Lecture 5: More Instructions, Procedure Calls

• Today’s topics:
  - Numbers, control instructions
  - Procedure calls
Memory Organization

• The space allocated on stack by a procedure is termed the activation record (includes saved values and data local to the procedure) – frame pointer points to the start of the record and stack pointer points to the end – variable addresses are specified relative to $fp as $sp may change during the execution of the procedure
• $gp points to area in memory that saves global variables
• Dynamically allocated storage (with malloc()) is placed on the heap
Recap – Numeric Representations

- **Decimal**
  \[35_{10} = 3 \times 10^1 + 5 \times 10^0\]

- **Binary**
  \[00100011_2 = 1 \times 2^5 + 1 \times 2^1 + 1 \times 2^0\]

- **Hexadecimal (compact representation)**
  \[0x23 \text{ or } 23_{\text{hex}} = 2 \times 16^1 + 3 \times 16^0\]

0-15 (decimal) → 0-9, a-f (hex)

<table>
<thead>
<tr>
<th>Dec</th>
<th>Binary</th>
<th>Hex</th>
<th>Dec</th>
<th>Binary</th>
<th>Hex</th>
<th>Dec</th>
<th>Binary</th>
<th>Hex</th>
<th>Dec</th>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>00</td>
<td>4</td>
<td>0100</td>
<td>04</td>
<td>8</td>
<td>1000</td>
<td>08</td>
<td>12</td>
<td>1100</td>
<td>0c</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>01</td>
<td>5</td>
<td>0101</td>
<td>05</td>
<td>9</td>
<td>1001</td>
<td>09</td>
<td>13</td>
<td>1101</td>
<td>0d</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>02</td>
<td>6</td>
<td>0110</td>
<td>06</td>
<td>10</td>
<td>1010</td>
<td>0a</td>
<td>14</td>
<td>1110</td>
<td>0e</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>03</td>
<td>7</td>
<td>0111</td>
<td>07</td>
<td>11</td>
<td>1011</td>
<td>0b</td>
<td>15</td>
<td>1111</td>
<td>0f</td>
</tr>
</tbody>
</table>
Instruction Formats

Instructions are represented as 32-bit numbers (one word), broken into 6 fields.

**R-type instruction**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>6</td>
<td>opcode</td>
</tr>
<tr>
<td>rs</td>
<td>5</td>
<td>source</td>
</tr>
<tr>
<td>rt</td>
<td>5</td>
<td>source</td>
</tr>
<tr>
<td>rd</td>
<td>5</td>
<td>dest</td>
</tr>
<tr>
<td>shamt</td>
<td>5</td>
<td>shift</td>
</tr>
<tr>
<td>funct</td>
<td>6</td>
<td>function</td>
</tr>
</tbody>
</table>

Example: `add $t0, $s1, $s2`

```
000000 10001 10010 01000 00000 100000
```

**I-type instruction**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>6</td>
<td>opcode</td>
</tr>
<tr>
<td>rs</td>
<td>5</td>
<td>source</td>
</tr>
<tr>
<td>rt</td>
<td>5</td>
<td>source</td>
</tr>
<tr>
<td>lw</td>
<td>16</td>
<td>constant</td>
</tr>
</tbody>
</table>

Example: `lw $t0, 32($s3)`

```
6 bits 5 bits 5 bits 16 bits
opcode rs rt constant
```
## Logical Operations

<table>
<thead>
<tr>
<th>Logical ops</th>
<th>C operators</th>
<th>Java operators</th>
<th>MIPS instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift Left</td>
<td><code>&lt;&lt;</code></td>
<td><code>&lt;&lt;</code></td>
<td><code>sll</code></td>
</tr>
<tr>
<td>Shift Right</td>
<td><code>&gt;&gt;</code></td>
<td><code>&gt;&gt;&gt;</code></td>
<td><code>srl</code></td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td><code>&amp;</code></td>
<td><code>&amp;</code></td>
<td><code>and, andi</code></td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td>`</td>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>Bit-by-bit NOT</td>
<td><code>~</code></td>
<td><code>~</code></td>
<td><code>nor</code></td>
</tr>
</tbody>
</table>
Control Instructions

• Conditional branch: Jump to instruction L1 if register1 equals register2: \texttt{beq register1, register2, L1}
  Similarly, \texttt{bne} and \texttt{slt} (set-on-less-than)

• Unconditional branch:
  \texttt{j L1}
  \texttt{jr $s0} \quad \text{(useful for big jumps and procedure returns)}

Convert to assembly:
\begin{verbatim}
if (i == j)
  f = g+h;
else
  f = g-h;
\end{verbatim}
Control Instructions

• Conditional branch: Jump to instruction L1 if register1 equals register2:  \texttt{beq} \ register1, \ register2, \ L1
  Similarly, \texttt{bne} and \texttt{slt} (set-on-less-than)

• Unconditional branch:
  \texttt{j} \ L1
  \texttt{jr} \ $s0 \quad \text{(useful for big jumps and procedure returns)}

Convert to assembly:
\begin{align*}
\text{if} \ (i \ == \ j) & \quad \text{bne} \ \$s3, \ \$s4, \ \text{Else} \\
\text{f} = \text{g}+\text{h}; & \quad \text{add} \ \$s0, \ \$s1, \ \$s2 \\
\text{else} & \quad \text{j} \ \text{End} \\
\text{f} = \text{g} - \text{h}; & \quad \text{Else:} \ \text{sub} \ \$s0, \ \$s1, \ \$s2 \\
\text{End:} & \quad \text{End}
\end{align*}
Example

Convert to assembly:

```assembly
while (save[i] == k)
    i += 1;
```

Values of i and k are in $s3 and $s5 and base of array save[] is in $s6
Example

Convert to assembly:

```assembly
while (save[i] == k)  
    i += 1;
```

Values of i and k are in $s3
and $s5 and base of array
save[] is in $s6

Loop:  
    sll  $t1, $s3, 2
    add  $t1, $t1, $s6
    lw   $t0, 0($t1)
    bne  $t0, $s5, Exit
    addi $s3, $s3, 1
    j    Loop
Exit:  
    sll  $t1, $s3, 2
    add  $t1, $t1, $s6
    lw   $t0, 0($t1)
    bne  $t0, $s5, Exit
    addi $s3, $s3, 1
    addi $t1, $t1, 4
    j    Loop
Exit:
Registers

• The 32 MIPS registers are partitioned as follows:

- Register 0 : $zero always stores the constant 0
- Regs 2-3 : $v0, $v1 return values of a procedure
- Regs 4-7 : $a0-$a3 input arguments to a procedure
- Regs 8-15 : $t0-$t7 temporaries
- Regs 16-23: $s0-$s7 variables
- Regs 24-25: $t8-$t9 more temporaries
- Reg 28 : $gp global pointer
- Reg 29 : $sp stack pointer
- Reg 30 : $fp frame pointer
- Reg 31 : $ra return address
Procedures

- Local variables, AR, $fp, $sp
- Scratchpad and saves/restores
- Arguments and returns
- jal and $ra