### Control

- Instruction that potentially changes the flow of program execution
- MIPS conditional branch instructions
  - `bne $s4, $s3, LABEL`  
    \[ \text{goto LABEL if } $s4 \neq $s3 \]
  - `beq $s4, $s3, LABEL`  
    \[ \text{goto LABEL if } $s4 = $s3 \]
- Example

```
if (i == h) h =i+j;
```

#### MIPS unconditional branch instruction (i.e., jump)

- `j LABEL`
- Example
  - `i, f, g, and h` are in registers $s4, $s5, $s6, $s7`

```
if (i == h) f=g+h;
else f=g–h;
```

- Example
  - `bne $s4, $s7, ELSE`
  - `add $s5, $s6, $s7`
  - `j EXIT`
  - `ELSE:
    - `sub $s5, $s6, $s7`
    - `EXIT:`
    - `...`
Control

- We have beq and bne; what about branch-if-less-than?
  - We have slt

  \[
  \text{if } (s1<s2) \text{ } t0=1; \\
  \text{else } t0=0; \\
  \text{slt } t0, s1, s2
  \]

- Can you make a “pseudo” instruction “blt $s1, $s2, LABEL”?
- Assembler needs a temporary register to do this
  - $at is reserved for this purpose

\[
\begin{align*}
\text{slt} & \quad \text{at}, s1, s2 & \# & \text{$at==1$ when $s1<s2$} \\
\text{bne} & \quad \text{at}, 0, \text{LABEL} & \# & \text{$at!=0$ implies $at==1$}
\end{align*}
\]

In-class exercise with control

- Let’s write an assembly language program:
  - Print “Big endian” if machine is big endian, otherwise print “Little endian”

- We can do this by declaring some 4 bytes (word) in memory
- Then, load the word
- Check which which byte was put in the lowest position

- See inclass3.asm
Checking for endianness

```assembly
.data
endian: .byte 0x1,0x2,0x3,0x4
m_lit: .asciiz "Little endian!\n"
m_big: .asciiz "Big endian!\n"
.text
la $a0,m_lit       # assume little end
la $s0,endian
lw $s1,0($s0)
andi $s1,$s1,0x1   # select low bit
bne $s1,$0,lit_end # LSB=1: little endian
la $a0,m_big
lit_end: li $v0,4  
syscall
```

Addresses in Branches

- Branches use **PC-relative addressing**
- **Target address**: Location (address) that the branch goes to. 
  ```assembly
  beq $t0,$0,LABEL
  ```
- Target address computed as:
  - next PC = (current PC + 4) + offset × 4
  - thus, **LABEL** is how many instructions to go forward (positive offset) or backward (negative offset) from current location.
- Branch uses I-format
  - Two register source operands for comparison
  - 16-bit immediate (offset) to specify target address
Addresses in Branches

```assembly
addi $t0,$t1,-100
beq $t0,$0,LABEL
addi $s0,$t0,1000
addi $s1,$t1,1100
sub $t0,$s1,$s0
LABEL: li $v0,1
add $a0,$t0,$0
syscall
```

```
addi $t0,$t1,-100
beq $t0,$0,LABEL
addi $s0,$t0,1000
addi $s1,$t1,1100
sub $t0,$s1,$s0
LABEL: li $v0,1
add $a0,$t0,$0
syscall
```

LABEL $= +3 instructions
I-format encoding: 0x110000003
opcode: 000100
regs: 01000 ($t0), 00000 ($0)
Addresses in Branches

```
addi $t0,$t1,-100
beq $t0,$0, LABEL
addi $s0,$t0,1000
addi $s1,$t1,1100
sub $t0,$s1,$s0
LABEL: li $v0,1
add $a0,$t0,$0
syscall
sub $t0,$s1,$s0
bne $t0,$0, LABEL
li $v0,10
add $a0,$t0,$0
syscall
```

LABEL = +3 instructions
I-format encoding: 0x110000003
opcode: 000100
regs: 01000 ($t0), 00000 ($0)

```
add $a0,$t0,$0
syscall
```

LABEL = -5 instructions
I-format encoding: 0x1500fffb
opcode: 000101
regs: 01000 ($t0), 00000 ($0)
Addresses in Branches

- PC = PC + 4 + offset * 4
- Why the multiply by 4?
- How can the hardware do this easily?
- Offset can be negative – Offset
- Represented as two’s complement negative
- Complete calculation is:
  \[ \text{PC} = \text{PC} + 4 + (\text{sign-extend}(	ext{offset}) \ll 2) \]

J-format

- The address of next instruction is obtained from PC and the immediate value
  - Next address = \{PC[31:28], IMM[25:0], 00\}
  - Address boundaries of 256MB
- Example
  - j 10000
    
    | Jump | op | 26-bit immediate |
    |------|----|------------------|
    | 2    | 2500                      |
**Control**

- What about comparisons against constant?
  - Can we use “beqi”?? Why not?

- Comparison variants with an immediate
  - `slti $s0,$s1,16`
  - `addi $s0,$s1,-20` ($s0==0 when $s1==20)

  ```
  if (a > 20) { a = a + 2; }
  ```

  ```
  slti $at,$s0,21 # $at==1 when a<=20
  bne $at,$0,LAB # $at!=0 implies $at==1, skip
  addi $s0,$s0,2
  ```

  ```
  LAB: ...
  ```

  # target when a<=20

**Control**

- Loops
  - “While” loops
    - Example on page 74
  - “For” loops
While Loop
while (condition == true) { some work; }

use comparison, branch to exit, and jump to continue
generally written as:

loop: check condition
branch if condition is false to exit
some work
j loop

exit:

char *str = "Hello World!";
char *s = str;
while (*s != '\0') { printf("%c", *s); s = s + 1; }

la $t0,str
loop: lb $a0,0($t0)
beq $a0,$0,exit
li $v0,11 # print character
syscall
addi $t0,$t0,1
j loop

exit:  See mips12.asm
**Do While Loop**

```c
char *str = "Hello World!";
char *s = str;
/* note: this breaks with the empty string! */
do { printf("%c",*s); s = s + 1; } while (*s != '\0')
```

```
l a  $t0,str
l b  $a0,0($t0)  # start the loop
l o o p:  l i  $v0,11   # print character
          s y s c a l l
          a d d i  $t0,$t0,1
          l b  $a0,0($t0)
          b n e  $a0,$0,loop
```

See mips11.asm
In-class Example

- Write code to reverse a null-terminated string

Examples of control flow

- Weather programs
  - weather.c
  - weather.asm - illustrates if-then-else and while loop
  - weather2.asm - a slightly improved version (still if-then-else)
  - weather3.asm - illustrates a “computed goto” (switch)
  - weather4.asm - illustrates an algorithm change, using a table

- sam12.asm
  - convert a 32-bit number into hex characters, which are displayed with the OS print string service

- We’ll see many more examples!