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 Tivo, Cisco routers, Nintendo 64, Sony PlayStation...
  - on “New Horizons” spacecraft that reached Pluto in July 2015!

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:
```c
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
  - 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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits of data</td>
<td>8 bits of data</td>
<td>8 bits of data</td>
<td>8 bits of data</td>
<td>8 bits of data</td>
<td>8 bits of data</td>
<td>8 bits of data</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.
- Valid byte addresses: 0, 1, 2, 3, 4, ..., $2^{32} - 1$
- Valid word addresses 0, 4, 8, ..., $2^{32}$
- Words are aligned

Registers hold 32 bits of data
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
  |------------------|
  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

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):
  ```
  | op | rs | rt | rd | shamt | funct |
  -------------------------------
  | 000000 | 10001 | 10010 | 01000 | 00000 | 100000 |
  ```
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)}

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

- Second part of answer (DECIMAL):

  \[
  \begin{array}{cccccc}
  \text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{funct} \\
  35 & 18 & 8 & 44 & \text{16 bit number} \\
  0 & 18 & 8 & 8 & 0 & 32 \\
  43 & 9 & 8 & 1200 & \\
  \end{array}
  \]

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

- Second part of answer (BINARY):

  \[
  \begin{array}{cccccc}
  \text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{funct} \\
  100011 & 01001 & 01000 & 0000 0100 1011 0000 \\
  000000 & 10010 & 01000 & 01000 & 0000 & 100000 \\
  101011 & 01001 & 01000 & 0000 0100 1011 0000 \\
  \end{array}
  \]
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