CSCI2467: Systems Programming Concepts

Slideset 2: Information as Data

Source: CS:APP Bryant & O'Hallaron (Section 2.1)

Course Instructors:

Matthew Toups
Caitlin Boyce

Course Assistants:

Saroj Duwal
David McDonald

Spring 2020

THE UNIVERSITY of

Course updates

- introlab due tonight, 11:59pm
  - Autolab handles due date, grace days, late penalties
- datalab out today - will be more challenging and time consuming
- Due in two weeks (Wednesday February 5), 11:59pm.
- Make sure Autolab works for you (both Intro Lab and Data Lab)
- As always: slides and resources available at `http://2467.cs.uno.edu`

Wrap-up

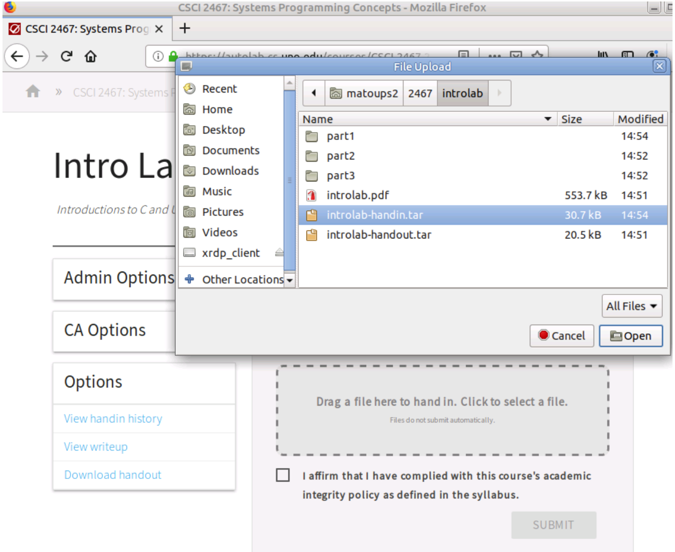
Now it's time to create the `introlab-handin.tar` file that is to be submitted to Autolab. To create the `tar` file we must first be sure that our current working directory contains the directories `part1 part2 part3`. Follow the steps below:

```
$ cd
$ cd 2467
$ ls
part1 part2 part3
$ tar cvf introlab-handin.tar part1 part2 part3
```

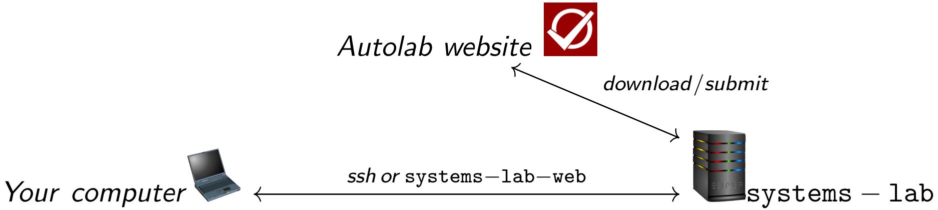
The first line moves us back to our home directory. We then enter the `2467` directory with the second line. The third line is to ensure that we are in the right location and can see our `part1 part2 part3` directories. Finally, the last line creates `introlab-handin.tar` which we will submit to Autolab.

To submit `introlab-handin.tar`, go back to where the lab handout was downloaded from Autolab. On the right hand side, check the box that confirms that you have adhered to the academic integrity policy then click the submit button. This will open up a file upload window where you will select the `introlab-handin.tar` file you just created. Refresh the page after a few seconds and you will see that the autograder has graded your work. You can see detailed grading information by clicking on one of the highlighted scores for parts 1, 2, or 3. Keep in mind that if you are unhappy with

