CS 64 Week 0 Lecture 1

Kyle Dewey

Overview

- Administrative stuff
- Class motivation
- Syllabus
- Working with different bases
- Bitwise operations
- Twos complement

Administrative Stuff

About Me

- 5th year Ph.D. candidate, doing programming languages research (automated testing)
- **Not** a professor; just call me Kyle
- Fourth time teaching; first time teaching CS64

About this Class

- See something wrong? Want something improved? Email me about it! (kyledewey@cs.ucsb.edu)
- I generally operate based on feedback

Bad Feedback

- This guy sucks.
- This class is boring.
- This material is useless.

–I can't do anything in response to this

Good Feedback

- This guy sucks, *I can't read his writing.*
- This class is boring, *it's way too slow.*
- This material is useless, *I don't see how it relates to anything in reality.*

- I can't fix anything if I don't know what's wrong

-I can actually do something about this!

Questions

- Which best describes you?
  - CS major
  - ECE major
  - Other

Office Hours
Placement

Class Motivation

```
int main(int argc, char** argv) {
   ...
}
```

–I just want to write my code

```
int main(int argc, char** argv) {
   ...
}
```

–Image source: http://media.firebox.com/pic/p5294_column_grid_12.jpg
–Have some magic happen

```
int main(int argc, char** argv) {
   ...
}
```

```
3.14956
```

-Image source: http://media.firebox.com/pic/p5294_column_grid_12.jpg
-And then get a result

```
int main(int argc, char** argv) {
   ...
}
```

3.14956

-Image source: http://dnr.wi.gov/eek/critter/reptile/images/turtleMidlandPainted.jpg
-But what if your magic isn't working fast enough?

```
int main(int argc, char** argv) {
   ...
}
```

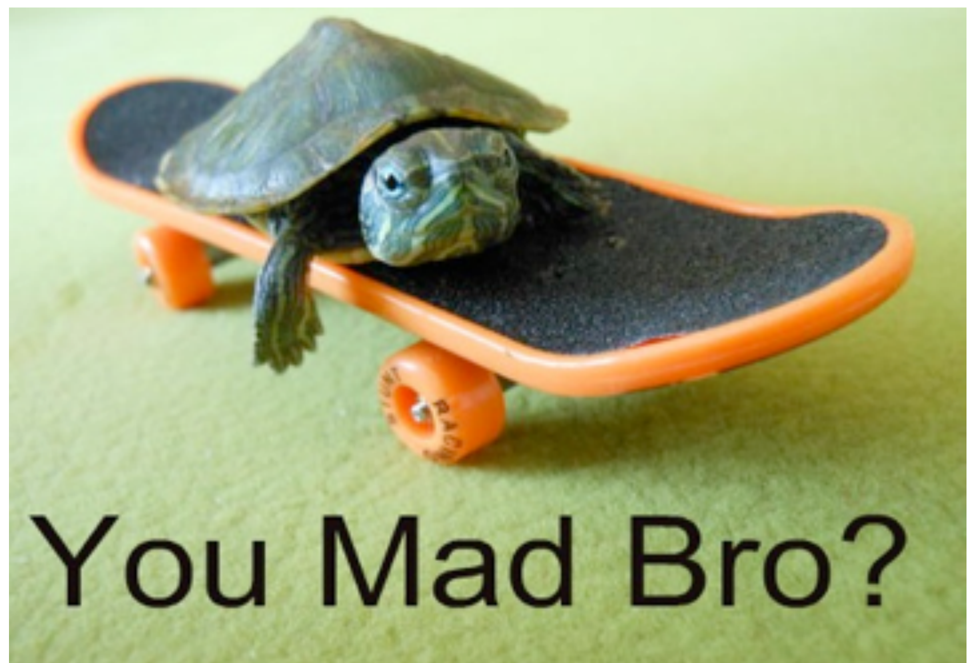
More Efficient Algorithms

3.14956

–Image source: http://dnr.wi.gov/eek/critter/reptile/images/turtleMidlandPainted.jpg
–Let's apply some better algorithms, improve time complexity, and so on...

```
int main(int argc, char** argv) {
   ...
}
```

More Efficient Algorithms




```
3.14956
```

-Image source: http://turtlefeed.tumblr.com/post/35444735335/ive-lost-track-of-how-many-turtle-on-skateboard
-...and we're left with a slightly faster turtle

Why are things still slow?

