Last Time

Sequential Logic

- D Flip Flops
- Shift registers
- Ripple Counters

Binary number system

Today

- Addition
- Representing negative numbers
- Multiplication with shift registers
- Arithmetic logic units

Binary Addition

Consider the following binary numbers:

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

How do we add these numbers?

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

1

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 1

And we have a carry now!

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 0 1

And we have a carry again!

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 0 0 1

and again!

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

1 0 0 0 1

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 1 0 0 0 1

One more carry!

Binary Addition

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 1 0 1 0 0 0 1

Binary Addition

Behaves just like addition in decimal, but:

- We carry to the next digit any time the sum of the digits is 2 (decimal) or greater

Implementing Binary Addition

How do we implement this in a circuit?

0 0 1 0 0 1 1 0

0 0 1 0 1 0 1 1

0 1 0 1 0 0 0 1

Let's focus on adding a single pair of digits

Implementing Binary Addition

Truth table

- A & B are the input binary values
- C_{in} is the carry from the previous digit
- Out is the output value for this digit
- C_{out} is the carry output to the next digit

C_{in}	A	B		C_{out}	Out
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

Output Value K-Map

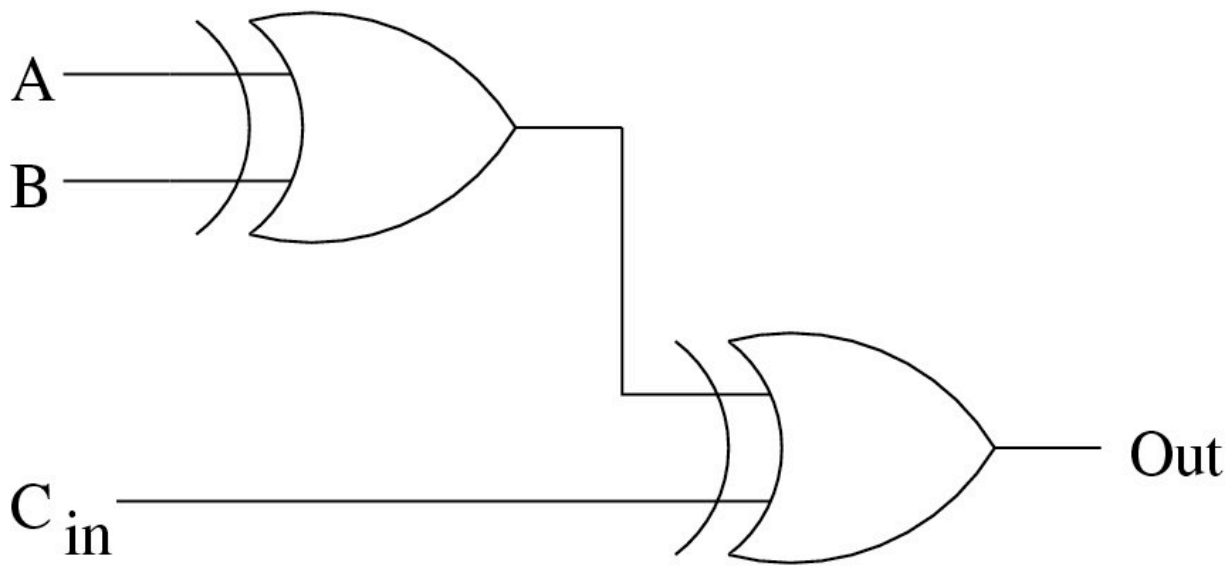
B A

