

Conditionals

Conditional Statements

Assembly does not have `if` statements

However, we have seen that we can move around in assembly code using branches and jumps

Only branches allow us to change behavior based on condition, so they will be key to implementing `if` statement

Conditional Statements

When we do not have something we want directly in assembly (e.g., conditionals, loops, functions), we establish a pattern of assembly code we can use in its place

You can think in high-level code and convert your code to assembly using these patterns

Remember – high-level code is always turned into assembly behind the scenes. Anything you can do in high-level code, you can do in assembly

Conditional Statements

We will see how to convert these two ideas to assembly:

`if`

`if/else`

Once you can do that, you can figure out `else if`, `switch`, and others

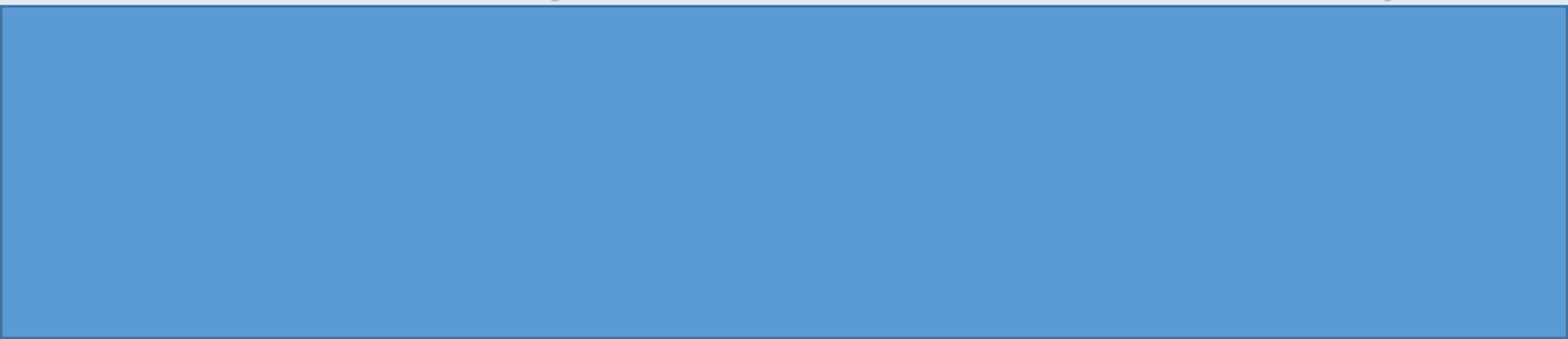
High-Level Code

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

f = f - i;
```

MIPS Assembly Code

```
## $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
```



High-Level Code

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

```
    f = g + h;
```

```
f = f - i;
```

MIPS Assembly Code

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j  
    bne $s3, $s4, L1    # if i != j, skip if block  
    add $s0, $s1, $s2    # if block: f = g + h  
L1:  
    sub $s0, $s0, $s3    # f = f - i
```

Implementing Conditionals

Assembly tests opposite case of high-level code!

Implementing Conditionals

General pattern is

```
if (not (high-level condition)): branch
```

```
[code if branch not taken –  
  i.e, high-level condition  satisfied]
```

```
target:
```

```
...rest of code...
```


Implementing Conditionals

Assembly tests opposite case of high-level code!

In previous example, used bne instead of beq
(i != j instead of i == j)

Could be written any number of ways, but testing the opposite condition usually leads to much cleaner assembly code

What if we do not follow this advice?

Don't do this

High-Level Code

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

f = f - i;
```

MIPS Assembly Code

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
        beq $s3, $s4, then    # if i == j, goto then
                                # need to skip "then"
                                # need to skip "then"
j either_way

then:
        add $s0, $s1, $s2    # if block: f = g + h

either_way:
        sub $s0, $s0, $s3    # f = f - i
```

If-Else

Write assembly for the following high-level code. Note that the final line is now in an `else` block

High-Level Code

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

else
    f = f - i;
```

If-Else

High-Level Code

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

MIPS Assembly Code

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
    bne $s3, $s4, else    # if i != j, branch to else
    add $s0, $s1, $s2     # if block: f = g + h
    j    L2              # skip past the else block
else:
    sub $s0, $s0, $s3     # else block: f = f - i
L2:
```

If-Else

Still testing the opposite case!

Magnitude Comparison

Up to now we've looked at equality branching

How can we instead compare based on inequality?

Use `slt` (set if less than)

Magnitude Comparison

High-Level Code

```
if (i < j)
    f = g + h;

f = f - i;
```

MIPS Assembly Code

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
    slt $t0, $s3, $s4    # is i < j?
    beq $t0, $0, L2      # if i >= j, skip if block
    add $s0, $s1, $s2    # if block: f = g + h

L2:
    sub $s0, $s0, $s3    # f = f - i
```

Magnitude Comparison

Still testing opposite case!

(Hopefully you are tired of hearing that by now)

Magnitude Comparison

This pattern will be helpful when we consider loops

```
while (i < 10) { ... }
```