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

- MIPS arithmetic
  - All instructions have 3 operands
  - Operand order is fixed

Example:

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

Design principles: to be found...
Design goals:
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:
  ```
  add $t0, $s1, $s2
  add $s0, $t0, $s3
  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

  ```
  Processor
  Control
  Datapath
  Memory
  Input
  Output
  ```

- 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.

0 1 2 3 4 5 6 ...

| 0 bits of data |
| 4 bits of data |
| 8 bits of data |
| 12 bits of data |
| 16 bits of data |
| 20 bits of data |
| 24 bits of data |
| 28 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.

```
    0           8       12
Registers hold 32 bits of data
32 bits of data
32 bits of data
32 bits of data
32 bits of data
32 bits of data
32 bits of data
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, ... $2^{30} - 4$
- Words are aligned
  - i.e., what are the least 2 significant bits of a word address?
Array layout in memory

Memory Instructions

- Load and store instructions
- Example:
  
  
  MIPS code:  
  
  ```
  lw $t0, 32($s3)
  add $t0, $s2, $t0
  sw $t0, 32($s3)
  ```

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

So far we’ve learned:

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

- Instruction | Meaning
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
</tr>
<tr>
<td>lw $s1, 100($s2)</td>
<td>$s1 = Memory[$s2+100]</td>
</tr>
<tr>
<td>sw $s1, 100($s2)</td>
<td>Memory[$s2+100] = $s1</td>
</tr>
</tbody>
</table>

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):
  
<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>000000</td>
<td>10001</td>
<td>10010</td>
<td>01000</td>
<td>00000</td>
<td>100000</td>
</tr>
</tbody>
</table>

Ex 2-1to 2-3
Machine Language

• Consider the load-word and store-word instructions,
  – What would the regularity principle have us do?
  – BIG IDEA: Make the common case fast
  – Principle #3: Good design demands a compromise

• Introduce a new type of instruction format
  – I-type for data transfer instructions
  – Example: \texttt{lw \$t0, 44(s2)}

  \begin{center}
  \begin{tabular}{|c|c|c|c|}
  \hline
  35  & 18 & 8 & 44 \\
  \hline
  \end{tabular}
  \end{center}

  op  rs rt 16 bit number

• 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:

  \begin{align*}
  \text{lw} & \$t0, 1200(\$t1) & \# \text{Temporary reg } \$t0 \text{ gets } A[300] \\
  \text{add} & \$t0, \$s2, \$t0 & \# \text{Temporary reg } \$t0 \text{ gets } h + A[300] \\
  \text{sw} & \$t0, 1200(\$t1) & \# \text{Stores } h + A[300] \text{ back into } A[300]
  \end{align*}

• Second part of answer (DECIMAL):

  \begin{center}
  \begin{tabular}{|c|c|c|c|c|c|}
  \hline
  op & rs & rt & rd & shamt & funct \\
  \hline
  35 & 9 & 8 & 1200 & & \\
  0 & 18 & 8 & 8 & 0 & 32 \\
  43 & 9 & 8 & 1200 & & \\
  \hline
  \end{tabular}
  \end{center}

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:

  \begin{align*}
  \text{lw} & \$t0, 1200(\$t1) & \# \text{Temporary reg } \$t0 \text{ gets } A[300] \\
  \text{add} & \$t0, \$s2, \$t0 & \# \text{Temporary reg } \$t0 \text{ gets } h + A[300] \\
  \text{sw} & \$t0, 1200(\$t1) & \# \text{Stores } h + A[300] \text{ back into } A[300]
  \end{align*}

• Second part of answer (BINARY):

  \begin{center}
  \begin{tabular}{|c|c|c|c|c|c|}
  \hline
  op & rs & rt & rd & shamt & funct \\
  \hline
  100011 & 01001 & 01000 & 0000 & 0100 & 1011 & 0000 \\
  000000 & 10010 & 01000 & 01000 & 0000 & 0000 & 10000 \\
  101011 & 01001 & 01000 & 0000 & 0100 & 1011 & 0000 \\
  \hline
  \end{tabular}
  \end{center}
Stored Program Computers

- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs – e.g., compilers, linkers, ...
- Binary compatibility allows compiled programs to work on different computers – Standardized ISAs
- 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

QUICK REVIEW

- Design Principles – 3 of them
- Arithmetic – Operands – Order – Location of data
- Register – MIPS provides
- Memory – Organization – Bits / Bytes / Words – Alignment