Chapter 2

MIPS Program Flow Instructions

MIPS-32 ISA Review

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point

3 Instruction Formats: all 32 bits wide

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>I format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>jump target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J format</td>
</tr>
</tbody>
</table>
## MIPS I-type Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Mnemonic</th>
<th>Format</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Immediate</td>
<td>ADDI</td>
<td>I</td>
<td>rs rd immediate</td>
</tr>
<tr>
<td>Add Immediate Unsigned</td>
<td>ADDIU</td>
<td>I</td>
<td>s $d immediate</td>
</tr>
<tr>
<td>Set on Less Than Immediate</td>
<td>SLTI</td>
<td>I</td>
<td>10s $d immediate</td>
</tr>
<tr>
<td>Set on Less Than Immediate Unsigned</td>
<td>SLTIU</td>
<td>I</td>
<td>11s $d immediate</td>
</tr>
<tr>
<td>And Immediate</td>
<td>ANDI</td>
<td>I</td>
<td>12s $d immediate</td>
</tr>
<tr>
<td>Or Immediate</td>
<td>ORI</td>
<td>I</td>
<td>13s $d immediate</td>
</tr>
<tr>
<td>Exclusive Or Immediate</td>
<td>XORI</td>
<td>I</td>
<td>14s $d immediate</td>
</tr>
<tr>
<td>Load Upper Immediate</td>
<td>LUI</td>
<td>I</td>
<td>150s $d immediate</td>
</tr>
<tr>
<td>Branch on Equal</td>
<td>BEQ</td>
<td>I</td>
<td>410 rt offset</td>
</tr>
<tr>
<td>Branch on Not Equal</td>
<td>BNE</td>
<td>I</td>
<td>510 rt offset</td>
</tr>
<tr>
<td>Branch on Less Than or Equal to Zero</td>
<td>BLEZ</td>
<td>I</td>
<td>610 rs 010 offset</td>
</tr>
<tr>
<td>Branch on Greater Than Zero</td>
<td>BGTZ</td>
<td>I</td>
<td>710 rs 010 offset</td>
</tr>
<tr>
<td>Branch on Less Than Zero</td>
<td>BLTZ</td>
<td>I</td>
<td>110 rs 010 offset</td>
</tr>
<tr>
<td>Branch on Greater Than or Equal to Zero</td>
<td>BGEZ</td>
<td>I</td>
<td>110 rs 110 offset</td>
</tr>
<tr>
<td>Branch on Less Than Zero and Link</td>
<td>BLTZAL</td>
<td>I</td>
<td>110 rs 16 offset</td>
</tr>
<tr>
<td>Branch on Greater Than or Equal to Zero and Link</td>
<td>BGEZAL</td>
<td>I</td>
<td>110 rs 17 offset</td>
</tr>
</tbody>
</table>

## MIPS I-type Instructions

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Mnemonic</th>
<th>Format</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Byte</td>
<td>LB</td>
<td>I</td>
<td>3210 rs rt offset</td>
</tr>
<tr>
<td>Load Halfword</td>
<td>LH</td>
<td>I</td>
<td>3310 rs rt offset</td>
</tr>
<tr>
<td>Load Word Left</td>
<td>LWL</td>
<td>I</td>
<td>3410 rs rt offset</td>
</tr>
<tr>
<td>Load Word</td>
<td>LW</td>
<td>I</td>
<td>3510 rs rt offset</td>
</tr>
<tr>
<td>Load Byte Unsigned</td>
<td>LBU</td>
<td>I</td>
<td>3610 rs rt offset</td>
</tr>
<tr>
<td>Load Halfword Unsigned</td>
<td>LHU</td>
<td>I</td>
<td>3710 rs rt offset</td>
</tr>
<tr>
<td>Load Word Right</td>
<td>LWR</td>
<td>I</td>
<td>3810 rs rt offset</td>
</tr>
<tr>
<td>Store Byte</td>
<td>SB</td>
<td>I</td>
<td>4D10 rs rt offset</td>
</tr>
<tr>
<td>Store Halfword</td>
<td>SH</td>
<td>I</td>
<td>4110 rs rt offset</td>
</tr>
<tr>
<td>Store Word Left</td>
<td>SWL</td>
<td>I</td>
<td>4210 rs rt offset</td>
</tr>
<tr>
<td>Store Word</td>
<td>SW</td>
<td>I</td>
<td>4310 rs rt offset</td>
</tr>
<tr>
<td>Store Word Right</td>
<td>SWR</td>
<td>I</td>
<td>4810 rs rt offset</td>
</tr>
</tbody>
</table>
MIPS Control Flow Instructions

