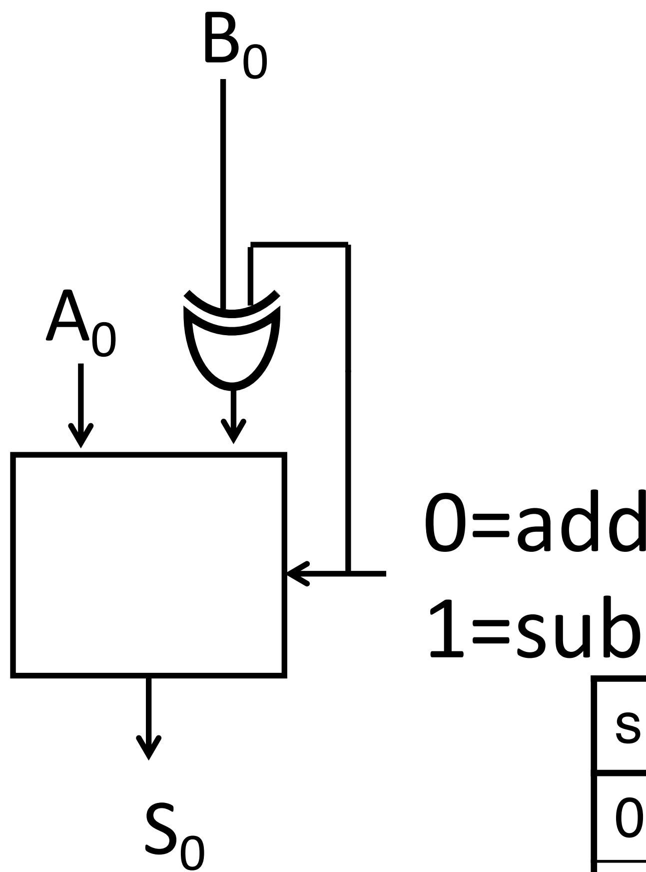


Note: 4-bit adder for illustrative purposes and may not represent the optimal design.

Put it together *better*

Two's Complement Adder with overflow detection





sub?	B_0	newB_0
0	0	0
0	1	1
1	0	1
1	1	0

Before: 2 inverters, 2 AND gates, 1 OR gate

After: 1 XOR gate

Takeaways

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

We write numbers as decimal or hex for convenience and need to be able to convert to binary and back (to understand what the 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 adding, where one operand is negated (two's complement; to negate: 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.

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)!