CS 101 Today...

Looking Back

Computing as composition

\textit{clay} == \textit{functions}

Looking Forward

Computing as representation

\textit{clay} == \textit{data} & \textit{bits}
Some legs to stand on...?

This is heady stuff!

decipher

max

encipher

rot(c,n)

sScore

letScore

ord

chr

Creating more and more capable compositions

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.

8 bits = 1 byte = 1 box

The same bits are in each container.

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

Base 10

What is 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

SAME!

101010

different
Write 123 in binary...

101010

$32 + 8 + 2 = 42$

each column represents the base's next power

123

1 hundred + 2 tens + 3 ones
Write 123 in binary...

Write 123 in binary...

Base 2

1010110

Base 10

42

4 tens + 2 ones

123

1 hundred + 2 tens + 3 ones
Binary math

Addition

Multiplication

Decimal math

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
</tbody>
</table>

www.youtube.com/watch?v=Nh7ypadVB-Wk
In binary, I’m an 11-eyed alien!

Name(s): ____________________________

Convert these two binary numbers to decimal:

\[ \begin{align*}
32 & \quad 16 & \quad 8 & \quad 4 & \quad 2 & \quad 1 \\
110011 & & & & & \\
10001000 & & & & & \\
\end{align*} \]

Convert these two decimal numbers to binary:

\[ \begin{align*}
32 & \quad 16 & \quad 8 & \quad 4 & \quad 2 & \quad 1 \\
11100 & & & & & \\
1100 & & & & & \\
\end{align*} \]

Add these two binary numbers:

\[ \begin{align*}
101101 & \\
+ & 1110 \\
\hline
110101 \\
\end{align*} \]

Multiply these binary numbers:

\[ \begin{align*}
101101 & \\
\times & 1110 \\
\hline
00000000 \\
01101010 \\
+ 01000000 \\
\hline
10110110 \\
\end{align*} \]

Extra! Can you figure out the last binary digit (bit) of 53 without determining any other bits? The last two? 3?
Convert these two binary numbers *to decimal*:

- \(110011\)
- \(10001000\)

Values in blue:

- \(32 + 16 + 2 + 1\) = 51
- \(128 + 8\) = 136

Convert these two decimal numbers *to binary*:

- \(28\)
- \(101\)\(_{10}\)

Syntax in orange:

- \(011100\)
- \(01100101\)

**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{align*}
\begin{array}{cccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 \\
+ & 1 & 1 & 1 & 0 & \hline
\hline
1 & 1 & 1 & 1 & 0 & 1
\end{array}
\end{align*}

Hint:
\[
\begin{array}{cc}
1 & \frac{529}{16} \\
+ & \frac{742}{16} \\
\hline
1271
\end{array}
\]
Do you remember this algorithm? It's the same!
Add these two binary numbers \textit{WITHOUT} converting to decimal!

\begin{align*}
\begin{array}{c}
\text{529} \\
+ \text{742}
\end{array}
\end{align*}

\begin{array}{ccccccc}
32 & 16 & 8 & 4 & 2 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 \\
+ & 1 & 1 & 1 & 0 \\
\hline
1 & 2 & 1 & 6 & 4 & 5 \\
\end{array}

\text{1271}

\text{Hint:}

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

\[
\begin{array}{cccccc}
 & 32 & 16 & 8 & 4 & 2 & 1 \\
101101 & & & & & & \\
\times & 1110 & & & & & \\
\hline
& 16 & 8 & 4 & 2 & 1 \\
101101 & & & & & & \\
\times & 1110 & & & & & \\
\hline
& 45 & 14 & & & & \\
\end{array}
\]

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

\[
\begin{array}{c}
529 \\
\times \ 42 \\
\hline
1058 \\
+ \ 2116 \\
\hline
22218
\end{array}
\]

A machine could - and probably **should** - be doing this!

Goal: 630
Multiply these two binary numbers **WITHOUT** converting to decimal!

\[
\begin{array}{c}
101101 \\
\times \\
1110 \\
\hline
0000000 \\
1011010 \\
10110100 \\
+ 101101000 \\
\hline
1001110110
\end{array}
\]

"partial products"

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...?

and what are the bases of the rest?
### Base Systems

<table>
<thead>
<tr>
<th>Base</th>
<th>Number</th>
<th>Digits</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>101010</td>
<td>0, 1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1120</td>
<td>0, 1, 2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>222</td>
<td>0, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>0, 1, 2, 3, 4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>0, 1, 2, 3, 4, 5, 6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>0, 1, 2, 3, 4, 5, 6, 7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>42</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>39</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>
All 42s!
Off base?

Base 12 –
"Duodecimal Society"
"Dozenal Society"

Base 20: Americas

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

Base 60 – Ancient Sumeria

Some of these bases are still echoing around...

