EE 109 Unit 14 – Control Flow

Branch Instructions

• Branches allow us to jump backward or forward in our code
• How? By manipulating the ________________
• Operation: ______________________

Some Examples
Two-Operand Compare & Branches

• Two-operand comparison is accomplished using the SLT/SLTI/SLTU (Set If Less-than) instruction followed by a BNE/BEQ
  – Syntax:  SLT Rd,Rs,Rt or SLT Rd,Rs,imm
    • If Rs < Rt then ______________________________
  – Use appropriate BNE/BEQ instruction to infer relationship

<table>
<thead>
<tr>
<th>Branch if...</th>
<th>SLT</th>
<th>BNE/BEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2 &lt; $3</td>
<td>SLT $1,$2,$3</td>
<td>B___ $1,$0,label</td>
</tr>
<tr>
<td>$2 ≤ $3</td>
<td>SLT $1,$3,$2</td>
<td>B___ $1,$0,label</td>
</tr>
<tr>
<td>$2 &gt; $3</td>
<td>SLT $1,$3,$2</td>
<td>B___ $1,$0,label</td>
</tr>
<tr>
<td>$2 ≥ $3</td>
<td>SLT $1,$2,$3</td>
<td>B___ $1,$0,label</td>
</tr>
</tbody>
</table>

Branch Pseudo-Instructions

• Rather than writing two instructions (SLT and BNE/BEQ) we can use provided pseudoinstructions

<table>
<thead>
<tr>
<th>Pseudo-instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLT Rt, Rs, label</td>
<td>Branch if less-than</td>
</tr>
<tr>
<td>BLE Rt, Rs, label</td>
<td>Branch if less-than or equal</td>
</tr>
<tr>
<td>BGT Rt, Rs, label</td>
<td>Branch if greater-than</td>
</tr>
<tr>
<td>BGE Rt, Rs, label</td>
<td>Branch if greater-than of equal</td>
</tr>
<tr>
<td>BLTU Rt, Rs, label</td>
<td>Branch if less-than (unsigned)</td>
</tr>
<tr>
<td>BLT Rt, imm, label</td>
<td>Branch if less-than immediate</td>
</tr>
</tbody>
</table>

Note: Pseudoinstructions are assembler-dependent. See MARS Help for more details.

Comparison with SLT

• Performing comparison with the SLT instruction is really accomplished by subtracting A-B and examining the sign of the result
  – if A-B = ____________, then A=B
  – if A-B = ____________, then A<B
  – If A-B = ____________, then A>B
• Determining if the result is positive or negative requires
  – knowing what system is being used
    • signed or unsigned?
  – if overflow occurred
    • when overflow occurs the sign of the result is incorrect (i.e. p+p = n or n+n = p)

SLT/SLTU Operation

• Use appropriate version based on system being used
  – SLT for ____________ operand
  – SLTU for ____________ operands
• An SLT instruction subtracts A-B and examine sign of the result and the overflow test to determine if it should set the result
Single-Operand Compare & Branches

• BGT, BLT, BGE, BLE are pseudoinstructions
  – Shorthand for an SLT and BEQ
• MIPS does have some single instructions to compare and branch all in one, but only for a single operand compared with 0
• Syntax: BccZ Rt, label
  – cc = {LT, LE, GT, GE}

<table>
<thead>
<tr>
<th>Branch Instruc.</th>
<th>Branch if...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLTZ $2,0</td>
<td>$2 &lt; 0</td>
</tr>
<tr>
<td>BLEZ $2,0</td>
<td>$2 ≤ 0</td>
</tr>
<tr>
<td>BGTZ $2,0</td>
<td>$2 &gt; 0</td>
</tr>
<tr>
<td>BGEZ $2,0</td>
<td>$2 ≥ 0</td>
</tr>
</tbody>
</table>

Branch Example 1

C Code

if A > B
| A = A + B |
| else |
| A = 1 |

MIPS Assembly

```assembly
.text
LW $t2,0($t0)
LW $t3,0($t1)
```

Branch Example 2

C Code

for(i=0; i < 10; i++)
| j = j + i; |

MIPS Assembly

```
.text
```

Another Branch Example

```
int dat[10];
for(i=0;i < 10;i++)
    data[i] = 5;
```