C \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

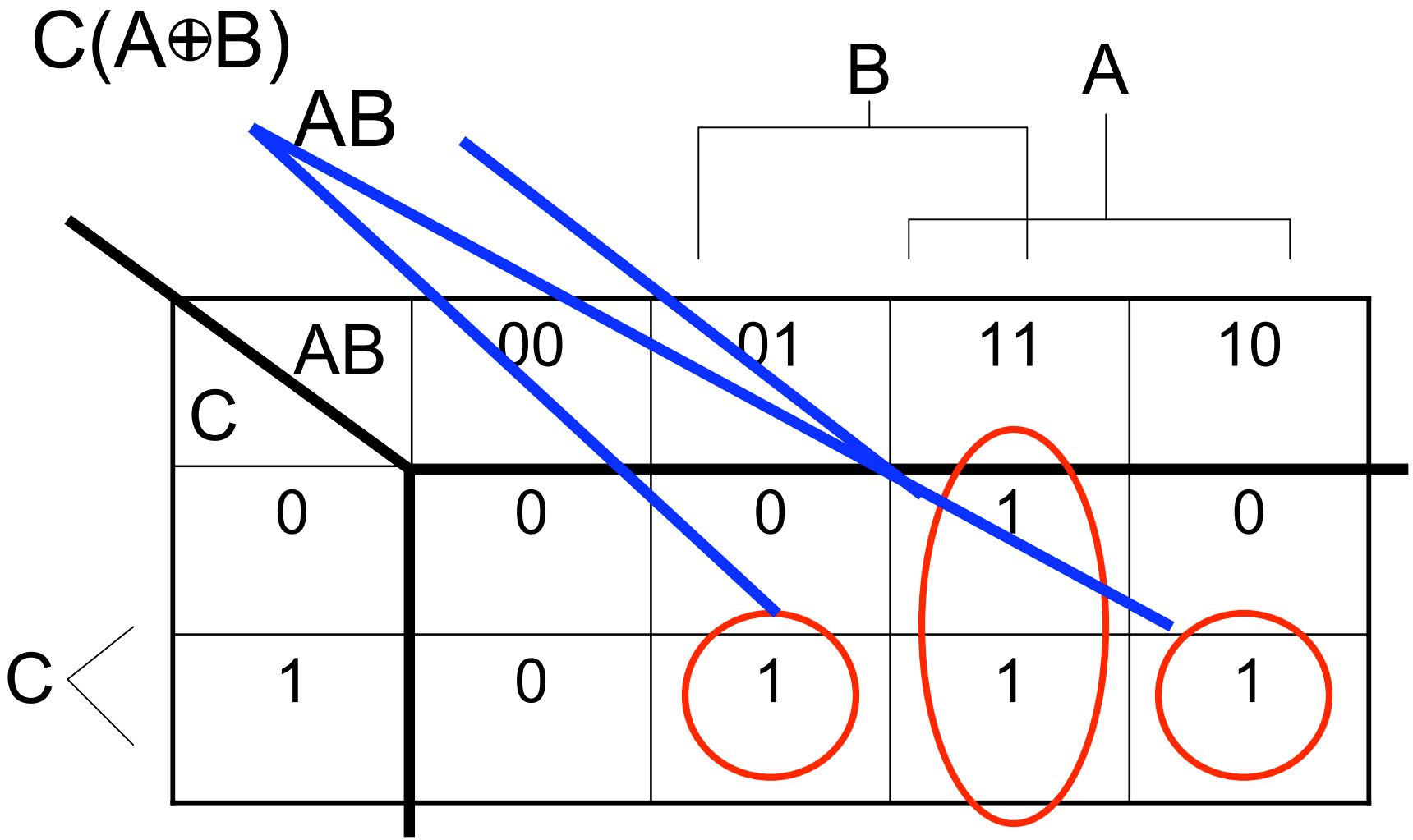
C

Output Value Implementation

$$Out = A \oplus B \oplus C_{in}$$



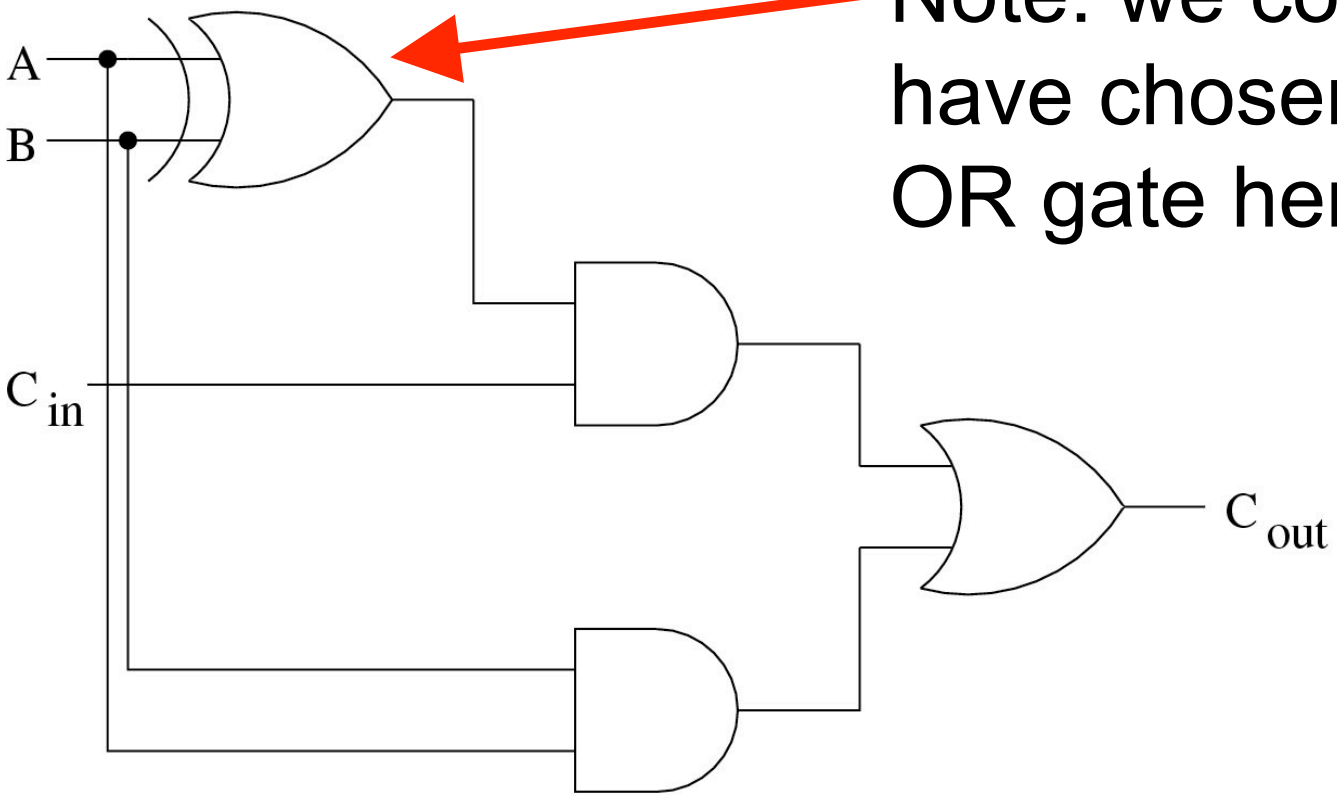
C_{out} K-Map



C_{out} Implementation

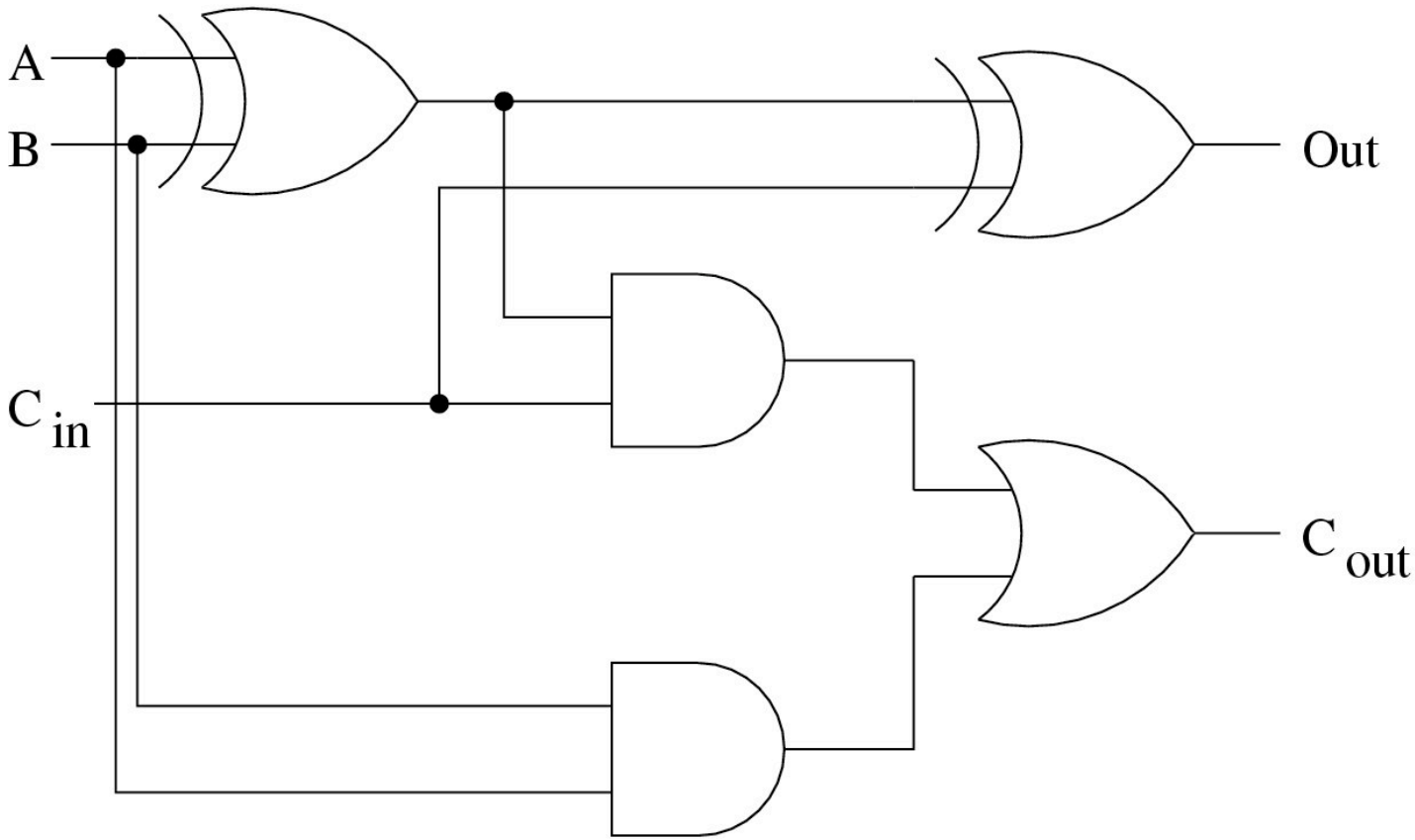
$$C_{out} = (A \oplus B)C_{in} + AB$$

Note: we could have chosen an OR gate here

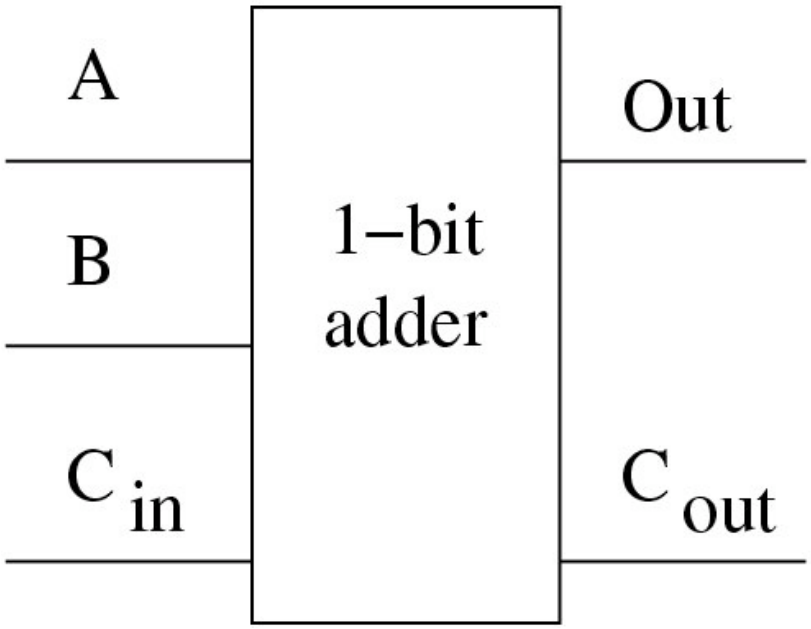


Complete Implementation

The use of the XOR allows us to share a gate between the two sides



The Final Product: An Adder "Black Box"

