Numbers and Arithmetic

Anne Bracy

CS 3410

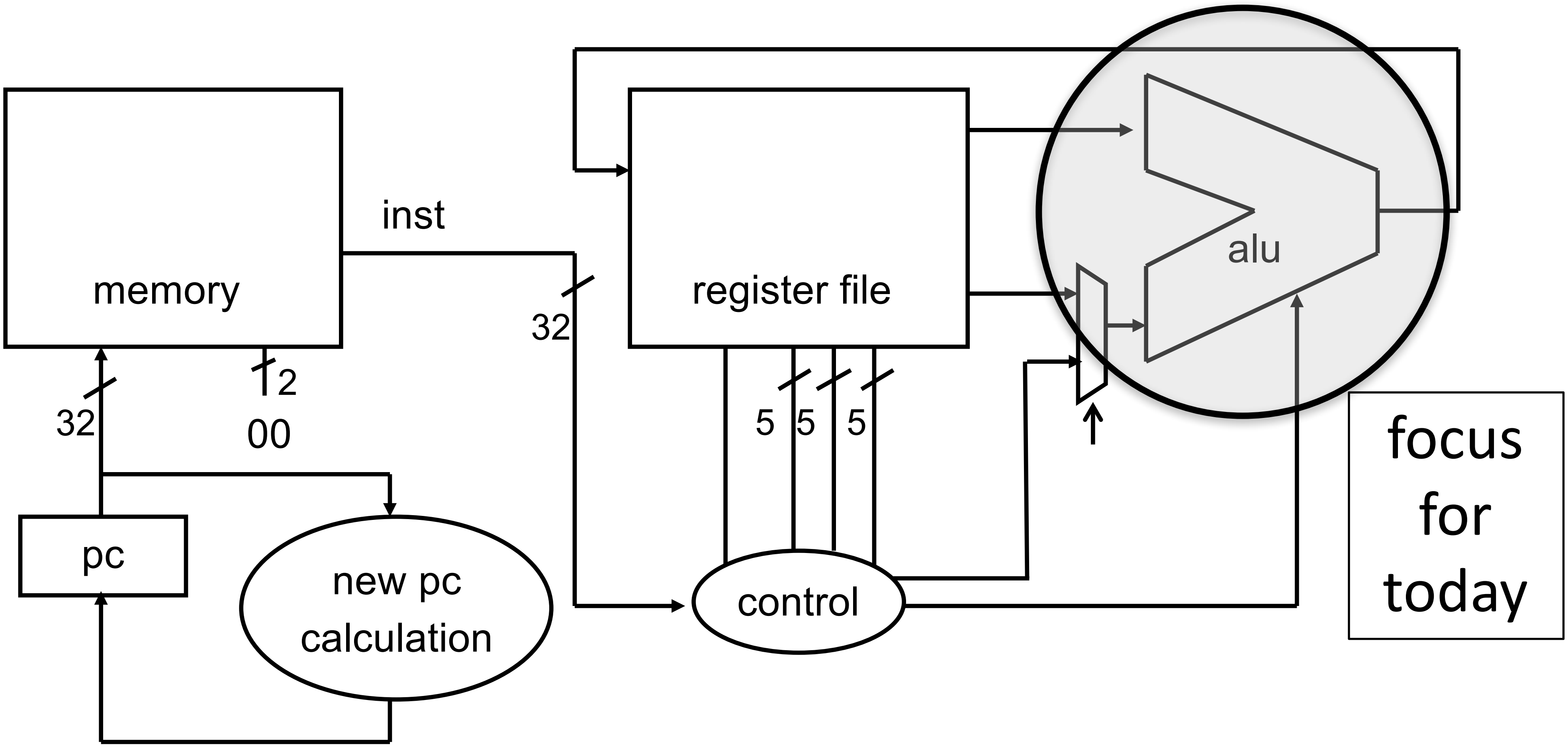
Computer Science

Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

See: Chapter 3 in your zybook

Big Picture: Building a Processor



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. $\underset{10^2}{\underline{6}}\ \underset{10^1}{\underline{3}}\ \underset{10^0}{\underline{7}}$ $\quad 6\cdot10^2 + 3\cdot10^1 + 7\cdot10^0 = 637$
- 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 $\underset{8^3}{\underline{1}}\ \underset{8^2}{\underline{1}}\ \underset{8^1}{\underline{7}}\ \underset{8^0}{\underline{5}}$ $\quad 1\cdot8^3 + 1\cdot8^2 + 7\cdot8^1 + 5\cdot8^0 = 637$
  - Base 16 — Hexadecimal 0x $\underset{16^2}{\underline{2}}\ \underset{16^1}{\underline{7}}\ \underset{16^0}{\underline{d}}$

Number Representations: Activity #1 Counting

How do we count in different bases?

