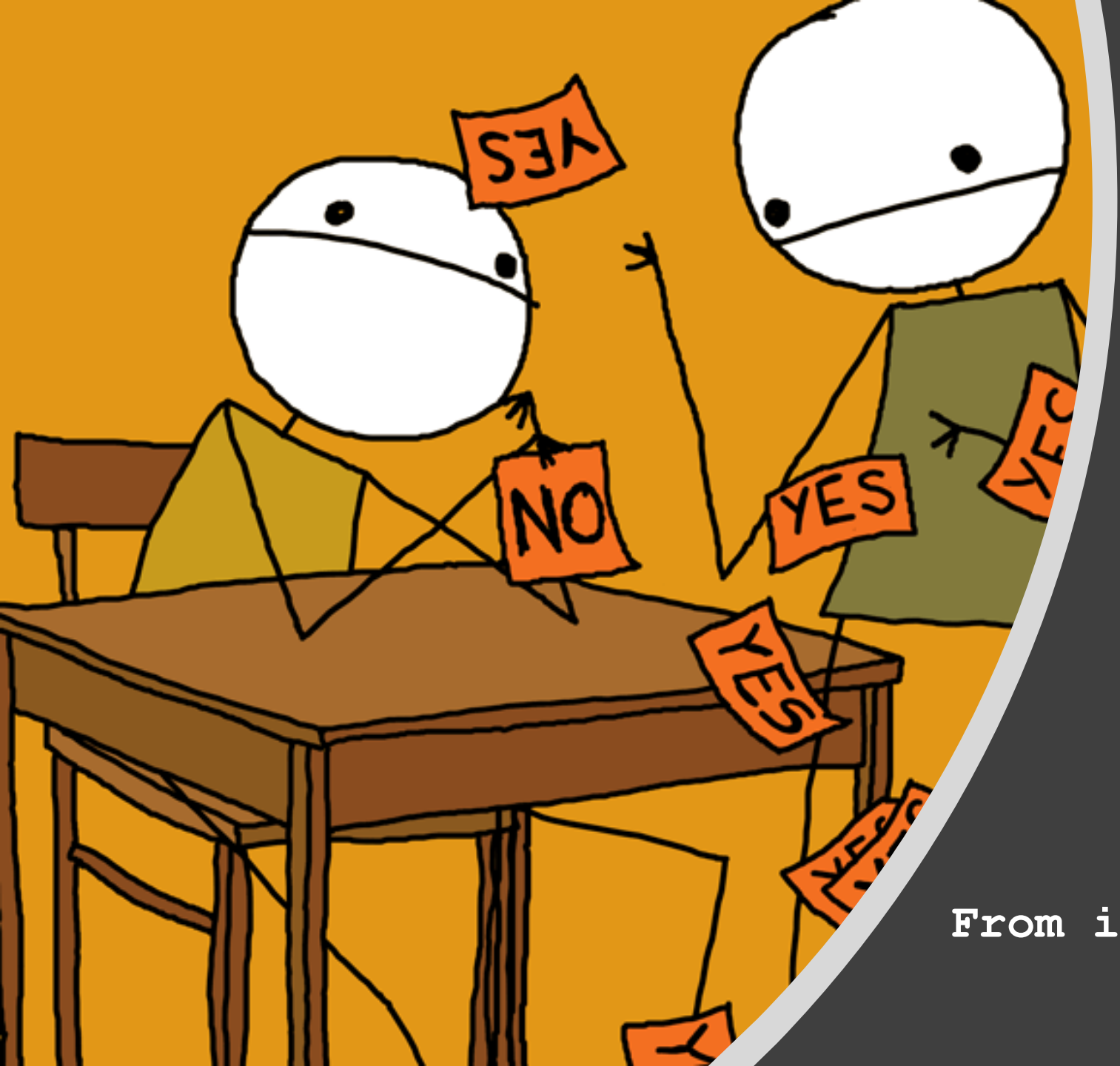
# Decisions in MIPS Assembly Language


- All instructions we've seen so far allow us to  manipulate data.
- To build a computer we must have the ability to make decisions.

