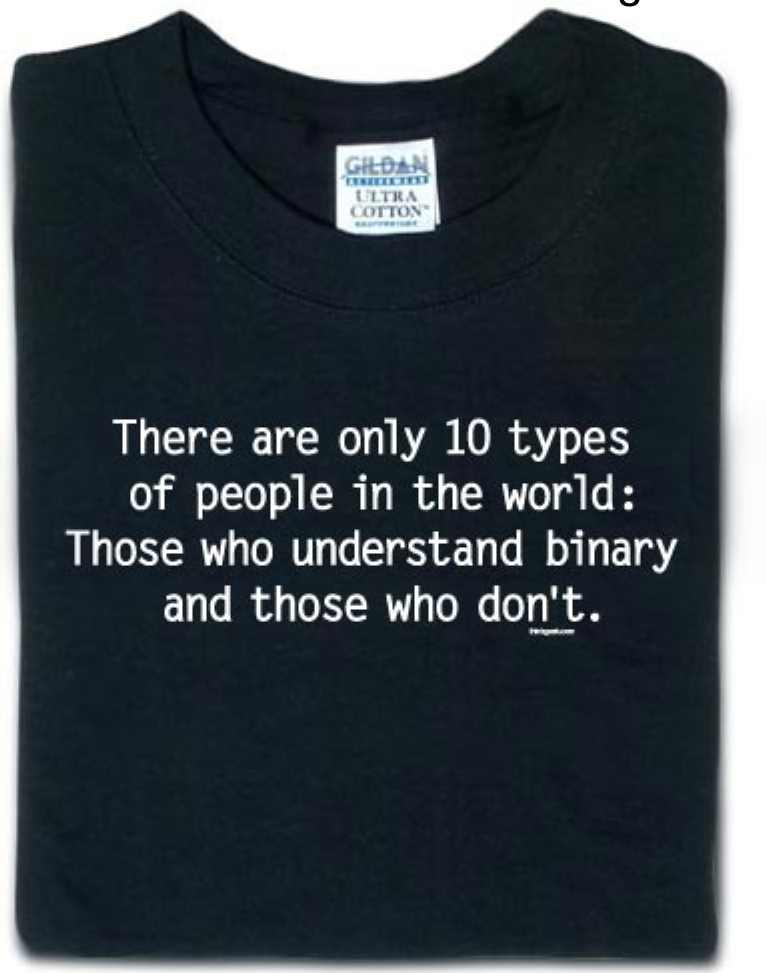
Today

www.thinkgeek.com

- Finish diodes
- Representing information



Representing Information Using Voltage

Representing Information Using Voltage

Analog representation: the precise voltage matters.

- Suppose we observed voltage ***v*** on a wire (e.g., an output from an accelerometer)
- The encoded quantity is some function of that voltage:

$$acceleration = f(v)$$

Representing Information Using Voltage

The simplest form assumes a linear relationship:

$$acceleration = \alpha\, v + \beta$$

Analog Encoding

Electrical noise in the circuit can alter the “true” voltage. E.g.:

- A device is turned on
- A motor is turned on or the direction is reversed

External sources can affect analog signals:

- Cell phones

Representing Information Using Voltage

Representing Information Using Voltage

- Digital representation: the value to be represented is binary – i.e., true or false
- For example, a bit b is:

$$b = \begin{cases} \text{true} & v > 2.5\,\text{Volts} \\ \text{false} & \text{otherwise} \end{cases}$$

Note: assuming a 5V based system

Representing Information Using Voltage

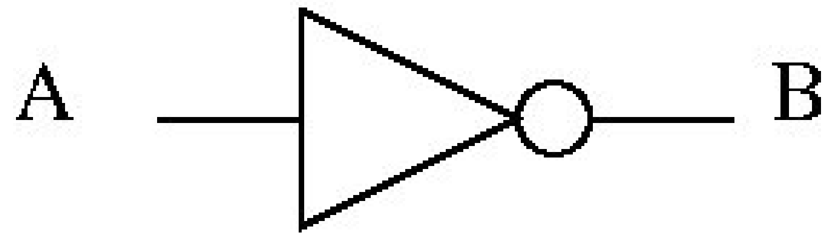
We typically use the shorthand:

$$
\begin{aligned}
0 \quad &= \text{false} \\
1 \quad &= \text{true}
\end{aligned}
$$

Computing In Binary
(i.e., Logic)

What is the Gate?