- Dec (base 10) Bin (base 2) Oct (base 8) Hex (base 16)

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 =

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)

$637 = \underset{\text{msb}}{0\text{o}}\ 117\underset{\text{lsb}}{5}$

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)

$637 = {}_{\text{msb}}10\ 0111\ 110{1}_{\text{lsb}}$ (or 0b10 0111 1101)

Convert base 10 → base 16

Base conversion via repetitive division

Divide by base, write remainder, move left with quotient

$637 \div 16 = 39$	remainder	13	lsb
$39 \div 16 = 2$	remainder	7	
$2 \div 16 = 0$	remainder	2	msb

637 = 0x 2 7 13 = $\boxed{\quad ? \quad}$

Thus, 637 = 0x27d

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

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. 0b1001111101

  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. 0b1001111101

  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
- Negative numbers and two's compliment
- Addition (two's compliment)
- 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

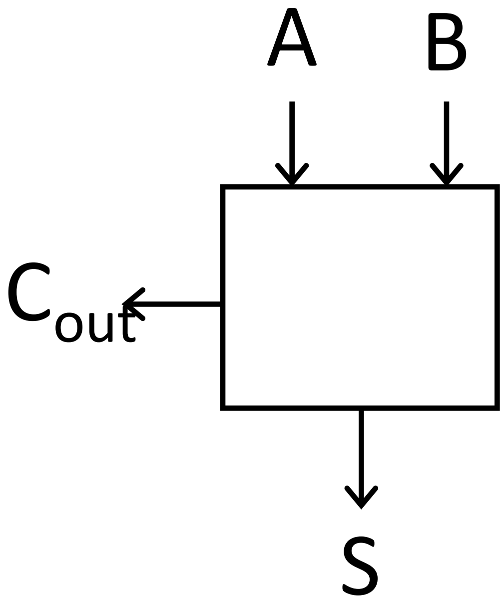