The magic box isn’t so magic

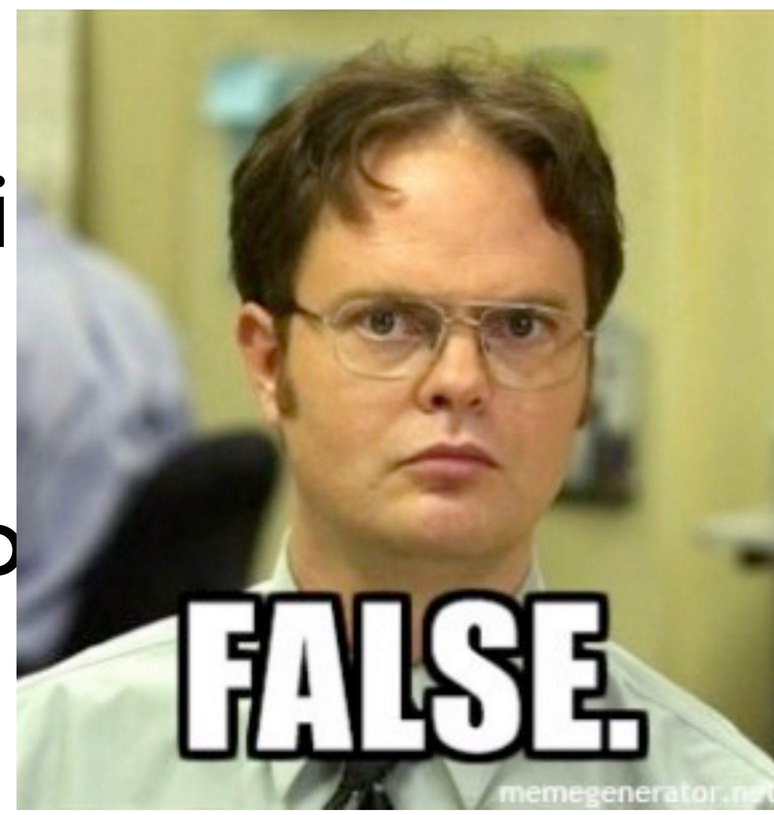
Array Access

`arr[x]`

- Constant time! (O(1))
- Where the **random** in random access memory comes from!

Array Access

`arr[x]`



- Constant ti
- Where the dom access memory co

–Image source: http://blog.fractureme.com/wp-content/uploads/2014/12/dwight-schrute-false-288x300.jpg

Array Access

- Memory is loaded as chunks into *caches*
    - Cache access is much faster (e.g., 10x)
    - Iterating through an array is fast
    - Jumping around any which way is slow
- Can change time complexity if accounted for
    - O(N^3) versus ~O(N^4)

-Matrix multiply is the example at the end. If you take the graduate-level parallel programming course, you'll watch a matrix multiply program seemingly nonsensically get around 5-6X faster by using a memory layout which looks asinine, but processors love

Instruction Ordering

```
int x = a + b;
int y = c * d;
int z = e - f;
```

```
int z = e - f;
int y = c * d;
int x = a + b;
```

-Two code snippets that appear to do the exact same thing
-Both should take the same amount of time, right?

