If you want to make an apple pie from scratch, you must first create the universe.

– Carl Sagan
C
-int x = 10;
-x = 2 * x + 15;

MIPS assembly language
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15

MIPS machine language
001000000000001010100000000000001010
00000000000000101010100101000010000000
0010000010100101000000000000001111
C

```c
int sum3(int v[]) {
    return v[0] +
    v[1] +
    v[2];
}

main() {
    ...
    int v[] = ...;
    int a = sum3(v);
    v[3] = a;
    ...
}
```

MIPS assembly language

```asm
sum3:
    lw    r9, 0(r5)
    lw    r10, 4(r5)
    lw    r11, 8(r5)
    add   r3, r9, r10
    add   r3, r3, r11
    jr     r31

main:
    ...
    addi  r5, r0, 1000
    jal   sum3
    sw    r3, 12(r5)
    ...
```
MIPS assembly language

```
sum3:
    lw   r9, 0(r5)
    lw   r10, 4(r5)
    lw   r11, 8(r5)
    add  r3, r9, r10
    add  r3, r3, r11
    jr   r31

main:
    ...
    addi  r5, r0, 1000
    jal   sum3
    sw    r3, 12(r5)
    ...
```

MIPS machine language

```
1000110010101001000000000000000000
1000110010101010100000000000000100
1000110010101010110000000000000100
00000001001010100001100001000000
00000000011010110001100001000000
00000011111100000000000000010000
...
...
...
001000000000010100000001111101000
000011000001000000000000000000000
101011001010001100000000000001100
...
```
Computer System = ?
Input +
Output +
Memory +
Datapath +
Control
1. Fetch
2. Decode
3. Execute

Function Unit

Control Unit

CPU

addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
lw r9, 0(r5)
lw r10, 4(r5)
add r3, r9, r10
sw r3, 12(r5)
Machine language represents program as numbers

- Store in / fetch from memory like other data
- 2 new registers:
  - Program counter (PC): address of next instruction
  - Instruction register (IR): current instruction

Revolutionary idea: a program is *just data*

→ von Neumann Architecture

Alternative:

- Separate memory systems for code and data
→ Harvard Architecture
1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute
1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute

0: lw r9, 4(r5)
4: addi r3, r9, 5
8: jr r31
12: ...
16: ...
20: addi r5, r0, 1000
24: jal 0
28: sw r3, 12(r5)
32: ...
...
...
1000: 10
1004: 20
1008: 30
1012: 40
...
MIPS R3000 ISA (Instruction Set Architecture)
Interface between hardware and software

- memory: load, store, ...
- computational: add, sub, mul, ...
- control: jump, branch, ...
- floating point, cpu and memory management, ...

### Instruction Formats

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rd</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OP</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10010001010101000000000000000100
lw r10, 4(r5)
1-bit Multiplexor
Computation
E.g. Multiplexor

State
E.g. Register

Gates
E.g. AND

Transistors

Logic and States
Why?
void A() {
    for (int i = 0; i < 4096; i++)
        for (int j = 0; j < 4096; j++)
            v[i][j] = f(v[i][j], i, j);
}

0.45 sec, (0.12 sec optimized)

void B() {
    for (int j = 0; j < 4096; j++)
        for (int i = 0; i < 4096; i++)
            v[i][j] = f(v[i][j], i, j);
}

4.05 sec (3.52 sec optimized)
The number of transistors integrated on a single die will double every 24 months...

– Gordon Moore, Intel co-founder, 1965

1971 – 2300 transistors – 1MHz – 4004
1990 – 1M transistors – 50MHz – i486
2001 – 42M transistors – 2GHz – Xeon
2004 – 55M transistors – 3GHz – P4
2007 – 290M transistors – 3GHz – Core 2 Duo
2009 – 731M transistors – 2GHz – Nehalem
Example 3: New Devices

Xilinx FPGA

NVidia GPU

Berkeley mote
Why?

- **Basic knowledge needed for all other areas of CS:**
  operating systems, compilers, ...
- **Levels are not independent**
  hardware design $\leftrightarrow$ software design $\leftrightarrow$ performance
- **Crossing boundaries is hard but important**
  device drivers
- **Good design techniques**
  abstraction, layering, pipelining, parallel vs. serial, ...
- **Understand where the world is going**
http://www.cs.cornell.edu/courses/cs3410

- Office Hours / Consulting Hours
- Lecture slides & schedule
- Logisim
- CSUG lab access (esp. second half of course)

Sections (choose one):

- T 2:55 – 4:10pm  Hollister 110
- W 3:35 – 4:50pm  Hollister 320
- R 11:40 – 12:55pm Hollister 401
- R 2:55 – 4:10pm  Hollister 401
- F 2:55 – 4:10pm  Snee 1150

- Will cover new material
- This week: intro to logisim
A) Love it  
B) Okay  
C) What?  
D) Whatever  
E) Please don’t
Grading:

• 4 Programming Assignments  (35 – 45%)
  – Work in groups of two

• 2 Prelims  (30 – 40%)

• 4-5 Homework Assignments  (20 – 25%)
  – Work alone

• Discretionary  (5%)
Academic Integrity:

• All submitted work must be your own (or your groups)
  – OK to study together, but do not share solutions

• Cite your sources

Stressed? Tempted? Lost?

• Come see me before due date!

Plagiarism in any form will not be tolerated