Carry-in Carry-out

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

Unsigned binary addition is pretty easy

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

1-bit Adder

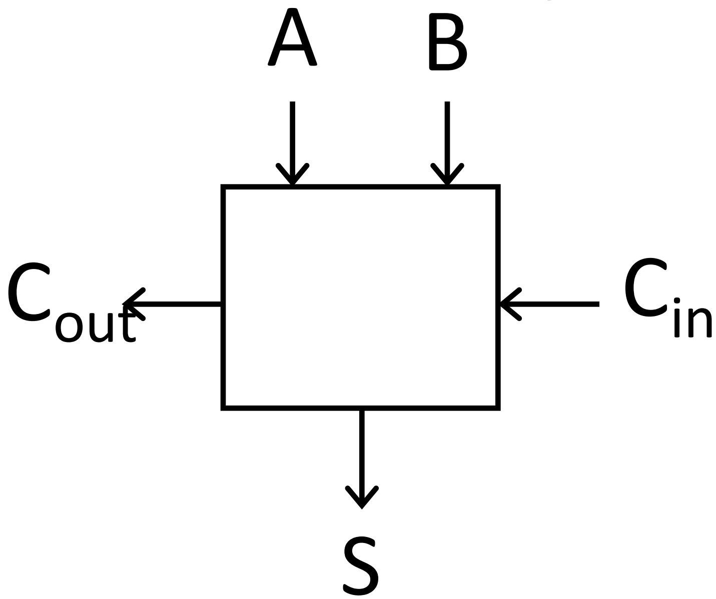


Half Adder

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

A	**B**	$\mathbf{C_{out}}$	**S**
0	0		
0	1		
1	0		
1	1		

1-bit Adder with Carry



Full Adder

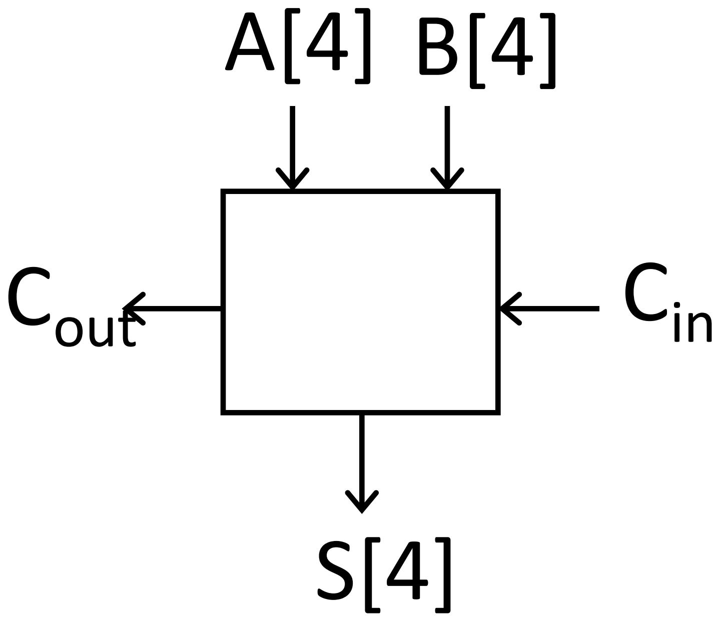
- Adds three 1-bit numbers
- Computes 1-bit result, 1-bit carry
- Can be cascaded

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:

1. Fill in Truth Table
2. Create Sum-of-Product Form
3. Minimize the equation
   - K-Maps
   - Algebraic Minimization
4. Draw the Circuits

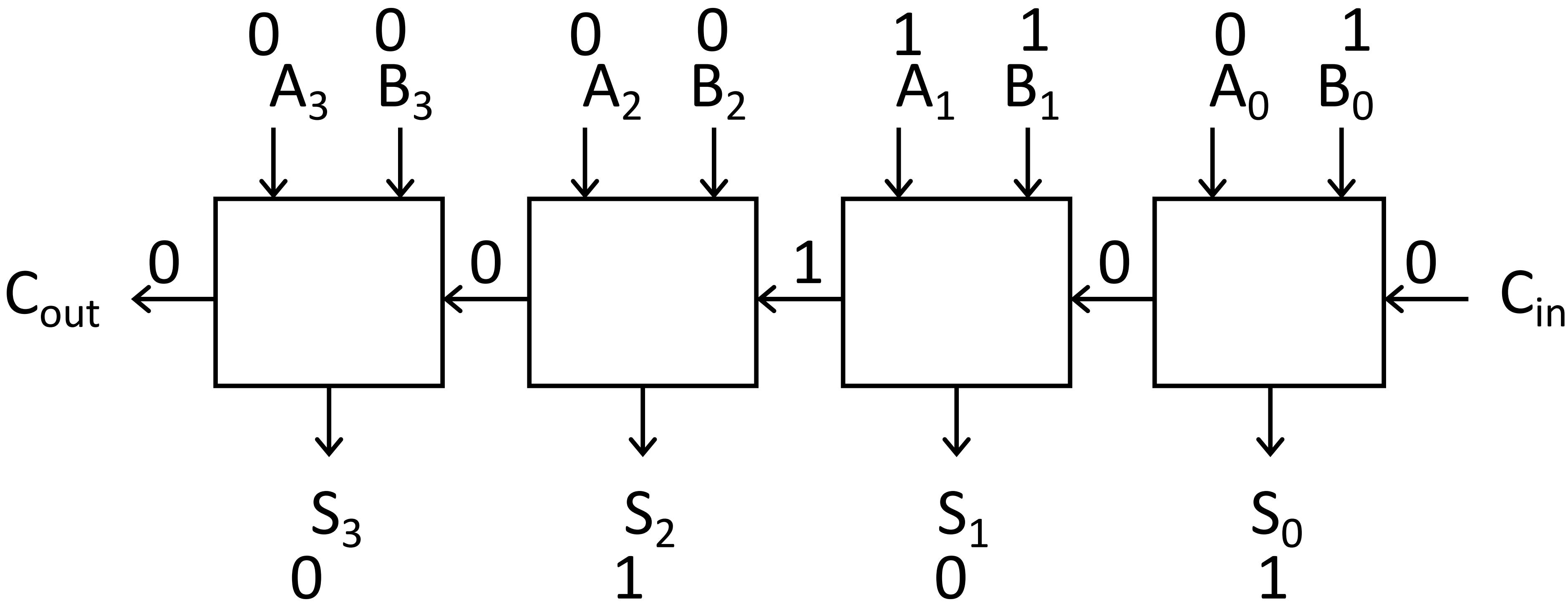
4-bit Adder



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 = result does not fit in 4 bits

iClicker Question

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)

1st Try: Sign/Magnitude Representation

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
- -2 + 1 = ???