- **MIPS conditional branch instructions:**
  
  - `bne $s0,$s1,Lbl`  # go to Lbl if $s0\neq$s1
  - `beq $s0,$s1,Lbl`  # go to Lbl if $s0=$s1

- **Ex:**
  
  ```
  if (i==j) h = i + j;
  bne $s0, $s1, Lbl1
  add $s3, $s0, $s1
  Lbl1: ...
  ```

- **Immediate Format (I format):**

<table>
<thead>
<tr>
<th>Offset Field</th>
<th>0x05</th>
<th>16</th>
<th>17</th>
<th>16 bit offset</th>
</tr>
</thead>
</table>

- **How is the branch destination address specified?**

---

Specifying Branch Destinations

- **Use a register (like in lw and sw) added to the 16-bit offset**
  
  - The register used is the Program Counter (the **PC**):
    - Its use is automatically *implied* by the instruction.
    - This type of addressing is called **PC relative**.
  
  - This limits the branch distance to $-2^{13}$ to $+2^{13}-1$ instructions from the branch instruction, but most branches are local anyway.

  ![Diagram](diagram.png)

  - **Diagram:**
    - **PC**
    - **offset**
    - **sign-extend**
    - **Add**
    - **branch dst address**
    - **from the low order 16 bits of the branch instruction**
Branching Far Away

- What if the branch destination is further away than can be captured in 16 bits?

- The assembler comes to the rescue – it inserts an unconditional jump to the branch target and inverts the condition

  \[
  \text{beq } \$s0, \$s1, L1
  \]

  becomes

  \[
  \text{bne } \$s0, \$s1, L2
  \]

  \[
  \text{j } L1
  \]

  \[
  L2:
  \]

In Support of Branch Instructions

- We have `beq`, `bne`, but what about other kinds of branches (e.g., branch-if-less-than)? For this, we need yet another instruction, `slt`

- Set on less than instruction:

  \[
  \text{slt } \$t0, \$s0, \$s1 \quad \# \text{ if } \$s0 < \$s1 \quad \text{then}
  \]

  \[
  \quad \# \quad \$t0 = 1 \quad \text{else}
  \]

  \[
  \quad \# \quad \$t0 = 0
  \]

- Instruction format (R format):

<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>17</th>
<th>8</th>
<th>0x24</th>
</tr>
</thead>
</table>

- Alternate versions of `slt`

  \[
  \text{slti } \$t0, \$s0, 25 \quad \# \text{ if } \$s0 < 25 \text{ then } \$t0 = 1 \ldots
  \]

  \[
  \text{sltu } \$t0, \$s0, \$s1 \quad \# \text{ if } \$s0 < \$s1 \text{ then } \$t0 = 1 \ldots
  \]

  \[
  \text{sltiu } \$t0, \$s0, 25 \quad \# \text{ if } \$s0 < 25 \text{ then } \$t0 = 1 \ldots
  \]
More Branch Instructions

- Can use `slt`, `beq`, `bne`, and the fixed value of 0 in register `$zero` to create other conditions:
  - less than
    - `blt $s1, $s2, Label`
    - `slt $at, $s1, $s2 # $at set to 1 if $s1 < $s2`
  - less than or equal to
    - `ble $s1, $s2, Label`
  - greater than
    - `bgt $s1, $s2, Label`
  - great than or equal to
    - `bge $s1, $s2, Label`

- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler:
  - Its why the assembler needs a reserved register (`$at - register 1`).

Other Control Flow Instructions

- MIPS also has an unconditional branch instruction or `jump` instruction:
  - `j label # go to label`

- Instruction Format (J Format):
  - `0x02 26-bit address`
  - from the low order 26 bits of the jump instruction
Instructions for Accessing Procedures

- MIPS procedure call instruction:
  \[ \text{jal } \text{ProcedureAddress} \quad \#\text{jump and link} \]

- Saves PC+4 in register $ra$ (register 31) to have a link to the next instruction for the procedure return.
- Machine format (J format):

  | 0x03 | 26 bit address |

- Then, return from the procedure with a:
  \[ \text{jr } $ra \quad \#\text{return} \]

- Instruction format (R format):

  | 0 | 31 | | 0x08 |

Six Steps in Execution of a Procedure

- Main routine (caller) places parameters in a place where the procedure (callee) can access them:
  - $a0 - a3$: four argument registers.
- Caller transfers control to the callee.
- Callee acquires the storage resources needed.
- Callee performs the desired task.
- Callee places the result value in a place where the caller can access them:
  - $v0 - v1$: two value registers for result values.
- Callee returns control to the caller:
  - $ra$: one return address register to return to the point of origin.
Recap

- MIPS branch instructions (i-type)
- MIPS jump instructions (j-type)
- Procedure calls
- Next – Booth’s recoding algorithm