Instruction Ordering

```
int x = a + b;
int y = c * d;
int z = e - f;
```

3 Milliseconds?

```
int z = e - f;
int y = c * d;
int x = a + b;
```

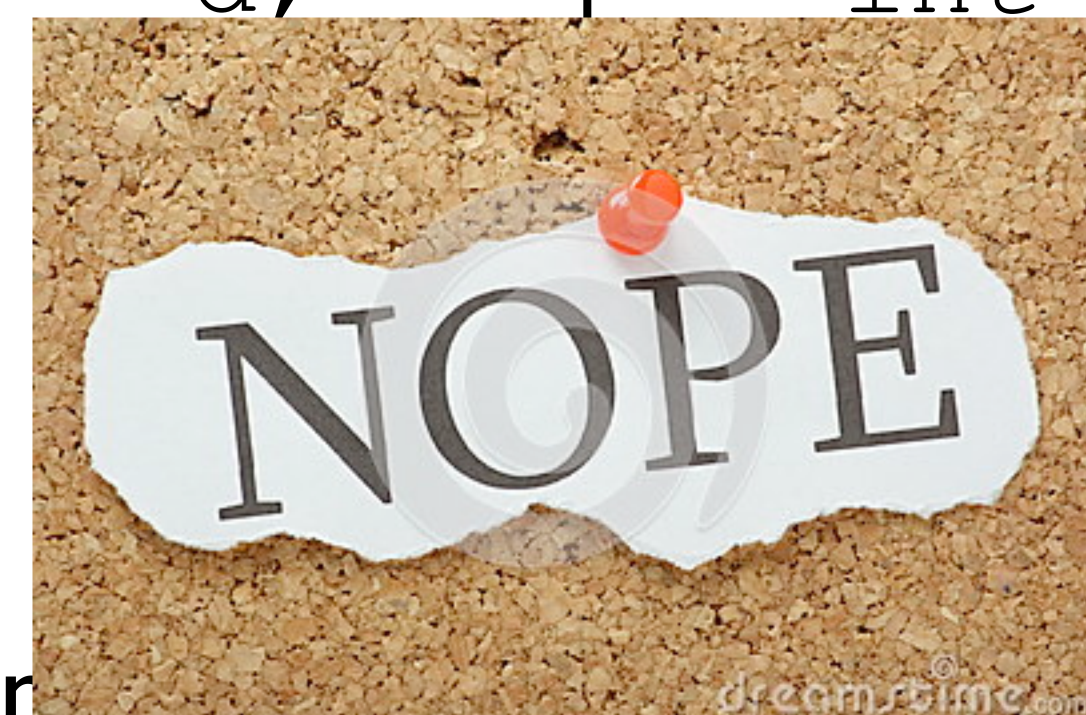
3 Milliseconds?

-Two code snippets that appear to do the exact same thing
-Both should take the same amount of time, right?

Instruction Ordering

```
int x = a + b;
int y = c * d;
int z = e
```

```
int z = e - f;
int y = c * d;
    x = a + b;
```



3 Millisecon

Milliseconds?

–Image source: http://www.dreamstime.com/stock-photo-nope-word-typed-scrap-torn-paper-pinned-to-cork-notice-board-word-well-known-meme-modern-slang-image43914016

Instruction Ordering

- Modern processors are *pipelined*, and can execute sub-portions of instructions in parallel
  - Depends on when instructions are encountered
- Some can execute whole instructions in different orders
- If your processor is from Intel, it is insane.

The Point

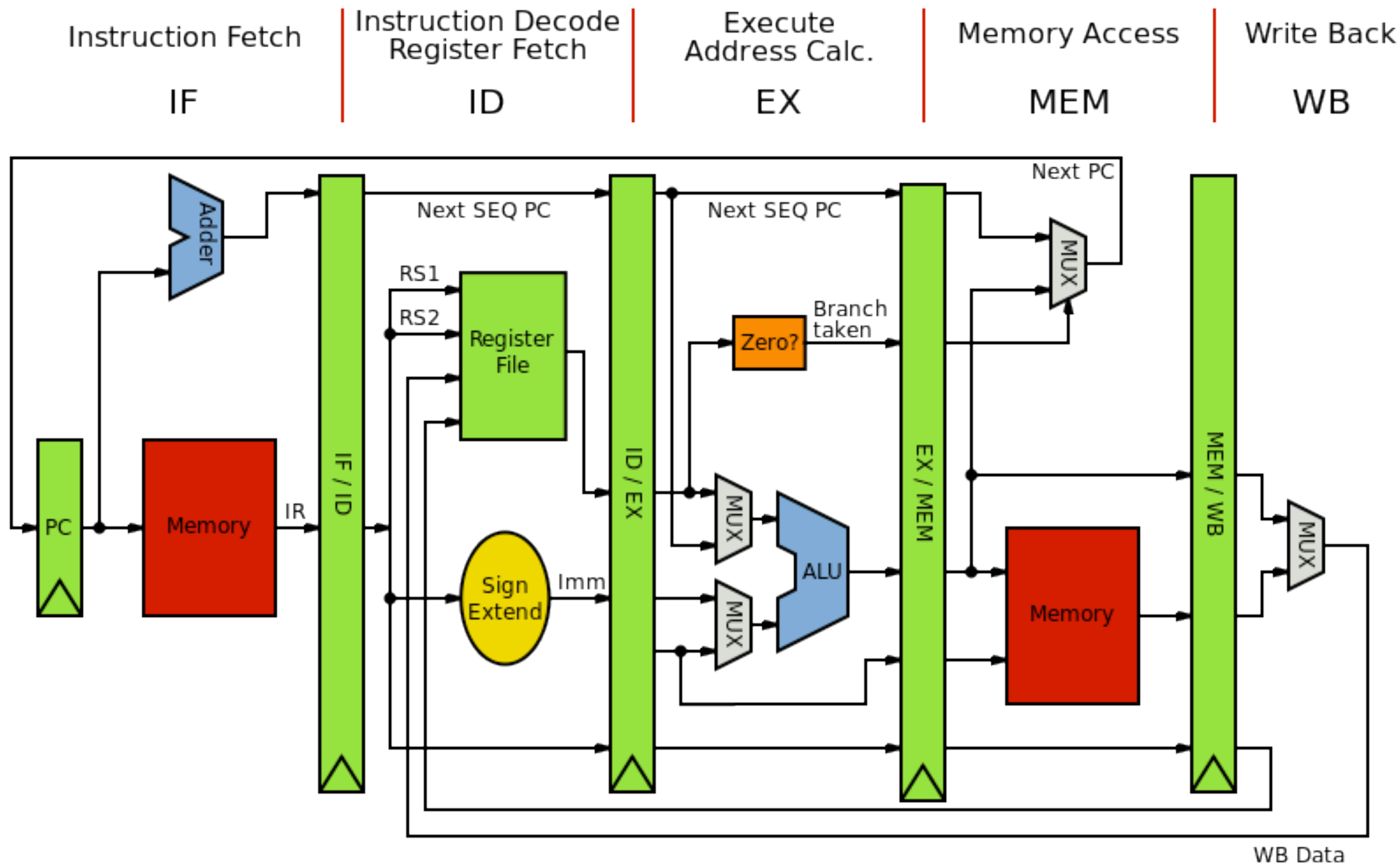
- If you really want performance, you need to know how the magic works
  - "But it scales!" - empirically, probably not
  - Chrome is fast for a reason
- If you want to write a naive compiler (CS160), you need to know some low-level details
- If you want to write a *fast* compiler, you need to know *tons* of low-level details

