COMP 122/L Lecture 1

Kyle Dewey

About Me

- My research:
  - Automated program testing + CS education
  - Programming language design (with JPL)
- Lots of experience with functional and logic programming
- Third time teaching this class, fifth time teaching this content

About this Class

- See something wrong? Want something improved? Email me about it! (kyle.dewey@csun.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

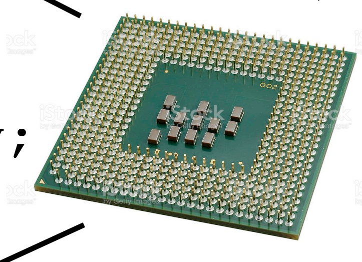
-I can actually do something about this!

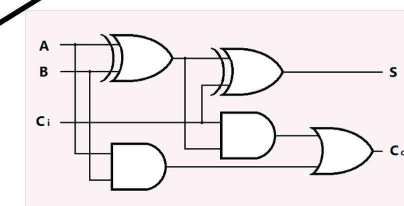
What's this Class About?

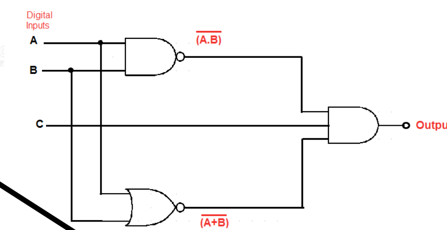
Software / hardware interface

```
int main() {
   ...
}
bool z = x && y;

while (...) {
   ...
}
```







Class Structure

- Numerical representation (what do we represent numbers on the machine?)
- Numerical operations (how does the processor do numeric operations?)
- Assembly (how do we talk directly to the processor?)
- Circuits (how can we build a processor?)

Class Motivation

```
public static void
main(String[] args) {
   ...
}
```

–I just want to write my code

```
public static void
main(String[] args) {
   ...
}
```

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

```
public static void
main(String[] args) {
   ...
}
```

3.14956

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

```
public static void
main(String[] args) {
   ...
}
```

3.14956

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

```
public static void
main(String[] args) {
   ...
}
```

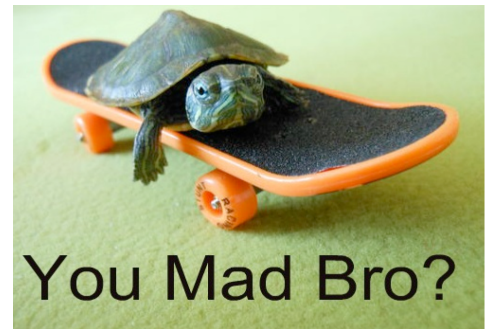
More Efficient Algorithms

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...

```
public static void
main(String[] args) {
   ...
}
```

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

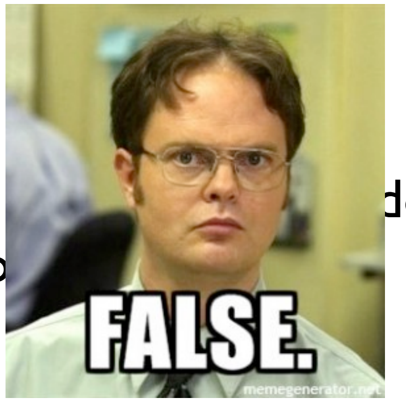
Array Access

`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 randomly is slow
- Can make code *exponentially* faster