- Logical Symbol:



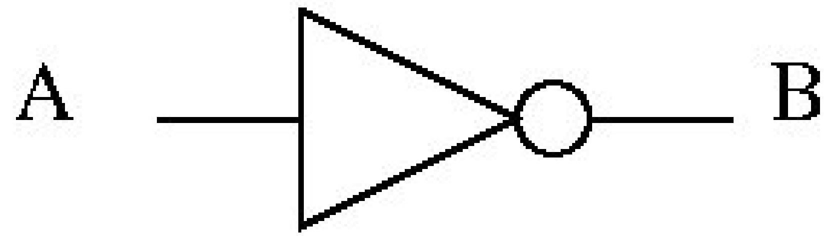
- Algebraic Notation:

- Truth Table:

A	B
0	
1	

The NOT Gate

- Logical Symbol:



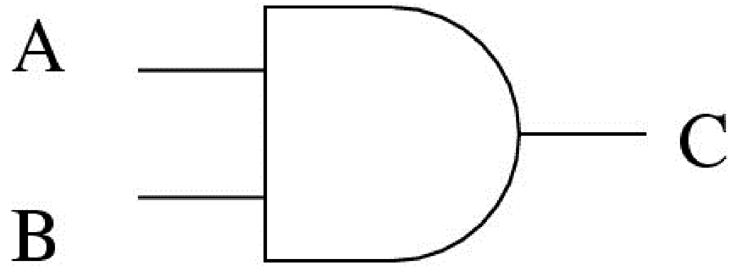
- Algebraic Notation: $B = \overline{A}$

- Truth Table:

A	B
0	1
1	0

And This Gate?

- Logical Symbol:



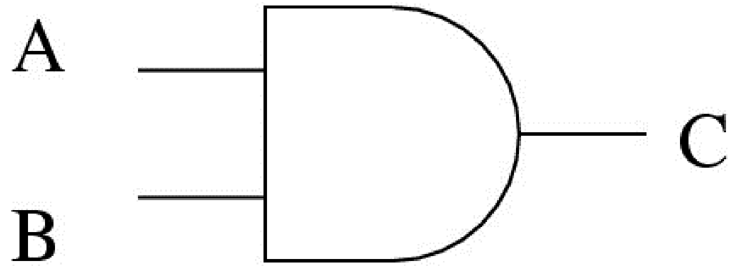
- Algebraic Notation: C = ?

- Truth Table:

A	B		C
0	0		
0	1		
1	0		
1	1		

The “AND” Gate

- Logical Symbol:



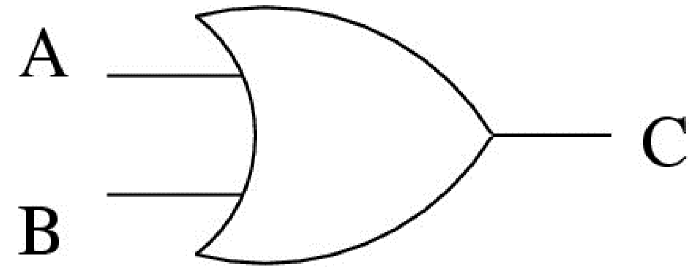
- Algebraic Notation: C = A*B = AB

- Truth Table:

A	B		C
0	0		0
0	1		0
1	0		0
1	1		1

And This Gate?

- Logical Symbol:



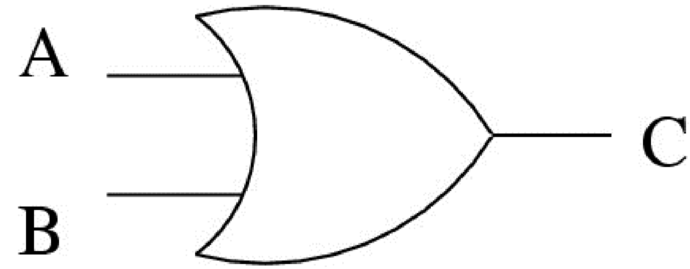
- Algebraic Notation: C = ?

