Representing instructions in the computer – Machine Language – Review

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

- Instruction Format:

\[
\begin{array}{cccccccc}
000000 & rs & rt & rd & shamt & funct \\
000000 & 10001 & 10010 & 01000 & 00000 & 100000
\end{array}
\]

Translating MIPS Assembly Language into Machine Language – Review

Example:

\[
g = h + A[100]
\]

Complied into

\[
lw \ \$t0, 400(\$t1) \\
add \ \$t0, \$s2, \$t0
\]

What is the MIPS machine language code?

\[
\begin{array}{ccccccc}
\text{op} & rs & rt & rd & shamt & funct \\
35 & 9 & 8 & 400 (16 bit) & 0 & 18 & 8 & 8 & 32
\end{array}
\]
### MIPS Instructions

#### conditional branching
- Decision making instructions
- alter the control flow, i.e., change the "next" instruction to be executed

```
beq $s0, $s1, label
   # if ($s0 == $s1) then go to label
bne $s0, $s1, label
   # if ($s0 != $s1) then go to label
```
- Both conditionals have 3 operands
  - 2 source registers for comparison
  - 1 destination label, which is a signed word offset from PC
- Note corresponding machine language below for "beq" example

#### condition setting
- Set less than operation has 3 operands
  - 2 source
  - 1 destination
- This is useful for setting a pre-condition for loop termination
- Note corresponding machine language below for "slt" example

#### jump via register
- Only 1 register operand
  - rs specifies an index into a jump table
  - Other operand fields are null

- Note the corresponding machine language for "jr" example

#### jump (unconditional)
- No register operands!
- Operand destination address is signed word offset from PC
- Note the corresponding machine language for "j" example

### MIPS Instructions

#### conditional branching

```
beq $s0, $s1, label
   # if ($s0 == $s1) then go to label
bne $s0, $s1, label
   # if ($s0 != $s1) then go to label
```

- Both conditionals have 3 operands
  - 2 source registers for comparison
  - 1 destination label, which is a signed word offset from PC

#### condition setting
- Set less than operation has 3 operands
  - 2 source
  - 1 destination
- This is useful for setting a pre-condition for loop termination

#### jump via register
- Only 1 register operand
  - rs specifies an index into a jump table
  - Other operand fields are null

#### jump (unconditional)
- No register operands!
Summary

MIPS

- simple instructions all 32 bits wide
- much can be done with instructions we have seen
- only three instruction formats

<table>
<thead>
<tr>
<th>R</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
</table>

- See summary on page 78

Summary

Registers & Memory

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>MIPS operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 registers</td>
<td>$a0-$a3, $v0-$v1, $zero</td>
<td>registers fit into 8-bit or 16-bit 32 registers</td>
<td>MIPS register space in Table 3.11</td>
</tr>
<tr>
<td>32-bit memory</td>
<td>Memory[0], Memory[1], ...</td>
<td>Accessed only by load/store instructions.</td>
<td>Memory holds data structures, such as arrays.</td>
</tr>
<tr>
<td>30-bit memory</td>
<td>Memory[0x0], Memory[0x1], ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

see table 2.12 page 77

Logical Operations

<table>
<thead>
<tr>
<th>Logical operation</th>
<th>MIPS instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift Left</td>
<td>sll</td>
</tr>
<tr>
<td>Shift right</td>
<td>srl</td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td>and</td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td>or</td>
</tr>
<tr>
<td>Bit-by-bit NOT*</td>
<td>nor</td>
</tr>
</tbody>
</table>

Logical Operations – Shift Example

- Suppose $s0$ contained
  0000 0000 0000 0000 0000 0000 0000 1001 = 9
  sll $t2$, $s0$, 4 #reg $s0$ << 4 bits
  $t0$ contains?
  0000 0000 0000 0000 1001 0000 0000 0000 = 144

- Shift operation is also equivalent to multiplication.
Logical Operations – Logic AND Example

• Suppose
  \$t2 \text{ contains } 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 1101 \ 0000 \ 0000
  \$t1 \text{ contains } 0000 \ 0000 \ 0000 \ 0000 \ 0011 \ 1100 \ 0000 \ 0000
  and \$t0, \$t1, \$t2 \ # \text{reg } \$t0 = \text{reg } \$t1 \& \text{reg } \$t2

• What is the content of \$t0 after the execution of the MIPS instruction?