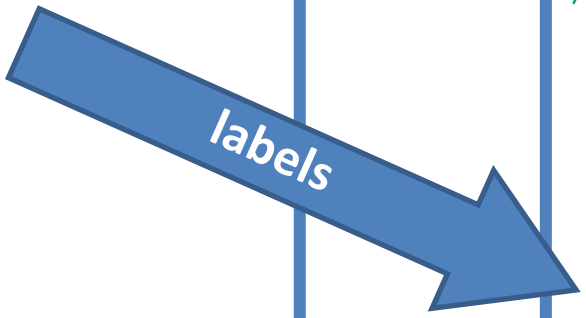



# Branches

From if-else/switch to assembly

# Conditional Statement in HLL

```
// if-else in C/Java
if (condition) clause
if (condition) {
  clause1
}
else {
  clause2
}
```




```
// C: Rewrite with goto
   if (condition) goto L1
   clause2
   goto L2
L1: clause1
L2:
```

Same meaning in C

No goto in Java

# Conditional Branches in MIPS

**Branch if (registers are) equal: `beq` reg1, reg2, label**

```
// C
if (reg1 == reg2)
    goto label1 ;
```

C to MIPS

```
# MIPS:
# go to label1 if $s1 == $s2
beq $s1 $s2 label1
```

**Branch if (registers are) not equal: `bne` reg1, reg2, label**

```
// C
if (reg1 != reg2)
    goto label1 ;
```

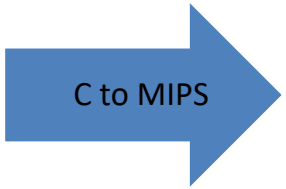
C to MIPS

```
# MIPS
# go to label1 if $s1 != $s2
bne $s1 $s2 label1
```

# Unconditional Branch

- **Jump Instruction**: Jump directly to a label

```
// C goto
goto label ;
```




```
# MIPS jump
j label
```

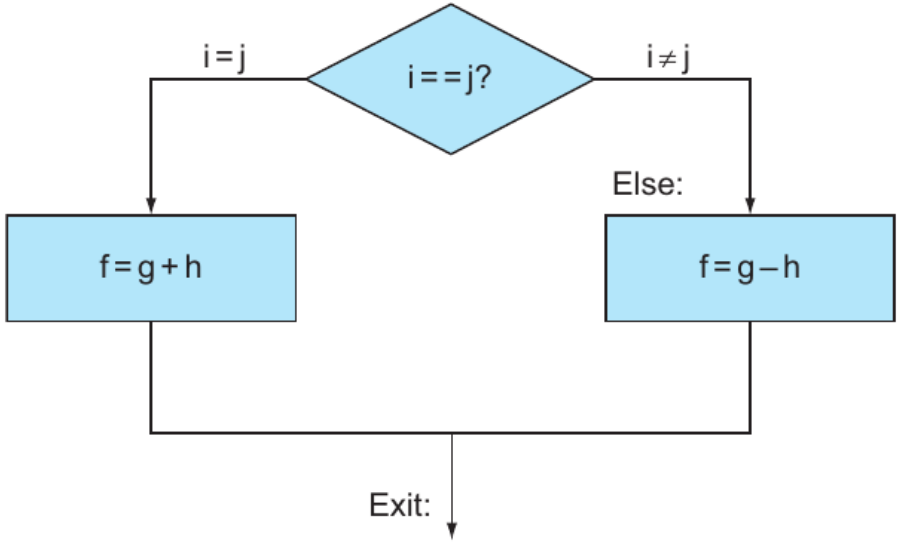
Technically, the following instruction is the same.

