CSE 141 – Computer Architecture
Summer Session 1, 2004

Lecture 1
Introduction & ISA

Pramod V. Argade
CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hours:
  Tue. 7:30 - 8:30 PM (AP&M 4141)
  Wed. 4:30 - 5:30 PM (AP&M 4141)

TA:
  Anjum Gupta (a3gupta@cs.ucsd.edu)
  Office Hour: Mon/Wed 12 - 1 PM
  Chengmo Yang (c5yang@cs.ucsd.edu)
  Office Hour: Mon/Thu 2 - 3 PM

Lecture: Mon/Wed. 6 - 8:50 PM, Center 109

Textbook: Computer Organization & Design
  Authors: Patterson and Hennessy

Web-page: http://www.cse.ucsd.edu/classes/su04/cse141
What is a Computer?

- Components:
  - Input (mouse, keyboard)
  - Output (display, printer)
  - Memory (disk drives, DRAM, SRAM, CD)
  - Network

- Our primary focus: the processor (datapath and control)
  - Implemented using millions of transistors
  - Impossible to understand by looking at each transistor

- Rapidly changing field:
  - Vacuum tube ➔ Transistor ➔ IC ➔ VLSI ➔ SoC

- Interesting history:
  - http://www.islandnet.com/~kpolsson/comphist/
Architecture: Dictionary Definition

- The art and science of designing and erecting buildings.
- A style and method of design and construction.
- Orderly arrangement of parts; structure.
- ...

What is Computer Architecture?

- **Hardware Designer**
  - Thinks about circuits, components, timing, functionality, ease of debugging

  “construction engineer”

- **Computer Architect**
  - Thinks about high-level components, how they fit together, how they work together to deliver performance.

  “building architect”
The Challenge of Computer Architecture

- The industry changes faster than any other.
  - design assuming processor technology trends
- The ground rules change every year.
  - new problems
  - new opportunities
  - different tradeoffs
- It’s all about making programs run faster than the current generation of processors
Forces on Computer Architecture

- Hardware Technology
- Programming Languages
- Applications
- Operating Systems
- Compiler Technology
- Compatibility

Computer Architecture
Moore’s Law*

- Gordon Moore made the observation in 1965, four years after planar integrated circuit was discovered.
- The “law” states that transistor density doubles every ~18 months

*http://www.intel.com/research/silicon/mooreslaw.htm
Computer Architecture’s Changing Definition

- 1950s to 1960s Computer Architecture
  - Computer Arithmetic

- 1970s to mid 1980s Computer Architecture
  - Instruction Set Design, especially ISA appropriate for compilers

- 1990s Computer Architecture
  - Design of CPU, memory system, I/O system, Multiprocessors
  - Instruction level Parallelism
Processors are everywhere!

- PC, Workstation
- Cell Phone
- DVD Player
- Digital Camera
- Car
- Watch
- Pacemaker
- ...

Pramod Argade
CSE 141, Summer Session 1, 2004
Why do you care about architecture?

- If you become a SW designer
  - Understand architecture to better utilize it
  - What are performance bottlenecks & who will fix them?
- If you become a HW designer
  - You may implement a processor architecture in your career
- If you become a System designer
  - You may make HW and SW partition decision
- You may have to make purchasing decisions
  - Buy from vendor A or B? Why?
- Everything amounts to cost/benefit tradeoff!
  - What are the techniques for making such tradeoffs?
Which is faster?

for (i=0; i<N; i=i+1)
  for (j=0; j<N; j=j+1) {
    r = 0;
    for (k=0; k<N; k=k+1)
      r = r + y[i][k] * z[k][j];
    x[i][j] = r;
  }

for (jj=0; jj<N; jj=jj+B)
  for (kk=0; kk<N; kk=kk+B)
    for (i=0; i<N; i=i+1) {
      for (j=jj; j<min(jj+B-1,N); j=j+1)
        r = 0;
      for (k=kk; k<min(kk+B-1,N); k=k+1)
        r = r + y[i][k] * z[k][j];
      x[i][j] = x[i][j] + r;
    }
Which is faster?

```
load R1, addr1
store R1, addr2
add R0, R2 -> R3
subtract R4, R3 -> R5
add R0, R6 -> R7
store R7, addr3
```

```
load R1, addr1
add R0, R2 -> R3
add R0, R6 -> R7
store R1, addr2
subtract R4, R3 -> R5
store R7, addr3
```
Levels of Abstraction

