ECE/CS 314: Computer Organization

"Build a Computer in Three Months"
"Some Assembly Required"

http://www.cs.cornell.edu/courses/cs314

Teaching Staff:

Instructor: Alan Demers <ademers@cs.cornell.edu>

TAs: see course web page

Undergraduate consulting when hw/projects begin.

Slides will be available through the course web page, and handed out in class.

Newsgroups: cornell.class.cs314.*

Administrivia

- Lectures: Olin 155 , TR 1:25-2:40
- WWW: Check announcements regularly!
- Grading:
  - 10% Homework (bi-weekly)
  - 30% 2 Prelims (15+15)
  - 55% 4 Projects (5+10+10+30), groups of 2
  - 5% My discretion
- Homework: Independent work will be required on all parts. Due in class, at the beginning of class.

Administrivia

Sections:

- Every day of the week, MTWRF 2:55-4:10
- Don't skip! New material presented here, esp. project-related.

Labs: PH 329

- Sign up for course account (see web page)
- Unix (FreeBSD) lab, so you can work remotely (`ssh`/VNC).
- Instructions on remote access on the lab page

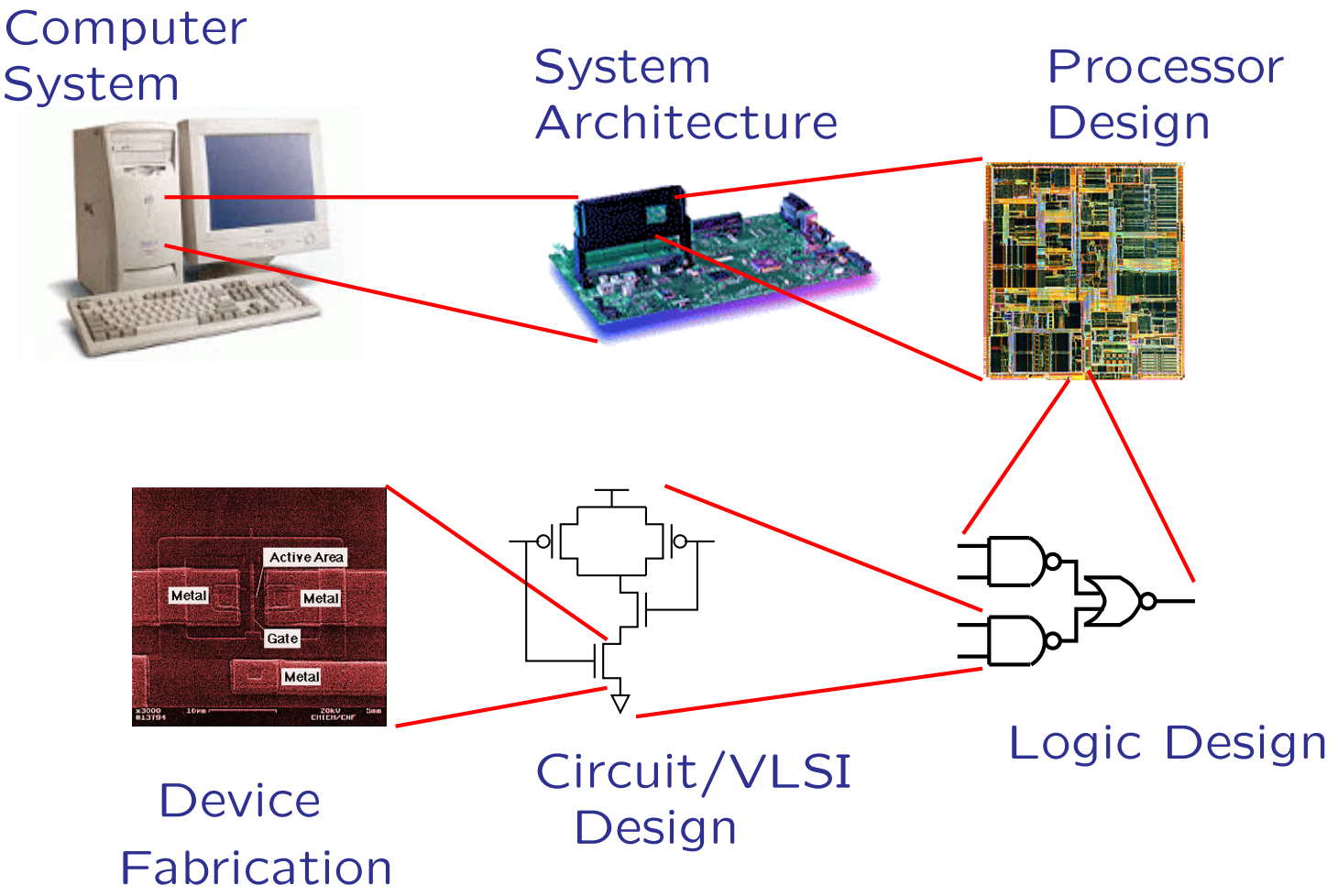
CSL

Administrivia

Instructor: Alan Demers

- Databases
- Distributed Systems

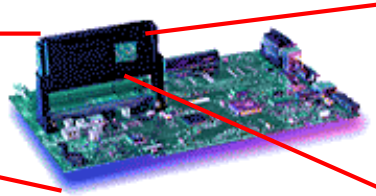
The Anatomy Of A Computer System



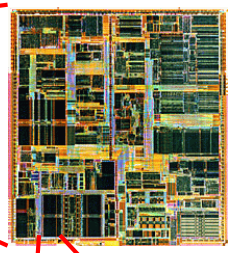
The Anatomy Of A Computer System

Computer System

System Architecture



Processor Design



```
while (event = getnext()) {
     /* process event */
     switch (event->type) {
       case BUTTONUP:
           win = event->W;
           if (!win) break;
           do_button (win);
           break;
       case BUTTONDOWN:
           ...
     }
...
```

Programming Language

```
jal _getnext
ori $a0,$0,0
lw $t0,8($v0)
lw $t0,12($t0)
beq $t0,0,0x401834
li $t1,4
beq $t0,$t1,0x4018a0
```

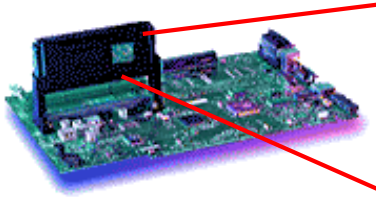
Assembly Language

```
0x0c004841
0x00000000