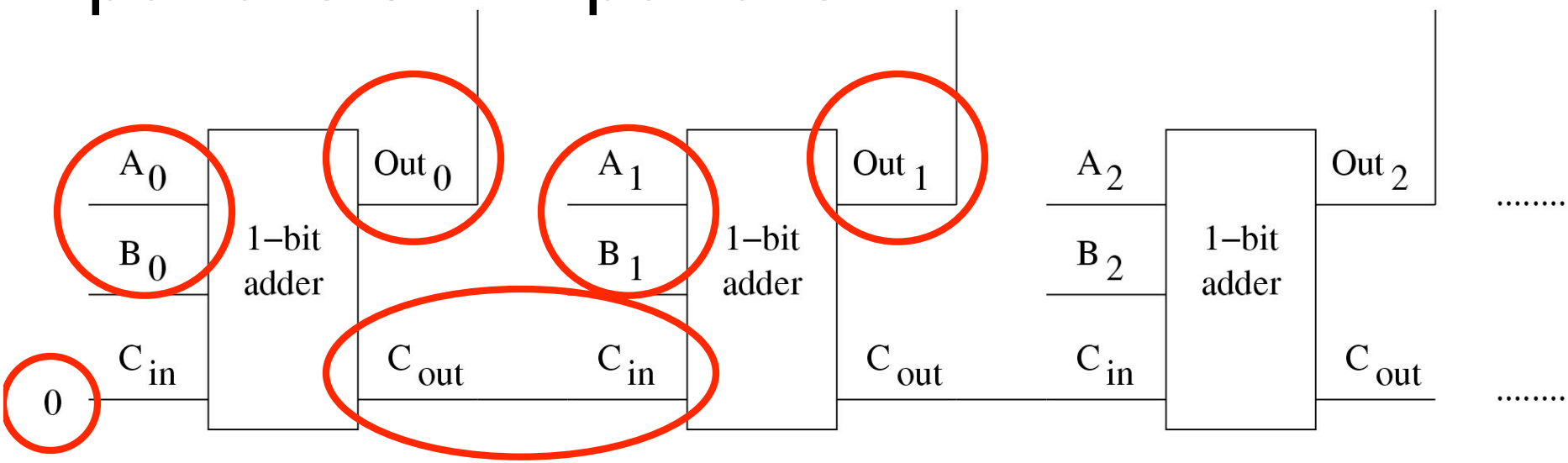


Now how do we construct an N-bit adder?

N-Bit Adder

Output bit 0 Output bit 1

Input bits 0 Input bits 1



1st carry is always zero

Carry propagates from one level to the next

Negative Numbers

So far we have only talked about representing non-negative integers

- What can we add to our binary representation that will allow this?

Representing Negative Numbers

One possibility:

- Add an extra bit that indicates the sign of the number
- We call this the “sign-magnitude” representation

Sign Magnitude Representation

+12 0 0 0 0 1 1 0 0

Sign Magnitude Representation

+12	0 0 0 0 1 1 0 0
-12	1 0 0 0 1 1 0 0

Sign Magnitude Representation

+12	0 0 0 0 1 1 0 0
-12	1 0 0 0 1 1 0 0

What is the problem with this approach?

Sign Magnitude Representation

What is the problem with this approach?

- Some of the arithmetic operators that we have already developed do not do the right thing

Sign Magnitude Representation

Operator problems:

- For example, we have already designed a counter (that implements an ‘increment’ operation)

-12 1 0 0 0 1 1 0 0

Sign Magnitude Representation

Operator problems:

-12 1 0 0 0 1 1 0 0

Increment

Sign Magnitude Representation

Operator problems:

-12 1 0 0 0 1 1 0 0

Increment

1 0 0 0 1 1 0 1

Sign Magnitude Representation

Operator problems:

-12 1 0 0 0 1 1 0 0

Increment

-13 1 0 0 0 1 1 0 1

!!!!

Representing Negative Numbers

An alternative:

- When taking the additive inverse of a number, invert all of the individual bits
- The leftmost bit still determines the sign of the number

One’s Complement Representation

12 0 0 0 0 1 1 0 0

Invert

-12 1 1 1 1 0 0 1 1

One's Complement Representation

12 → -12

0 0 0 0 1 1 0 0

↓ Invert

1 1 1 1 0 0 1 1

↓ Increment

1 1 1 1 0 1 0 0

One's Complement Representation

12 0 0 0 0 1 1 0 0

Invert

-12 1 1 1 1 0 0 1 1

Increment

-11 1 1 1 1 0 1 0 0

One’s Complement Representation

What problems still exist?