- Delving into the depths reveals more information
- An abstraction omits unneeded detail, helps us cope with complexity

```plaintext
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
SW $15, 0($2)  
LW $16, 4($2)  
SW $16, 0($2)  
SW $15, 4($2)
```

```
10001100011000100000000000000000  
10001100111100100000000000001000  
10101100111100100000000000000000  
10101100011000100000000000001000
```

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```
The five classic components of computers

- Control
- Datapath
- Memory
- Input
- Output
Computer Architecture Topics

Input/Output and Storage
- Disks, WORM, Tape
- RAID

Emerging Technologies
- Interleaving
- Bus protocols

Memory Hierarchy
- DRAM
- L2 Cache
- L1 Cache

VLSI
- Instruction Set Architecture
- Addressing, Protection, Exception Handling

Instruction Set Architecture
- Pipelining, Hazard Resolution, Superscalar, Reordering, Prediction, Speculation, Vector, DSP

Pipelining and Instruction Level Parallelism
What you can expect to get out of this class

- Understand fundamental concepts in computer architecture
- Understand impact on application performance.
- Evaluate architectural descriptions of processors.
- Gain experience designing a working CPU completely from scratch.
- Learn techniques used to evaluate advanced architecture design
Course Outline

- Instruction Set Architecture
- Computer System Performance and Performance Metrics
- Computer Arithmetic and Number System
- CPU Architecture
- Pipelining
- Memory Hierarchy and Caches
- Virtual Memory
Course Work and Grading

- **Course Work**
  - Weekly homework assignments
    - No late assignments accepted
    - No re-grading of homework or quizzes

- **Grading**
  - 10% homework
    - Lowest homework score will be dropped
  - 25% weekly quizzes (10 minutes, every Thursday)
    - Lowest quiz score will be dropped
  - 25% midterm
  - 40% final (which will cover the whole quarter)
  - Tests & Quizzes
    - Closed books and no notes
## CSE141 Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Time</th>
<th>Room</th>
<th>Topic</th>
<th>Quiz topic</th>
<th>Homework Due</th>
</tr>
</thead>
</table>
| 1         | Mon. 6/28| 6 - 8:50 PM | Center 109| Introduction, Ch. 1  
ISA, Ch. 3                                                          | -                   | -            |
| 2         | Wed. 6/30| 6 - 8:50 PM | Center 109| Performance, Ch. 2  
Arithmetic, Ch. 4                                                      | ISA Ch. 3           | #1           |
| 3         | Mon. 7/5 | No Class    |          | July 4th Holiday                                                     | -                   | -            |
| 4         | Wed. 7/7 | 6 - 8:50 PM | Center 109| Arithmetic, Ch. 4 Cont.  
Single-cycle CPU Ch. 5                                              | Performance Ch. 2   | #2           |
| 5         | Mon. 7/12| 6 - 8:50 PM | Center 109| Single-cycle CPU Ch. 5 Cont.  
Multi-cycle CPU Ch. 5                                                | Arithmetic, Ch. 4   | #3           |
| 6         | Tue. 7/13| 7:30 - 8:50 PM | Center 109| Multi-cycle CPU Ch. 5 Cont.  
(July 5th make up class)                                               | -                   | -            |
| 7         | Wed. 7/14| 6 - 8:50 PM | Center 109| Single and Multicycle CPU Examples and Review for Midterm           | Single-cycle CPU Ch. 5 | -          |
| 8         | Mon. 7/19| 6 - 8:50 PM | Center 109| Mid-term Exam Exceptions                                             | -                   | #4           |
| 9         | Tue. 7/20| 7:30 - 8:50 PM | Center 109| Pipelining Ch. 6  
(July 5th make up class)                                               | -                   | -            |
| 10        | Wed. 7/21| 6 - 8:50 PM | Center 109| Hazards, Ch. 6                                                       | -                   | -            |
| 11        | Mon. 7/26| 6 - 8:50 PM | Center 109| Memory Hierarchy & Caches Ch. 7                                     | Hazards Ch. 6       | #5           |
| 12        | Wed. 7/28| 6 - 8:50 PM | Center 109| Virtual Memory, Ch. 7  
Course Review                                                          | Cache Ch. 7         | #6           |
| 13        | Sat. 7/31| 7 - 10 PM   | Center 109| Final Exam                                                           | -                   | -            |
Announcements

