Lecture 5: More Instructions, Procedure Calls

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

- The format of a store instruction:

\[
\text{sw} \quad \$t0, \quad 8(\$t3)
\]  

- source register
- destination address
- any register

A constant that is added to the register in parentheses
Example

```
int a, b, c, d[10];

addi $gp, $zero, 1000  # assume that data is stored at
                     # base address 1000; placed in $gp;
                     # $zero is a register that always
                     # equals zero
lw  $s1, 0($gp)       # brings value of a into register $s1
lw  $s2, 4($gp)       # brings value of b into register $s2
lw  $s3, 8($gp)       # brings value of c into register $s3
lw  $s4, 12($gp)      # brings value of d[0] into register $s4
lw  $s5, 16($gp)      # brings value of d[1] into register $s5
```
Example

Convert to assembly:

Convert to assembly:


Assembly (same assumptions as previous example):
\[
\begin{align*}
    lw & \quad $s0, 0($gp) \quad \# \quad a \text{ is brought into } $s0 \\
    lw & \quad $s1, 20($gp) \quad \# \quad d[2] \text{ is brought into } $s1 \\
    add & \quad $s2, $s0, $s1 \quad \# \quad \text{the sum is in } $s2 \\
    sw & \quad $s2, 24($gp) \quad \# \quad $s2 \text{ is stored into } d[3]
\end{align*}
\]

Assembly version of the code continues to expand!
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)
  \(0x\ 23\ \text{or}\ 23_{\text{hex}} = 2 \times 16^1 + 3 \times 16^0\)

  0-15 (decimal) \(\rightarrow\) 0-9, a-f (hex)

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

\[
\text{add} \quad \$t0, \$s1, \$s2
\]

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
</tr>
</tbody>
</table>

**I-type instruction**

\[
\text{lw} \quad \$t0, 32(\$s3)
\]

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>constant</td>
</tr>
</tbody>
</table>

\[
(\$s3) \quad (\$t0)
\]
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 (with $zero)</code></td>
</tr>
</tbody>
</table>
Control Instructions

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

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

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

Convert to 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:

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

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