CS 101 Today...

Looking Back

Computing as composition

*clay == functions*

Looking Forward

Computing as representation

*clay == data & bits*

Jotto Corner

<table>
<thead>
<tr>
<th>5C guess</th>
<th>ZD guess</th>
<th>HS guess</th>
<th>ZD guess</th>
</tr>
</thead>
<tbody>
<tr>
<td>party: 3</td>
<td>diner: 1</td>
<td>alien: 2</td>
<td>diner: 2</td>
</tr>
<tr>
<td>??????:</td>
<td>savvy:</td>
<td>??????:</td>
<td>savvy:</td>
</tr>
</tbody>
</table>

Our top-10 list of binary jokes...

On a scale of 1 to 10, how likely is it that this question is using binary?

(?)

What's a 4?

There are only 10 types of people in the world: Those who understand binary and those who don't.

Santiago

Jeff
Some legs to stand on...?

This is heady stuff!

Creating more and more capable compositions

Diagram:

- max
- encipher
- rot(c,n)
- ord
- chr
- sScore
- letScore

Program organization
Some legs to stand on!

It looks like I'm ahead of this...

creating more and more capable compositions

how are even these fundamentals physically realized?!
Binary Storage & Representation

The SAME bits can represent different pieces of data, depending on type.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 0000</td>
<td>32</td>
<td>20</td>
<td>(blank) (str)</td>
</tr>
<tr>
<td>0010 0001</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>0010 0010</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>0010 0011</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>0010 0100</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>0010 0101</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>0010 0110</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>0010 0111</td>
<td>39</td>
<td>27</td>
<td>*</td>
</tr>
<tr>
<td>0010 1000</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>0010 1001</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>0010 1010</td>
<td>42</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>0010 1011</td>
<td>43</td>
<td>2B</td>
<td>+</td>
</tr>
</tbody>
</table>

8 bits = 1 byte = 1 box

value:
name: C

value:
name: X

00101010

The same bits are in each container.

C = "X " 42

But why these bits?
What is 42?
What is 42?

Base 10

42

4 tens + 2 ones

123

1 hundred + 2 tens + 3 ones
Value (semantics)
stuff we care about (what things mean)

Syntax
stuff we need to communicate
Value  (semantics)
stuff we care about (what things *mean*)

Syntax
stuff we need *to communicate*
Write 123 in binary...

123 = 1(64) + 1(32) + 1(16) + 0(8) + 0(4) + 1(2) + 1(1)

123 base 10 = 1 hundred + 2 tens + 3 ones

123 base 2 = 101010

Base 2:
- ONEs column
- TWOs column
- FOURs column
- EIGHTs column
- SIXTEENs column
- THIRTYTWOs column

Base 10:
each column represents the base's next power

42 base 10 = 4 tens + 2 ones

123 base 10 = 1(100) + 2(20) + 3(1)
Write 123 in binary...

Base 2

1010110

Base 10

42

4 tens + 2 ones

123

1 hundred + 2 tens + 3 ones
Binary math

[Diagram showing binary addition and multiplication tables]

Decimal math

[Table showing decimal addition and multiplication facts]

www.youtube.com/watch?v=Nh7xapVB-Wk
In binary, I'm an 11-eyed alien!

**Convert these two binary numbers to decimal:**

- **Binary:** 110011
  - **Decimal:** 32 + 16 + 8 + 2 + 1 = 51

- **Binary:** 10001000
  - **Decimal:** 128 + 8 = 136

**Convert these two decimal numbers to binary:**

- **Decimal:** 28
  - **Binary:** 011000

- **Decimal:** 101
  - **Binary:** 101

**Add these two binary numbers:**

- **Binary:** 101101 + 1110 = 101100

**Multiply these binary numbers:**

- **Binary:** 101101
  - **Binary:** 1110
  - **Result:** 101101

**Extra!** Can you figure out the last binary digit (bit) of 53 without determining any other bits? The last two? 😲

**Name(s): ____________________________**

**Hint:** Remember these algorithms? They're the same in binary!
Convert these two binary numbers to decimal:

110011

Values in blue:

32 + 16 + 2 + 1 = 51

10001000

128 + 8 = 136

Convert these two decimal numbers to binary:

28

Values in blue:

011100

10110

101,10

Extra! Can you figure out the last binary digit (bit) of 53 without determining any other bits? The last two? 3?

We'll return to this in a bit...
Add these two binary numbers \textit{WITHOUT} converting to decimal!