- **Reading Assignment:**
  - Performance, Chapter 2
  - Arithmetic, Chapter 4

- **Homework**
  3.1, 3.2, 3.3, 3.7, 3.9, 3.14, 3.17.
  Due Wed., June 30 in class

- **Quiz**
  **When:** Wed., June 30, First 10 minutes of the class
  **Topic:** ISA, Chapter 3
  **Need:** Paper, pen, calculator

- CSE 141 Lab Assignment has been posted on the web page
  http://www.cse.ucsd.edu/classes/su04/cse141L
Instruction Set Architecture (ISA)
General Considerations
Stored Program Concept

- Instructions are bits
- Programs are stored in memory
  — to be read or written just like data
- Fetch & Execute Cycle
  - Instructions are fetched and put into a special register
  - Bits in the register "control" the subsequent actions
  - Fetch the “next” instruction and continue
Memory Organization

- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.

```
0   8 bits of data
1   8 bits of data
2   8 bits of data
3   8 bits of data
4   8 bits of data
5   8 bits of data
6   8 bits of data
...
```
Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32 bits of data</td>
</tr>
<tr>
<td>4</td>
<td>32 bits of data</td>
</tr>
<tr>
<td>8</td>
<td>32 bits of data</td>
</tr>
<tr>
<td>12</td>
<td>32 bits of data</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- $2^{32}$ bytes with byte addresses from 0 to $2^{32}-1$
- $2^{30}$ words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are aligned
  - i.e., what are the least 2 significant bits of a word address?
Endian-ness

- **Big Endian:** address of most significant byte = word address
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- **Little Endian:** address of least significant byte = word address
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

```
0 1 2 3          0 1 2 3
| msb | | | |    | msb | | | |
  0   1   2   3          0 1 2 3
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12
| 0x00 | 0x11 | 0x22 | 0x33 |
  0   1   2   3
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12
| 0x00112233 | 0x44556677 |
  0   1   2   3
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12
| 0x33 | 0x22 | 0x11 | 0x00 |
  0   1   2   3
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12
| 0x00112233 | 0x44556677 |
  0   1   2   3
```

Word Data

```
...Big-endian
```

```
...Little-endian
```
Addressing: Alignment

• Alignment: require that objects fall on address that is multiple of their size.

• MIPS requires address alignment
  • Word addresses must be multiple of 4
  • Half word addresses must be multiple of 2
The Instruction Execution Cycle

- **Instruction Fetch**
  - Obtain instruction from program storage

- **Instruction Decode**
  - Determine required actions and instruction size

- **Operand Fetch**
  - Locate and obtain operand data

- **Execute**
  - Compute result value or status

- **Result Store**
  - Deposit results in storage for later use

- **Next Instruction**
  - Determine successor instruction
Instruction Set Architecture (ISA)

- Instructions: Words of a machine’s language
- Instruction Set: Machine’s vocabulary
- ISA: A very important abstraction
  - Interface between hardware and low-level software
  - Standardizes instructions, machine language bit patterns, etc.
  - Advantage: different implementations of the same architecture
  - Disadvantage: sometimes prevents using new innovations

True or False? Binary compatibility is extraordinarily important.