**There is an important difference. We will see in MIPS representation!**

```
# beq version
beq $0, $0, label
```

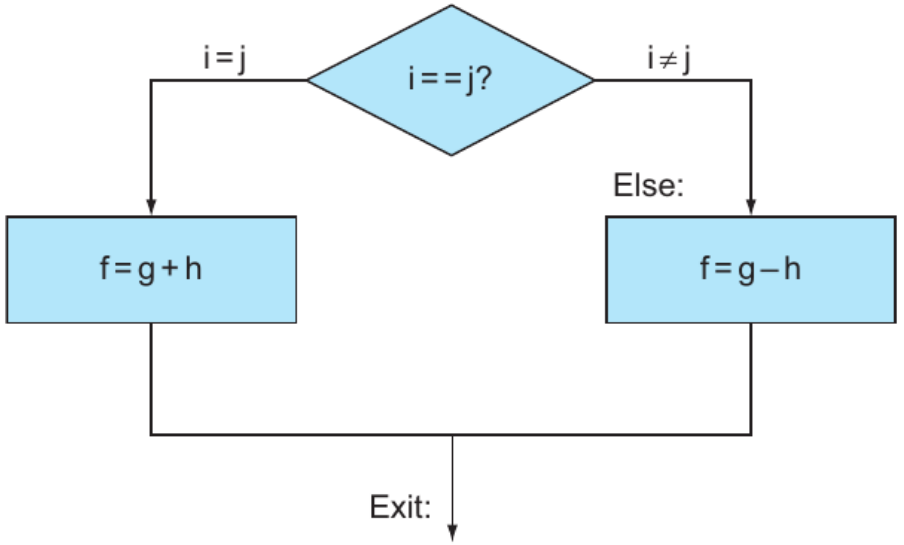
# Conditional Statement in HLL

```
// C and Java
if ( i == j ) {
     f = g + h ;
} else {
     f = g - h ;
}
```



# Compiling `if-else` into MIPS

```
// C and Java
if ( i == j ) {
     f = g + h ;
} else {
     f = g - h ;
}
```




***compiler*** automatically creates labels to handle decisions (branches).

| Registers | |
|---|---|
| $s0 | f |
| $s1 | g |
| $s2 | h |
| $s3 | i |
| $s4 | j |

```
# MIPS
      beq $s3, $s4, True    # branch i == j
      sub $s0, $s1, $s2     # f = g - h (false)
      j   Exit              # jump to Exit
True: add $s0, $s1, $s2     # f = g + h (true)
Exit:
```

# The Switch Statement in HLL

Choose among four alternatives depending on whether `k` has the value 0, 1, 2 or 3.

```
// Switch Statement
switch (k) {
   case 0: f=i+j; break ;
   case 1: f=g+h; break ;
   case 2: f=g-h; break ;
   case 3: f=i-j; break ;
}
```

```
// Rewrite it with if-else
if      (k==0) f = i + j ;
else if (k==1) f = g + h ;
else if (k==2) f = g - h ;
else if (k==3) f = i - j ;
```

# The switch-statement

- How would you code this in MIPS?

```
     bne  $s5, $0, L1      # if k != 0, branch to L1
     add  $s0, $s3, $s4    # if k == 0, f = i + j
     j    Exit             # end of case, exit
L1:  addi $t0, $s5, -1     # $t0 = k-1
     bne  $t0, $0, L2      # if k != 1, branch to L2
     add  $s0, $s1, $s2    # if k == 1, f = g + h
     j    Exit             # end of case, exit
L2:  addi $t0, $s5, -2     # $t0 = k - 2
     bne  $t0, $0, L3      # if k != 2, branch to L3
     sub  $s0, $s1, $s2    # if k == 2, f = g - h
     j    Exit             # end of case, exit
L3:  addi $t0, $s5, -3     # $t0 = k - 3
     bne  $t0, $0, Exit    # if k != 3, branch to Exit
     sub  $s0, $s3, $s4    # if k == 3, f = i - j
Exit:
```