0000 = +0

1000 = -0

IBM 7090

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
- -7: 7 $\Rightarrow$ 0111 $\Rightarrow$ 1000 $\Rightarrow$ 1001
- -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 (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 vs. Unsigned

4 bit Two's Complement -8 … 7

4 bit Two's Complement	Binary	4 bit Unsigned Binary
-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

4 bit Unsigned Binary 0 … 15

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 $0 \ldots 2^N-1$
  - eg: 8 bits $\Rightarrow$ 0…255
- signed (two's complement): $-(2^{N-1}) \ldots (2^{N-1} - 1)$
  - E.g.: 8 bits $\Rightarrow$ (1000 0000) … (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 as usual, ignore the sign (it just works)

Examples

1 + -1 =

-3 + -1 =

-7 + 3 =

7 + (-3) =

What is wrong with the following additions?

7 + 1   -7 + -3   -7 + -1

Binary Subtraction

Why create a new circuit?

Just use addition using two's complement math

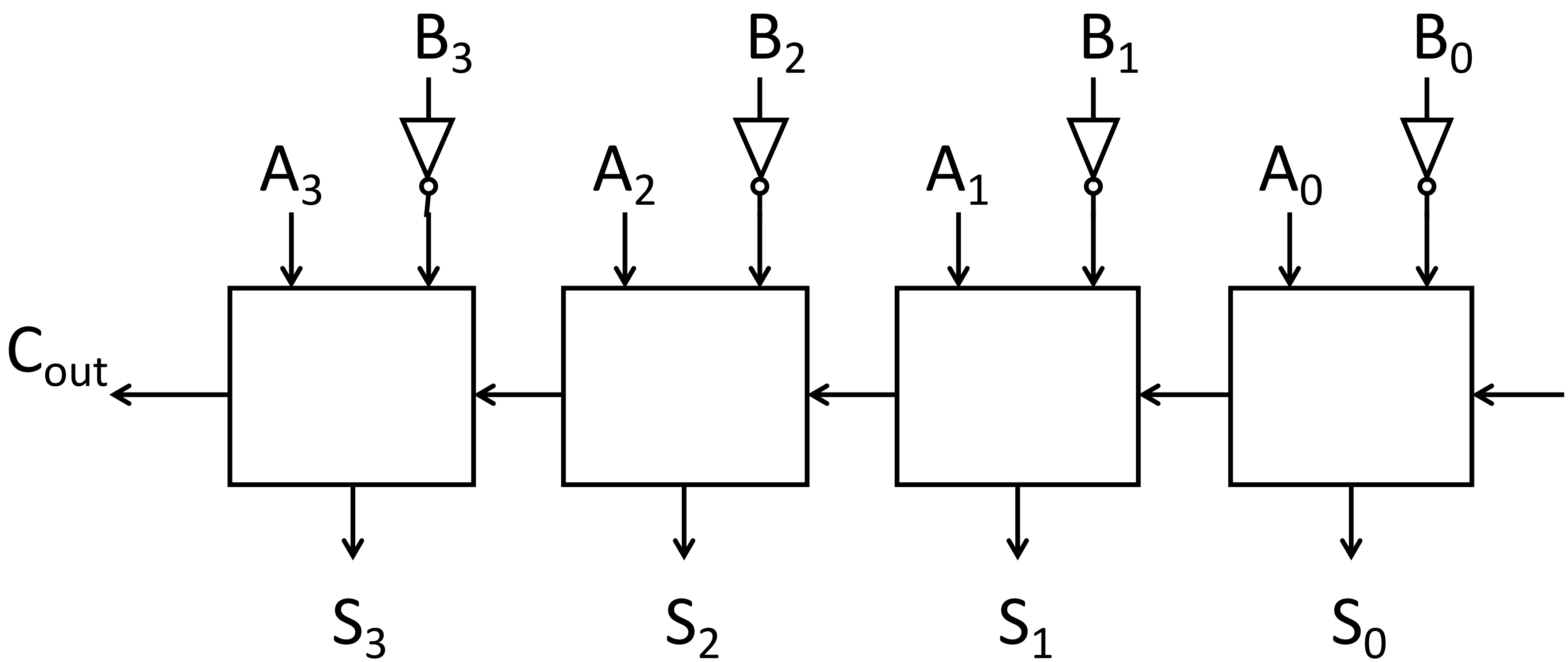
- How?