- Truth Table:

A	B		C
0	0		
0	1		
1	0		
1	1		

The “OR” Gate

- Logical Symbol:



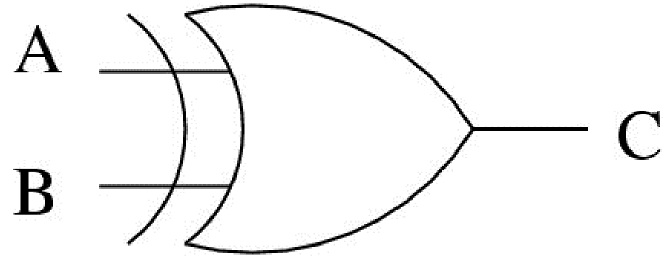
- Algebraic Notation: C = A+B

- Truth Table:

A	B		C
0	0		0
0	1		1
1	0		1
1	1		1

Exclusive OR ("XOR") Gates

- Logical Symbol:



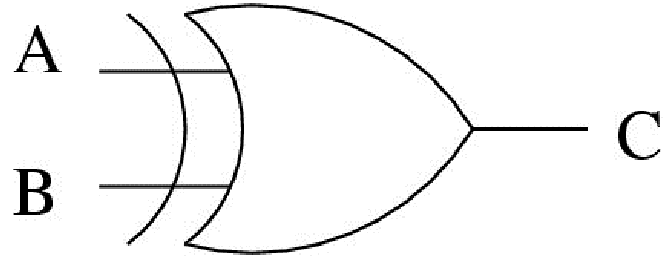
- Algebraic Notation: C = A⊕B

- Truth Table:

A	B		C
0	0		
0	1		
1	0		
1	1		

Exclusive OR (“XOR”) Gates

- Logical Symbol:



- Algebraic Notation: C = A⊕B

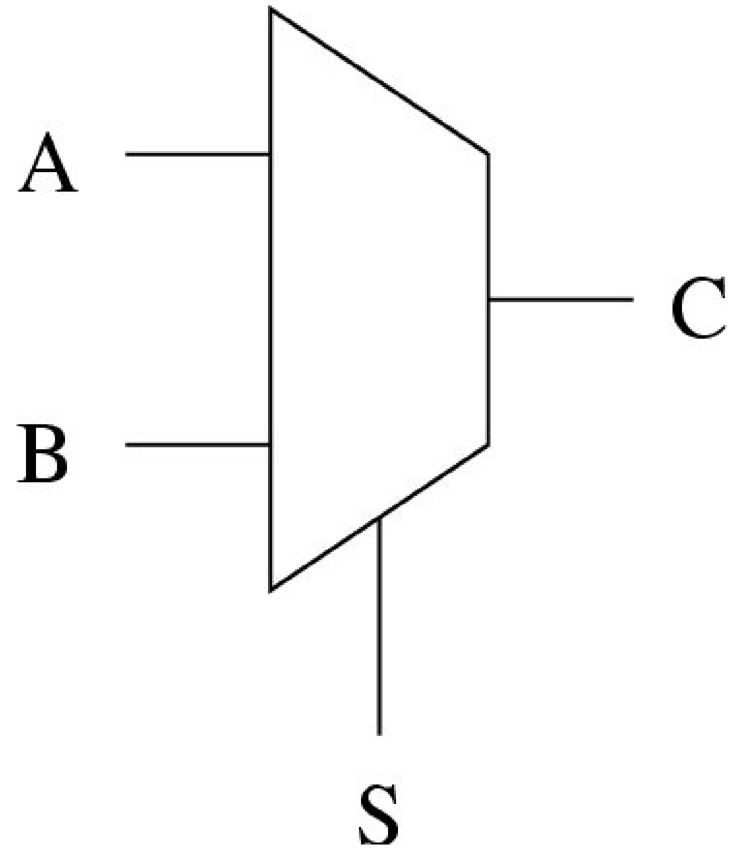
- Truth Table:

