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
- Fifth time teaching; second 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.
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
Questions

• Which best describes you?
  • CS major
  • ECE major
  • Other
Office Hours
Placement
Class Motivation
int main(int argc, char** argv) {
    
    }

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

Tuesday, January 5, 16
int main(int argc, char** argv) {
    ...
}

3.14956
int main(int argc, char** argv) {
    ...
}

3.14956
int main(int argc, char** argv) {
    ...
}

More Efficient Algorithms

3.14956
int main(int argc, char** argv) {
    ...
}

More Efficient Algorithms

You Mad Bro?

3.14956
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

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

\[ \text{arr}[x] \]
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 $\sim O(N^4)$
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;
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;

3 Milliseconds?
3 Milliseconds?
Instruction Ordering

\[
\begin{align*}
\text{int } x &= a + b; \\
\text{int } y &= c \times d; \\
\text{int } z &= e - f;
\end{align*}
\]

3 Milliseconds?
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
So Why Digital Design?
So Why Digital Design?
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?
What’s In a Number?
What’s In a Number?

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>
What’s In a Number?

<table>
<thead>
<tr>
<th>Hundreds</th>
<th>Tens</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
What’s In a Number?

<table>
<thead>
<tr>
<th></th>
<th>Hundreds</th>
<th>Tens</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Value</td>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Tuesday, January 5, 16
Question

- Why did we go to tens? Hundreds?

<table>
<thead>
<tr>
<th>Hundreds</th>
<th>Tens</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Answer

- Because we are in decimal (base 10)

<table>
<thead>
<tr>
<th>Hundreds</th>
<th>Tens</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another View

123
## Another View

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \times 10^2$</td>
<td>$2 \times 10^1$</td>
<td>$3 \times 10^0$</td>
</tr>
</tbody>
</table>

Tuesday, January 5, 16
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

231
Conversion from Some Base to Decimal

\[
\begin{array}{c|c}
10 & 231 \\ \hline
23 & 1
\end{array}
\]
Conversion from Some Base to Decimal

<table>
<thead>
<tr>
<th>10</th>
<th>23</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Conversion from Some Base to Decimal

<table>
<thead>
<tr>
<th>10</th>
<th>23</th>
<th>23</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>23</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Now for Binary

<table>
<thead>
<tr>
<th>Eights</th>
<th>Fours</th>
<th>Twos</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Tuesday, January 5, 16
Now for Binary

<table>
<thead>
<tr>
<th></th>
<th>Eights</th>
<th>Fours</th>
<th>Twos</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1 \times 2^3$</td>
<td>$0 \times 2^2$</td>
<td>$1 \times 2^1$</td>
<td>$0 \times 2^0$</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Question

• What is binary 0101 as a decimal number?
Answer

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

<table>
<thead>
<tr>
<th></th>
<th>Eights</th>
<th>Fours</th>
<th>Twos</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 x 2^3</td>
<td>0</td>
<td>1 x 2^2</td>
<td>0 x 2^1</td>
<td>1 x 2^0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Tuesday, January 5, 16
From Decimal to Binary

• What is decimal 57 to binary?
From Decimal to Binary

57
From Decimal to Binary

\[
\begin{array}{c|c
}
2 & 57 \\
\hline
 & 28 \\
\end{array}
\]

Remainder

1
From Decimal to Binary

\[
\begin{array}{c|c}
2 & 57 \\
2 & 28 \\
2 & 14 \\
\end{array}
\]

Remainder
\[
\begin{array}{c}
1 \\
0 \\
\end{array}
\]
From Decimal to Binary

<table>
<thead>
<tr>
<th>2</th>
<th>57</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

Remainder:

1
0
0
From Decimal to Binary

2 | 57
2 | 28
2 | 14
2 | 7
  | 3

Remainder
  1
  0
  0
  1
### From Decimal to Binary

<table>
<thead>
<tr>
<th>2</th>
<th>57</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
From Decimal to Binary

<table>
<thead>
<tr>
<th>2</th>
<th>57</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Octal

- Octal is base 8
- Same idea
Octal Example

• What is 172 octal in decimal?
Octal Example

172
## Octal Example

| 1 | 7 | 2 |
## Octal Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sixty-fours</strong></td>
<td><strong>Eights</strong></td>
<td><strong>Ones</strong></td>
</tr>
<tr>
<td>1 x $8^2$</td>
<td>7 x $8^1$</td>
<td>2 x $8^0$</td>
</tr>
</tbody>
</table>
## Octal Example