0x34040000
0x8c480008
0x00000000
0x8d08000c
0x10001834
0x00000000
0x24090004
0x11090002
...
```

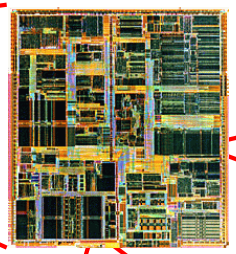
Machine Instructions

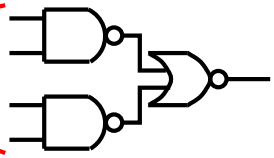
What 314 Is About

System Architecture



Processor Design





Logic Design

```
jal _getnext
ori $a0,$0,0
lw $t0,8($v0)
lw $t0,12($t0)
beq $t0,0,0x401834
li $t1,4
beq $t0,$t1,0x4018a0
```

Assembly Language

```
0x0c004841
0x00000000
0x34040000
0x8c480008
0x00000000
0x8d08000c
0x10001834
0x00000000
0x24090004
0x11090002
...
```

Machine Instructions

Why Should You Know This?

- **Feeds into a number of areas in CS/ECE.** *Operating systems, compilers, languages, circuit design, computer architecture, parallel computer architecture*
- **Important to know how different parts of the system fit together and influence each other.**
- **Impress your friends.**
- **Understand technology news.** *"So why does the new Pentium IV run slower at a higher clock rate?"*
- **It's fun!**

Topics

- Assembly Language Programming
- Processor Design
  - Instruction sets, with emphasis on MIPS
  - Data representation and arithmetic
  - Performance evaluation
  - CPU implementation
- Memory and I/O
  - Caches and virtual memory
  - Buses and I/O systems

Projects

Four projects:

- Assembly language programming
- Processor simulator
- Simple sequential finite state machines
- Pipelined processor
  - At the logic design level
  - Fixed instruction set, plus (unseen) benchmarks

Introduction To Assembly Language

Three basic parts of a computation:

- Data: values being read or produced by the computation
- Operations: add, subtract, multiply, shift, etc.
- Control: what operations should be performed

Introduction To Assembly Language

Assembly language: Description of the basic operations that the computer can perform.