- Part of the architecture that is visible to the programmer
  - opcodes (available instructions)
  - number and types of registers
  - instruction formats
  - storage access, addressing modes
  - exceptional conditions
The Instruction Set Architecture

- Is the interface between all the software that runs on the machine and the hardware that executes it.

- Provides a “level of abstraction” in both directions
- Modern instruction set architectures:
  80x86/Pentium/K6, MIPS, SPARC, PowerPC, ARM, Tensilica, ...
Key ISA decisions

- Operations
  - how many?
  - which ones
- Operands
  - how many?
  - location
  - types
  - how to specify?
- Instruction format
  - size
  - how many formats?

\( y = x + b \)

(add \ r1, r2, r5)

how does the computer know what 0001 0100 1101 1111 means?
Accessing the Operands

- Operands are generally in one of two places:
  - Registers (how many, how wide?)
  - Memory (how many locations?)

- Registers are
  - Easy to specify
  - Close to the processor
  - Provide fast access
  - Can read two operands and write one result per clock cycle

- The idea that we want to access registers whenever possible led to *load-store architectures*.
  - Normal arithmetic instructions only access registers
  - Only access memory with explicit loads and stores
## Basic ISA Classes

### Comparing the Number of Instructions

Code sequence for $C = A + B$ for four classes of instruction sets:

<table>
<thead>
<tr>
<th>Stack</th>
<th>Accumulator</th>
<th>Register (register-memory)</th>
<th>Register (load-store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push A</td>
<td>Load A</td>
<td>Load R1,A</td>
<td>Load R1,A</td>
</tr>
<tr>
<td>Push B</td>
<td>Add B</td>
<td>Add R1,B</td>
<td>Load R2,B</td>
</tr>
<tr>
<td>Add</td>
<td>Store C</td>
<td>Store C, R1</td>
<td>Add R3,R1,R2</td>
</tr>
<tr>
<td>Pop C</td>
<td></td>
<td></td>
<td>Store C,R3</td>
</tr>
</tbody>
</table>
MIPS Instruction Set Architecture
Key Points

- MIPS is a general-purpose register, load-store, fixed-instruction-length architecture.
- MIPS is optimized for fast pipelined performance, not for low instruction count
- Four principles of ISA
  - Simplicity favors regularity: regular instruction set
  - Smaller is faster: small number of formats, registers
  - Good design demands compromise
  - Make the common case fast
Overview of MIPS ISA

- 3-operand, load-store architecture
- 32 general-purpose registers (integer, floating point)
  - R0 always equals zero.
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits.
- Registers are 32-bits wide (word)
- Register, immediate, base+displacement, PC-relative and pseudo-direct addressing modes
- Fixed 32-bit instructions
- 3 instruction formats will be covered in the class
Is this sufficient?

- Measurements on the VAX show that these addressing modes (immediate, direct, register indirect, and base+displacement) represent 88% of all addressing mode usage.

- Similar measurements show that 16 bits is enough for the immediate 75 to 80% of the time.

- Measurements also show that 16 bits is enough of a displacement 99% of the time.

- ISA is based on gathering usage statistics.
MIPS Instructions

- **Arithmetic**
  - add, sub, addi

- **Logical**
  - (not covered here)

- **Data transfer**
  - lw, sw, lb, sb, lui

- **Conditional branch**
  - beq, bne, slt, slti

- **Unconditional jump**
  - j, jr
## Policy of Use Conventions

<table>
<thead>
<tr>
<th>Name</th>
<th>Register number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>0</td>
<td>the constant value 0</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>2-3</td>
<td>values for results and expression evaluation</td>
</tr>
<tr>
<td>$a0-$a3</td>
<td>4-7</td>
<td>arguments</td>
</tr>
<tr>
<td>$t0-$t7</td>
<td>8-15</td>
<td>temporaries</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>16-23</td>
<td>saved</td>
</tr>
<tr>
<td>$t8-$t9</td>
<td>24-25</td>
<td>more temporaries</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>global pointer</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>stack pointer</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>frame pointer</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>return address</td>
</tr>
</tbody>
</table>
MIPS Arithmetic Instructions

