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
bgez   $8, Target      # branch to Target if $8 >= 0
```

Target:
MIPS Branch Instructions

\texttt{beq, bne, bgtz, bltz, bgez, blez}

are the only conditional branch opcodes

Use \texttt{slt} (set on less then) for \texttt{>, <, >=, <=} comparisons between two registers

\begin{verbatim}
  slt   rd, rs, rt  # if rs < rt, rd = 1; else rt = 0
\end{verbatim}

An example:

\begin{itemize}
  \item branch if the first register operand is less than the second
    \begin{verbatim}
    slt   $8, $9, $10# $8 = 1 if $9 < $10; otherwise $8 = 0
    bne   $8, $0, L1  # branch to L1 if $8 = 1
    \end{verbatim}
\end{itemize}

Can use a pseudo-instruction as a shortcut:

A "pseudo instruction" is one inserted by the assembler but is not really implemented in hardware. For example:

\begin{verbatim}
blt   $8, $10, L1   # if $9 < $10 branch to L1
\end{verbatim}

Assembler automatically inserts:

\begin{verbatim}
  slt $at, $9, $10
  bne $at, $0, L1
\end{verbatim}

Other pseudo operations: \texttt{bgt, bge, ble, mult} (when 3 arguments are used).
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
```

- **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)

```
bne  $s0, $s1, Exit
[  5 ][ 16 ][ 17 ][ (Exit - PC+4)/4 ]
```
MIPS Jump Instructions

Jump instructions: unconditional transfer of control

\[ \text{j} \quad \text{target} \quad \# \text{jump go to the specified target address} \]

\[ \text{jr} \quad \text{rs} \quad \# \text{jump register go to the address stored in rs} \]
\hspace{1em} \text{(called an indirect jump)}

\[ \text{jal} \quad \text{target} \quad \# \text{jump and link go to the target address;} \]
\hspace{1em} \# \text{save PC+4 in } \$\text{ra} \]

\[ \text{jalr} \quad \text{rs, rd} \quad \# \text{jump and link register go to the address store} \]
\hspace{1em} \# \text{in } \text{rs; rd = PC+4, default rd is } \$\text{ra} \]

Examples:

\[ \text{jal procedureAddress} \] calls a procedure

\[ \text{jr } \$31 \text{ (or jr } \$\text{ra}) \] returns from a procedure

\[ \text{jr } \$8 \] implements a case statement

\hspace{1em} \bullet \text{ where the target addresses for the different cases are in a}
\hspace{1em} \text{table (jump address table)}

\hspace{1em} \bullet \text{$8$ contains one such entry}
J-type Format for Jumps

J-type format used for unconditional jumps

31 26
[ opcode ] [ address ]
25 0

- **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

\[ j \quad 10000 \]
\[ [ \ 2 \ ] [ \ 10000 \ ] \]
If/then/else Example

The C version

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

An assembly language version:

```assembly
# 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       # f = g + h
j Exit                 # jump out of the if

Else:
    sub $s0,$s1,$s2    # f = g - h

Exit:
```