# Loops

**Q: How did the programmer die in the shower?**

**A: He read the shampoo bottle instructions: Lather. Rinse. Repeat.**

# Loops in C and Assembly

HLL has three types of loops: while, do-while, for. Each can be rewritten as the other

MIPS: There are multiple ways to write a loop with conditional branch

# Loops in HLL: 3 ways

**Example: Sum of Series**

**sum = 1 + 2 + 3 + 4 + 5**

```
// while
int i = 1 ;
int N = 5 ;
int sum = 0 ;

while ( i<=N ) {
   sum += i ;
   i++ ;
}
```

```
// for
int i = 1 ;
int N = 5 ;
int sum = 0 ;

for (i=1 ; i<=N ; i++)
   sum += i ;
```

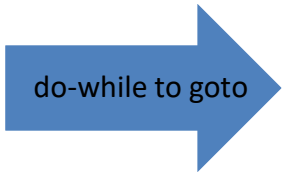
```
// do-while
int i = 1 ;
int N = 5 ;
int sum = 0 ;

do {
   sum += i ;
   i++ ;
} while (i<=N) ;
```

# From do-while to goto

**Example: Sum of Series**

**sum = 1 + 2 + 3 + 4 + 5**

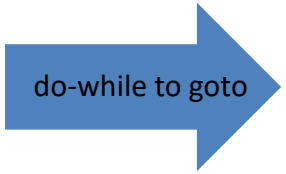
```
int i = 1 ;
int N = 5 ;
int sum = 0 ;
// do-while loop in C
do {
   sum = sum + i ;
   i = i + 1 ;
} while ( i != N ) ;
```




```
int i = 1 ;
int N = 5 ;
int sum = 0 ;

// Rewrite it with goto in C
Loop: sum = sum + i ;
      i = i + 1 ;
if ( i != N )
  goto Loop ;
```

# From do-while to MIPS assembly

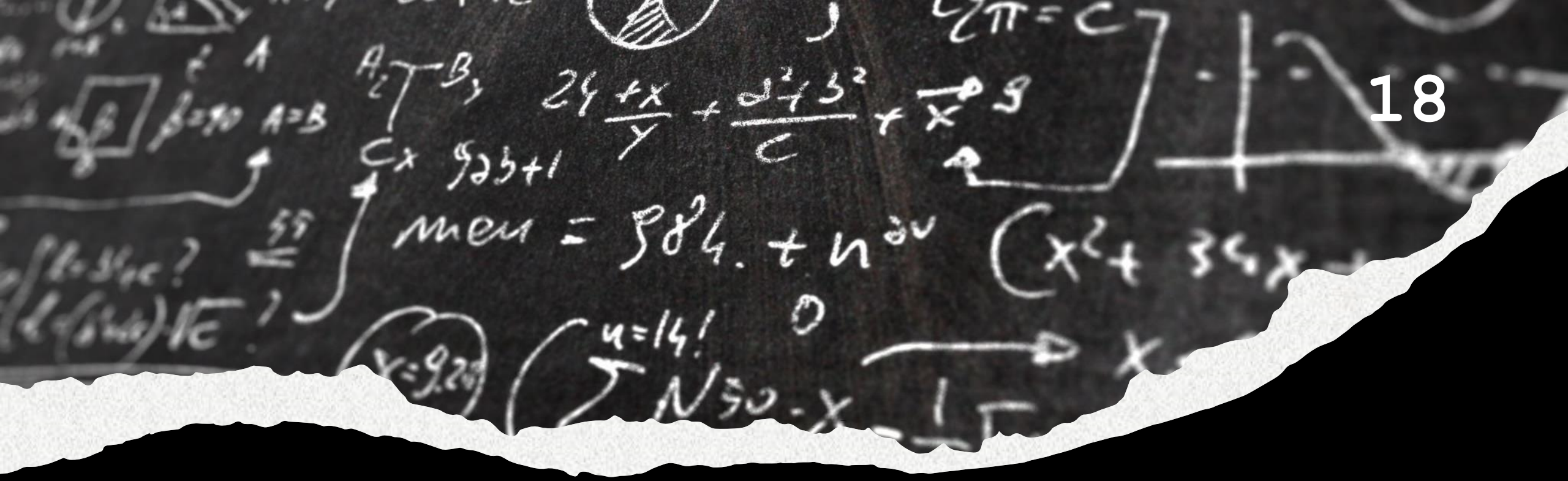
```
// do-while loop in C
do {
   sum = sum + i ;
   i = i + 1 ;
} while ( i != N ) ;
```




```
// Rewrite it with goto in C
Loop: sum = sum + i ;
      i = i + 1 ;
if ( i != N )
  goto Loop ;
```

