Announcements

- **Discussions Sections for 141:**
  - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
  - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)

- **Reading Assignment**
  - Chapter 2. Instructions: Language of the Computer
    - Sec. 2.1 - 2.9, 2-16

- **Homework 2:** Due Mon., April 11 in class
  - 2.2, 2.4, 2.6, 2.29, 2.30, 2.32, 2.47

- **Quiz**
  - When: Mon., April 11th, First 10 minutes of the class
  - Topic: ISA, Chapter 2
  - Need: Paper, pen

---

Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mon.  3/28</td>
<td>Introduction, Ch. 1</td>
<td>Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Wed.  3/30</td>
<td>Performance, Ch. 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Mon.  4/4</td>
<td>ISA, Ch. 2</td>
<td>Performance</td>
<td></td>
<td>ISA #2</td>
</tr>
<tr>
<td>4</td>
<td>Wed.  4/6</td>
<td>Arithmetic, Ch. 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Mon.  4/11</td>
<td>Arithmetic, Ch. 3</td>
<td>ISA #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Wed.  4/13</td>
<td>Single cycle CPU, Ch. 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Mon.  4/18</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Arithmetic #3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Wed.  4/20</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Mon.  4/25</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU #4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Wed.  4/27</td>
<td>Review for the Midterm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Mon.  5/2</td>
<td>Mid-term Exam</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Wed.  5/4</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Mon.  5/9</td>
<td>Pipelining, Ch. 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Wed.  5/11</td>
<td>Data and control hazards, Ch. 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Mon.  5/16</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards #5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Wed.  5/18</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Mon.  5/23</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache #6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Wed.  5/25</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mon.  5/30</td>
<td>No Class</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Wed.  6/1</td>
<td>Course Review</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Mon.  6/6</td>
<td>Final Exam</td>
<td></td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

---

Notes:

- Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
- Office Hour:
  - Mon. 5:00 - 6:00 PM (AP&M 2444)
- TAs:
  - Bars Arslan: baraslan@cs.ucsd.edu
  - Raid Ayoub: rayoub@cs.ucsd.edu
  - Leo Porter: lporter@cs.ucsd.edu
- Textbook: Computer Organization & Design
  - Authors: Patterson and Hennessy
- Web-page: http://www.cse.ucsd.edu/classes/sp05/cse141
Instruction Set Architecture (ISA)
General Considerations

- 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

Stored Program Concept

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.

Memory Organization
- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.
  - 0-3 bytes of data
  - 4-7 bytes of data
  - 8-11 bytes of data
  - 12-15 bytes of data
  - ...
- Words are aligned on 4-byte boundary
  - i.e., what are the least 2 significant bits of a word address?
- 32 bits address
  - $2^{32}$ bytes with byte addresses from 0 to $2^{32}-1$
  - $2^{30}$ words with byte addresses 0, 4, 8, ..., $2^{30}$.4
### Endian-ness: How to address bytes within words?

- **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)

### 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

1. **Instruction Fetch**
   - Obtain instruction from program storage
2. **Instruction Decode**
   - Determine required actions and instruction size
3. **Operand Fetch**
   - Locate and obtain operand data
4. **Execute**
   - Compute result value or status
5. **Result Store**
   - Deposit results in storage for later use
6. **Next Instruction**
   - Determine successor instruction

### The Instruction Set Architecture

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

- **Micro-code**
- **B0 system**
- **Digital Design**
- **Circuit Design**

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

Key ISA Decisions

- Instruction length
  - Fixed length
  - Variable length
- Registers
  - How many?
- Operand access
  - Register
  - Memory
- Instruction format
  - Meaning of group of bits within machine instruction
- Operands
  - How many per instruction, size (byte, word,..)
- Operations
  - ADD, SUB, MUL, ...

Accessing the Operands

- Operands are generally in one of two places:
  - Registers: fast on-chip storage (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: |
|------------------------|-----------------|-----------------|-----------------|
| Stack | Accumulator | Register | Register |
| Push A | Load A | Load R1,A | Load R1,A |
| Push B | Add B | Add R1,B | Load R2,B |
| Add | Store C | Store C, R1 | Add R3,R1,R2 |
| Pop C | | | Store C,R3 |
MIPS Instruction Set Architecture

- Typical “RISC” Instruction Set designed in 1980’s
- MIPS is found in products from:
  - Silicon Graphics
  - NEC
  - Cisco
  - Broadcom
  - Nintendo
  - Sony
  - Ti
  - Toshiba
- We will study various implementations of MIPS instruction set
  - Later part of the course!