One's Complement Representation

What problems still exist?

- We have two distinct representations of 'zero':

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

One's Complement Representation

What problems still exist?

- We can't directly add a positive and a negative number:

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 0

One’s Complement Representation

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 0
	↓
	0 0 0 0 0 1 1 0

One's Complement Representation

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 0
6	0 0 0 0 0 1 1 0

!!!!

Representing Negative Numbers

An alternative:

(a little intuition first)

0 0 0 0 0 0 0 0

Decrement

Representing Negative Numbers

An alternative:

(a little intuition first)

0 0 0 0 0 0 0 0

Decrement

1 1 1 1 1 1 1 1

Representing Negative Numbers

An alternative:

(a little intuition first)

0 0 0 0 0 0 0 0

Decrement

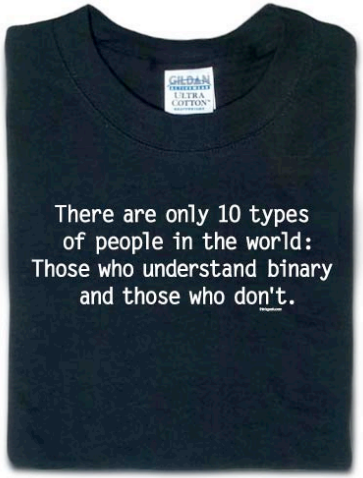
Define this as

-1 1 1 1 1 1 1 1 1

Last Time

Binary Numbers

- Conversion to/from decimal
- Addition (and circuits for addition)
- Representing negative numbers
  - Sign magnitude
  - One's complement
  - Two's complement



Today

- Finish 2s complement
- Subtraction
- Multiplication
- Arithmetic logical units
- Start on computer architectures
  - Memory systems

Administrivia

- My office hours are canceled Wednesday morning (afternoon will stay the same)
- Homework 1 is due on Thursday @ 5:00

- Power supply module for your digital circuits

Representing Negative Numbers

A few more numbers:

3	0 0 0 0 0 0 1 1
2	0 0 0 0 0 0 1 0
1	0 0 0 0 0 0 0 1
0	0 0 0 0 0 0 0 0
-1	1 1 1 1 1 1 1 1
-2	1 1 1 1 1 1 1 0
-3	1 1 1 1 1 1 0 1

Two's Complement Representation

In general, how do we take the additive inverse of a binary number?

Two's Complement Representation

In general, how do we take the additive inverse of a binary number?

- Invert each bit and then add '1'

Two's Complement Representation

Invert each bit and then add '1'

5 0 0 0 0 0 1 0 1

Two's complement

-5 1 1 1 1 1 0 1 1

Two's Complement Representation

Now: let's try adding a positive and a negative number:

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 1

Two's Complement Representation

Now: let's try adding a positive and a negative number:

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 1
	0 0 0 0 0 1 1 1

Two's Complement Representation

Now: let's try adding a positive and a negative number:

12	0 0 0 0 1 1 0 0
+	+
-5	1 1 1 1 1 0 1 1
7	0 0 0 0 0 1 1 1

Two’s Complement Representation

Two’s complement is used for integer representation in today’s processors

Two’s Complement Representation

Two’s complement is used for integer representation in today’s processors

One oddity: we can represent one more negative number than we can positive numbers

Implementing Subtraction

How do we implement a ‘subtraction’ operator?

(e.g., A – B)