A	B		C
0	0		0
0	1		1
1	0		1
1	1		0

2-Input Multiplexer

A multiplexer is a device that selects between two input lines

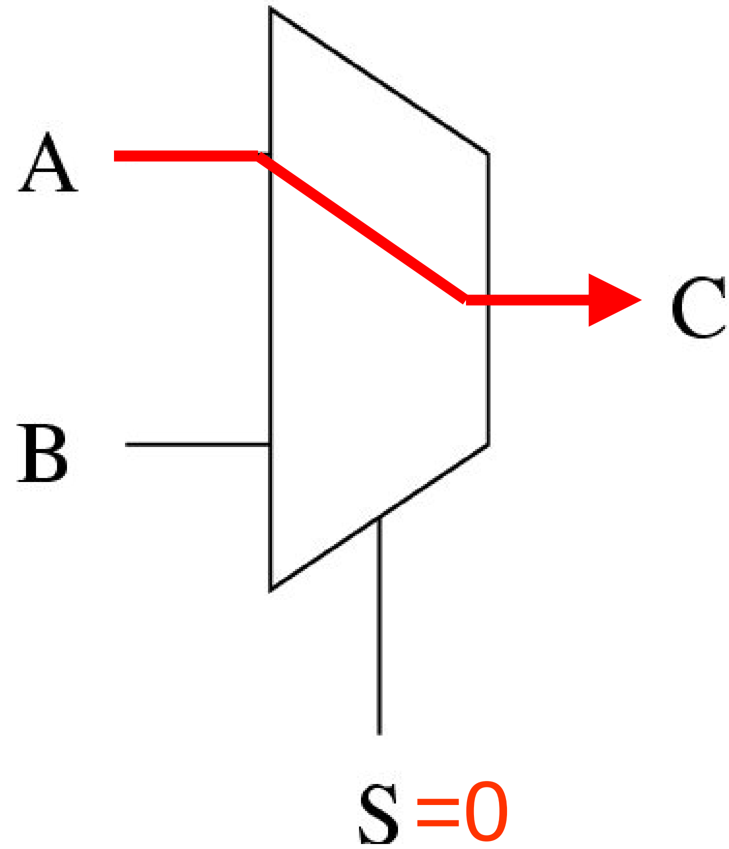
- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1



2-Input Multiplexer

A multiplexer is a device that selects between two input lines

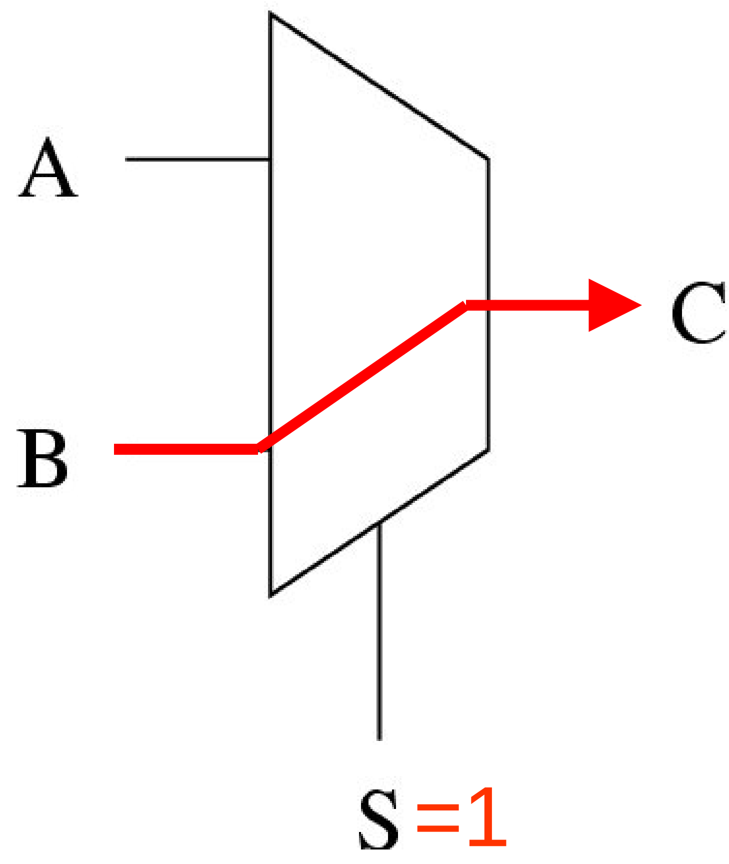
- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1