<table>
<thead>
<tr>
<th></th>
<th>Sixty-fours</th>
<th>Eights</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1 \times 8^2$</td>
<td>$7 \times 8^1$</td>
<td>$2 \times 8^0$</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>8 8 8 8 8 8 8 8</td>
<td>1 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(56)</td>
<td></td>
</tr>
</tbody>
</table>
### Octal Example

**Answer:** 122

<table>
<thead>
<tr>
<th></th>
<th>Sixty-fours</th>
<th>Eights</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1 \times 8^2$</td>
<td>$7 \times 8^1$</td>
<td>$2 \times 8^0$</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>8 8 8 8 8 8 8 8</td>
<td>1 1</td>
</tr>
<tr>
<td></td>
<td>(56)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From Decimal to Octal

• What is 182 decimal to octal?
From Decimal to Octal

182
From Decimal to Octal

\[
\begin{array}{c|c}
8 & 182 \\
\hline
 & 22 \\
\end{array}
\]

Remainder

6
From Decimal to Octal

<table>
<thead>
<tr>
<th>8</th>
<th>182</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Remainder

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
From Decimal to Octal

\[
\begin{array}{c|c}
8 & 182 \\
 8 & 22 \\
 8 & 2 \\
 0 & \text{Remainder} \\
\end{array}
\]

\[
\begin{array}{c|c}
& 6 \\
& 6 \\
& 2 \\
\end{array}
\]
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 IAF hexadecimal in decimal?
# Hexadecimal Example

<table>
<thead>
<tr>
<th>I</th>
<th>A</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A</td>
<td>F</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Two-fifty-sixes</td>
<td>Sixteens</td>
<td>Ones</td>
</tr>
</tbody>
</table>
# Hexadecimal Example

<table>
<thead>
<tr>
<th></th>
<th>Two-fifty-sixes</th>
<th>Sixteens</th>
<th>Ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$1 \times 16^2$</td>
<td>$10 \times 16^1$</td>
<td>$15 \times 16^0$</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Hexadecimal Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A</td>
<td>F</td>
</tr>
<tr>
<td><strong>Two-fifty-sixes</strong></td>
<td><strong>Sixteens</strong></td>
<td><strong>Ones</strong></td>
</tr>
<tr>
<td>$1 \times 16^2$</td>
<td>$10 \times 16^1$</td>
<td>$15 \times 16^0$</td>
</tr>
<tr>
<td>256</td>
<td>16 16 16 16 16 (160)</td>
<td>1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A (10)</td>
<td>1010</td>
</tr>
<tr>
<td>B (11)</td>
<td>1011</td>
</tr>
<tr>
<td>C (12)</td>
<td>1100</td>
</tr>
<tr>
<td>D (13)</td>
<td>1101</td>
</tr>
<tr>
<td>E (14)</td>
<td>1110</td>
</tr>
<tr>
<td>F (15)</td>
<td>1111</td>
</tr>
</tbody>
</table>
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: |

\[(1001 \mid 0101) = 1101\]
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: ~

\[ \sim 1001 = 0110 \]
• 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 $N$ 0s on the right

1001
Shift Left

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

$$1001 << 2 = 100100$$
Shift Left

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

$$1001 \ll 2 = 100100$$
Shift Left as Multiplication

- Equivalent decimal operation:

234
Shift Left as Multiplication

- Equivalent decimal operation:

\[ 234 \ll 1 = 2340 \]
Shift Left as Multiplication

- Equivalent decimal operation:

  \[ 234 \ll 1 = \]
  \[ 2340 \]

  \[ 234 \ll 2 = \]
  \[ 23400 \]
Multiplication

- Shifting left $N$ positions multiplies by $(\text{base})^N$
- 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

$$234 \ll 2 = 23400$$
Shift Right

• Move all the bits $N$ positions to the right, subbing in \textit{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 $N$ 0s or $N$ (whatever the leftmost bit is)s on the left

• Two different forms

\[ 1001 \gg 2 = \]

\textit{either} 0010 \textit{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

\[ 234 \gg 1 = 23 \]
Two Forms of Shift Right

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

\[1100 \ (\text{arithmetic}) \gg 1 = 1110\]
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

  $5 \text{ (decimal)} = 0101 \text{ (binary)}$
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

\[ \sim 0101 = 1010 \]
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

\[ \sim 1011 = 0100 \]
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