Implementing Subtraction

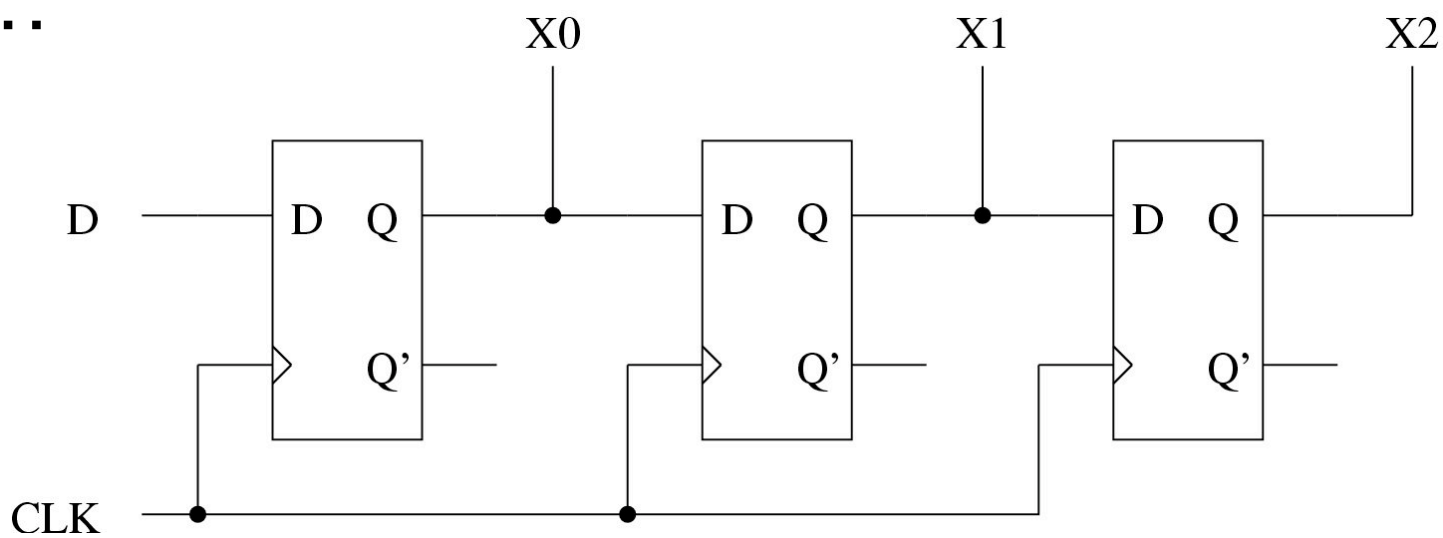
How do we implement a ‘subtraction’ operator?

(e.g., A – B)

- Take the 2s complement of B
- Then add this number to A

Shift to a Related Topic

Recall the shift register we designed last week…



- What arithmetic operator does this implement?

Shift Registers

By shifting all of the bits in a number to the "left", we multiply the number by 2

There is a caveat here: what is it?

Shift Registers

What is the caveat?

- If we are representing a negative number, we cannot shift the most significant bit (MSB)
  - This could change the sign of the number

Shift Registers

What is the caveat?

- In either case (signed or unsigned number), if we shift a ‘1’ off the end then we exceed the representational capability of the set of bits

- We force the programmer to plan for this

Shift Registers

What happens when we shift to the right instead of the left?

Shift Registers

What happens when we shift to the right instead of the left?

- We divide the number by 2 (dropping the remainder)

Implementing Multiplication

- Multiplying arbitrary integers is more complicated
- One technique: “unroll” the multiplication into a sequence of adds

Implementing Multiplication

- One technique: “unroll” the multiplication into a sequence of adds:
  - 5 * 9 becomes:
  - 9 + 9 + 9 + 9 + 9

