\section*{SI 232}
\section*{SlideSet #3: Control Flow (more chapter 2)}

\subsection*{Conditional Control}
\begin{itemize}
  \item Decision making instructions
    \begin{itemize}
      \item alter the control flow,
      \item i.e., change the "next" instruction to be executed
    \end{itemize}
  \item MIPS conditional branch instructions (I - type):
    \begin{verbatim}
    bne $t0, $t1, Label
    beq $t0, $t1, Label
    \end{verbatim}
  \item Example:
    \begin{verbatim}
    if (i == j)
      h = i + j;
    \end{verbatim}
  \item Assembly Code:
    \begin{verbatim}
    bne $s0, $s1, Label
    add $s3, $s0, $s1
    Label: ....
    \end{verbatim}
\end{itemize}

\subsection*{Unconditional Control}
\begin{itemize}
  \item MIPS unconditional branch instructions:
    \begin{verbatim}
    j label
    \end{verbatim}
  \item New type of instruction (J-type)
    \begin{itemize}
      \item op code is 2 (no function field)
    \end{itemize}
  \item Example:
    \begin{verbatim}
    if (i!=j)
      h=i+j;
      add $s3, $s4, $s5
    else
      h=i-j;
    \end{verbatim}
    \begin{verbatim}
    Label1: sub $s3, $s4, $s5
    Label2: ...
    \end{verbatim}
\end{itemize}
Example

- What is the MIPS assembly code for the following:
  If \( j == i \)  \( f = g + h \);
  Else  \( f = g - h \);

Variables \( f \) to \( j \) are assigned to registers \$s0\) to \$s4

So far:

- Instruction  Meaning
  
  - Add \( $s1, $s2, $s3 \)  \( $s1 = $s2 + $s3 \)
  - Sub \( $s1, $s2, $s3 \)  \( $s1 = $s2 - $s3 \)
  - Lw \( $s1, 100($s2) \)  \( $s1 = \text{Memory}[\$s2+100] \)
  - Sw \( $s1, 100($s2) \)  \( \text{Memory}[\$s2+100] = $s1 \)
  - Beq \( $s4, $s5, L \)  Next instr. is at Label if \$s4 == \$s5\)
  - Bne \( $s4, $s5, L \)  Next instr. is at Label if \$s4 != \$s5\)
  - J Label  Next instr. is at Label

- Formats:

  \[
  \begin{array}{c|cccc}
  \text{R} & \text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{func} \\
  \hline
  1 & \text{op} & \text{rs} & \text{rt} & 16 \text{ bit address} \\
  3 & \text{op} & 26 \text{ bit address} \\
  \end{array}
  \]

Exercise #1

- What is the MIPS assembly code for the following:
  If \( g != j \)  \( h = g - h \);
  Else  \( h = g + h \);

Variables \( f \) to \( j \) are assigned to registers \$s0\) to \$s4

Exercise #2

- What is the MIPS assembly code for the following:
  If \( j == h \)  \( g = i + j \);

Variables \( f \) to \( j \) are assigned to registers \$s0\) to \$s4
Exercise #3

- What is the MIPS assembly code for the following:
  
  if ( (j == h) && (f != i) )  g = i + j;

  Variables f to j are assigned to registers $s0 to $s4

  $f  $s0
  $g  $s1
  $h  $s2
  $i  $s3
  $j  $s4

Exercise #4

- What is the MIPS assembly code for the following:

  if ( ( (g != h) && (f == i) ) ||
       ( (g == h) && (j == i) ) )
    g = i + j;

  Variables f to j are assigned to registers $s0 to $s4

  $f  $s0
  $g  $s1
  $h  $s2
  $i  $s3
  $j  $s4

Control Flow – Branch if less than

- We have: beq, bne, what about Branch-if-less-than?
- New instruction:

  if $s1 < $s2 then
    $t0 = 1
  else
    $t0 = 0

  $t0 is a R-type instruction (function code 42)

Example

- What is the MIPS assembly code to test if variable a ($s0) is less than variable b($s1) and then branch to Less: if the condition holds?

  if (a < b)
    go to Less;
  ....

  Less: ....
Pseudoinstructions

- Example #1: Use `slt` instruction to build "blt $s1, $s2, Label"
  - Pseudoinstruction" that assembler expands into several real
  instructions
  - Note that the assembler needs a register to do this
  - What register should it use?
  - Why not make `blt` a real instruction?

- Example #2: ‘Move’ instruction
  - ‘move $t0, $t1’
  - Implementation?