MIPS ISA: 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 good compromises: e.g. fixed length instructions
  - Make the common case fast: encode constant within the instruction

Overview of MIPS ISA

- Fixed 32-bit instructions
- 3-operand, load-store architecture
- 32 general-purpose registers
  - Registers are 32-bits wide (word)
  - R0 always equals zero.
- Addressing modes
  - Register, immediate, base+displacement, PC-relative and pseudo-direct addressing modes
- 3 instruction formats will be covered in the class
Notes on MIPS Assembly

- Comments start with “#”
- Destination first (except for “store” instructions)
  - ADD $t0, $s1, $s2  # $t0 = $s1 + $s2
- Register access $n: register n (e.g. $1 is register 1)
- Register naming convention

<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 Instructions (Covered in this course)

- Arithmetic
  - add, sub, addi
- Logical
  - sll, srl, and, andi, or, ori, nor
- Data transfer
  - lw, sw, lb, sh, lui
- Conditional branch
  - beq, bne, slt, slti
- Unconditional jump
  - j, jr
- Jump and link
  - jal

MIPS Arithmetic Instructions

- 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
- Instruction format:
  - \( \text{op} \) \( rs \) \( rt \) \( rd \) \( shamt \) \( funct \)

  - 6 bits  5 bits  5 bits  5 bits  6 bits

  - Design Principle: simplicity favors regularity. Why?
  - Design Principle: smaller is faster. Why?

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
  - 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:
  - `add $s3, $t0, 4`
  - `slti $s0, $t1, 10`
- Instruction Format
  - \( \text{op} \) \( rs \) \( rt \) \( 16 \text{ bit address} \)
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
  \[ \text{lui $t0, 1010101010101010} \]
- Then must get the lower order bits right, i.e.,
  \[ \text{ori $t0, $t0, 1010101010101010} \]

Memory Access Instructions

- Load and store instructions
- Example:
  \[
  \begin{align*}
  \text{C code:} & \quad A[8] = h + A[8]; \\
  \text{MIPS code:} & \quad \text{lw} \quad $t0, 32($s3) \\
  & \quad \text{add} \quad $t0, \quad $s2, \quad $t0 \\
  & \quad \text{sw} \quad $t0, \quad 32($s3)
  \end{align*}
  \]
- Store word has destination last
- Remember arithmetic operands are registers, not memory!
- Instruction format:
  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{16 bit address} \\
  \end{array}
  \]

Control Transfer Instructions

- Decision making instructions
  - Alter the control flow,
  - i.e., change the "next" instruction to be executed
- MIPS conditional branch instructions:
  \[
  \begin{align*}
  \text{bne} \quad $t0, \quad $t1, \quad \text{Label} \\
  \text{beq} \quad $t0, \quad $t1, \quad \text{Label}
  \end{align*}
  \]
- Example: if (i==j) h = i + j;
  \[
  \begin{align*}
  \text{bne} \quad $s0, \quad $s1, \quad \text{Label} \\
  \text{add} \quad $s3, \quad $s0, \quad $s1 \\
  \text{Label:} \quad \ldots
  \end{align*}
  \]
- Instruction format:
  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{16 bit address} \\
  \end{array}
  \]

MIPS Conditional Branches

- MIPS branches use PC-relative addressing
- BEQ, BNE
  \[
  \begin{align*}
  \text{BEQ} \quad $1, \quad $2, \quad \text{addr} \Rightarrow \text{if} (r1 == r2) \text{goto addr} \\
  \text{MIPS has no Branch-If-Less-Than} \\
  \text{Set-Less-Than, SLT} \Rightarrow \text{if} (r2 < r3) r1 = 1; \text{else} r1 = 0 \\
  \text{BEQ, BNE, SLT combined with S0 can implement all branch conditions:} \\
  \text{always, never, } \langle, \rangle, \langle<, \rangle>, \langle\text{unsigned}, \rangle\langle\text{unsigned}, \rangle, \ldots
  \end{align*}
  \]
MIPS Jump Instructions

- Need to transfer control
  - Jump to an absolute address
  - Jump to an address in a register
  - Jump-And-Link to do procedure call and return
- Jump example:
  - \( j \ 0 \times 20000 \) \( \Rightarrow \) PC = 0x20000
- Jump and Link example
  - jal 0x40000 \( \Rightarrow \) $31 = PC+4, PC = 0x40000
- Jump register example:
  - jr $31 \( \Rightarrow \) PC = $31 (This is return instruction!)