-A bunch of Chrome is written using low-level machine instructions (assembly)
-Ruby on Rails is horrendously slow, and is built on the idea of scaling up. A startup I know of beat a 50 node Rails cluster using one machine. Even in more typical settings, typically it's something like 10 Rails nodes to one optimized node. Twitter used to run Rails, but found that it was too slow to handle the sort of scale that it handles now.

So Why Digital Design?

-Image source: http://media.firebox.com/pic/p5294_column_grid_12.jpg
-It's to turn this

So Why Digital Design?



-Image source: https://en.wikipedia.org/wiki/MIPS_instruction_set#/media/File:MIPS_Architecture_%28Pipelined%29.svg
-...into this

So Why Digital Design?

- Basically, circuits are the programming language of hardware
    - Yes, everything goes back to physics

Syllabus

Working with Different Bases

What's In a Number?

- Question: why exactly does 123 have the value 123? As in, what does it *mean*?

-Not a philosophy question
-This is actually kind of brain-melting, but once this is understood everything else becomes second-nature

What's In a Number?

123

–Start with 123

What's In a Number?

1	2	3

-Break it down into its separate digits

What's In a Number?

1	2	3
Hundreds	Tens	Ones

-Values of each digit

What's In a Number?

1	2		3		
Hundreds	Tens		Ones		
100	10	10	1	1	1

–Values of each digit

Question

- Why did we go to tens? Hundreds?

1	2	3
Hundreds	Tens	Ones
100	10 10	1 1 1

Answer

- Because we are in decimal (base 10)

1	2	3
Hundreds	Tens	Ones
100	10 10	1 1 1

Another View

123

Another View

1	2	3

–Break it down into its separate digits

Another View

1	2	3
1×10^2	2×10^1	3×10^0

–Values of each digit

Conversion from Some Base to Decimal

- Involves repeated division by the value of the base
  - From right to left: list the remainders
  - Continue until 0 is reached
  - Final value is result of reading remainders from bottom to top
- For example: what is 231 decimal to decimal?