\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}

\begin{array}{cccccc}
101101 & 45 \\
+ 1110 & 14 \\
\hline
1110101 & 59 \\
\end{array}

\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}

\text{Hint:} \quad \frac{\begin{array}{c}
529 \\
+ 742 \\
\hline
1271 \\
\end{array}}{}

Do you remember this algorithm? It's the same!
Add these two binary numbers *WITHOUT* converting to decimal!

```
+ 101101
+ 1110
---
111010
```

**Hint:**

```
529
+ 742
---
1271
```

Do you remember this algorithm? It's the same!
Multiply these two binary numbers **WITHOUT** converting to decimal!

```
101101
* 1110
```

```
+ 2116
-----
22218
```

**Goal**

```
529
* 42
1058
+ 2116
22218
```

**Hint:**
Do you remember this algorithm? It's the same!

A machine could - and probably *should* - be doing this!
Multiply these two binary numbers **WITHOUT** converting to decimal!

```
  101101
*  1110
---------
  000000
 1011010
10110100
+ 101101000
---------
1001110110
```

**Hint:**
Do you remember this algorithm? It's the same!

```
529
* 42
---
1058
+ 2116
---
22218
```

"partial products"

**Goal**

A machine could - and probably **should** - be doing this!
There are 10 kinds of "people" in the universe:
- those who know ternary,
- those who don't, and
- those who think this is a binary joke!
Which of these isn't 42...?

- base 2: $101010 (\text{digits: 0, 1})$
- base 3: $1120 (\text{digits: 0, 1, 2})$
- base 10: $42 (\text{digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9})$
- base 11: $39 (\text{digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9})$
- base 16: $11 (\text{digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F})$

and what are the bases of the rest?
<table>
<thead>
<tr>
<th>Base</th>
<th>Number</th>
<th>Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>101010</td>
<td>0, 1</td>
</tr>
<tr>
<td>3</td>
<td>1120</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>4</td>
<td>222</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0, 1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>0, 1, 2, 3, 4, 5, 6</td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8</td>
</tr>
<tr>
<td>10</td>
<td>42</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>11</td>
<td>39</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A</td>
</tr>
<tr>
<td>16</td>
<td>2A</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F</td>
</tr>
</tbody>
</table>

All 42s!
All 42s!
Our Mascot, the Panda

Off base?

Base 12 –
"Duodecimal Society"
"Dozenal Society"

Base 20: Americas

Olmec base-20 numbers
E. Mexico, ~ 300 AD

Base 27: New Guinea

Telefol is a language spoken by the Telefol people in Papua New Guinea, notable for possessing a base-27 numeral system.

Base 60 – Ancient Sumeria

Some of these bases are still echoing around...
But why binary?
Ten symbols is too many!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

What digits are these?

Ouch!
Ten symbols is too many!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

This is too difficult to replicate billions of times

What digits are these?

Ouch!
Two symbols is easiest!

A computer has to differentiate *physically* among all its possibilities.

10 symbols ~ 10 different voltages

0 1

2 symbols ~ 2 different voltages

0 1

What digits are these?
Two symbols is easiest!

A computer has to differentiate physically among all its possibilities.

ten symbols ~ ten different voltages

two symbols ~ two different voltages

What digits are these?
Ternary computers?

50 of these *Setun* ternary machines were made at Moscow U. ~ 1958

This project was discontinued in 1970... *though not because of the ternary design!*
Eye-catching submissions...

ASCII wanderings...

Noah H.

Jack W.
Spirals...
Rainbow tri-blade boomerang.jpg

Violet and blue-studded snowflake kite.jpg
a turtle-drawn portrait from turtle graphics ...
Back to bits… not the original name...
"bit" first appeared in print in 1948 (Claude Shannon)

early document allocating different bits to control or data portions of a processor's work

Extra! Can you figure out the last binary digit (bit) of 53 without determining any earlier bits? The last two? three? All of them?
Lab 5: Computing in binary

base 2  |  base 10
---------|---------
... 4 2 1 | 100 10 1 = 141

This first step of left-to-right conversion into binary is tricky to program... Why?

You mean aside from having to think in binary?
Lab 5: Computing in binary

This first step of left-to-right conversion into binary is tricky to program... *Why?*

It's tricky to find the largest power needed...
in the end, we need "53"-worth of value

```
... 32s  16s  8s  4s  2s  1s
...  16s  8s  4s  2s  1s
...   8s  4s  2s  1s
...   4s  2s  1s
...   2s  1s
```

Extra!  Can you figure out the last binary digit (bit) of 53 without determining any earlier bits?  The last two?  three?  All of them?
Lab 5: Converting to binary...

base 10

100 10 1
141

= base 2

128 64 32 16 8 4 2 1

What does the fact that 141 is ODD tell us?!

Try right-to-left!