Binary Subtraction

Two's Complement Subtraction

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

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

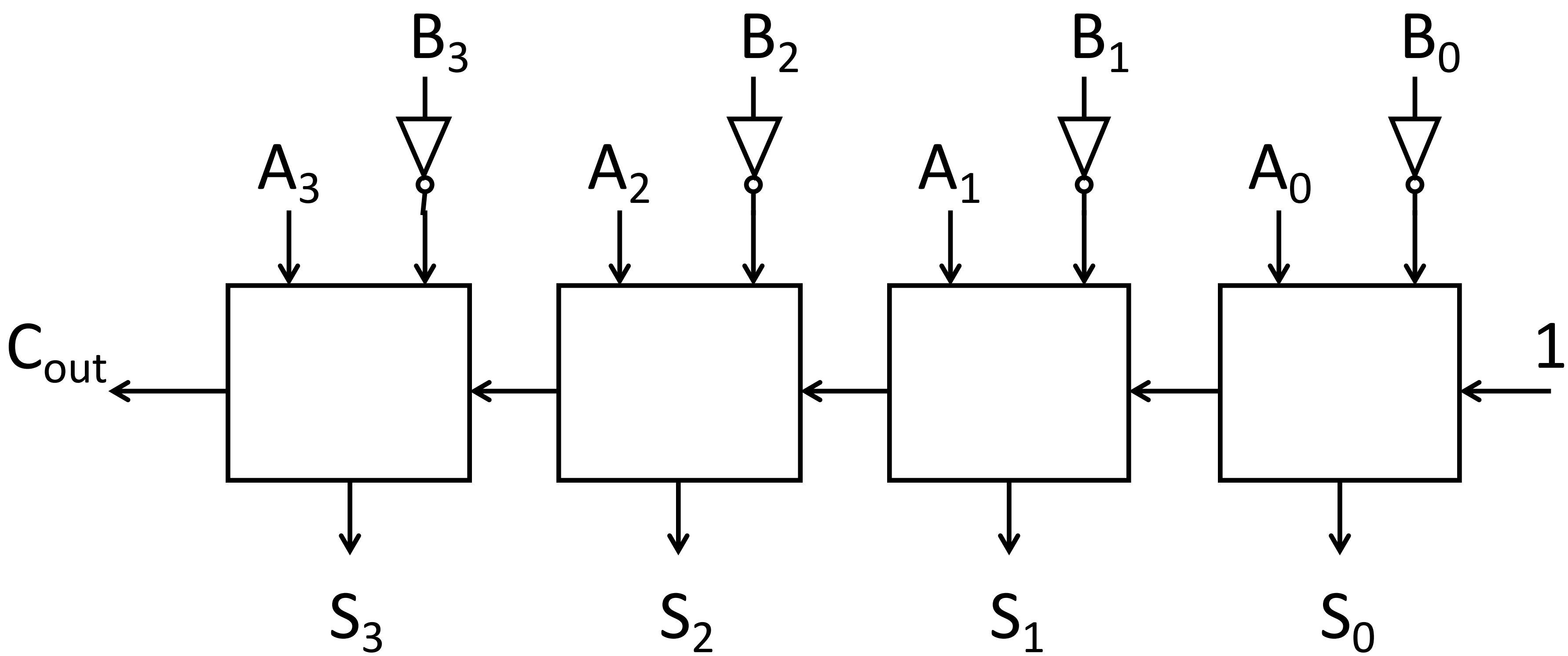


Binary Subtraction

Two's Complement Subtraction

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

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



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)
- Detecting and handling overflow

Overflow

When can overflow occur?