Unconditional Jumps

- MIPS unconditional branch instructions:
  \( j \ label \)
- Instruction format:
  \[
  \begin{array}{cccccc}
  \text{op} & 26 & \text{bit address} \\
  \hline
  \text{J} & 26 & \text{bit address} \\
  \end{array}
  \]
- Jump uses pseudo-direct addressing mode

MIPS Instruction Format & Machine code

- Instruction format
  \[
  \begin{array}{cccccc}
  \text{R} & 6 & \text{bits} & 5 & \text{bits} & 5 & \text{bits} & 5 & \text{bits} & 6 & \text{bits} \\
  \hline
  \text{I} & \text{op} & 26 & \text{bit address} \\
  \text{J} & \text{op} & 26 & \text{bit address} \\
  \end{array}
  \]
- The opcode tells the machine which format
- Machine instruction add \( r1, r2, r3 \) has
  - opcode=0, funct=32, rs=2, rt=3, rd=1, sa=0
  - Machine code:
    \[
    \begin{array}{cccccccc}
    000000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 \\
    000000 & 01000 & 00001 & 00000 & 10000 & 00000 & 00000 & 00000 \\
    000000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 \\
    000000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 & 00000 \\
    \end{array}
    \]
- Expected to assemble and disassemble machine code

Summary of MIPS Instructions

- Summary of MIPS instructions:
<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</td>
<td>Loads constant in upper 16 bits</td>
</tr>
<tr>
<td>Conditional</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>branch set on less than</td>
<td>slt  $s1, $s2, $s3</td>
<td>if ($s2 &lt; $s3) ( \text{set} ) $s1 = 1; else $s1 = 0</td>
<td>Compare less than; for beq, bne</td>
</tr>
<tr>
<td></td>
<td>branch set less than immediate</td>
<td>slti  $s1, $s2, 100</td>
<td>if ($s2 &lt; 100) ( \text{set} ) $s1 = 1; else $s1 = 0</td>
<td>Compare less than constant</td>
</tr>
<tr>
<td>Unconditional</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 $ra</td>
<td>( \text{ra} = \text{PC} + 4; \text{go to 10000} )</td>
<td>For procedure call</td>
</tr>
</tbody>
</table>
### MIPS Addressing Modes

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>e.g. <code>addi $t0, $t1, 4</code></td>
</tr>
<tr>
<td>Register</td>
<td>e.g. <code>sub  $t0, $t1, $t2</code></td>
</tr>
<tr>
<td>Base</td>
<td>e.g. <code>lw $t0, 4( $t2)</code></td>
</tr>
<tr>
<td>PC-relative</td>
<td>e.g. <code>beq $t1, $t2, 32</code></td>
</tr>
<tr>
<td>Pseudodirect</td>
<td>e.g. <code>j 0x1000</code></td>
</tr>
</tbody>
</table>

Note: Parts 3 and 4 in Figure 2.24 (page 101) Edition 3 are incorrect!

### 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
  - Used by the processor at run time
- 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
    - instructions from 1 to 54 bytes long!
Intel IA-32 Architecture

- 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
- 1999 Pentium II (same architecture)
- 2000 Pentium 4 (144 new multimedia instructions)
- 2001 Itanium (new ISA which executes x86 code)

A dominant architecture: 80x86

- See Section 2-16 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

Summary

- Instruction complexity is only one variable
  - Lower instruction count vs. higher CPI / lower clock rate
- Four principles of ISA
  - Simplicity favors regularity: regular instruction set
  - Smaller is faster: small number of formats, small number of registers
  - Good design demands good compromises: fixed length instructions
  - Make the common case fast: encode constant within the instruction
- Instruction set architecture
  - A very important abstraction indeed!
- In subsequent lectures we will show how to implement a subset of this ISA

Announcements

- Discussions Sections for 141:
  - Fridays, 10:00 - 10:50 am, Peterson Hall 104 (Chris)
  - Fridays, 2:00 - 2:50 pm, Peterson Hall 104 (Leo)
- Reading Assignment
  - Chapter 2. Instructions: Language of the Computer
    Sec. 2.1 - 2.9, 2-16
- Homework 2: Due Mon., April 11 in class
  2.2, 2.4, 2.6, 2.29, 2.30, 2.32, 2.47
- Quiz
  When: Mon., April 11th, First 10 minutes of the class
  Topic: ISA, Chapter 2
  Need: Paper, pen