How to submit introlab
Using Autolab website



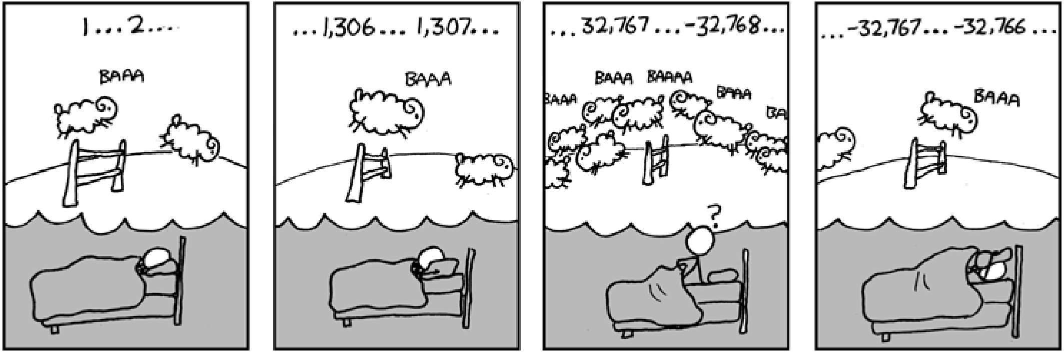
Handing in (introlab)



Overview

- Course notes
1. Preview
2. Bits and Bytes
   - Representing information as bits
   - Bit-level manipulations
     - Boolean Algebra
     - Logical operators
     - Shift operators
3. Up next: Integer Values
   - Signed and Unsigned ints

ints are not Integers



Source: xkcd.com

$\mathbb{Z}$ is infinitely large, computer memory is not.

This is the fundamental challenge!

ints are not Integers and floats are not Reals

- Is $x^2 \geq 0$?
  - Floating point? Yes!
  - Int?

    $40000 * 40000 \to 1600000000$

    $50000 * 50000 \to ??$
- Is $(x + y) + z = x + (y + z)$?
  - Int (signed or unsigned): Yes!
  - Float?

    $3.2 + (1e20 - 1e20) \to 3.2$

    $(3.2 + 1e20) - 1e20 \to ??$

Computer Arithmetic

- Does not generate random values
  - Arithmetic operations have important mathematical properties
- Cannot assume all "usual" mathematical properties
  - Due to finiteness of representations
  - `int` operations satisfy *ring* properties:
    Commutativity, associativity, distributivity
  - Floating point operations satisfy *ordering* properties:
    Monotonicity, values of signs
- Observation
  - You need to understand which abstractions apply in which contexts

Overview

- Course notes

1. Preview
2. Bits and Bytes
   - Representing information as bits
   - Bit-level manipulations
     - Boolean Algebra
     - Logical operators
     - Shift operators
3. Up next: Integer Values
   - Signed and Unsigned ints

Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways computers determine what to do (instructions) and represent and manipulate numbers, sets, text, etc

```
00100011 01101001 01101110 01100011 01101100 01110101  #inclu
01100100 01100101 00100000 00111100 01110011 01110100  de <st
01100100 01101001 01101111 00101110 01101000 00111110  dio.h>
00001010 00001010 01101001 01101110 01110100 00100000  ..int
01101101 01100001 01101001 01101110 00101000 00101001  main()
00001010 01111011 00001010 00100000 00100000 00100000  .{.
00100000 01110000 01110010 01101001 01101110 01110100   print
01100110 00101000 00100010 01101000 01100101 01101100  f("hel
01101100 01101111 00101100 00100000 01110111 01101111  lo, wo
01110010 01101100 01100100 01011100 01101110 00100010  rld\n"
00101001 00111011 00001010 00100000 00100000 00100000  );.
00100000 01110010 01100101 01110100 01110101 01110010   retur
01101110 00100000 00110000 00111011 00001010 01111101  n 0;.}
00001010                                               .
```

