Control Instructions

Do not execute the next PC value.
Transfer control to another part of the instruction space.

Two groups of instructions:
• branches
  • conditional transfers of control
  • the target address is close to the current PC location
    • branch distance from the incremented PC value fits
      into the immediate field
  • for example, loops, if statements
• jumps
  • unconditional transfers of control
  • the target address is far away from the current PC location
  • for example, subroutine calls

MIPS Branch Instructions

Branch instructions: conditional transfer of control
• Compare on:
  • equality or inequality of two registers
    Opcode rs, rt, target
    rs, rt: the registers to be compared
    target: the branch target
  • >, <, ≥, ≤ of a register & 0
    Opcode rs, target
    rs: the register to be compared with an implicit 0
    target: the branch target
  • Branch to a target that is a signed displacement (expressed in
    number of instructions) from the instruction following the branch

Some examples:
  beq $8, $9, Target  # branch to Target if $8 == $9
  bnez $8, Target     # branch to Target if $8 ≥ 0

Target:
MIPS Branch Instructions

**beq, bne, bgtz, bltz, bgez, blez**

are the only conditional branch instructions

Use **slt** (set on less than) for >, <, ≥, ≤ comparisons between two registers

\[
\text{slt } \text{rd, rs, rt} \quad \# \text{ if } rs < rt, \text{rd} = 1; \text{else rd} = 0
\]

An example:

- branch if the first register operand is less than the second

\[
\text{slt } \$8, \$9, \$10 \quad \# \text{ if } \$9 < \$10; \text{otherwise } \$8 = 0
\]

\[
\text{bne } \$8, \$0, \text{L1} \quad \# \text{ branch to L1 if } \$9 < \$10
\]

Can use a **pseudoinstruction** as a shortcut:

- an instruction provided by the assembler but not implemented in the hardware

\[
\text{blt } \$9, \$10, \text{L1} \quad \# \text{ pseudoinstruction for branch to L1}
\]

\[
\# \text{ if } \$9 < \$10
\]

assembler expands to:

\[
\text{slt } \$at, \$9, \$10
\]

\[
\text{bne } \$at, \$0, \text{L1}
\]

(note the use of $at ($1) by the assembler)

also **bgt, bge, ble**

---

Branch Distance

Extending the displacement of a branch target address

- offset is a signed 16-bit offset
  - represents a number of **instructions**, not bytes
- added to the incremented PC
- target address is a **word** address, not a byte address
  - bottom 2 bits are zero
  - in assembly language, use a symbolic target address

Why can you do this?

- each instruction is a fixed length, i.e., 4 bytes

What does it buy you?

- 4 times the branch distance
Branch Distance

Branch offset is a decent size
- 16-bit offset
- added to the incremented PC
- represents a word address

But what if it is too small to reach the branch target?
- assembler inserts an unconditional jump
- the conditional branch branches to the original false path code
  (condition evaluated to false) or falls through to the jump

Example:
```
beq $s0, $s1, L1
```
changes to:
```
bne $s0, $s1, L2
j L1
```
`L2:` the false path: the original fall through code

I-type Format for Branches

I-type format used for conditional branches

```
31 26 20 16

   opcode rs rt immed

   25 21 15 0

bne $s0, $s1, Exit
```

- `opcode` = operation
  - `opcode` = control instruction
- `rs, rt` = source operands
- `immed` = address offset in words, $\pm 2^{15}$
  - hardware sign-extends when uses (replicate msb)
  - target address = PC + (immed*4)
MIPS Jump Instructions

Jump instructions: unconditional transfer of control

**j**  target  # jump
go to the specified target address

**jr**  rs  # jump register
go to the address stored in rs
(called an indirect jump)

**jal**  target  # jump and link
go to the target address; save PC+4 in $ra

**jalr**  rs, rd  # jump and link register
go to the address stored in rs; rd = PC+4
default rd is $ra

Examples:

jal procedureAddress calls a procedure
jr $31 (or jr $ra) returns from a procedure
jr $8 implements a case statement
  • where the target addresses for the different cases are in a
table (jump address table)
  • $8 contains one such entry

I-type Format for Jumps

J-type format used for unconditional jumps

```
31 2625 0
   opcode      address
```

• **opcode** = operation
  • opcode = data transfer instruction
• **address** = partial address in words
  • bottom 2 bits are zero (jumping to a word/instruction boundary)
  • top 4 bits come from the PC

```
  j 10000
     2      10000
```
If/then/else Example

The C version

```c
if (i == j)
    f = g + h;
else f = g - h;
```

An assembly language version:

```
i in $s3, j in $s4
f in $s0, g in $s1, h in $s2
```
```
bne $s3,$s4,Else       # go to Else if i not = j
add $s0,$s1,$s2  # = g + h
j Exit              # jump out of the if
```
```
Else: sub $s0,$s1,$s2  # = g - h
Exit:
```