Base 27: New Guinea

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

Base 20:

10 11 12 13 14 15 16 17 18 19 20

10 20 30 40 50

E. Mexico, ~300 AD
But *why* binary?
Ten symbols is too many!

A computer has to differentiate \textit{physically} among all its possibilities.

ten symbols $\sim$ ten different voltages

\textit{This is too difficult to replicate billions of times}

\begin{center}
\begin{tabular}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{tabular}
\end{center}

What digits are these?

Ouch!
Ten symbols is too many!

A computer has to differentiate \textit{physically} among all its possibilities.

ten symbols \sim ten different voltages

This is too difficult to replicate billions of times

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?
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!*
"bit" first appeared in print in 1948 (Claude Shannon)

early document allocating different bits to control or data portions of a processor's work
Lab 4: Computing in binary

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 4: 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

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 4: Converting to binary...

\[
\begin{array}{c|c}
\text{base 10} & \text{base 2} \\
141 & 10001101 \\
\hline
128 & 1 \\
64 & 0 \\
32 & 0 \\
16 & 0 \\
8 & 1 \\
4 & 1 \\
2 & 0 \\
1 & 1 \\
\end{array}
\]

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

Try right-to-left!

141 = 10001101

answer
Lab 4: Computing in binary

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

\[
\text{numToBinary}(\ N \ )
\]

\[
\text{binaryToNum}(\ S \ )
\]

we need to \textit{represent} binary numbers with \textit{strings}

\[
\text{n2b}(141)
\]

\[
\text{b2n}('10001101')
\]

Right-to-left works!
Lab 4: Computing in binary

```python
def numToBinary(N):
    if N == 0:
        return ''
    elif N % 2 == 0:  # even N
        return numToBinary(N // 2) + '0'
    else:  # odd N
        return numToBinary(N // 2) + '1'
```

If N is even, what is the final bit?
If N is odd, what is the final bit?

---

**base 10**  

| 100 | 10 | 1 | 141 |

**base 2**  

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 10001101 |

How much VALUE is left to convert!?
Lab 4: Binary conversion

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

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** (`|`)

**Bitwise and**

\[
\begin{align*}
5 : & 101 \\
6 : & 110 \\
\& : & 100
\end{align*}
\]

\[
5 \& 6 \Rightarrow 4
\]

**Bitwise or**

\[
\begin{align*}
5 : & 101 \\
6 : & 110 \\
\| : & 111
\end{align*}
\]

\[
5 \| 6 \Rightarrow 7
\]

**Bitwise and**

\[
\begin{align*}
11 : & 1011 \\
5 : & 0101 \\
\& : & \text{odd!}
\end{align*}
\]

\[
11 \& 5 \Rightarrow 1
\]

**Bitwise or**

\[
\begin{align*}
11 : & 1011 \\
5 : & 0101 \\
\| : & 1111
\end{align*}
\]

\[
11 \| 5 \Rightarrow 15
\]
Reasoning, *bit by bit*

**left-shift**

- Left-shift by 1
  - 11
  - 110
  - $3 \ll 1 = 6$

- Left-shift by 2
  - 11
  - 1100
  - $3 \ll 2 = 12$

**right-shift**

- Right-shift by 1
  - 101010
  - 10101
  - $42 \gg 1 = 21$

- Right-shift by 2
  - 1010
  - $42 \gg 2$
Being bit-wise

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

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

\[
\begin{align*}
N &= 14 \\
N &= 9 \\
N/4 &= 14 \\
N*7 &= 170 \\
N*17 &= 5 \\
N%16 &= \text{left-shift} \\
14 &\text{ or } 9 \\
14 &\text{ and } 9 \\
\end{align*}
\]
**Being bit-wise**

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

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

- \( N \div 4 \)
- \( N \times 7 \)
- \( N \times 17 \)
- \( N \mod 16 \)

**Try these for a bit...**

\[ \begin{align*}
14 &\quad 1110 \\
9 &\quad 1001 \\
14 \mid 9 &\quad 15 \\
14 \& 9 &\quad 8 \\
\end{align*} \]
Intel x86 processor instructions and their speeds (2016)

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.
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?

\[
\begin{align*}
N \ll 4 & \Rightarrow N \gg 2 \\
N \times 7 & \Rightarrow N \ll 3 - N \\
N \times 17 & \Rightarrow N \ll 4 + N \\
N \% 16 & \Rightarrow N - (N \gg 4) \ll 4)
\end{align*}
\]
Intel x86 processor instructions and their speeds (2014)

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?

\[
\begin{align*}
N/\!\!/4 & \implies N \gg 2 \\
N \ast 7 & \implies (N \ll 3) - N \\
N \ast 17 & \implies (N \ll 4) + N \\
N \% 16 & \implies N - (N \gg 4) \ll 4
\end{align*}
\]