-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;
```

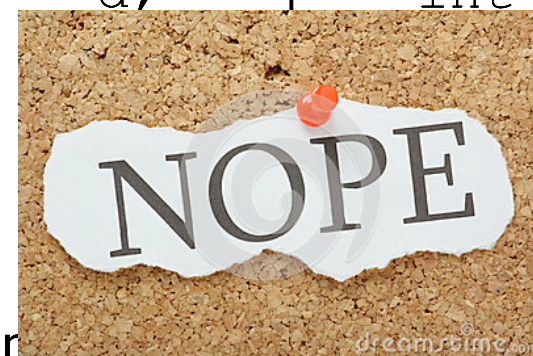
3 Milliseconds?

-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 Millisecor

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
- Processors executing x86(_64) are complex

The Point

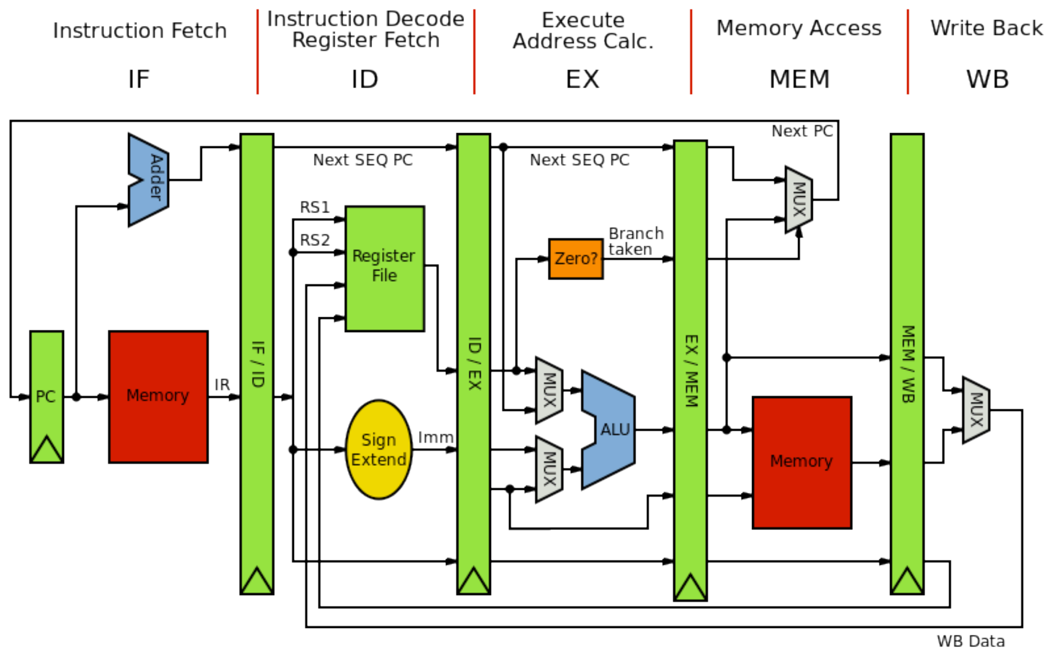
- If you really want performance, you need to know how the magic works
  - "But it scales!" - restrictions apply
  - Chrome is fast for a reason
- If you want to write a naive compiler, you need to know some low-level details
- If you want to write a *fast* compiler, you need to know *tons* of low-level details

-A bunch of Chrome is written using low-level machine instructions (assembly)
-Software has to be built to scale from the start. The first step to scaling is to make the code run fast on a single core; scaling is a performance multiplier, so it makes sense to start with high performance
-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 Circuits?

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

So Why Circuits?



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

So Why Circuits?

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

Lecture vs. Lab

- They're graded as if it's one class (single grade)
- Many days won't be a 35 minute lecture with a 35 minute lab (depends on where we are and what we're doing)
  - Sometimes more lecture will be needed, other times more lab is needed

Syllabus

Working with Different Bases

Base-10 (Decimal)

- Our number system is **base-10**; we have 10 possible digits for each position in a number: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - 192, 9034, 42, 118, ...
- Why?

Base-2 (Binary)

- Only two digits: 0, 1
  - 010, 1101, 11100101, ...
- Extremely popular in computing - why?

Why Care?

- Processors natively "speak" binary
- If you want to speak directly to the processor, you have to speak it's language (to some degree)

- Direct binary is admittedly very inconvenient, because the numbers get BIG. We often instead use shorthands which have a one-to-one translation to binary. This'll make more sense when we get to assembly.

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

Conversion from Some Base to Decimal

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

Conversion from Some Base to Decimal

	Remainder
10 | 231	
10 | 23	1
10 | 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	Fours	Twos	Ones
1×2^3	0×2^2	1×2^1	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	Fours	Twos	Ones
0×2^3	1×2^2	0×2^1	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
$2 \,\underline{\big\vert\; 57}$	
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

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

1	A	F

Hexadecimal Example

1	A	F
Two-fifty-sixes	Sixteens	Ones

Hexadecimal Example

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