Policy of Use Conventions

<table>
<thead>
<tr>
<th>Name</th>
<th>Register number</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
<td>the constant value 0</td>
</tr>
<tr>
<td>v0-v1</td>
<td>2-3</td>
<td>values for results and expression evaluation</td>
</tr>
<tr>
<td>a0-a3</td>
<td>4-7</td>
<td>arguments</td>
</tr>
<tr>
<td>t0-t32</td>
<td>8-15</td>
<td>temporaries</td>
</tr>
<tr>
<td>t0-t32</td>
<td>16-23</td>
<td>saved</td>
</tr>
<tr>
<td>t0-t32</td>
<td>24-25</td>
<td>more temporaries</td>
</tr>
<tr>
<td>sp</td>
<td>28</td>
<td>global-pointer</td>
</tr>
<tr>
<td>fp</td>
<td>29</td>
<td>black pointer</td>
</tr>
<tr>
<td>ra</td>
<td>31</td>
<td>frame pointer</td>
</tr>
<tr>
<td>at</td>
<td>32</td>
<td>return address</td>
</tr>
</tbody>
</table>

\$at = Register #1 – reserved for assembler
\$t0, \$t1 = Register #26, 27 – reserved for OS

Constants

- Small constants are used quite frequently
  - e.g., $A = A + 5;
  $B = B + 1;
  $C = C - 18;
- Possible solution
  - put ‘typical constants’ in memory and load them.
  - And create hard-wired registers for constants like zero, one.
- Problem?

- MIPS Instructions:
  addi $29, $29, 4
  slli $8, $18, 10
  addi $29, $29, 6
  ori $29, $29, 4
- How do we make this work?

<table>
<thead>
<tr>
<th>Type</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
</tr>
</thead>
</table>

How about larger constants?

- We’d like to be able to load a 32 bit constant into a register
- Must use two instructions, new “load upper immediate” instruction

```
  lui $t0, 1010101010101010
  1010101010101010 0000000000000000
  0000000000000000 1010101010101010
  1010101010101010 1010101010101010
```

- Then must get the lower order bits right, i.e.,

```
  ori $t0, $t0, 1010101010101010
  0000000000000000 1010101010101010
  1010101010101010 1010101010101010
  1010101010101010 1010101010101010
```
Assembly Language vs. Machine Language

- Assembly provides convenient symbolic representation
  - much easier than writing down numbers
  - e.g., destination first
- Machine language is the underlying reality
  - e.g., destination is no longer first
- Assembly can provide 'pseudoinstructions'
- When considering performance you should count

Memory – Byte Order & Alignment

- **Endian**
  - Processors don't care
  - Big: 0,1,2,3
  - Little: 3,2,1,0
  - Network byte order:

- Alignment
  - require that objects fall on address that is multiple of
  - Legal word addresses:
  - Legal byte addresses:

Looping

- We know how to make decisions, but:
  - Can we set up a flow that allows for multiple iterations?
  - What high level repetition structures could we use?
  - What MIPS instructions could we use?
- "Basic block"
  - Sequence of instructions __________________________ except possibly __________________________

Looping Example

**C Code**

do {
  g = g + A[i]; //vars g to j in $s1 to $s4
  i = i + 1; // $s5 holds base address of A
} while (i <= h)

**Assembly Language**

Loop:  add $t1, $s3, $s3  #
          add $t1, $t1, $t1  #
          add $t1, $t1, $s5  #
          lw $t0, 0($t1)  #
          add $s1, $s1, $t0  #
          add $s3, $s3, $s4  #
          bne $s3, $s2, Loop  #

G  $s1
H  $s2
I  $s3
J  $s4
&A  $s5
Exercise #1

a.) What is the MIPS assembly code for the following:

\[
\begin{align*}
d & \cdot \text{do} \\
g & = g + j; \\
\} \text{ while } (g < h);
\end{align*}
\]

Variables \( f \) to \( j \) are assigned to registers \( $s0 \) to \( $s4 \)
Use \( $v0, $v1 \) as temporaries if needed

b.) Did your solution use any pseudo-instructions?

Exercise #2

a.) What is the MIPS assembly code for the following:

\[
\begin{align*}
d & \cdot \text{do} \\
g & = g + j; \\
\} \text{ while } (g < 100);
\end{align*}
\]

Variables \( f \) to \( j \) are assigned to registers \( $s0 \) to \( $s4 \)
Use \( $v0, $v1 \) as temporaries if needed

b.) Did your solution use any pseudo-instructions?

Exercise #3

a.) What is the MIPS assembly code for the following:

\[
\begin{align*}
\text{while } (g < i) \{ \\
g & = g + j;
\}
\end{align*}
\]

Variables \( f \) to \( j \) are assigned to registers \( $s0 \) to \( $s4 \)
Use \( $v0, $v1 \) as temporaries if needed

b.) Did your solution use any pseudo-instructions?

Exercise #4

a.) What is the MIPS assembly code for the following:

\[
\begin{align*}
\text{while } (g > i) \{ \\
g & = g + 3;
\}
\end{align*}
\]

Variables \( f \) to \( j \) are assigned to registers \( $s0 \) to \( $s4 \)
Use \( $v0, $v1 \) as temporaries if needed

b.) Did your solution use any pseudo-instructions?