lecture 9

MIPS assembly language 2

- register names
- pseudo instructions
- immediate versions
- signed and unsigned
- shifting

February 8, 2016
### R (Register-Transfer)

<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>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

E.g.: add, sub

### I (Immediate)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

E.g.: lw, sw, beq

### J (Jump)

<table>
<thead>
<tr>
<th>op</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>

E.g.: j
Register Names

We will learn other register names later.

$zero$ always has value 0.

(\$0\)

"temporary"

\$t0, \$t1, \ldots \$t7

\(\$8, \$9, \ldots \$15\)

"save"

\$s0, \$s1, \ldots \$s7

\(\$16, \$17, \ldots \$23\)
$t$ registers: why "temporary"?

(C instruction) \[ g = g + h + i \]

(MIPS instructions)

\begin{align*}
\text{add} & \quad \text{\$t0, \$s1, \$s2} \\
\text{add} & \quad \text{\$s1, \$t0, \$s3}
\end{align*}
Conditional Branch (last lecture)
e.g. "branch equals" (beq)

(C instruction)

```c
    if (a != b)  // if a == b
        f = g + h  // then don't do it
```

(MIPS instructions)

```mips
    beq $s1, $s2, Exit1
    add $s3, $s4, $s5

Exit 1:
```
Conditional Branch

e.g. "branch not equals" (bne)

(C instruction)

```c
if (a == b) // if a != b
    f = g + h // then don't do it.
```

(MIPS instructions)

```assembly
bne $s1, $s2, Exit1
add $s3, $s4, $s5
```

Exit 1:
Other Conditional Branches?

if \( a \leq b \)      bgt ...  
if \( a < b \)        bge ... ? 
if \( a \geq b \)      blt ... .
if \( a > b \)        ble ... 

These instructions don't exist in MIPS.

Why not? What to do?
Instead use "set less than" $\text{slt}$

$\text{slt}$

Assigns $t_o = \begin{cases} 1, & \text{if } s_1 < s_2 \\ 0, & \text{if } s_1 \geq s_2 \end{cases}$
blt $sl, $s2, Exit1

slt bne $to, $sl, $s2
    $to, $zero, Exit1
How to express inequalities using only "<" and "not"?

\[
\begin{align*}
a & \geq b & \iff & \overline{a < b} \\
a & > b & \iff & b < b \\
a & \leq b & \iff & b < a
\end{align*}
\]
<table>
<thead>
<tr>
<th>Desired</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bge</code></td>
<td><code>$so, $sl, Exit</code></td>
</tr>
<tr>
<td><code>bgt</code></td>
<td><code>$so, $sl, Exit</code></td>
</tr>
<tr>
<td><code>ble</code></td>
<td><code>$so, $sl, Exit</code></td>
</tr>
</tbody>
</table>
To avoid headaches, MIPS assemblers allow you to use pseudoinstructions such as \texttt{blt}, \texttt{ble}.

Assemblers (e.g. MARS) convert these to real MIPS instructions.
Why was MIPS designed this way?

The benefits of having fewer instructions available outweighs the cost of having more instructions in each program.

(recall: RISC - reduced instruction set computer)
lecture 9

MIPS assembly language 2

- register names
- pseudo instructions
- immediate versions
- signed and unsigned
- shifting
f = \text{addi} \text{ so, sl, -13}
if (i < 3)

```assembly
slti $t0, $s0, 3
beg $t0, $zero, Exit1
```
Q: What is the difference here?

A: The "overflow" conditions (recall Exercises 2 Q 12)

e.g. 01...... signed -> overflow

+ 01...... unsigned -> no overflow

1......
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

```
addi   $s0, $s1, -357
```

```
addiu  $s0, $s1, 50000
```

between $2^{15}$ and $2^{16}-1$
Example

Let \( \{\) $s_0$ \(,\) $s_1$ \(\}\) \(\) transitions:

\[
\begin{array}{c}
\text{slt} \to, \; \{ \text{sltu} \to, \; \{ \text{to}, \; \{ \text{so}, \; \{ s_1 \to = 0 \}
\end{array}
\]

\[
\begin{array}{c}
\text{sltu} \to, \; \{ \text{to}, \; \{ \text{so}, \; \{ s_1 \to = 1 \}
\end{array}
\]
So exist a
Exercise:

MIPS doesn't have these two. Why not?
Sign Extension

```
add i $16, $17, 3
```

```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
add i $16, $17, -3
```

```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Data path.... (coming soon...)
Manipulating Bits

How to put some 32 bit pattern e.g. 0x37b1 fa93 into a register?

lui $s0, 0x37b1
ori $s0, $s0, 0xfa93

Q: What happens if you switch the order of those two instructions?
Shifting Bits

# shift left logical
Sll $s0, $s1, 7

# shift right logical
Sr1 $s0, $s0, 8

(You might have thought this was "I" format, but in fact it is "R" format. Why?)
Announcements

- A1 due Wednesday night
- A2 will be posted Wednesday (MIPS programming)
- no office hours today for me