- Design Principle: simplicity favors regularity. Why?
- All instructions have 3 operands
- Operand order is fixed (destination first)
  Example:
  
  C code: \( A = B + C \)
  
  MIPS code: `add $s0, $s1, $s2`
  
  (associated with variables by compiler)
- Operands must be registers, only 32 registers provided
- Design Principle: smaller is faster. Why?
- Instruction format:

<table>
<thead>
<tr>
<th></th>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
</tr>
</tbody>
</table>
How to specify constants?

- Small constants are used quite frequently (50% of operands)
  e.g., \[ A = A + 5; \]
  \[ B = B + 1; \]
  \[ C = C - 18; \]

- Solutions? Why not?
  - Put 'typical constants' in memory and load them.
  - Create hard-wired registers (like $zero) for constants like one.
  - Specify constant in the instruction: this commonly used

- MIPS Instructions:
  \[ \text{addi} \; \$s3, \; \$t0, \; 4 \]
  \[ \text{slti} \; \$s0, \; \$t1, \; 10 \]

- Instruction Format
  6 bits 5 bits 5 bits
  \[ \text{I} \; \text{op} \; \text{rs} \; \text{rt} \; \text{16 bit address} \]
How about larger constants?

- We'd like to be able to load a 32 bit constant into a register.
- Must use two instructions, new "load upper immediate" instruction:
  
  ```
  lui $t0, 1010101010101010
  ```
  
  Then must get the lower order bits right, i.e.,
  
  ```
  ori $t0, $t0, 1010101010101010
  ```
Memory Access Instructions

- Load and store instructions
- Example:
  

  MIPS code:
  
  \[
  \begin{align*}
  &\text{lw} \; \$t0, \; 32(\$s3) \\
  &\text{add} \; \$t0, \; \$s2, \; \$t0 \\
  &\text{sw} \; \$t0, \; 32(\$s3) \\
  \end{align*}
  \]

- Store word has destination last
- Remember arithmetic operands are registers, not memory!
- Instruction format:

  \[
  \begin{array}{c|c|c|c}
  \text{op} & \text{rs} & \text{rt} & \text{16 bit address} \\
  \hline
  \text{I} & \text{6 bits} & \text{5 bits} & \text{5 bits} \\
  \end{array}
  \]
Control Transfer Instructions

- Decision making instructions
  - Alter the control flow,
  - i.e., change the "next" instruction to be executed

- MIPS conditional branch instructions:
  - `bne $t0, $t1, Label`
  - `beq $t0, $t1, Label`

- Example: if (i==j) h = i + j;
  - `bne $s0, $s1, Label`
  - `add $s3, $s0, $s1`
  - `Label: ....`

- Instruction format:
  - `I` 6 bits  5 bits  5 bits  16 bit address
Unconditional Jumps

- MIPS unconditional branch instructions:
  \[ j \ label \]

- Example:
  \[
  \begin{align*}
  \text{if (i!=j) & beq } & \text{ $s4, } s5, \text{ Lab1} \\
  \text{ h=i+j;} & \text{ add } & \text{ $s3, } s4, \text{ $s5} \\
  \text{ else} & \text{ j Lab2} \\
  \text{ h=i-j;} & \text{ Lab1:sub } & \text{ $s3, } s4, \text{ $s5} \\
  \text{ Lab2:...} & 
  \end{align*}
  \]

- Instruction format:

  \[
  \begin{array}{c|c|c}
  \hline
  \text{J} & \text{op} & \text{26 bit address} \\
  \hline
  \end{array}
  \]
MIPS Instruction Format & Machine code

- Instruction format

<table>
<thead>
<tr>
<th></th>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
</tr>
<tr>
<td>I</td>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>16 bit address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>op</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26 bit address</td>
</tr>
</tbody>
</table>