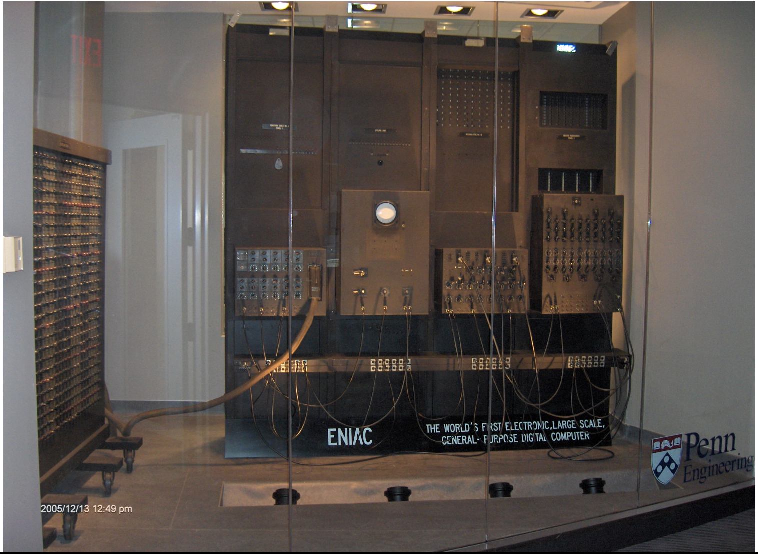
Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways computers determine what to do (instructions) and represent and manipulate numbers, sets, text, etc

```
001000110110100101101110011000110110110001110101  #inclu
011001000110010100100000001111000111001101110100  de <st
011001000110100101101111001011100110100000111110  dio.h>
000010100000101001101001011011100111010000100000  ..int
011011010110000101101001011011100010100000101001  main()
000010100111101100001010001000000010000000100000  .{.
001000000111000001110010011010010110111001110100   print
011001100010100000100010011010000110010101101100  f("hel
011011000110111100101100001000000111011101101111  lo, wo
011100100110110001100100010111000110111000100010  rld\n"
001010010011101100001010001000000010000000100000  );.
001000000111001001100101011101000111010101110010   retur
011011100010000000110000001110110000101001111101  n 0;.}
00001010                                               .
```

Why bits?

Photo ©2005 Paul W Shaffer, University of Pennsylvania



ENIAC

Electronic Computer Flashes Answers, May Speed Engineering

By T. R. KENNEDY Jr.
Special to THE NEW YORK TIMES.

PHILADELPHIA, Feb. 14—One of the war's top secrets, an amazing machine which applies electronic speeds for the first time to mathematical tasks hitherto too difficult and cumbersome for solution, was announced here tonight by the War Department. Leaders who saw the device in action for the first time heralded it as a tool with which to begin to rebuild scientific affairs on new foundations.

Such instruments, it was said, could revolutionize modern engineering, bring on a new epoch of industrial design, and eventually eliminate much slow and costly trial-and-error development work now deemed necessary in the fashioning of intricate machines. Heretofore, sheer mathematical difficulties have often forced designers to accept inferior solutions of their problems, with higher costs and slower progress.

The "Eniac," as the new electronic speed marvel is known, virtually eliminates time in doing such jobs. Its inventors say it computes a mathematical problem 1,000 times faster than it has ever been done before.

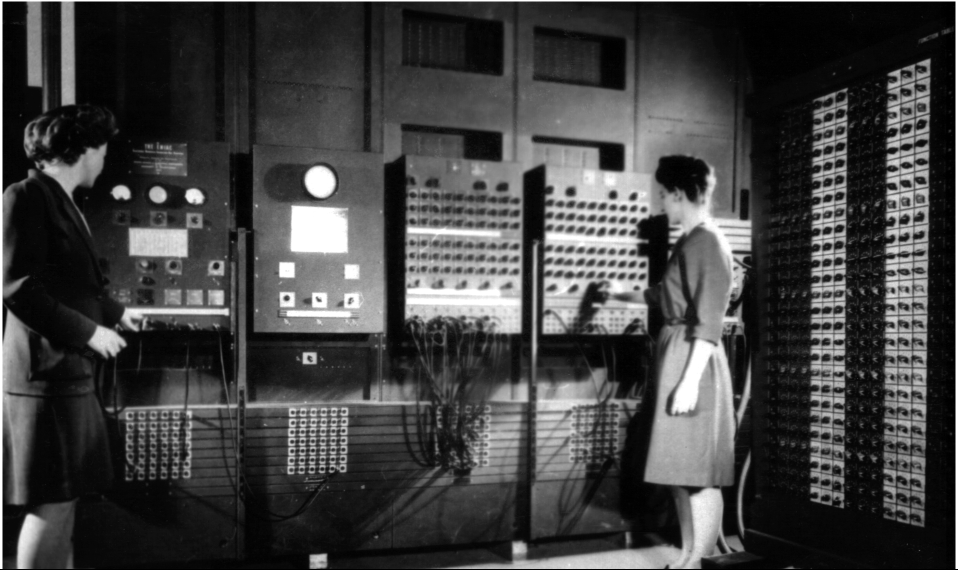
The machine is being used on a problem in nuclear physics.