2-Input Multiplexer

A multiplexer is a device that selects between two input lines

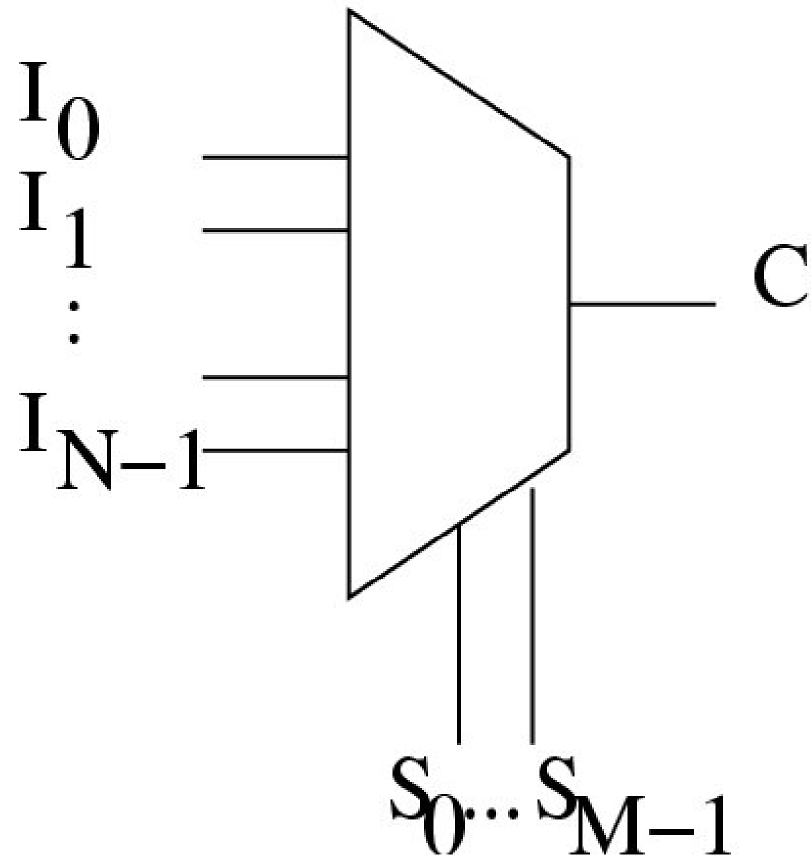
- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1



N-Input Multiplexer

Suppose we want to select from between N different inputs.

- This requires more than one select line. How many?



N-Input Multiplexer

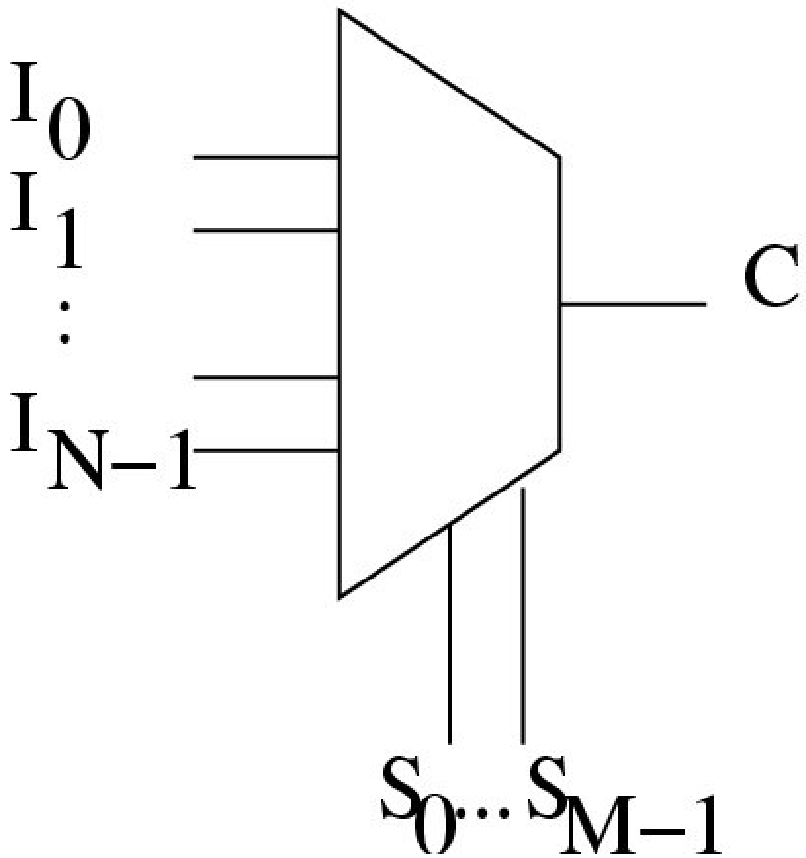
How many select lines?

- $M = \log_2 N$

  or

- $N = 2^M$

What would the N=8 implementation look like?



Back to Binary…

With a binary digit, we can only represent two different values…

How do we represent more?

