CSE 410: Computer Systems

- Instructor: Gretta Bartels (gretta@cs)
  - Office Hours: M 1:30-2:20, F 2:30-3:20, Sieg 226D
- TAs: Ruth Anderson (rea@cs) and Maureen Chesire (maureen@cs)
  - Office Hours: Ruth W 1:30 (Sieg 226A), Maureen Th 2:30 (Sieg 226B)
- Web page: http://www.cs.washington.edu/410
- Mailing List: cse410@cs

Administrative

- Textbooks
  - Architecture component: *Computer Organization and Design: The Hardware/Software Interface,* 2nd edition. Hennessy and Patterson.
  - Operating Systems component: *Operating Systems Concepts,* 5th edition. Silberschatz and Galvin
- Anonymous feedback

More Administrative

- Schedule (on web page):
  - weeks 1+2: Intro, MIPS assembly
  - week 3+4: Speeding things up (pipelining, the memory hierarchy)
  - Midterm: Wednesday, October 27th
  - weeks 5-10: Operating systems
  - week 11: Wrap up, review
  - Final: Wednesday, December 15th, 8:30am

Yet More Administrative

- Grading
  - 9 Homeworks: 40% (drop lowest)
  - Midterm: 25%
  - Final: 35%
- Homework policy
  - due in class on Monday
  - no late work accepted
  - high level collaboration only

9/27: Lecture Topics

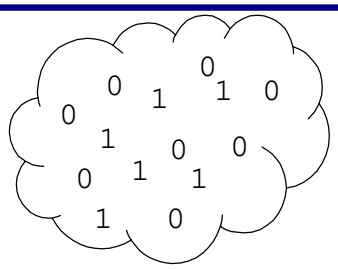
- Administrative stuff
- An overview of computer architecture
  - Organization vs. architecture
  - Levels of abstraction
  - The hierarchy of computer languages
  - Some example architectures
- An introduction to MIPS

Architecture Overview

- How do you talk to a computer?
- What do computers have in them?
- How do computers execute programs?
- How can we speed up execution?
- What are today's hot topics in architecture?

Computerese

- *Bits* = binary digits
- *Instruction* = set of bits forming a command




```
01110101110101001101000110010110
```

- People used to write these by hand...

Machine Language is Tedious

- Writing this is hard
- Debugging this is *impossible*

```
01101000111001100110001
10101010101101010110101
01010101101010001010000
11111010100010101001010
10101000010010101110010
10101010011010101010100
01010101101010001010101
01010100010101011101010
10101001010100100000010
10101001001010101010111
00001010100010101001010
```

Solution #1: Assembly

- Assembly language is just like machine language, but more comfortable for humans

```
01110101110101001101000110010110
```

becomes

```
add A, B
```

Assembly is Also Tedious

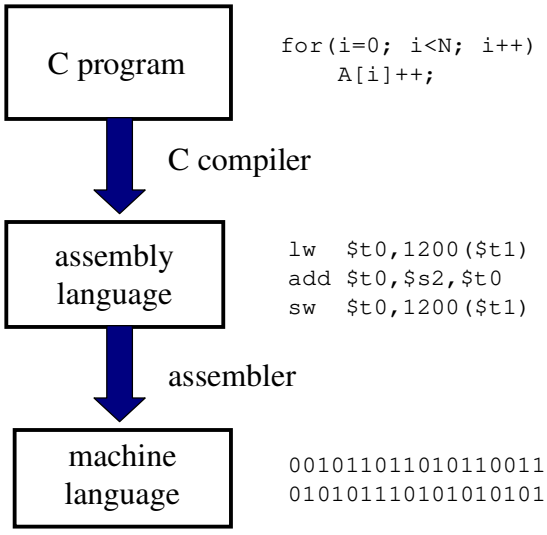
- Each line corresponds to one instruction
- Procedure call is a pain
- Forces the programmer to think like the computer

```
subu    $sp,$sp,32
sw      $ra,20($sp)
sw      $fp,16($sp)
addu    $fp,$sp,28
li      $a0,10
jal     fact
la      $a0,$LC
move    $a1,$v0
jal     printf
lw      $ra,20($sp)
lw      $fp,16($sp)
addu    $sp,$sp,32
```

Sol. #2: High Level Languages

- Programmer can think more naturally
- Different languages for different uses
- Portability
- Enable software reuse (*libraries*)

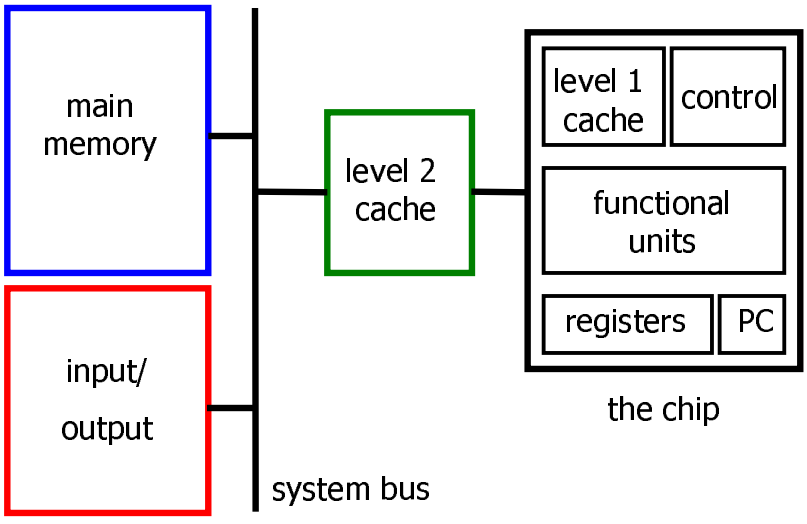
Tower of Babel



Organization vs. Architecture

- *Architecture*: interface between hardware and software
  - e.g. the instruction set, registers, how to access memory
- *Organization*: components and connections
  - e.g. how "mult" is implemented
- Many organizations for one architecture
  - Intel x86, Pentium, Pentium Pro