The Eniac, known more formally as "the electronic numerical integrator and computer," has not a single moving mechanical part. Nothing inside its 18,000 vacuum tubes and several miles of wiring moves except the tiniest elements of matter-electrons. There are, however, mechanical devices associated with it which translate or "interpret" the mathematical language of man to terms understood by the Eniac, and vice versa.

Ceremonies dedicating the machine will be held tomorrow night at a dinner given a group of Government and scientific men at the University of Pennsylvania, after

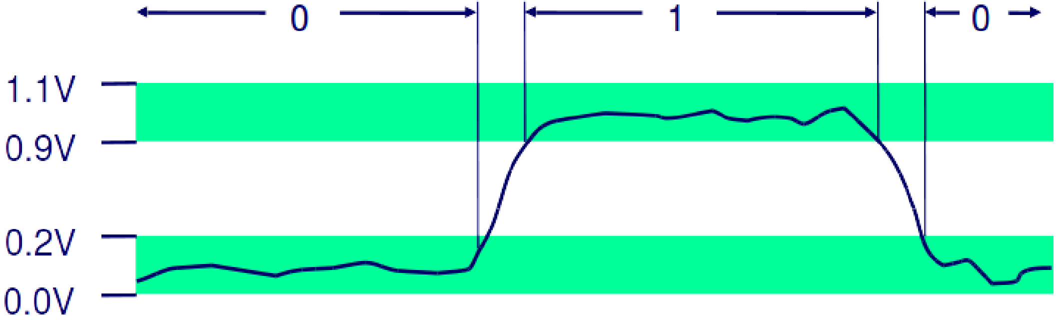
5. Column 3

Why bits?



Why bits?

- Electronic Implementation
- Easy to store with bistable elements
- Reliably transmitted on noisy and inaccurate wires



Counting in base-2 (binary)

Base 2 Number Representation (not characters or strings)

- Represent 2467_{10} as 100110100011_2

value	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
value	2048	1024	512	256	128	64	32	16	8	4	2	1
Bits	1	0	0	1	1	0	1	0	0	0	1	1
add:	2048	+		256	+128	+	32	+			2+	1
Sum:	2467											

- Represent 1.20_{10} as $1.0011001100110011[0011]\ldots_2$

value	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	$\ldots$		
value	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$	$\ldots$		
Bits	1	0	0	1	1	0	0	1	1	0	0		1	1

Encoding Byte Values

- 1 Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
- Hexadecimal: Base 16 representation
  - Use characters 0 to 9 and A to F
  - Write `FA 1D 37 B1` in C as:
    `0xFA1D37B1`
    `0xfa1d37b1`
- Important to get comfortable with this notation
  - Used in all subsequent labs
  - Practice problems 2.1, 2.2, 2.3, 2.4 will help you build your hex-literacy

hex	decimal	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Example Data Representations

	Size in Bytes		
C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	-	-	10/16
pointer	4	8	8

Course notes

1 Preview

2 Bits and Bytes
- Representing information as bits
- Bit-level manipulations
  - Boolean Algebra
  - Logical operators
  - Shift operators

3 Up next: Integer Values

Boolean Algebra

Algebraic representation of logic, developed by Boole in 1850s
Encodes "True" as 1 and "False" as 0

Binary AND:
$A \& B = 1$ when
both $A = 1$ and $B = 1$

&	0	1
0	0	0
1	0	1

Binary OR:
$A | B = 1$ when
either $A = 1$ or $B = 1$

|	0	1
0	0	1
1	1	1

Binary NOT (complement):
$\sim A = 1$ when $A = 0$

~	1
0	1
1	0

Exclusive-Or (XOR):
$A \wedge B = 1$ when *either* $A = 1$ or $B = 1$ but *not* both

∧	0	1
0	0	1
1	1	0

Based on **Figure 2.7** in CS:APP3e

Boolean Algebra extended

The connection between Boolean algebra and digital logic was first proposed by Claude Shannon in a 1937 Master's thesis.