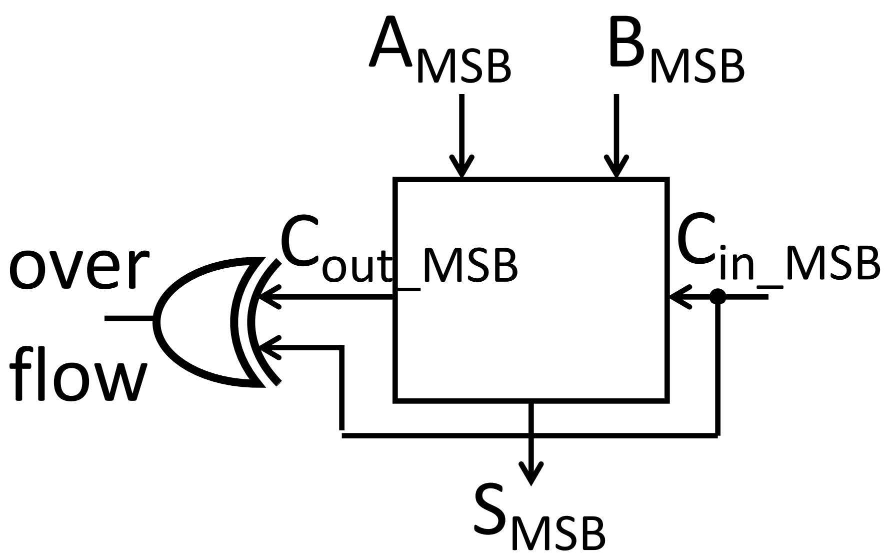
- adding a negative and a positive?

- adding two positives?

- adding two negatives?

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

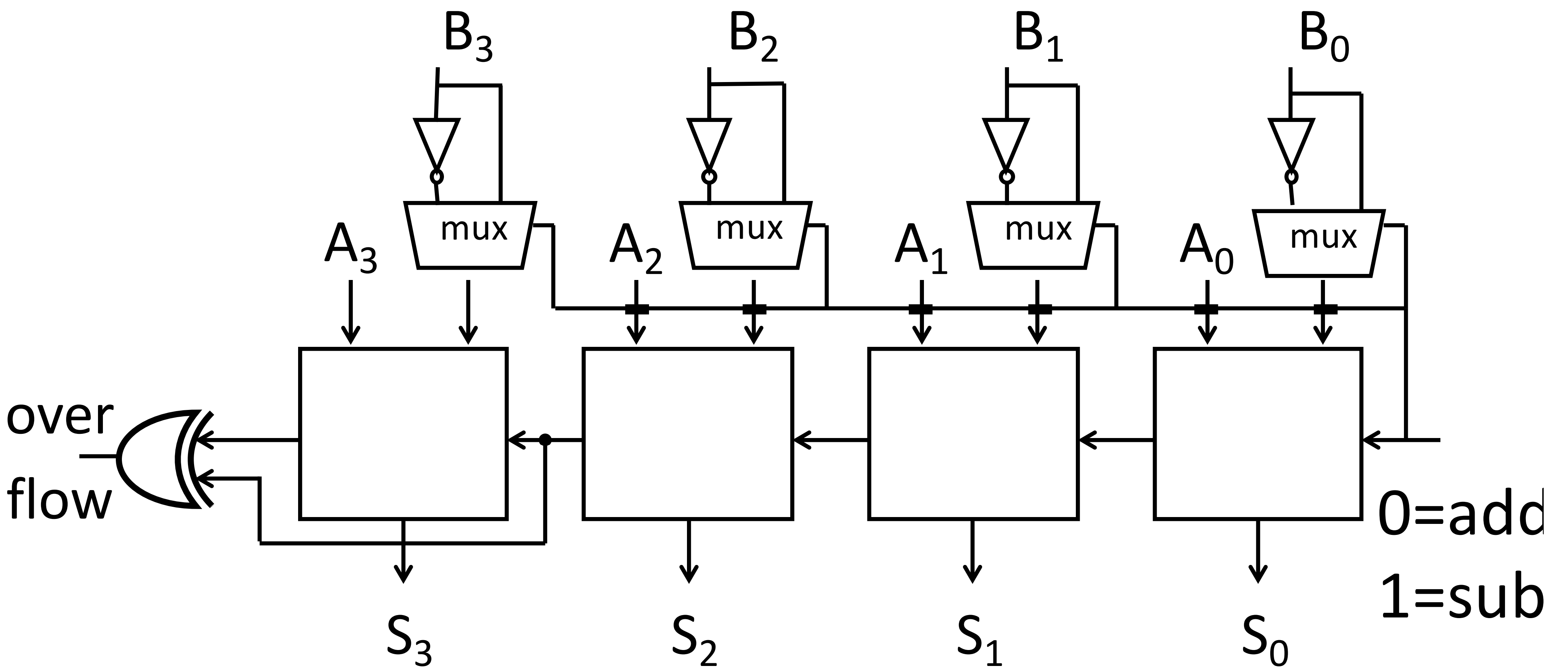
Wrong Sign

Rule of thumb:

- Overflow happened iff msb's carry in != carry out

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.

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)!
- Next time, state and finite state machines…