- The opcode tells the machine which format
- So Machine instruction `add r1, r2, r3` has
  - opcode=0, funct=32, rs=2, rt=3, rd=1, sa=0
  - Machine code:
    000000 00010 00011 00001 00000 100000
    0000 0000 0100 0011 0000 1000 0010 0000
    0x00430820

- Expected to assemble and disassemble machine code
So far:

- **Instruction** | **Meaning**

  - `add $s1,$s2,$s3`#$s1 = $s2 + $s3$
  - `sub $s1,$s2,$s3`#$s1 = $s2 - $s3$
  - `lw $s1,100($s2)`#$s1 = Memory[$s2+100]$
  - `sw $s1,100($s2)`#Memory[$s2+100] = $s1
  - `bne $s4,$s5,LNext`#instr. is at Label if
    - #$s4 != $s5
  - `beq $s4,$s5,LNext`#instr. is at Label if
    - #$s4 == $s5
  - `j Label`#instr. is at Label
Summary of MIPS Instructions

### MIPS operands

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 registers</td>
<td>$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at</td>
<td>Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.</td>
</tr>
<tr>
<td>2^30 memory words</td>
<td>Memory[0], Memory[4], .... Memory[4294967292]</td>
<td>Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.</td>
</tr>
</tbody>
</table>

### MIPS assembly language

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>add</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
<td>Three operands; data in registers</td>
</tr>
<tr>
<td></td>
<td>subtract</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
<td>Three operands; data in registers</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>addi $s1, $s2, 100</td>
<td>$s1 = $s2 + 100</td>
<td>Used to add constants</td>
</tr>
<tr>
<td>Data transfer</td>
<td>load word</td>
<td>lw $s1, 100($s2)</td>
<td>$s1 = Memory[$s2 + 100]</td>
<td>Word from memory to register</td>
</tr>
<tr>
<td></td>
<td>store word</td>
<td>sw $s1, 100($s2)</td>
<td>Memory[$s2 + 100] = $s1</td>
<td>Word from register to memory</td>
</tr>
<tr>
<td></td>
<td>load byte</td>
<td>lb $s1, 100($s2)</td>
<td>$s1 = Memory[$s2 + 100]</td>
<td>Byte from memory to register</td>
</tr>
<tr>
<td></td>
<td>store byte</td>
<td>sb $s1, 100($s2)</td>
<td>Memory[$s2 + 100] = $s1</td>
<td>Byte from register to memory</td>
</tr>
<tr>
<td></td>
<td>load upper immediate</td>
<td>lui $s1, 100</td>
<td>$s1 = 100 * 2^16</td>
<td>Loads constant in upper 16 bits</td>
</tr>
<tr>
<td>Conditional branch</td>
<td>branch on equal</td>
<td>beq $s1, $s2, 25</td>
<td>if ($s1 == $s2) go to PC + 4 + 100</td>
<td>Equal test; PC-relative branch</td>
</tr>
<tr>
<td></td>
<td>branch on not equal</td>
<td>bne $s1, $s2, 25</td>
<td>if ($s1 != $s2) go to PC + 4 + 100</td>
<td>Not equal test; PC-relative branch</td>
</tr>
<tr>
<td></td>
<td>set on less than</td>
<td>slt $s1, $s2, $s3</td>
<td>if ($s2 &lt; $s3) $s1 = 1; else $s1 = 0</td>
<td>Compare less than; for beq, bne</td>
</tr>
<tr>
<td></td>
<td>set less than immediate</td>
<td>slti $s1, $s2, 100</td>
<td>if ($s2 &lt; 100) $s1 = 1; else $s1 = 0</td>
<td>Compare less than constant</td>
</tr>
<tr>
<td>Unconditional jump</td>
<td>jump</td>
<td>j 2500</td>
<td>go to 10000</td>
<td>Jump to target address</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>jr $ra</td>
<td>go to $ra</td>
<td>For switch, procedure return</td>
</tr>
<tr>
<td></td>
<td>jump and link</td>
<td>jal 2500</td>
<td>$ra = PC + 4; go to 10000</td>
<td>For procedure call</td>
</tr>
</tbody>
</table>
MIPS Addressing Modes