Logical Operations – Logic NOT Example

• The designers of MIPS decided to include the instruction NOR instead of NOT. Show that if one operand is zero, then it is equivalent to NOT.
  nor \$t0, \$t1, \$t3  \ # \text{reg } \$t0 = \sim (\text{reg } \$t1 \lor \text{reg } \$t3)

  A \text{ NOR } 0 = \text{ NOT } (A \text{ OR } 0) = \text{ NOT } (A)

  \text{A + 0 = A . 0 = A (DeMorgan’s)}

Special Registers

\textit{pc} – program counter

• “pc” holds the address of the \textit{next instruction} to be executed.
• Not to be confused with personal computer-PC.

Special Registers

\textit{$sp$ – stack pointer ($29$)}

In \textit{MIPS} machines, part of main memory is reserved for a \textit{stack}.

\textbf{Stack:} A data structure for spilling registers.

The address of the top element of the stack is stored (by convention) in the \textit{"stack pointer"} register, \$sp. The stack grows downward in terms of memory addresses.

\textbf{MIPS} does not provide “push” and “pop” instructions. Instead, they must be done explicitly by the programmer.
## Implementing the equivalent of push $r3 on
## the stack in MIPS

```plaintext
addi $sp, $sp, -4     # Decrement stack pointer by 4
sw $r3, 0($sp)        # Save $r3 to stack
```

## Implementing the equivalent of popping
## $r3 off the stack in MIPS

```plaintext
lw  $r3, 0($sp)         # Copy from stack to $r3
addi $sp, $sp, 4       # Increment stack pointer by 4
```

## MIPS Instructions

### immediate operands

- **lui $t0, 255**     # Load upper half of $t0 with 255
- **addi $s1, $s2, 100** # Add 100 to $s1
- **slti $s1, $s2, 10**  # If $s2 < 10 then $s1 = 1 else $s1 = 0

- Most immediate instructions have 3 operands
  - 1 source 16-bit signed constant & 1 source register
  - 1 destination register
- Immediate instructions pack small constants into instruction
- Note corresponding machine language below for “lui” example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Immediate Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lui $t0, 255</td>
<td>255</td>
<td>Load upper half of $t0 with 255</td>
</tr>
<tr>
<td>addi $s1, $s2, 100</td>
<td>100</td>
<td>Add 100 to $s1</td>
</tr>
<tr>
<td>slti $s1, $s2, 10</td>
<td>10</td>
<td>If $s2 &lt; 10 then $s1 = 1 else $s1 = 0</td>
</tr>
</tbody>
</table>

## Review Instructions

### Table 2.12 page 77

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>lui $t0, 255</td>
<td>255</td>
<td>Load upper half of $t0 with 255</td>
<td></td>
</tr>
<tr>
<td>addi $s1, $s2, 100</td>
<td>100</td>
<td>Add 100 to $s1</td>
<td></td>
</tr>
<tr>
<td>slti $s1, $s2, 10</td>
<td>10</td>
<td>If $s2 &lt; 10 then $s1 = 1 else $s1 = 0</td>
<td></td>
</tr>
</tbody>
</table>

- 1 source 16-bit signed constant & 1 source register
- 1 destination register

### Example

<table>
<thead>
<tr>
<th>sp</th>
<th>rs</th>
<th>rt</th>
<th>Immediate</th>
<th>16 bit address offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 1 1</td>
<td>11 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

CNU Fall 2006  CPSC330 CompOrg: Dr. Gerousis  Lec2b - Instructions 17

CNU Fall 2006  CPSC330 CompOrg: Dr. Gerousis  Lec2b - Instructions 18

CNU Fall 2006  CPSC330 CompOrg: Dr. Gerousis  Lec2b - Instructions 19

CNU Fall 2006  CPSC330 CompOrg: Dr. Gerousis  Lec2b - Instructions 20
Next class...

- Finishing up instructions
- System calls
- SPIM simulator and examples

Chapter 2 HW due 09/20

- Exercises 2.1, 2.2, 2.3, 2.26, 2.27, 2.34
  + Using SPIM, write and debug a program to input two numbers, then it computes and prints their sum, their difference, their product, and their quotient. Use the SPIM system calls read_int (5, $a0), print_int (1, $v0). Only the MIPS code should be submitted to WebCT.
- Exam 1 - 09/27