| Registers | |
|---|---|
| $s1 | i |
| $s2 | N |
| $s3 | sum |

```
                # MIPS code
Loop: add  $s3, $s3, $s1   # sum = sum + i
      addi $s1, $s1, 1     # i = i + 1
      bne  $s1, $s2, Loop  # go to Loop if i != N
```



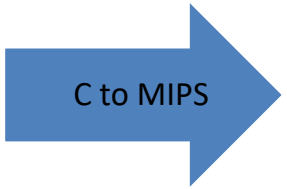
# Inequalities

So far, we only test equalities. What about inequalities?

# Inequalities in MIPS

- beq and bne only tested equalities

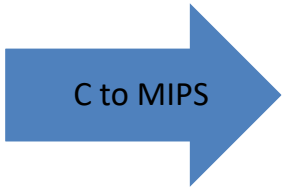
```
if ( i == j )
if ( i != j )
```




```
beq $s1 $s2 label1
bne $s1 $s2 label1
```

- We need to test <, <=, >, >=

```
if ( i <  j )
if ( i <= j )
if ( i >  j )
if ( i >= j )
```



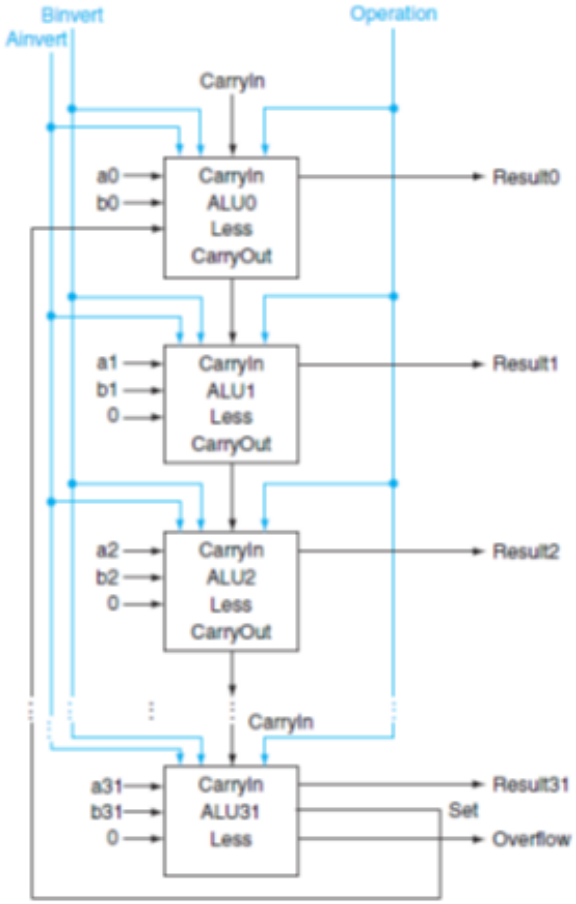
# Inequalities in MIPS: `slt`

- Syntax:

  **`slt reg1, reg2, reg3`**

  - Compare `reg2` and `reg3`
  - Place the result in `reg1`

```
// HLL style
if ( reg2 < reg3 )
   reg1 = 1 ;
else
   reg1 = 0 ;
```




Remember "**S**et on **L**ess **T**han" From ALU?