141 = 10001101

answer
Lab 5: Computing in binary

You'll write these right! (-to-left)

<table>
<thead>
<tr>
<th>base 10</th>
<th>base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>'10001101'</td>
</tr>
</tbody>
</table>

Right-to-left works!

numToBinary( N )

binaryToNum( S )

we need to represent binary numbers with strings

n2b(141)

b2n('10001101')
Lab 5: Computing in binary

def numToBinary( N ):
    if N == 0:
        return ''
    elif N%2 == 0:
        return numToBinary( N ) + '0'
    else:
        return numToBinary( N ) + '1'

How much VALUE is left to convert!?

If N is even, what is the final bit?

If N is odd, what is the final bit?
def numToBinary( N ):
    if N == 0:    return ''
    elif N%2 == 0:
        return numToBinary( N//2 ) + '0'
    else:
        return numToBinary( N//2 ) + '1'

Lab 5: Binary conversion

empty string means 0

If N is even, what is the final bit?

If N is odd, what is the final bit?

How much VALUE is left to convert!?
Reasoning, *bit by bit*

- **Left-shift** (`<<`)
- **Right-shift** (`>>`)
- **And** (`&`)
- **Or** (`|`)

**And** (both) and **Or** (either)

**Bitwise and**
- 5: 101
- 6: 110
- $5 \& 6 \\and 100$
- 4

**Bitwise or**
- 5: 101
- 6: 110
- $5 \mid 6 \\| 111$
- 7

**Bitwise and**
- 11: 1011
- 5: 0101
- $11 \& 5 \\& 0001$
- 1

**Bitwise or**
- 11: 1011
- 5: 0101
- $11 \mid 5 \\| 1111$
- 15
Reasoning, *bit by bit*

left-shift by 1

<table>
<thead>
<tr>
<th>11</th>
<th>3 (&lt;&lt;) 1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

left-shift by 2

<table>
<thead>
<tr>
<th>11</th>
<th>3 (&lt;&lt;) 2</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

right-shift by 1

<table>
<thead>
<tr>
<th>101010</th>
<th>42 (&gt;&gt;) 1</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1010</th>
<th>42 (&gt;&gt;) 2</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

What does *left-shifting* do to the value of a #?

What does *right-shifting* do to the value of a #?
Being bit-wise

In processors shifts, ands, ors, adds, and subtractions are very fast, whereas multiplying, dividing, and mod, which are relatively slow.

Try these for a bit...

14: 1110
9: 1001

14 | 9
14 & 9

You don't need to convert to binary for these three...

7 << 1
5 << 4
170 >> 2

left-shift
right-shift

You do need to use binary for these two!

14
9

Given this, what is a way to compute these expressions using only fast operations, maybe in combination?

N // 4
N*7
N*17
N%16
Being bit-wise

In processors shifts, ands, ors, adds, and subtractions are very fast, whereas multiplying, dividing, and mod, which are relatively slow.

Given this, what is a way to compute these expressions using only fast operations, maybe in combination?

\[
\begin{align*}
N \div 4 \\
N \times 7 \\
N \times 17 \\
N \% 16
\end{align*}
\]

Try these for a bit...

<table>
<thead>
<tr>
<th>14:</th>
<th>1110</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:</td>
<td>1001</td>
</tr>
</tbody>
</table>

14 | 9 \rightarrow 15

14 \text{ or} 9

14 \text{ and} 9 \rightarrow 8

170 >> 2 \rightarrow 42

5 \ll 4 \rightarrow 80

7 \ll 1 \rightarrow 14

You don't need to convert to binary for these three...

You do need to use binary for these two!

1111

1000

left-shift

right-shift

Let's first look at why you'd bother ... !?
In processors **shift**, **and**, **or**, **add**, and **subtract** are **much faster** than **multiply**, **divide**, and **mod**, which are **relatively slow**.

---

Old Microsoft *systems-interview* question, #42:

42. Give a fast way to multiply a number by 7.

42. How would you go about finding out where to find...
Given this, what is a way to compute these statements using combinations from only the fast operations above?

\[ \frac{N}{4} \quad \Rightarrow \quad N \gg \_ \]

\[ N \times 7 \quad \Rightarrow \]

\[ N \times 17 \quad \Rightarrow \]

\[ N \% 16 \quad \Rightarrow \]
In processors **shift**, and, **or**, **add**, and **subtract** are **much faster** than multiply, divide, and **mod**, which are **relatively slow**.

Given this, what is a way to compute these statements using combinations from only the **fast** operations above?

\[
\begin{align*}
N \div 4 & \rightarrow N \gg 2 \\
N \times 7 & \rightarrow (N \ll 3) - N \\
N \times 17 & \rightarrow (N \ll 4) + N \\
N \% 16 & \rightarrow N - ( (N \gg 4) \ll 4 )
\end{align*}
\]