CS 201 Lecture 13
Assembly Code, Loops, and Labels
Spring 2014

© John A. Stratton, 2014
Plan for today

• Loops in instructions
• Assembly code (because we’re tired of bits)
• Labels and the Assembler
Loops in instructions

• Loops always involve a branch backwards
  – Go back and repeat instructions already done
• Several options for loop implementation, matching the different loop patterns
  – For, while, do-while

Common while-loop approach
• Test loop condition: branch past body if false
• At end of body, unconditional jump back to start
  – Retest condition
Translate the following into instructions

```java
int a = 0;
for (int i = 0; i < 100; i++) {
    a = a + i;
}
```

Put “a” into R10, and “i” into R8
R10 ← R10 AND 0
R8 ← R8 AND 0
R9 ← R8 + 100
BranchEQ R8, R9, ?
R10 ← R10 + R8
R8 ← R8 + 1
BranchEQ R8, R8, ?
R10 ← R10 AND 0
R8 ← R8 AND 0
R9 ← R8 + 100
BranchEQ R8, R9, 12
R10 ← R10 + R8
R8 ← R8 + 1
BranchEQ R8, R8, -16
Assembly Code

• RTL has the problem of being unspecific
  – Sometimes multiple options of how to accomplish the same operation with different instructions
  – Some desired RTL operations may have no single instruction

• Typing bits is hard
  – How many mistakes do you think would happen if all code was manually typed in binary?

• Assembly code: because people got tired of typing bits
MIPS Assembly Conventions

\(<\text{instruction}> \ <\text{operands}>\)

• Instruction is a word or mnemonic for an opcode
  – MIPS Instructions include, `add`, `addi`, `loadw`, `sub`, `bne`, `j`

• Operands are identifiers for
  – Registers: starting with a $ (like $0)
  – Immediates: type number directly (usually decimal and hexadecimal values supported)
  – Composite memory references: immediate(baseR)
    e.g. 4($0) for accessing memory at R0 + 4
Example RTL from last time

R5 ← M[R10 + 0]
R6 ← M[R10 + 4]
R5 ← R5 + R6
R6 ← M[R10 + 8]
R5 ← R5 + R6
R6 ← M[R10 + 12]
R5 ← R5 + R6
R5 ← 00, R5 (31:2)
M[R10 + 0] ← R5
Example from last time

lw $5, ($10)
lw $6, 4($10)
add $5, $5, $6
lw $6, 8($10)
add $5, $5, $6
lw $6, 12($10)
add $5, $5, $6
sra $5, $5, 2
sw $5, 0($10)
Labels

- What is the most annoying thing left to deal with in assembly code
  - Personal preference, I would say branch/address calculations
- Labels are a marker for an address
  - The assembler determines where the address is, and the appropriate number for offsets
Previous example

R10 ← R10 AND 0
R8 ← R8 AND 0
R9 ← R8 + 100

StartofLoop:
  BranchEQ R8, R9, EndOfLoop
  R10 ← R10 + R8
  R8 ← R8 + 1
  BranchEQ R8, R8, StartofLoop

EndOfLoop:
Substitute R10 for label Data:

lw $5, Data
lw $6, Data + 4
add $5, $5, $6
lw $6, Data + 8
add $5, $5, $6
lw $6, Data + 12
add $5, $5, $6
sra $5, $5, 2
sw $5, Data

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01010</td>
<td>00101</td>
<td>0000000000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100011</td>
<td>01010</td>
<td>00110</td>
<td>00000000000000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>00101</td>
<td>00110</td>
<td>00101</td>
<td>00000</td>
<td>10000</td>
</tr>
<tr>
<td>100011</td>
<td>01010</td>
<td>00110</td>
<td>00000000000001000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>00101</td>
<td>00110</td>
<td>00101</td>
<td>00000</td>
<td>10000</td>
</tr>
<tr>
<td>100011</td>
<td>01010</td>
<td>00110</td>
<td>00000000000001100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>00101</td>
<td>00110</td>
<td>00101</td>
<td>00000</td>
<td>10000</td>
</tr>
<tr>
<td>000000</td>
<td>00101</td>
<td>00110</td>
<td>00101</td>
<td>00010</td>
<td>10000</td>
</tr>
<tr>
<td>101011</td>
<td>01010</td>
<td>00101</td>
<td>0000000000000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>