Conversion from Some Base to Decimal

Conversion from Some Base to Decimal

	Remainder
$10 \overline{)231}$	
23	1

Conversion from Some Base to Decimal

	Remainder
$10 \,\underline{)\,231}$	
$10 \,\underline{)\,23}$	1
2	3

Conversion from Some Base to Decimal

	Remainder
$10 \,\lfloor 231$	
$10 \,\lfloor 23$	1
$10 \,\lfloor 2$	3
0	2

–Final value: 231 (reading remainders from bottom to top)

Now for Binary

- Binary is base 2
- Useful because circuits are either on or off, representable as two states, 0 and 1

Now for Binary

1010

Now for Binary

1	0	1	0

Now for Binary

Eights	Fours	Twos	Ones
1	0	1	0

Now for Binary

1	0	1	0
Eights 1×2^3	Fours 0×2^2	Twos 1×2^1	Ones 0×2^0
8	0	2	0

Question

- What is binary 0101 as a decimal number?

Answer

- What is binary 0101 as a decimal number?
  - 5

0	1	0	1
Eights 0×2^3	Fours 1×2^2	Twos 0×2^1	Ones 1×2^0
0	4	0	1

From Decimal to Binary

- What is decimal 57 to binary?

From Decimal to Binary

57

From Decimal to Binary

	Remainder
$2 \overline{\smash{)}57}$	
28	1

From Decimal to Binary

	Remainder
2 | 57	
2 | 28	1
14	0

From Decimal to Binary

	Remainder
2 | 57	
2 | 28	1
2 | 14	0
7	0

From Decimal to Binary

	Remainder
2 | 57	
2 | 28	1
2 | 14	0
2 | 7	0
3	1

From Decimal to Binary

	Remainder
2 | 57	
2 | 28	1
2 | 14	0
2 | 7	0
2 | 3	1
1	1

From Decimal to Binary

	Remainder
2 | 57	
2 | 28	1
2 | 14	0
2 | 7	0
2 | 3	1
2 | 1	1
0	1

Octal

- Octal is base 8
- Same idea

Octal Example

- What is 172 octal in decimal?

Octal Example

172

Octal Example

1	7	2

–Break it down into its separate digits

Octal Example

1	7	2
Sixty-fours 1×8^2	Eights 7×8^1	Ones 2×8^0

–Break it down into its separate digits

Octal Example

1	7	2
Sixty-fours 1×8^2	Eights 7×8^1	Ones 2×8^0
64	8 8 8 8 8 8 8 (56)	1 1

–Break it down into its separate digits

Octal Example

Answer: 122

1	7	2
Sixty-fours 1×8^2	Eights 7×8^1	Ones 2×8^0
64	8 8 8 8 8 8 8 (56)	1 1

-Break it down into its separate digits

From Decimal to Octal

- What is 182 decimal to octal?

From Decimal to Octal

182

From Decimal to Octal

	Remainder
$8 \,\underline{\lfloor 182}$	
22	6

From Decimal to Octal

	Remainder
8 | 182	
8 | 22	6
2	6

From Decimal to Octal

	Remainder
8 | 182	
8 | 22	6
8 | 2	6
0	2

Hexadecimal

- Base 16
- Binary is horribly inconvenient to write out
- Easier to convert between hexadecimal (which is more convenient) and binary
  - Each hexadecimal digit maps to four binary digits
  - Can just memorize a table

Hexadecimal

- Digits 0-9, along with A (10), B (11), C (12), D (13), E (14), F (15)

Hexadecimal Example

- What is 1AF hexadecimal in decimal?

Hexadecimal Example

I	A	F

Hexadecimal Example

1	A	F
Two-fifty-sixes	Sixteens	Ones

Hexadecimal Example

1	A	F
Two-fifty-sixes	Sixteens	Ones
1×16^2	10×16^1	15×16^0

Hexadecimal Example

1	A	F
Two-fifty-sixes	Sixteens	Ones
1×16^2	10×16^1	15×16^0
	16 16 16 16 16 16 16 16 16 16	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
256	(160)	(15)

Hexadecimal to Binary

- Previous techniques all work, using decimal as an intermediate
- The faster way: memorize a table (which can be easily reconstructed)

Hexadecimal to Binary

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

Hexadecimal	Binary
8	1000
9	1001
A (10)	1010
B (11)	1011
C (12)	1100
D (13)	1101
E (14)	1110
F (15)	1111

-0x1AF: 0001 1010 1111
-0101 1010: 0x5A

Bitwise Operations

Bitwise AND

- Similar to logical AND (`&&`), except it works on a bit-by-bit manner
- Denoted by a single ampersand: `&`

```
(1001 &
 0101)=
 0001
```

Bitwise OR

- Similar to logical OR (`||`), except it works on a bit-by-bit manner
- Denoted by a single pipe character: `|`

```
(1001 |
 0101)=
 1101
```