Can operate on *bit vectors*, applying operation *bitwise*

```
    01101001     01101001     01101001
&   01010101  |  01010101  ^  01010101  ~  01010101
    --------     --------     --------     --------
    01000001     01111101     00111100     10101010
```

105 & 85
= 65 ??

(Bitwise operations look strange when using decimal representations!)

Boolean Algebra and finite sets

Width *w* bit vector represents subsets of {0, ... , w-1}
$a_j = 1$ if $j \in A$

01101001 { 0 , 3 , 5 , 6 }
76543210

01010101 { 0 , 2 , 4 , 6 }
76543210

Operations (on the two sets given above):

&	Intersection	01000001	{ 0 , 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }

Some useful properties of Boolean Algebra

Shared properties

Property	Integer ring	Boolean algebra
Commutativity	$a + b = b + a$ $a \times b = b \times a$	$a \mid b = b \mid a$ $a \mathbin{\&} b = b \mathbin{\&} a$
Associativity	$(a + b) + c = a + (b + c)$ $(a \times b) \times c = a \times (b \times c)$	$(a \mid b) \mid c = a \mid (b \mid c)$ $(a \mathbin{\&} b) \mathbin{\&} c = a \mathbin{\&} (b \mathbin{\&} c)$
Distributivity	$a \times (b + c) = (a \times b) + (a \times c)$	$a \mathbin{\&} (b \mid c) = (a \mathbin{\&} b) \mid (a \mathbin{\&} c)$
Identities	$a + 0 = a$ $a \times 1 = a$	$a \mid 0 = a$ $a \mathbin{\&} 1 = a$
Annihilator	$a \times 0 = 0$	$a \mathbin{\&} 0 = 0$
Cancellation	$-(-a) = a$	$\tilde{}(\tilde{}a) = a$

Unique to Rings

Inverse	$a + -a = 0$	—

Unique to Boolean Algebras

Distributivity	—	$a \mid (b \mathbin{\&} c) = (a \mid b) \mathbin{\&} (a \mid c)$
Complement	— —	$a \mid \tilde{}a = 1$ $a \mathbin{\&} \tilde{}a = 0$
Idempotency	— —	$a \mathbin{\&} a = a$ $a \mid a = a$
Absorption	— —	$a \mid (a \mathbin{\&} b) = a$ $a \mathbin{\&} (a \mid b) = a$
DeMorgan's laws	— —	$\tilde{}(a \mathbin{\&} b) = \tilde{}a \mid \tilde{}b$ $\tilde{}(a \mid b) = \tilde{}a \mathbin{\&} \tilde{}b$

Bit Masks

	10010101	data
&	00011100	mask
=	00010100	result
Unwanted bits are "masked out":	~~000~~101~~00~~	

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- !0x41 ⇒ 0x00
- !0x00 ⇒ 0x01
- !!0x41 ⇒ 0x01
- 0x69 && 0x55 ⇒ 0x01
- 0x69 || 0x55 ⇒ 0x01
- a && 5/a (will never divide by zero)
- p && *p (avoids null pointer access)

Shift operators

- Left Shift: $x << y$
  - Shift bitvector x left y positions
    (Throw away extra bits on left)
  - Fill with 0s on right
- Right Shift: $x >> y$
  - Shift bitvector x right y positions
    (Throw away extra bits on right)
  - ⋆ Logical shift: fill with 0s on left
  - ⋆ Arithmetic shift: Replicate most significant bit on left
- Undefined: Shift < 0 or $\geq$ word size

Example 1

Argument x	01100010
$<< 3$	00010000
Log. $>> 2$	00011000
Arith. $>> 2$	00011000

Example 2

Argument x	10100010
$<< 3$	00010000
Log. $>> 2$	00101000
Arith. $>> 2$	11101000

Course notes

1 Preview

2 Bits and Bytes
- Boolean Algebra
- Logical operators
- Shift operators

3 Up next: Integer Values
- Signed and Unsigned ints

Coming up next. . .