What are these operations? It depends on the computer!
Most support: add, subtract, multiply, shift, compare, jump

- VAX: polyf
- x86: push, pop, multimedia operations
- Sparc: save, restore

"Real Programmers can write assembly code in any language."

CPU Organization

In a modern processor, we have:

- **Registers:** r0, r1, r2, ...
  - Binary encoding, holds N bits of information where $N = 8, 16, 32, 64, 128$
  - Like variables in C
- **Functional units:**
  - Given operation + data produce results
- **Control unit:**
  - Controls logical sequence of operations

Examples Of MIPS Instructions

C Statement:

```
int foo; foo = 15; foo = foo + 7;
```

MIPS Assembly Language:

```
ori $1,$0,15            # set foo to 15
addiu $1,$1,7           # add 7 to foo
```

(register 1 holds the value of foo)

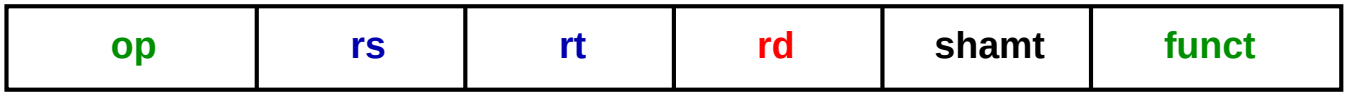
MIPS Machine Instructions:

```
00110100000000010000000000001111
00100100001000010000000000000101
```

Machine Instructions

Encoding Assembly Language:

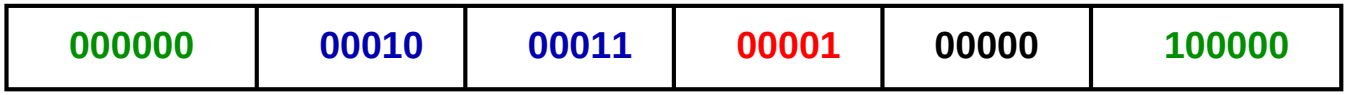
```
add rd, rs, rt          # rd = rs + rt
```



6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Example:

```
add $1, $2, $3
```



Binary Encoding:

$$(00010)_{\text{base 2}} = 0\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 2$$

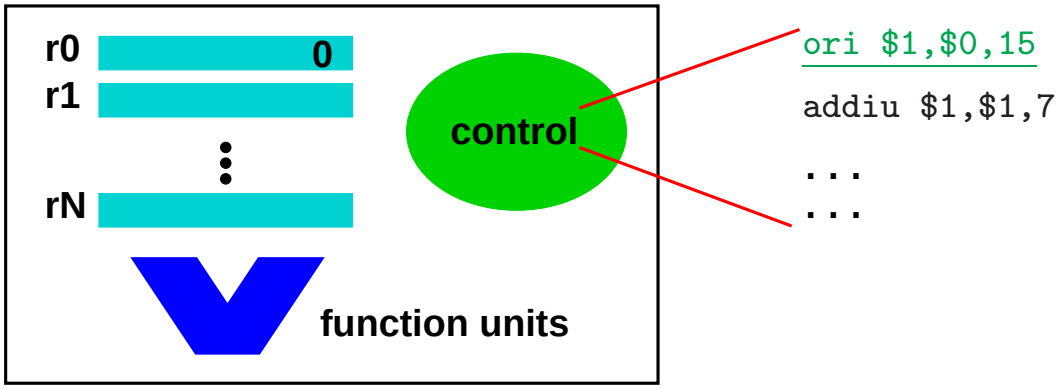
Sequencing: The Control Unit

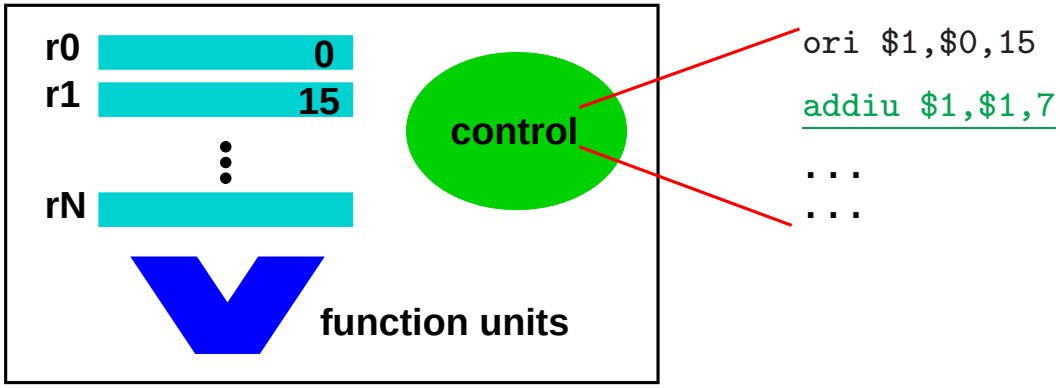
Interprets machine instructions, telling function units what to do.

