COMPUTER ORGANIZATION AND DESIGN

The Hardware/Software Interface

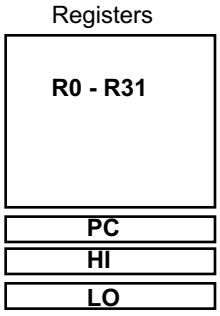
5th Edition

Chapter 2

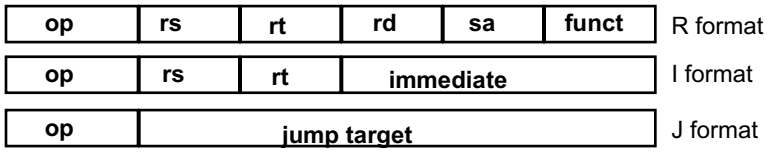
MIPS Program Flow Instructions

MIPS-32 ISA Review

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



3 Instruction Formats: all 32 bits wide



MIPS I-type Instructions

Add Immediate	ADDI	I	8_{10}	rs	rd	immediate
Add Immediate Unsigned	ADDIU	I	9_{10}	$s	$d	immediate
Set on Less Than Immediate	SLTI	I	10_{10}	$s	$d	immediate
Set on Less Than Immediate Unsigned	SLTIU	I	11_{10}	$s	$d	immediate
And Immediate	ANDI	I	12_{10}	$s	$d	immediate
Or Immediate	ORI	I	13_{10}	$s	$d	immediate
Exclusive Or Immediate	XORI	I	14_{10}	$s	$d	immediate
Load Upper Immediate	LUI	I	15_{10}	0_{10}	$d	immediate

Branch on Equal	BEQ	I	4_{10}	rs	rt	offset
Branch on Not Equal	BNE	I	5_{10}	rs	rt	offset
Branch on Less Than or Equal to Zero	BLEZ	I	6_{10}	rs	0_{10}	offset
Branch on Greater Than Zero	BGTZ	I	7_{10}	rs	0_{10}	offset

Branch on Less Than Zero	BLTZ	I	1_{10}	rs	0_{10}	offset
Branch on Greater Than or Equal to Zero	BGEZ	I	1_{10}	rs	1_{10}	offset
Branch on Less Than Zero and Link	BLTZAL	I	1_{10}	rs	16	offset
Branch on Greater Than or Equal to Zero and Link	BGEZAL	I	1_{10}	rs	17	offset

MIPS I-type Instructions

Instruction name	Mnemonic	Format	Encoding			
Load Byte	LB	I	32_{10}	rs	rt	offset
Load Halfword	LH	I	33_{10}	rs	rt	offset
Load Word Left	LWL	I	34_{10}	rs	rt	offset
Load Word	LW	I	35_{10}	rs	rt	offset
Load Byte Unsigned	LBU	I	36_{10}	rs	rt	offset
Load Halfword Unsigned	LHU	I	37_{10}	rs	rt	offset
Load Word Right	LWR	I	38_{10}	rs	rt	offset
Store Byte	SB	I	40_{10}	rs	rt	offset
Store Halfword	SH	I	41_{10}	rs	rt	offset
Store Word Left	SWL	I	42_{10}	rs	rt	offset
Store Word	SW	I	43_{10}	rs	rt	offset
Store Word Right	SWR	I	46_{10}	rs	rt	offset

MIPS Control Flow Instructions

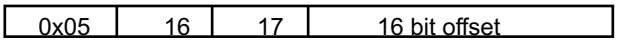
- MIPS *conditional branch* instructions:

```
bne $s0,$s1,Lbl      #go to Lbl if $s0≠$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:   ...
```

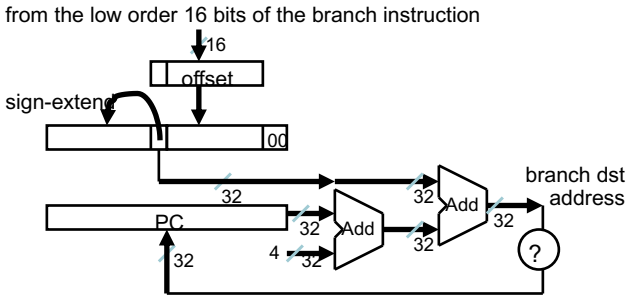
- Immediate Format (I format):



- 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.



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

```
        beq   $s0, $s1, L1
```

becomes

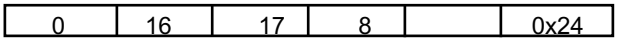
```
        bne   $s0, $s1, L2
        j     L1
L2:
```

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:

```
slt $t0, $s0, $s1     # if $s0 < $s1     then
                      # $t0 = 1          else
                      # $t0 = 0
```

- Instruction format (**R** format):



- Alternate versions of `slt`

```
slti $t0, $s0, 25     # if $s0 < 25 then $t0=1 ...
sltu $t0, $s0, $s1    # if $s0 < $s1 then $t0=1 ...
sltiu $t0, $s0, 25    # if $s0 < 25 then $t0=1 ...
```

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
bne  $at, $zero, Label  #$s1 < $s2
```

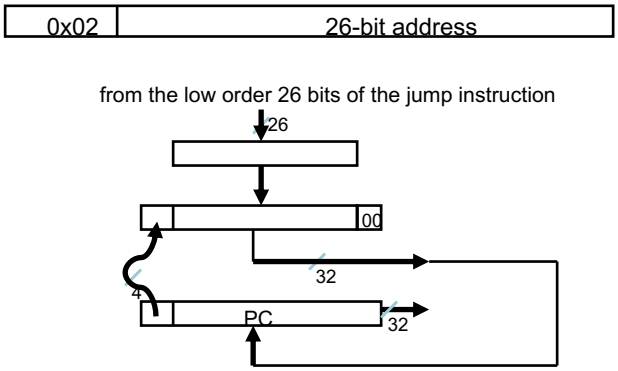
  - less than or equal to `ble $s1, $s2, Label`
  - greater than `bgt $s1, $s2, Label`
  - great than or equal to `bge $s1, $s2, Label`
- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler:
  - Its why the assembler needs a reserved register (`$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):

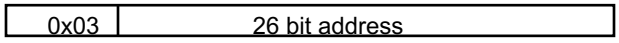


Instructions for Accessing Procedures

- MIPS *procedure call* instruction:

```
jal   ProcedureAddress     #jump and link
```

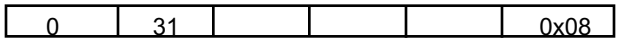
- Saves PC+4 in register $ra (register 31) to have a link to the next instruction for the procedure return.
- Machine format (**J** format):



- Then, *return* from the procedure with a:

```
jr    $ra                  #return
```

- Instruction format (**R** format):



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