Integers: unsigned, signed, negation, arithmetic (Sections 2.2-2.3)

Encoding Integer values

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Signed

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Change: Sign bit!

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

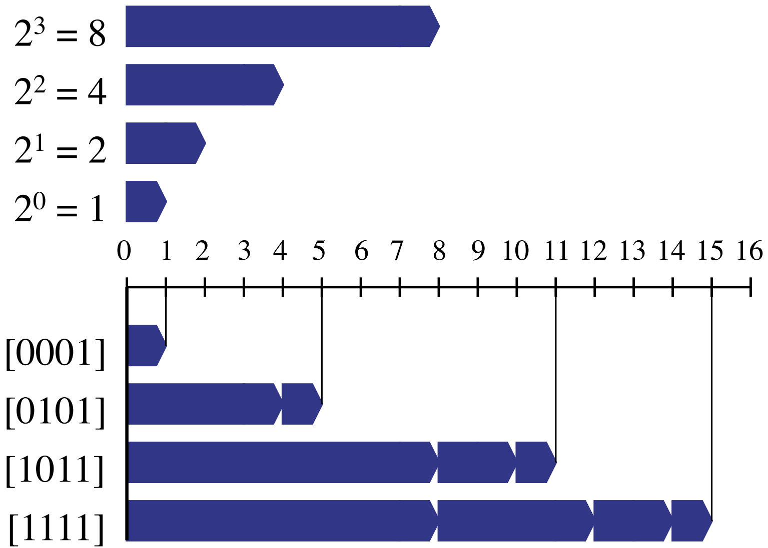
Change: Sign bit!

- Example using `short` in C (2 bytes):

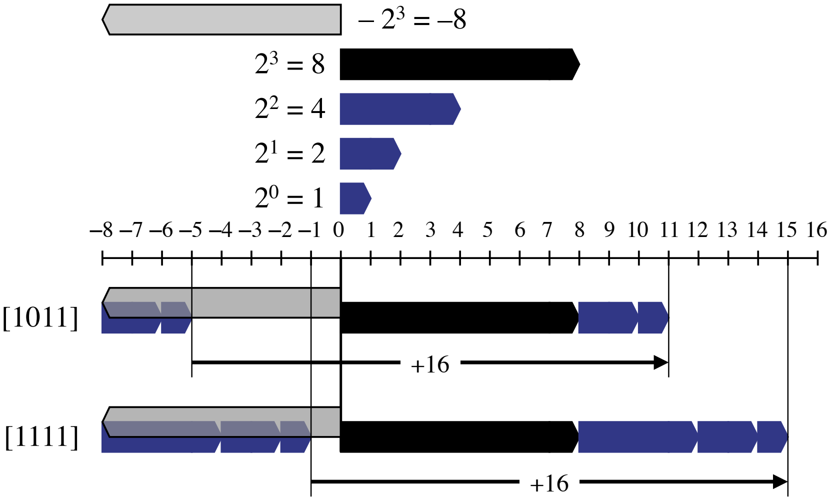
Decimal	Hex	Binary
2467	09A3	00001001 10100011
-2467	F65D	11110110 01011101

- This is called *Two's complement*
- Sign bit indicates sign
  - 0 for non-negative
  - 1 for negative

Unsigned Integers



Signed Integers



Back to the 2's complement encoding example

`short int x=`	2467:	00001001 10100011
`short int y=`	-2467:	11110110 01011101

Weight	2467		-2467	
1	1	1	1	1
2	1	2	0	0
4	0	0	1	4
8	0	0	1	8
16	0	0	1	16
32	1	32	0	0
64	0	0	1	64
128	1	128	0	0
256	1	256	0	0
512	0	0	1	512
1024	0	0	1	1024
2048	1	2048	0	0
4096	0	0	1	4096
8192	0	0	1	8192
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum:		2467		-2467

Coming up next. . .

Integers: unsigned, signed, negation, arithmetic (Sections 2.2-2.3)