CSC258H Lab 9: Control Flows

1 Introduction

When programming in a high-level language like Python, Java, or even C, we use three main control-flow constructs: branches, loops, and function calls. Last week, we learned how to create programs in assembly for the MIPS architecture, but these programs did not include any "control". This week, we will write programs with branches and loops to see how these constructs are implemented in assembly. When we finish this lab, we hope you will have a better appreciation for the structures a high level language provides!

Like last week, take your time on this lab and ask your neighbors or the TAs questions whenever you see something you don't understand. We will be learning how to write functions next week, so it's important that you be comfortable with the syntax for system calls and branch and memory instructions. If you need a reference, please check the extra resources posted on the course webpage. I've found Larus's guide particularly useful for a high level overview.

This week's lab assumes that you completed all of the material from last week. Make sure to complete the tasks in last week's lab before lab time this week.

Required Submissions: This lab requires the submission of two assembly files ("`lab9a.s`", "`lab9b.s`", and "`lab9c.s`") and a lab report ("`lab9report.pdf`") to MarkUs by **Tuesday, March 30, 10:00 PM**. Please read the syllabus regarding the late policy. All submitted work must be completed **individually**.

2 If-Else

An If-Else statement (or *branch* or *conditional statement*) is a control structure that creates conditionally executed code. The structure relies on a *predicate* – an expression that evaluates to a Boolean value – `True` or `False`. The *if* is always followed by a block that is executed when the predicate is `True`. This is the *Then* block. The Then block is optionally followed by an *Else* block that is executed if the predicate is `False`.

In assembly, If-Else is implemented using `labels` and `branch` operations. If necessary, the predicate is simplified using normal arithmetic operators. Then, it is evaluated using a `branch`. The `branch` uses a label to specify what the next instruction to execute should be if the predicate evaluates to `True`. If it evaluates to `False`, then the next instruction is executed (as normal). Here is an assembly implementation of a branch:

```
# if x < 5 {
#     y = 1
# }
# else {
#     y = 2
# }

IF:                     # This label isn't required but is added for clarity.
    addi $t1, $t0, -4   # Prepare to evaluate x - 4 <= 0.
    bgtz $t1, ELSE      # Branch to the label ELSE if the predicate is False.
```

```
THEN:                   # This label isn't required but is added for clarity.
    li $t2, 1
    j DONE

ELSE:
    li $t2, 2

DONE:                   # This label marks the end of the If-Else.
```

Note that the predicate is evaluated in an odd way. Branches in MIPS can only compare if two values are equal (beq) or not equal (bne), if one value is $<= 0$ (blez), or if one value is > 0 (bgtz). Some arithmetic has to be done to make most comparisons into comparisons to zero. Furthermore, the branch in the example checks whether the predicate is `False` and then branches to the `Else` block. If you want to change the branch to be checking whether the condition is `True`, you'll need to swap the positions of the THEN and ELSE blocks. To better understand the flow, use the flowchart technique that we used in the lecture (see Week 9 lecture slides).

TODO #1: Make a copy of your program from last week's lab, which prompted the user for numbers. Name the file `lab9a.s`. Modify that program to check whether the number provided by the user is an odd number or an even number. If the user input is an odd number, print "THIS IS ODD"; otherwise print "THIS IS EVEN". (*Hint*: use `andi` to check if a number is odd or even.)

3 Loops

Loops are very similar to branches. If we start with a branch with a Then block and no Else block, then the difference is that we name the Then block the loop's *body* and may execute it multiple times. This means that the bottom of the Then block is an unconditional branch back to the top of the loop. For example:

```
# x = 0
# while x < 5 {
#     x = x + 1
# }

LOOPINIT:               # Many loops have an initialization section.
    li $t0, 0
WHILE:                  # The loop checks the condition, then evaluates the body.
    addi $t1, $t0, -4
    bgtz $t1, DONE

    addi $t0, $t0, 1
    j WHILE
DONE:                   # This label marks the end of the loop.
```

This code breaks the loop into three parts. First, the initialization block sets up loop variables. Second, the loop's predicate is evaluated, and if the predicate is false, control jumps to the code

after the loop. Third, the loop body is evaluated, and an unconditional branch is made to the top of the loop.

TODO #2: Make a copy of `lab9a.s` and name it `lab9b.s`. Modify your program so that it repeatedly asks the user for input until the user provides an even number; or if the user has entered odd numbers for N times, then print "TOO MANY TIMES" and exit. Make N a parameter in the `.data` section with value 5. (*Hint:* Use the `.word` keyword to create space for an integer. Use `lw` to load the word into a register.)

4 TODO #3: Product

Write a new program `lab9c.s` that first asks the user for an integer N. N represents the number of integers to be multiplied. After getting a legal value N, the program asks for an integer N times and then prints their product. (You may, for simplicity, assume that the user only provides positive integers and their product is small enough to be represented by 32 bits.)

5 Lab Report

Include the following in your lab report named "`lab9report.pdf`".

1. Your name and student number.
2. A screenshot of your MARS console showing that `lab9a.s` is working correctly.
3. A screenshot of your MARS console showing that `lab9b.s` is working correctly.
4. A screenshot of your MARS console showing that `lab9c.s` is working correctly.

6 Summary of TODOs

Below is a short summary of the steps to be completed for this lab:

1. Before the lab, read through the handout and get familiar with the procedure.
2. Implement the If-Else program that checks parity.
3. Implement the Loop program that promotes until getting an even number.
4. Implement the Product program, and show it to your TA.
5. Complete the lab report.
6. Submit `lab9a.s`, `lab9b.s`, `lab9c.s`, and `lab9report.pdf` to MarkUs before the deadline.

Evaluation (3 marks in total):

- 1 mark for the code and report for If-Else;
- 1 mark for the code and report for Loop;
- 1 mark for the code and report for Product.