Control flow code in MIPS

Chapter 2C

If statement

Notice that the Boolean statement of if-statement has been complemented. The reason is assembler executes the code line-by-line starting from if-block and in the case of else statement, we want to skip executing if-block, therefore executing only else block.

```
if ($t0 < $t1) {
    // code block #1
} else {
    // code block #2
}
```

```
if:
    bge $t0, $t1, else
    #
    # code block #1
    #
    b end_if
else:
    #
    # code block #2
    #
end_if:
```

Else if statement

Notice that both Boolean statement have been complemented.
We can think of else-if block as an another if statement (nested) inside outer else block.

```
if ($t0 < $t1) {
    // code block #1
} else if ($t0 == $t1) {
    // code block #2
} else {
    // code block #3
}
```

```
if:
    bge $t0, $t1, else_if
    #
    # code block #1
    #
    b end_if
else_if:
    bne $t0, $t1, end_else_if
    #
    # code block #2
    #
    b end_if
end_else_if:
    #
    # code block #3
    #
end_if:
```

While loop

Notice that the Boolean statement have been complemented here.
The reason is that we have an unconditional branch to beginning of the block that will only be executed when Boolean statement is false (complement is false).

```
while ($t0 < $t1) {
    // code block
}
```

```
while:
    bge $t0, $t1, end_while
    #
    # code block
    #
    b while

end_while:
```

Do-while loop

Notice that the Boolean statement <u>have not</u> been complemented here.
The reason to not complementing the Boolean statement is if the Boolean statement is false, as assembly executes the code line-by-line, we automatically stop executing the code block.

```
do {

    // code block

} while ($t0 < $t1)
```

```
do_while:
    #
    # code block
    #
    blt $t0, $t1, do_while

end_do_while:
```

For loop

Notice that the Boolean statement have been complemented here.
Also, incrementing the counter (i.e. $t0) should be done after the code block in order to simulate for-loop control flow correctly.

```
for ($t0 = 0; $t0 < $t1; $t0++) {
    // code block
}
```

```
    li $t0, 0
for:
    bge $t0, $t1, end_for
    #
    # code block
    #
    addi $t0, $t0, 1

    b for

end_for:
```

Miscellaneous notes

Difference between branch and jump:

- Branches allow for conditions. But allowing for conditions takes up more bits in the instruction. Therefore, a branch's address is only 2^16 bits and only allows you to branch 2^15−1 instructions backward or 2^15 instructions forward.

- A jump is unconditional and the bits saved by leaving out the condition can be used for the address. A jump allows for a 26 bit address and so can jump much further in the code than a branch. At the expense of not being conditional.

Difference between branch unconditionally (b) and jump unconditionally (j):
Branches (b) use a PC-relative displacement while jumps (j) use absolute addresses. The distinction is important for position-independent code. Also, only jumps can be used for indirect control transfer (jr, using a register value). **In short, use only (b) for if and loop statements and avoid using (j) in your code.**

Common mistakes

- **There is no "immediate" branch instruction. For example, `beq $t0, 2, else`**
  It is technically a **syntax error**, however, QtSPIM will resolve the issue by replacing it with:

```
li $at, 2   # which itself is a macro for: ori $at, $0, 2
beq $t0, $at, else
```

  Programmer should not access `$at` register. It is an assembler temporary register and it is generally reserved for use in pseudo-instructions (macro instructions). We need to load constant value 2 into a temporarily registers and then use it in branch. Corrected code:

```
li $t9, 2
beq $t0, $t9, else
```

- **There is no "immediate" multiplication, division and subtraction. For example:**

```
# erroneous code        -|-  resolved by QtSPIM

    mul $t0, $t0, 2          # li $at, 2
                             # mul $t0, $t0, $at

    div $t0, $t0, 2          # li $at, 2
                             # div $t0, $t0, $at

    sub $t0, $t0, 2          # addi $t0, $t0, -2
```