Bitwise XOR

- Exclusive OR, denoted by a carat: ^
- Similar to bitwise OR, except that if both inputs are `1` then the result is `0`

```
(1001 ^
 0101)=
 1100
```

Bitwise NOT

- Similar to logical NOT (`!`), except it works on a bit-by-bit manner
- Denoted by a tilde character: `~`

```
~1001 =
 0110
```

Shift Left

- Move all the bits `N` positions to the left, subbing in `N` `0`s on the right

Shift Left

- Move all the bits `N` positions to the left, subbing in `N` `0`s on the right

`1001`

Shift Left

- Move all the bits `N` positions to the left, subbing in `N` `0`s on the right

```
1001 << 2 =
100100
```

Shift Left

- Useful as a restricted form of multiplication
- Question: how?

```
1001 << 2 =
100100
```

Shift Left as Multiplication

- Equivalent decimal operation:

```
234
```

Shift Left as Multiplication

- Equivalent decimal operation:

```
234 << 1 =
2340
```

Shift Left as Multiplication

- Equivalent decimal operation:

```
234 << 1 =
2340

234 << 2 =
23400
```

Multiplication

- Shifting left `N` positions multiplies by `(base)`^{N}
- Multiplying by 2 or 4 is often necessary (shift left 1 or 2 positions, respectively)
- Often a whooole lot faster than telling the processor to multiply
- Compilers try hard to do this

```
234 << 2 =
23400
```

Shift Right

- Move all the bits `N` positions to the right, subbing in **either** `N` `0`s or `N` `1`s on the left
  - Two different forms

Shift Right

- Move all the bits `N` positions to the right, subbing in **either** `N` `0`s or `N` (whatever the leftmost bit is)s on the left
  - Two different forms

```
1001 >> 2 =
either 0010 or 1110
```

Shift Right Trick

- Question: If shifting left multiplies, what does shift right do?

Shift Right Trick

- Question: If shifting left multiplies, what does shift right do?
  - Answer: divides in a similar way, but truncates result

Shift Right Trick

- Question: If shifting left multiplies, what does shift right do?
  - Answer: divides in a similar way, but truncates result

Shift Right Trick

- Question: If shifting left multiplies, what does shift right do?
  - Answer: divides in a similar way, but truncates result

```
234 >> 1 =
23
```

Two Forms of Shift Right

- Subbing in `0`s makes sense
- What about subbing in the leftmost bit?
    - And why is this called "arithmetic" shift right?

```
1100 (arithmetic)>> 1 =
1110
```

Answer...Sort of

- Arithmetic form is intended for numbers in *twos complement*, whereas the non-arithmetic form is intended for *unsigned* numbers

Twos Complement

Problem

- Binary representation so far makes it easy to represent positive numbers and zero
- Question: What about representing negative numbers?

Twos Complement

- Way to represent positive integers, negative integers, and zero
- If `1` is in the *most significant bit* (generally leftmost bit in this class), then it is negative

Decimal to Twos Complement

- Example: -5 decimal to binary (twos complement)

Decimal to Twos Complement

- Example: -5 decimal to binary (twos complement)
- First, convert the magnitude to an unsigned representation

Decimal to Twos Complement

- Example: -5 decimal to binary (twos complement)
- First, convert the magnitude to an unsigned representation

5 (decimal) = 0101 (binary)

Decimal to Twos Complement

- Then, take the bits, and negate them

Decimal to Twos Complement

- Then, take the bits, and negate them

```
0101
```

Decimal to Twos Complement

- Then, take the bits, and negate them

```
~0101 =
 1010
```

Decimal to Twos Complement

- Finally, add one:

Decimal to Twos Complement

- Finally, add one:

```
1010
```

Decimal to Twos Complement

- Finally, add one:

```
1010 + 1 =
1011
```

Twos Complement to Decimal

- Same operation: negate the bits, and add one

Twos Complement to Decimal

- Same operation: negate the bits, and add one

```
1011
```

Twos Complement to Decimal

- Same operation: negate the bits, and add one

```
~1011 =
 0100
```

Twos Complement to Decimal

- Same operation: negate the bits, and add one

```
0100
```

Twos Complement to Decimal

- Same operation: negate the bits, and add one

```
0100 + 1 =
0101
```

Where Is Twos Complement From?

- Intuition: try to subtract 1 from 0, in decimal
  - Involves borrowing from an invisible number on the left
  - Twos complement is based on the same idea