A Final Example

```
char A[] = "hello world";
char B[50];
while(A[i] != 0){
    B[i] = A[i]; i++;
}
B[i] = 0;
```

Branch Machine Code Format

- Branch instructions use the I-Type Format
- Operation: \( PC = PC + \{\text{disp.},00\} \)
- Displacement notes
  - Displacement is the value that should be added to the PC so that it now points to the desired branch location
  - Processor appends two 0's to end of disp. since all instructions are 4-byte words
    - Essentially, displacement is in units of words
    - Effective range of displacement is an 18-bit signed value = ±128KB address space (i.e. can't branch anywhere in memory...but long branches are rare and there is a mechanism to handle them)
**Branch Displacement**

- To calculate displacement you must know where instructions are stored in memory (relative to each other)
  - Don’t worry, assembler finds displacement for you...you just use the label

```
.text
ADD $8, $0, $0
ADDI $7, $0, 10
LOOP: SLTI $1, $8, 10
BEQ $1, $0, NEXT
ADD $9, $9, $8
ADDI $8, $8, 1
BEQ $0, $0, LOOP

NEXT: ----
```

**MIPS Assembly**

**Calculating Displacements**

- **Disp. = [(Addr. of Target) – (Addr. of Branch + 4)] / 4**
  - Constant 4 is due to the fact that by the time the branch executes the PC will be pointing at the instruction after it (i.e. plus 4 bytes)
  - Following slides will show displacement calculation for `BEQ $1,$0,NEXT`

```
.text
ADD $8, $0, $0
ADDI $7, $0, 10
ADDI $9, $9, $8
ADDI $8, $8, 1
BEQ $0, $0, LOOP

NEXT: ----
```

**MIPS Assembly**

- **Disp. = [(Addr. of Target) – (Addr. of Branch + 4)] / 4**
  - Constant 4 is due to the fact that by the time the branch executes the PC will be pointing at the instruction after it (i.e. plus 4 bytes)

```
.text
ADD $8, $0, $0
ADDI $7, $0, 10
ADDI $9, $9, $8
ADDI $8, $8, 1
BEQ $0, $0, LOOP

NEXT: ----
```

**MIPS Assembly**

**Calculating Displacements**

- If the BEQ does in fact branch, it will add the displacement `(0x03, 00) = 00000C) to the PC (A+0x10) and thus point to the ---- instruction (A+0x1C)

```
.text
ADD $8, $0, $0
ADDI $7, $0, 10
ADDI $9, $9, $8
ADDI $8, $8, 1
BEQ $0, $0, LOOP

NEXT: ----
```

**MIPS Assembly**
Another Example

- Disp. = [(Addr. of Label) – (Addr. of Branch + 4)] / 4
- Disp. = (A+0x04) – (A+0x14 + 4) = 0x04 – 0x18 = 0xFFEC / 4 = 0xFFFFB

Jump Instructions

- Jumps provide method of branching beyond range of

- Syntax: _______________

  - Operation: PC = __________
  - Address is appended with two 0’s just like branch displacement yielding a 28-bit address with upper 4-bits of PC unaffected

- New instruction format: J-Type

Jump Example

- Take 28 LSB’s of target address, remove LSB’s (which are 0’s) and store 26-bits in the jump instruction

Jump Register

- ‘jr’ instruction can be used if a full 32-bit jump is needed or variable jump address is needed

- Syntax: JR rs
  - Operation: PC = R[s]
  - R-Type machine code format

- Usage:
  - Can load rs with an immediate address
  - Can calculate rs for a variable jump (class member functions, switch statements, etc.)
jr Example

- Take whatever value is in the source register and places it in the PC (jumping you to that address)

```
.text
ADD      $8, $0, $0
LUI      $6, 0xffff
ORI      $6, $6, 0x0010
JR       $6
...
```

0xffff0010:

- **Value of $6**
  - FFFF0010

- **PC before exec. of jr:**
  - 00400010

- **PC after exec. of jr:**