Quick Recap on Administrative Stuff

- Recitations are Mandatory, but you can go to any recitation
- You can miss up to four, but let me know ahead of time
- No cell phones please, or laptops (but a tablet for taking notes is fine)
- Feel free to email or text me anytime

My sincerest apologies to the 10 AM section, since we ran out of time. Here's a quick recap, as well as what we would have covered.

**Binary & Hexadecimal Numbers**

We covered this pretty thoroughly (a little too thoroughly perhaps), so most of it I won't review. However, I learned something cool about converting decimal to binary.

Let's use 713 as an example.
I always converted decimal to binary by hand by dividing by powers of 2.

\[ \text{dec } 713 = \text{ bin } 1011001001 \]

(You can also just view this as repeatedly subtracting the biggest power of 2 that will fit.)

However, you can also successively divide by 2.

\[ \text{dec } 713 = \text{ bin } 1001 \]

Which is pretty neat! Thanks for sharing that with us.
Terminology:

1 bit := 1 binary digit
1 byte := 8 bits
1 word := in RV32, 32 bits

MODEL OF COMPUTATION

(Also called the architecture, or more specifically, instruction set architecture)

In 6.004, we use RISC-V. RISC-V comes in a few flavors; we use specifically RV32I.

The only data type is the word.

We can only store 32 words at a time; each word is held in a register.

There are basically only three types of things that our computer can do — register-register arithmetic operations, load/store operations, and branches/jumps.
REGISTER-REGISTER ARITHMETIC

We can do things like addition, subtraction, shifts, bitwise-AND, bitwise-OR, and a few others.

Example:

1. reg 0 ← reg 0 shifted left by reg 2
2. reg 3 ← reg 0 + reg 1

```
reg 0    0...001101 (0d13)
reg 1    0...000110 (0d6)
reg 2    0...000001 (0d1)
reg 3    0...000000 (0d0)
```

1. LEFT SHIFT

```
reg 0    0...011010 (0d26)
reg 1    0...000110 (0d6)
reg 2    0...000001 (0d1)
reg 3    0...000000 (0d0)
```

2. +

```
reg 0    0...011010 (0d26)
reg 1    0...000110 (0d6)
reg 2    0...000001 (0d1)
reg 3    0...0100000 (0d32)
```

DONE
32 registers of 32 bits each is not a lot — for instance, it’s not enough to store a $33 \times 33$ pixel image.

So there’s this huge array of memory (called “memory”, “main memory”, or “RAM”) that lets us store extra stuff.

Main Memory is arranged as a giant list of bytes.

(we could also draw this more helpfully as)
Example:

**PSEUDO-RISCV CODE**

1. reg5 ← Load 4-byte word from 0x2C
2. reg5 ← Load byte from 0x40
3. STORE reg6 at 0x4B
BRANCHES AND JUMPS

Programs are just a list of steps or operations; sometimes we may want to skip a few steps, go back a few steps (in order to loops), go to different parts of the program based on certain conditions, &c.

Conditions are things like "register 2 = 0?" or "reg 7 > reg 9?".

Terminology:
A skip that always happens (there is no condition) is called a jump.
A skip that may or may not happen (there is a condition) is called a branch.
Example:

**PSEUDO - RISCV CODE**

Start of Loop:  \[ r12 \leftarrow r12 - 1 \]

Go to “Start of loop” if \( r12 > 0 \)

\[ r13 \leftarrow 000000000000CAFE \]

Example:

**PSEUDO - RISCV CODE**

Start of Loop:  \[ r17 \leftarrow r17 \times 2 \]

Go to “continue” if \( R17 \geq R19 \)

Continue:  \[ r18 \leftarrow DEADBEF \]
PUTTING IT ALL TOGETHER

Let's write factorial! Here's a python version:

\[
\begin{align*}
n &= 7 \\
result &= 1 \\
\text{while } n > 0: \\
\quad result &= result \times n \\
\quad n &= n - 1 \\
n &= result
\end{align*}
\]

First, we need to decide which registers to use for "n" and "result" — let's just use r1 and r2!

<table>
<thead>
<tr>
<th>&quot;n&quot;</th>
<th>r1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;result&quot;</td>
<td>r2</td>
<td>1</td>
</tr>
</tbody>
</table>

Next, let's write the body of the loop:

\[
\begin{align*}
r2 &\leftarrow r2 \times r1 \\
r1 &\leftarrow r1 - 1
\end{align*}
\]

There are a few ways to do the loop; here's one way.

start:
\[
\begin{align*}
r2 &\leftarrow r2 \times r1 \\
r1 &\leftarrow r1 - 1 \\
\text{Branch to "start" if } r1 \text{ not equal to zero}
\end{align*}
\]

\[
r1 \leftarrow r2
\]

See you on Friday!