It’s All (just) Bits

1) Numbers are bits
2) Text is bits
3) Formatted text is bits
4) Pictures are bits
5) Sounds are bits

Paradigm Shift?

Bits can be operated on!
Example: adding two bits

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+0</td>
<td>0 0</td>
</tr>
<tr>
<td>0+1</td>
<td>0 1</td>
</tr>
<tr>
<td>1+0</td>
<td>0 1</td>
</tr>
<tr>
<td>1+1</td>
<td>1 0</td>
</tr>
</tbody>
</table>

Built using http://logic.ly by Josh Tynjala

multiplexer: chooses 1 bit out of 2 using bits!

built using http://logic.ly by Josh Tynjala

how to build a multiplexer out of gates

built using http://logic.ly by Josh Tynjala
Computer Architecture in 1 Slide

• A program is a sequence of instructions
• Both data and program sit in memory (RAM)
• At the heart of your computer is a processor, which simply executes instructions in order, one at a time
• A processor has a few registers for storing intermediate results (like scratch paper)
• There are only a few possible instructions and a few registers; anything more complex is built up from there

Example Instructions

1. R1 ← 100
2. R2 ← 9
3. R3 ← R1 + R2
4. R8 ← 218
5. R6 ← R8/2

Instructions can be encoded as bits: need a code for each operation (e.g., +=1, −→2, /=3) write down the operation code, the values, and the register numbers

Memory

• But we have lots of data; it doesn’t all fit into the few registers. It sits in memory.
• Big idea: address it.

 load into R1 the value in memory at address 153

Illustration of multiplexer: chooses 1 bit out of 2

Illustration of choosing 1 bit out of 4

Set selectors to: 00

0

0

...
choosing 1 bit out of 4
Set selectors to: 01 (bottom to top)

choosing 1 bit out of 4
Set selectors to: 10 (bottom to top)

choosing 1 bit out of 4
Set selectors to: 11 (bottom to top)

Addresses:
000 001 010 011 100 101 110 111

choosing 1 bit out of 8

Program using memory
1. load into R1 the value in memory at address 153

How many layers of multiplexers?
Program using memory
1. load into R1 the value in memory at address 153
2. R1 ← R1 − 32
3. store the value in R1 into memory at address 153

It’s All (just) Bits

**SO WHAT?**