Basic loop:

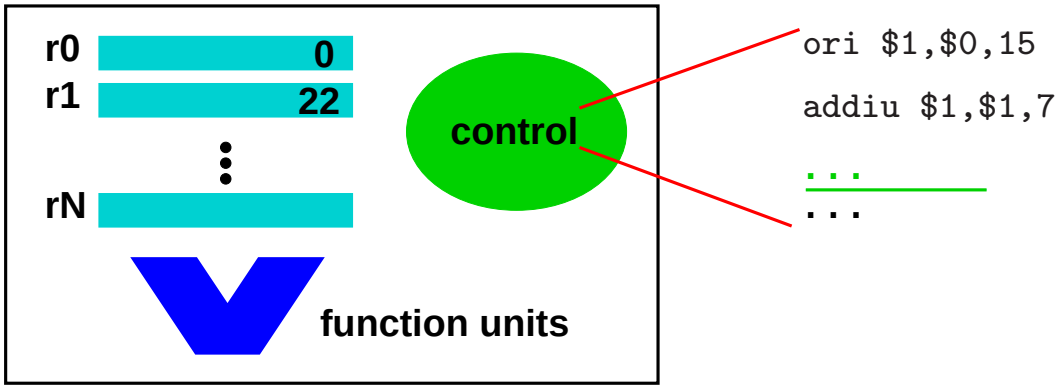
1. Fetch machine instruction
2. Decode it, examine fields
3. Execute the specified operation

A Simple Computer





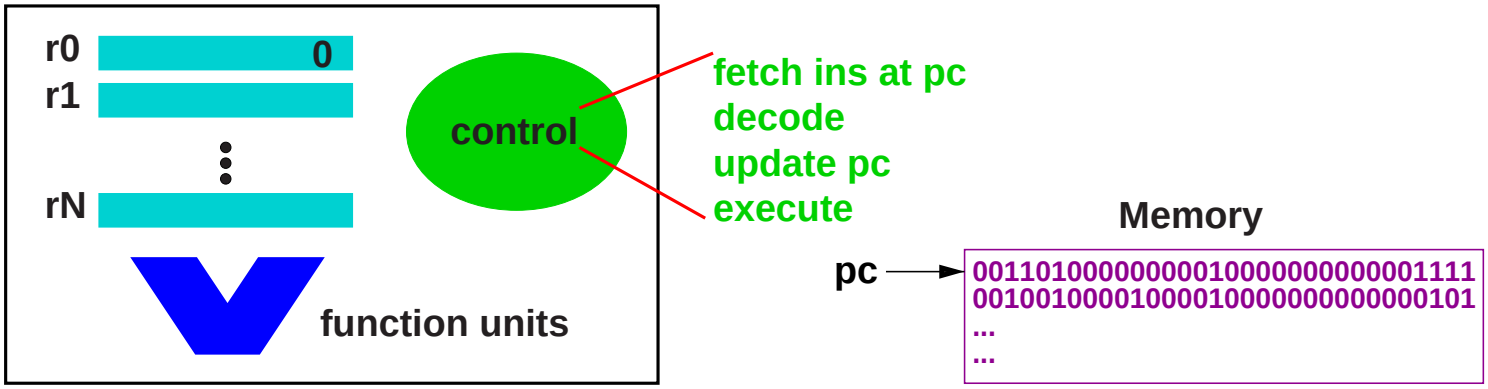
A Simple Computer



New Idea: Instructions are numbers too!

- Memory is a linear array of "registers"
- Store instructions in memory (von Neumann architecture)

A Simple Computer



New register `pc`, the *program counter*

1. Fetch machine instruction at `pc`
2. Decode it, examine fields
3. Update `pc`
4. Execute the specified operation

A Simple Computer