Chapter Goals

- Teach a subset of MIPS assembly language
- Introduce the stored program concept
- Explain how MIPS instructions are represented in machine language
- Illustrate basic instruction set design principles

Instructions:

- Language of the Machine
- More primitive than higher level languages
- Very restrictive
e.g., MIPS Arithmetic Instructions
- We'll be working with the MIPS instruction set architecture
  - similar to other architectures developed since the 1980's
  - used by NEC, Nintendo, Silicon Graphics, Sony

Design principles: to be found...

Design goals:
MIPS arithmetic

- All instructions have 3 operands
- Operand order is fixed

Example:

C code: \[ A = B + C \]
MIPS code: \[ \text{add } $s0, $s1, $s2 \]

MIPS arithmetic

- Design Principle #1: simplicity favors regularity. Why?

- Of course this complicates some things...

C code: \[ A = B + C + D; \]
\[ E = F - A; \]
MIPS code: \[ \text{add } $t0, $s1, $s2 \]
\[ \text{add } $s0, $t0, $s3 \]
\[ \text{sub } $s4, $s5, $s0 \]

Registers vs. Memory

- Design Principle #2: smaller is faster. Why?

- Therefore, arithmetic instruction operands must be “registers”
  - And only 32 registers provided

- Compiler associates variables with registers
- What about programs with lots of variables?

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  32 bits of data
  4  32 bits of data
  8  32 bits of data
  12 32 bits of data

  ...  
- $2^{32}$ 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?

So far we’ve learned:

- MIPS
  - loading words but addressing bytes
  - arithmetic on registers only

  Instruction | Meaning
  --- | ---
  add $s1$, $s2$, $s3$ &nbsp; &nbsp; &nbsp; $s1 = s2 + s3$
  sub $s1$, $s2$, $s3$ &nbsp; &nbsp; &nbsp; $s1 = s2 - s3$
  lw $s1$, 100($s2) &nbsp; &nbsp; &nbsp; $s1 = \text{Memory}[$s2+100$]
  sw $s1$, 100($s2) &nbsp; &nbsp; &nbsp; \text{Memory}[$s2+100$] &nbsp; &nbsp; &nbsp; $s1$

Memory Instructions

- Load and store instructions
- Example:

  MIPS code: &nbsp; lw $t0, 32($s3)
             &nbsp; add $t0, $s2, $t0
             &nbsp; sw $t0, 32($s3)

  For lw/sw, address always = register value + offset
  How about this?
  add $t0, 32($s3), $t0

Machine Language

- Instructions, like registers and words of data, are also 32 bits long
  - Example: add $t0$, $s1$, $s2$
  - registers have numbers, $t0=8$, $s1=17$, $s2=18$

  Instruction Format (r-type):
  000000 10001 10010 01000 00000 100000
  op | rs | rt | rd | shamt | funct
Consider the load-word and store-word instructions,
- What would the regularity principle have us do?
- Principle #3: Make the common case fast
- Principle #4: Good design demands a compromise

Introduce a new type of instruction format
- I-type for data transfer instructions
- Example: `lw $t0, 44($s2)`

<table>
<thead>
<tr>
<th>35</th>
<th>18</th>
<th>8</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>16 bit number</td>
</tr>
</tbody>
</table>

Where's the compromise?

---

**Example Part 1**

- What is the machine code for the following:
  \[ A[300] = h + A[300]; \]
  - Variable h is assigned register $s2
  - Array A base address is assigned register $t1
  - Do the assembly code first, then machine language instructions, and then machine code

**Example Part 2**

- What is the machine code for the following: \[ A[300] = h + A[300]; \]
  - Variable h is assigned register $s2 & Array A base address is assigned register $t1

First part of answer:

| lw $t0, 1200($t1) | # Temporary reg $t0 gets A[300] |
| add $t0, $s2, $t0 | # Temporary reg $t0 gets h + A[300] |
| sw $t0, 1200($t1) | # Stores h + A[300] back into A[300] |

Second part of answer (DECIMAL):

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>9</td>
<td>8</td>
<td>1200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>18</td>
<td>8</td>
<td>0</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>9</td>
<td>8</td>
<td>1200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second part of answer (BINARY):

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01001</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
<td>1011 0000</td>
</tr>
<tr>
<td>000000</td>
<td>10010</td>
<td>01000</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
</tr>
<tr>
<td>101011</td>
<td>01001</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
<td>1011 0000</td>
</tr>
</tbody>
</table>

**Example Part 3**

- What is the machine code for the following: \[ A[300] = h + A[300]; \]
  - Variable h is assigned register $s2 & Array A base address is assigned register $t1

First part of answer:

| lw $t0, 1200($t1) | # Temporary reg $t0 gets A[300] |
| add $t0, $s2, $t0 | # Temporary reg $t0 gets h + A[300] |
| sw $t0, 1200($t1) | # Stores h + A[300] back into A[300] |

Second part of answer (BINARY):

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01001</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
<td>1011 0000</td>
</tr>
<tr>
<td>000000</td>
<td>10010</td>
<td>01000</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
</tr>
<tr>
<td>101011</td>
<td>01001</td>
<td>01000</td>
<td>0000</td>
<td>01000</td>
<td>1011 0000</td>
</tr>
</tbody>
</table>
Stored Program Concept

- Instructions are composed of bits / bytes / words
- Programs are stored in memory — to be read or written just like data

Quick Review

- Design Principles
  - 4 of them
- Arithmetic
  - Operands
  - Order
  - Location of data
- Register
  - MIPS provides
- Memory
  - Organization
  - Bits / Bytes / Words
  - Alignment