# Inequalities in MIPS: from *goto* to MIPS

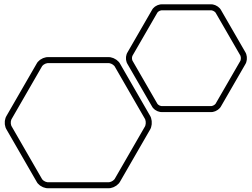
```
// C
if ( g < h )
    goto Less ;
```

| Registers | |
|---|---|
| $s0 | g |
| $s1 | h |
| $t0 | |

```
# MIPS: branch to Less if  $s0 < $s1
slt $t0, $s0, $s1    # if $s0<$s1 (g<h), $t0 = 1
bne $t0, $0,  Less   # branch if $t0 != 0
```

$0 always contains 0

bne and beq often use it for comparison after an slt instruction.



# Inequalities in MIPS

We have now seen slt for <, what about >, <= and >= ?

MIPS philosophy: **Simpler is Better!** Can we implement them using just slt and beq/bne

# Four Combinations of `slt` and `beq/bne`

```
slt $t0, $s0, $s1    # $t0 = 1 if $s0 < $s1 (g < h)
bne $t0, $0, Less    # if $t0 != 0, goto Less ( g < h)
```

```
slt $t0, $s0, $s1    # $t0 = 1 if $s0 < $s1 (g < h)
beq $t0, $0, Geq     # if $t0 == 0, goto Geq ( g >= h )
```

```
slt $t0, $s1, $s0    # $t0 = 1 if $s1 < $s0 (h < g)
bne $t0, $0, Gtr     # if $t0 != 0 goto Gtr ( g > h )
```

```
slt $t0, $s1, $s0    # $t0 = 1 if $s1 < $s0 (h < g)
beq $t0, $0, Leq     # if $t0 == 0, goto Leq ( g <= h)
```

# Pseudo-instructions for Inequalities

**Too complicated? Good News!**
**MARS translates pseudo-instructions into MIPS instructions**

**PSEUDOINSTRUCTION SET**

| NAME | MNEMONIC | OPERATION |
|---|---|---|
| Branch Less Than | `blt` | if(R[rs]<R[rt]) PC = Label |
| Branch Greater Than | `bgt` | if(R[rs]>R[rt]) PC = Label |
| Branch Less Than or Equal | `ble` | if(R[rs]<=R[rt]) PC = Label |
| Branch Greater Than or Equal | `bge` | if(R[rs]>=R[rt]) PC = Label |
| Load Immediate | `li` | R[rd] = immediate |
| Move | `move` | R[rd] = R[rs] |

# Inequalities with Immediates

# Immediates in Inequalities

- **Syntax:**

  **slti Result Source Immediate**

- Result = 1 if Source < Immediate, or 0 otherwise
- slti is the immediate version of slt

```
// C
if ( g >= 1 )
   goto Loop ;
```

```
# MIPS
slti $t0, $s0, 1     # $t0 = 1 if $s0 < 1
beq  $t0, $0,  Loop  # goto Loop if $t0 == 0
```

# Unsigned Immediates in Inequalities

- Syntax:

```
sltu  Result Source1 Source2
sltiu Result Source Immediate
```

- Set result to 1 or 0 depending on unsigned comparisons

```
# MIPS
sltu  $t0, $s0, $s1    # $t0 = 1 if $s0 < $s1
sltiu $t0, $s0,  5     # $t0 = 1 if $s0 < 5
```

# Immediates in Inequalities

```
slt   $t0, $s0, $s1
sltu  $t1, $s0, $s1
```

**Assume**

**$s0 = 0xFFFF FFFA**

**$s1 = 0x 0000 FFFA**

**What is value of $t0, $t1?**

# Review and More Information

- High-level languages
  - Conditional statement: `if-else, switch`
  - `Loop: while, do-while, for`
- MIPS uses conditional branches:
  - `Equality: beq, bne`
  - Inequality: `slt, slti, sltu, sltiu`
  - Jump: `j`
- Textbook Section 2.7
- **Try it out in MARS**