Computer Organization



Components of Computers

- The processor, or the chip, includes:
  - **Functional units:** logic to manipulate bits
  - **Control:** logic to control the manipulation
  - **Registers** and **Program Counter (PC)**
  - **First level cache**
- Second level cache
- Memory
- Other devices for input and output: disks, etc.

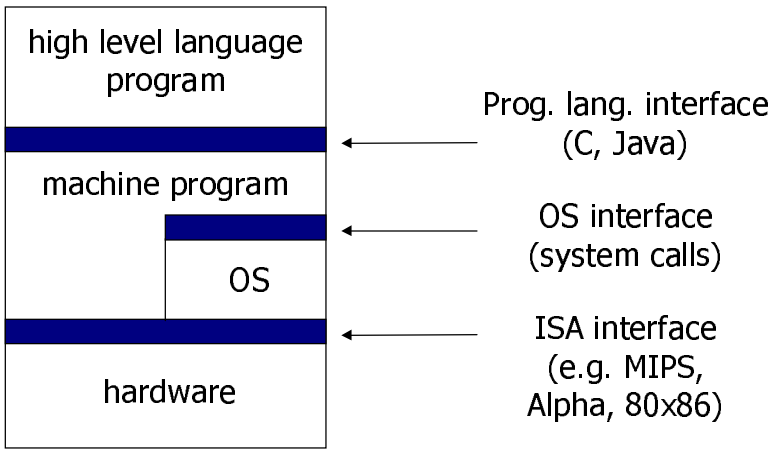
Instruction Set Architecture (ISA)

- All of the specifications necessary to program the machine
  - What instructions does the machine understand?
  - How do the instructions need to be formatted into bits?
  - How many registers?
  - How big is memory?
  - How is memory addressed?
- This should become clearer with MIPS

Architecture Families

- IBM 360, 370, etc.
- IBM PowerPC (601, 603, etc.)
- DEC PDP-11, VAX
- Intel x86 (80286, 80386, 80486, Pentium, etc.)
- Motorola 680x0
- MIPS Rx000, SGI
- Sun Sparc
- Dec Alpha (21x64)

The Bigger Picture



Instruction Sets

- An assembly language has a "vocabulary" of commands
  - add, subtract, shift, branch, etc.
- The set of commands forms the *instruction set*
- Computer architects argue about what should be included

RISC vs. CISC

- Some instruction sets are large and have complex instructions: CISC
  - *Complex Instruction Set Computer*
- Other sets are small and have only simple instructions: RISC
  - *Reduced Instruction Set Computer*
- More on this after you've seen MIPS

MIPS

- The Instruction Set Architecture (ISA) we'll be studying
- A RISC architecture
- Popular for real life
  - NEC, Nintendo, SGI, Sony
- Popular for classes like this one
  - reasonably simple and elegant
  - about 100 instructions total

Operations

- Arithmetic
  - add, sub, mult, div
- Data transfer
  - lw, sw, lb, sb
- Conditional branch
  - beq, bne, slt
- Jump
  - j, jr, jal

Arithmetic Operations

Desired result: a = b + c

MIPS assembly:

```
add a, b, c
```

- Conventions:
  - Always three variables
  - Result goes into first variable
- How do you add up more than two variables?

Desired result: a = b + c + d

MIPS assembly:

```
???
```

More Than Two Variables

Desired result: a = (b + c) + d

MIPS assembly:

```
add a, b, c
add a, a, d
```

- Form more complex operations by combining simple ones
- You can reuse the same variable in more than one place

A Complex Arithmetic Example

Desired result:

f = (g + h) - (i + j)

MIPS assembly:

Registers

- Variables are a high-level language concept
- In assembly, we use *registers*
- A special place on the chip that can hold a (32-bit) word
- There are only a few of them
- They are very fast to access

How Many Registers?

- MIPS has 32 registers
- Intel x86 has only 4 or 8 general-purpose registers
- Why does the number matter?
  - Any data you use has to be in a register
  - If you're using 5 data items and have 4 registers, that's a pain

How Many Registers? cont.

- If registers are so great, why not have lots?
  - space is at a premium
  - the more you have, the less efficient your design
    - they might all get slower
    - it takes more bits to describe which one you mean

MIPS Registers (p. A-23)

Register name	Usage
$zero	constant 0
$at	assembler use only
$v0-$v1	expression evaluation
$a0-$a3	function arguments
$t0-$t9	temporary
$s1-$s7	saved temporary
$k0-$k1	reserved for OS
$gp	pointer to global area
$sp	stack pointer
$fp	frame pointer
$ra	return address