CS 382
Fall 2024

Computer Architecture
and Organization

Lab 5

1 Task 1: Simple Conditionals

The first task of this lab is to write a program that check that if the given three sides can form a right triangle. A right triangle is the one that has an angle of 90 degrees, and can be verified by the following formula:

$$c^2 = a^2 + b^2$$

where a, b, and c are the lengths of three sides of the triangle.

The data segment is declared as follows:

```
.data
side_a:  .quad    3
side_b:  .quad    4
side_c:  .quad    5
yes:     .string  "It is a right triangle.\n"
len_yes: .quad    . - yes  // Calculate the length of string yes
no:      .string  "It is not a right triangle.\n"
len_no:  .quad    . - no   // Calculate the length of string no
```

where you can assume `side_c` is always the longest side. If it is a right triangle, you'd need to print the string `"It is a right triangle.\n "`; otherwise `"It is not a right triangle.\n "`.

To print a string, you might want to review and borrow the code from lab 2.

Requirements

- **Note** your code is a **complete** assembly program (not just a sequence of instructions). It should be able to assemble, link, and execute without error and warnings. When executed, the program should finish without problems. If your code cannot assemble, you get no credit – this is the same as C programs that cannot be compiled;
- `MUL` instruction can be used for multiplications;
- You can use either `CBZ` / `CBNZ` or `B.EQ` / `B.NE` for branching;
- Avoid using registers X29 and X30;
- You have to put comments on each line of instruction;
- Put your name and honor code pledge at the top of your code in comments.

2 Task 2: More Conditionals

This task is to get familiar with other conditional branching instructions such as `B.LE`. You are given the starter code as follows (it can also be downloaded from Canvas):

```
.text
.global _start
.extern scanf
```

```

_start:
    ADR   X0, fmt_str   // Load address of formated string
    ADR   X1, left      // Load &left
    ADR   X2, right     // Load &right
    ADR   X3, target    // Load &target
    BL    scanf         // scanf("%ld %ld %ld", &left, &right, &target);

    ADR   X1, left      // Load &left
    LDR   X1, [X1]      // Store left in X1
    ADR   X2, right     // Load &right
    LDR   X2, [X2]      // Store right in X2
    ADR   X3, target    // Load &target
    LDR   X3, [X3]      // Store target in X3

    /* Your code here */

exit:
    MOV   X0, 0         // Pass 0 to exit()
    MOV   X8, 93        // Move syscall number 93 (exit) to X8
    SVC   0             // Invoke syscall

.data
    left:    .quad     0
    right:   .quad     0
    target:  .quad     0
    fmt_str: .string   "%ld%ld%ld"
    yes:     .string   "Target is in range\n"
    len_yes: .quad     . - yes  // Calculate the length of string yes
    no:      .string   "Target is not in range\n"
    len_no:  .quad     . - no   // Calculate the length of string no
```

The code given to you calls C library function `scanf()` to receive three signed integers from keyboard (seperated by space or newline). The first and second integers represent a numerical range `(left,right)` **(exclusive)**, while the third integer represents a target number. Your code should check if `target` is inside the range. If so, it should print the message labeled as `yes`; otherwise `no`. You can assume `left` is always **smaller** than `right`.

Again, to print messages, you'd need to invoke system calls learned in lab 3.

Because we used C library function here, during linking, you'd need flag `-lc` in the command. Related information can be found in Appendix B.2.3.3 in the textbook.

Requirements

- **Note** your code is a **complete** assembly program (not just a sequence of instructions). It should be able to assemble, link, and execute without error and warnings. When executed, the program should finish without problems. If your code cannot assemble, you get no credit – this is the same as C programs that cannot be compiled;
- Your code should complete the task successfully without errors. Please do check edge cases;
- You have to put comments on each line of instruction;
- Put your name and honor code pledge at the top of your code in comments.

3 Grading

The lab will be graded based on a total of 10 points, 5 for task 1 and 5 for task 2. The following lists deductibles, and the lowest score is 0 – no negative scores:

- ► Task 1:
  - **-5:** the code does not assemble, or the program terminates abnormally/unsuccessfully;
  - **-5:** the code is generated by compiler;
  - **-5:** the code does not attempt the task;
  - **-5:** the code cannot be explained clearly in person;
  - **-5:** no output is printed;
  - **-5:** not using conditionals;
  - **-3:** the right triangle checking result is wrong;
  - **-1:** one or more instructions is missing comments;
  - **-1:** no pledge and/or name.
- ► Task 2:
  - **-5:** the code does not assemble, or the program terminates abnormally/unsuccessfully;
  - **-5:** the code is generated by compiler;
  - **-5:** the code does not attempt the task;
  - **-5:** the code cannot be explained clearly in person;
  - **-5:** no output is printed;
  - **-5:** not using conditionals;
  - **-1:** 1 point off for each of the following test cases:
    - `-5 6 3`;
    - `-5 10 -30`;
    - `-6 -2 0`;
    - `3 4 3`;
    - `10 30 30`;
    - `-20 20 -20`;
  - **-1:** one or more instructions is missing comments;
  - **-1:** no pledge and/or name.

Earlybird Extra Credit: 2% of extra credit will be given if the lab is finished by Wednesday 11:59PM EST (1 day before the lab deadline). For specific policy, see syllabus.

Attendance: check off at the end of the lab to get attendance credit.

Deliverable

Assembly code for both tasks in two separate `.s` files. No need to zip all files; just submit both files separately on Canvas.