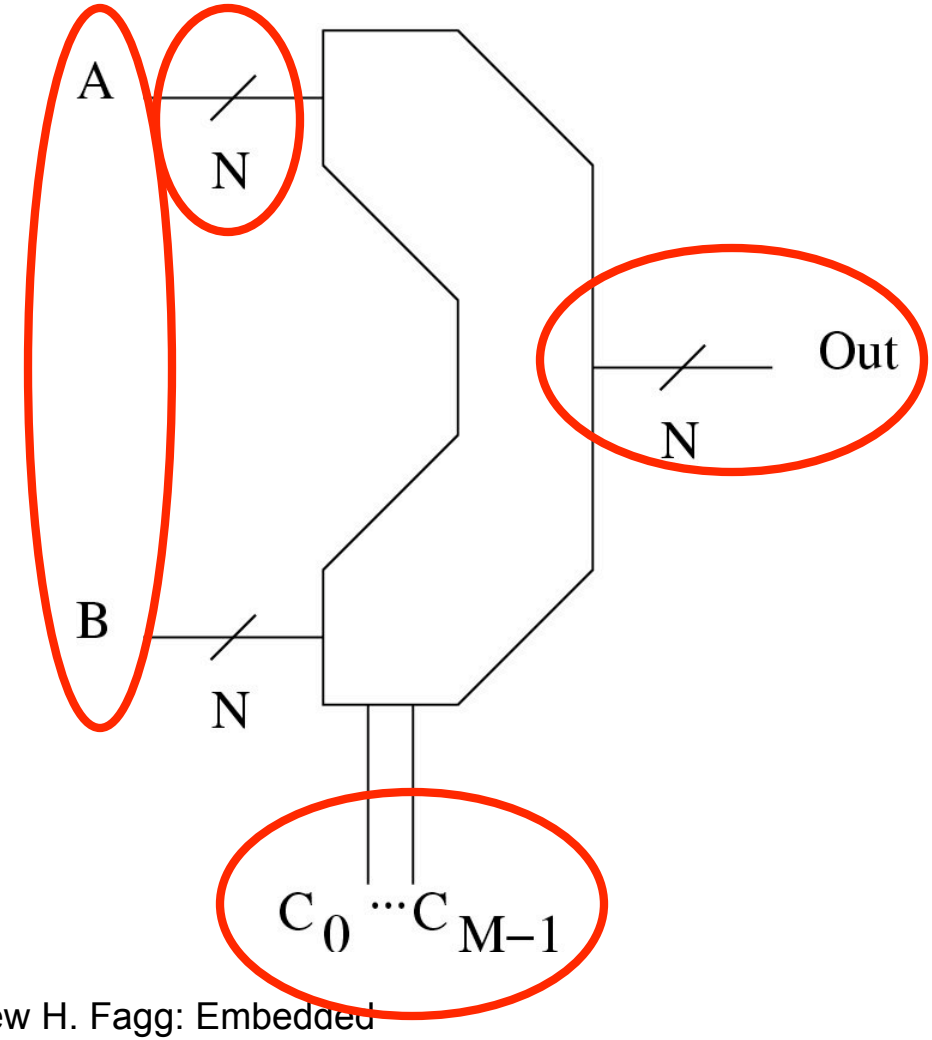
A General Arithmetic Logic Unit (ALU)

Two input parameters

N-bits wide (so N wires)

Output

Control lines determine the operator



ALU Operators

Common operators include:

- Addition
- Subtraction
- Bit-wise OR / AND / XOR / NOT
- Additive inverse
- Shift left/right by K bits

Some of these operators only use one parameter

Next Time

- A little more arithmetic
- Processor design
  - Registers/memory and addressing
  - Data/address busses
  - Machine-level instructions

Other Useful Number Systems

You already know:

- Decimal – base 10
- Binary – base 2

Other Useful Number Systems

You already know:

- Decimal – base 10
- Binary – base 2

But it is common to also see:

- Octal – base 8
- Hexadecimal – base 16

Other Number Systems

Decimal	Binary	Octal	Hex
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Decimal	Binary	Octal	Hex
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

Binary to Hex Conversion

What is the hex equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1

Binary to Hex Conversion

What is the hex equivalent of:

0 1 1 | 0 0 0 1 | 1 1 0 0 | 1 0 0 1

Partition the binary digits into groups of four – starting from the right-hand-side

Binary to Hex Conversion

What is the hex equivalent of:

0 1 1 | 0 0 0 1 | 1 1 0 0 | 1 0 0 1

3 1 C 9

Convert the individual groups

Binary to Hex Conversion

In C notation (the programming language), we will write:

0x31C9

Binary to Octal Conversion

What is the octal equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1

Binary to Octal Conversion

What is the octal equivalent of:

0 1 1 | 0 0 0 | 1 1 1 | 0 0 1 | 0 0 1

Partition the binary digits into groups of **three** – starting from the right-hand-side

Binary to Octal Conversion

What is the octal equivalent of:

0 1 1 | 0 0 0 | 1 1 1 | 0 0 1 | 0 0 1

3 0 7 1 1

Convert the individual groups

Binary to Octal Conversion

In C notation (the programming language), we will write:

030711

Octal or Hex to Binary

How do we perform this type of conversion?

Octal or Hex to Binary

How do we perform this type of conversion?

- For each octal or hex digit, convert to the binary equivalent (3 or 4 binary digits, respectively)
- Append the binary digits together

Binary Notation in C

How would we write a binary constant in C?

Binary Notation in C

How would we write a binary constant in C?

0b0110001111001001