1. Immediate addressing  
   \textbf{e.g.} \texttt{addi} $s0, s1, 4

2. Register addressing  
   \textbf{e.g.} \texttt{sub} $s0, s1, s2

3. Base addressing  
   \textbf{e.g.} \texttt{lw} $s0, 4( s2)

4. PC-relative addressing  
   \textbf{e.g.} \texttt{beq} $s1, s2, 32

5. Pseudodirect addressing  
   \textbf{e.g.} \texttt{j} 0x1000

\begin{tabular}{|c|c|c|}
\hline
\textbf{op} & \textbf{rs} & \textbf{rt} \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{op} & \textbf{rs} & \textbf{rt} & \textbf{rd} \\
\hline
\end{tabular}

\begin{tabular}{|c|}
\hline
\textbf{op} \\
\hline
\end{tabular}

\begin{tabular}{|c|}
\hline
\textbf{op} \\
\hline
\end{tabular}
Assembly Language vs. Machine Language

- Assembly provides convenient symbolic representation
  - Much easier than writing down numbers
  - e.g., destination first
- Machine language is the underlying reality
  - e.g., destination is no longer first
- Assembly can provide 'pseudo-instructions'
  - e.g., “move $t0, $t1” exists only in Assembly
  - would be implemented using “add $t0,$t1,$zero”
- When considering performance you should count real instructions
Other Issues

- Things we are not going to cover
  - support for procedures
  - linkers, loaders, memory layout
  - stacks, frames, recursion
  - manipulating strings and pointers
  - interrupts and exceptions
  - system calls and conventions

- We've focused on architectural issues
  - basics of MIPS assembly language and machine code
  - we’ll build a processor to execute these instructions.
Alternative Architectures

- Design alternative:
  - Provide more powerful operations
  - Goal is to reduce number of instructions executed
  - Danger is a slower cycle time and/or a higher CPI

- Sometimes referred to as “RISC vs. CISC”
  - Virtually all new instruction sets since 1982 have been RISC
  - VAX: minimize code size, make assembly language easy

- We’ll look at 80x86
80x86

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions
  (mostly designed for higher performance)
- 1997: MMX is added

“This history illustrates the impact of the “golden handcuffs” of compatibility”
“an architecture that is difficult to explain and impossible to love”
A dominant architecture: 80x86

- See your textbook for a more detailed description
- Complexity:
  - Instructions from 1 to 17 bytes long
  - One operand must act as both a source and destination
  - One operand can come from memory
  - Complex addressing modes
    e.g., “base or scaled index with 8 or 32 bit displacement”
- Saving grace:
  - The most frequently used instructions are not too difficult to build
  - Compilers avoid the portions of the architecture that are slow

“What the 80x86 lacks in style is made up in quantity, making it beautiful from the right perspective”
Summary

- Instruction complexity is only one variable
  - Lower instruction count vs. higher CPI / lower clock rate
- Design Principles:
  - Simplicity favors regularity
  - Smaller is faster
  - Good design demands compromise
  - Make the common case fast
- Instruction set architecture
  - A very important abstraction indeed!
- In next lectures we will show how to implement a subset of this ISA
Announcements

- Reading Assignment:
  - Performance, Chapter 2
  - Arithmetic, Chapter 4

- Homework
  3.1, 3.2, 3.3, 3.7, 3.9, 3.14, 3.17.
  Due Wed., June 30 in class

- Quiz
  When: Wed., June 30, First 10 minutes of the class
  Topic: ISA, Chapter 3
  Need: Paper, pen, calculator

- CSE 141 Lab Assignment has been posted on the web page
  [http://www.cse.ucsd.edu/classes/su04/cse141L](http://www.cse.ucsd.edu/classes/su04/cse141L)