Back to Binary…

How do we represent more?

- As in the decimal number system, we introduce multiple digits…

Binary Encoding

How do we convert from binary to decimal in general?

B2	B1	B0		decimal
0	0	0		0
0	0	1		1
0	1	0		2
0	1	1		3
1	0	0		4
1	0	1		5
1	1	0		6
1	1	1		7

Binary to Decimal Conversion

$$value = B_0 + B_1 * 2^1 + B_2 * 2^2 + B_3 * 2^3 + \ldots$$

$$value = \sum_{i=0}^{N-1} B_i * 2^i$$

How do we convert from decimal to binary?

Decimal to Binary Conversion

```
int value;

For each i: B[i] = 0

for(i = 0; value > 0; ++i) {
 B[i] = remainder of: value/2;
 value = value/2;
}
```

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

Binary Counting…

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 discussed 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

12	0 0 0 0 1 1 0 0
	↓ Invert
-12	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 1 1 1 1 1 1 1

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

$\downarrow$

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

!!!!

Today

- Two's complement numbers
- Binary math
- Bit-wise operators

www.thinkgeek.com

Representing Negative Numbers

An alternative:

(a little intuition first)

0 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 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 0

Decrement

Define this as

-1   1 1 1 1 1 1 1 1

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

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?

0b0110000111001001

Bit-Wise Operators

If A and B are bytes, what does this code mean?

```
C = A & B;
```

Bit-Wise Operators

If A and B are bytes, what does this code mean?

```
C = A & B;
```

The corresponding bits of A and B are ANDed together

Bit-Wise AND

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
?	C = A & B

Bit-Wise AND

0 1 0 1 1 1 1 0 A

1 0 0 1 1 0 1 1 B

C = A & B

Bit-Wise AND

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
0	C = A & B

Bit-Wise AND

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
1 0	C = A & B

Bit-Wise AND

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
0 0 0 1 1 0 1 0	C = A & B

Logical AND

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
???	C = A && B

Logical AND

0 1 0 1 1 1 1 0	A
true	
1 0 0 1 1 0 1 1	B
???	C = A && B

Logical AND

0 1 0 1 1 1 1 0 → ***true*** A

1 0 0 1 1 0 1 1 → ***true*** B

??? C = A && B

Logical AND

0 1 0 1 1 1 1 0		*true*	A
1 0 0 1 1 0 1 1		*true*	B
???		*true*	C = A && B

Logical AND

0 1 0 1 1 1 1 0	***true***	A
1 0 0 1 1 0 1 1	***true***	B
0 0 0 0 0 0 0 1	***true***	C = A && B

Logical AND

0 1 0 1 1 1 1 0		***true***	A
1 0 0 1 1 0 1 1		***true***	B
0 0 0 0 0 0 0 1		***true***	C = A && B

NOTE: we are assuming an 8-bit value

Representing Logical Values

Most of the time, we represent logical values using a multi-bit value. (e.g., using 8 or 16 bits). The rules are:

- A value of zero is interpreted as ***false***
- A non-zero value is interpreted as ***true***

Representing Logical Values

A logical operator will give a result of ***true*** or ***false***:

- ***false*** is represented with a value of zero (0)
- ***true*** is represented with a value of one (1)

Other Operators

LOGICAL / Bit-Wise

- OR: LOGICAL `||`, Bit-Wise `|`
- NOT: LOGICAL `!`, Bit-Wise `~`
- XOR: Bit-Wise `^`
- Shift left: Bit-Wise `<<`
- Shift right: Bit-Wise `>>`

When coding: keep this distinction straight

Putting the Bit-Wise Operators to Work: Bit Manipulation

Assume a variable A is declared as such:

```
uint8_t A;
```

What is the code that allows us to set bit 2 of A to 1? (we start counting bits from 0)

Bit Manipulation

What is the code that allows us to set bit 2 of A to 1? (we start counting bits from 0)

```
A = A | 4;
```

Bit Manipulation

What is the code that allows us to set bit 2 of A to 0?

Bit Manipulation

What is the code that allows us to set bit 2 of A to 0?

```
A = A & 0xFB;
```

or

```
A = A & ~4;
```

Bit Shifting

```
uint8_t A = 0x5A;
uint8_t B = A << 2;
uint8_t C = A >> 5;
```

What are the values of B and C?

What mathematical operations have we performed?