Decisions in ly Language

ions we've seen so far manipulate data.

- To mputer we must have the ability to make decisions.

Decisions in High-Level Languages

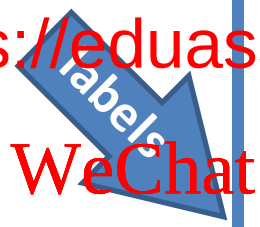
- Conditional Statements : `if`, `if-else`, `switch`
- Loops: `while`, `do while`, `for`
- Equality and Inequal `>` `<=` `>=`

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
          se2
                2
          se1
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
    $s2 label1
```


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

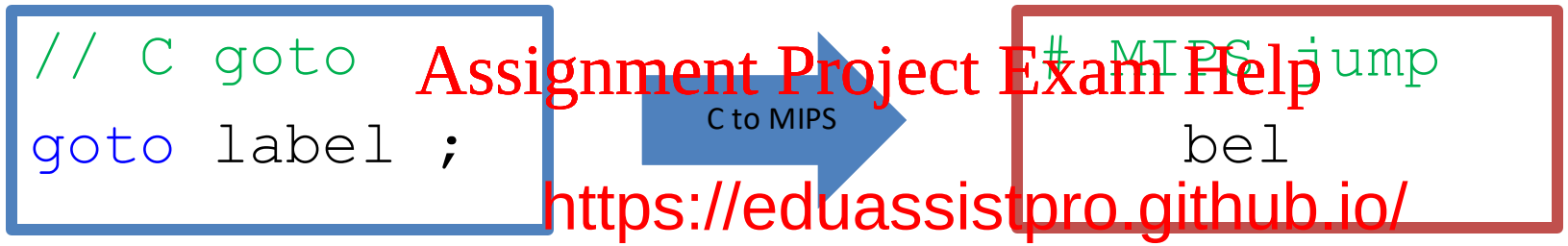
```
// C
if (reg1 != reg2)
    goto label1 ;
```

C to MIPS

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

Unconditional Branch

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



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 ;
}
```

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:
```

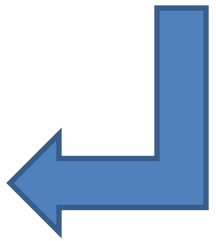
compiler automatically creates labels to handle decisions (branches).

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 ;
       se 1: f=g+h; break ;
         sh 2: f=g-h; break ;
              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 ;
```



Loops

Q: How did the programmer die
wer?

A the shampoo bottle
. Lather. Rinse.
Repeat.

Loops in C and Assembly

HLL has three types of loops: while, do-w can be rewritten a

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

Loops in HHL: 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
in
in
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 ) ;
```

do-while to goto

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

do-while to goto

```
// 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 )
```

C to MIPS

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


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

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

C to MIPS

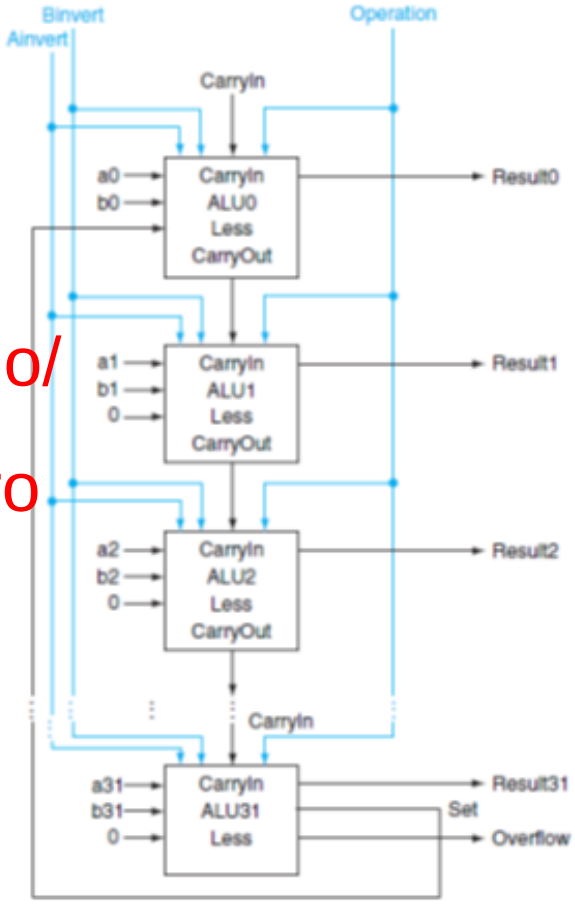
Inequalities in MIPS: `slt`

- Syntax:

  `slt reg1, reg2, reg3`

  - Compare reg2 a
  - Place the resu


```
// 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

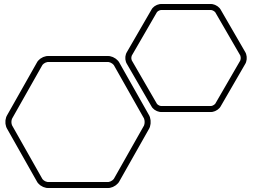
```
// C
if ( g < h )
    goto Less ;
```


Registers	
$s0	g
$s1	h
$t0	

```
# MIPS: branch to Le          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                    $s1 (g < h)
beq $t0, $0, Geq                  Geq ( g >= h )
```



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

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

Inequalities with

Immediates in Inequalities

- **Syntax:**

  slti Result Source Immediate

- Result = 1 if Sourc 0 otherwise
- slti is the immedia


```
// 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
sltui Result                 e
```


- Set result to 1 or 0 ned comparisons

```
# MIPS
slti  $t0, $s0, $s1    # $t0 = 1 if $s0 < $s1
sltui $t0, $s0,  5     # $t0 = 1 if $s0 < 5
```

Immediates in Inequalities

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

Assume

FFFF FFFA

0000 FFFA

Wh e of $t0, $t1?

Review and More Information

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