

CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Homework 2

Name (Print): _____ RHIT Username: _____

Put your answers in the spaces reserved below the questions. **Do not** write your answers on **separate sheets of paper**.

1. (8 points) Assume **x** is a variable of type pointer (i.e., an address) that points to one-byte unsigned number. Further, assume **b** is a variable of size **4 bytes** and is stored at memory address **0xA80**.

Suppose these variables are given values as follows.

b = 0x2F552;

x = 0xA81;

Hint: This problem is about how to write data in memory with different endianness. You have to pad zeros to the number to make it a 4 bytes.

- 1.1 Suppose the system is **big-endian**. Draw a diagram (as an array, with addresses as indices) of memory space where **b** is stored and their contents, and depict **x** as a pointer (arrow) to the appropriate byte of **b**. What is the (hexadecimal) value that **x** points to?

- 1.2 Now suppose the system is **little-endian**. Draw a new picture, contrasting with part (a). What is the (hexadecimal) value that **x** points to?

2. (4 points)

2.1 (2 points.) Suppose register `x0` holds the address of an `int`. Which of the following lines correctly load the `int` into a register? [Circle all correct answers.]

(a) `ldr w0, [x0]`

(b) `ldr w1, [x0]`

(c) `ldr x0, [x0]`

(d) `ldr x1, [x0]`

2.2 (2 pts) The following assembly function takes as input the address of an integer. The function should increment that integer (increase its value by 1).

However, this function **has a tricky bug** in it. When tested on a little-endian system (like our VMs) the function appears to work. But when tested on a big-endian system it does the wrong thing. The bug is a simple mistake that beginners often make.

Your task: Edit / mark up / change the code below to fix the bug.

```
1 // input: x0 - address of an integer
2 increment:
3     ldr x1, [x0]           // fetch the integer
4     add x1, x1, #1         // increment it
5     str x1, [x0]           // store the result
6     ret                   // integer is now 1 bigger
```

3. (36 points) Assembly Coding:

- On your Linux, pull changes to your Git repository.
- Complete and test all the functions in the `problems.s` file in the `homework2` subdirectory. You will need to write appropriate ARM assembly code, using a text editor such as `vim`.
 - The comments tell you what needs to be completed.
 - Use `make test_asm` (or just `make`) to compile your homework and the tests.
 - Once built, you can run the `test_asm` executable using the command: `./test_asm`.
 - Refer to the ARM handout and ARM examples on the schedule page for assembly instruction details.
 - Debugging tips:
 - * If you get an error like *file format not recognized*, use `make clean` to clean up your old non-working files, then run `make` again. If you still get the error, there's likely a bug in your assembly code.
 - * If you see error messages like **segmentation fault** or **core dump**, it is most likely caused by an attempt to access memory (`ldr` or `str`) with an invalid address.
- When finished, submit your modified `problems.s` to **Gradescope**. To do this, first use `git` on your VM to add, commit, and push your code to your Git repo. Then in your local clone of the repo, use `git` to pull the changes. Finally, upload the file from your local filesystem to Gradescope. (See Lab 1 instructions for help with the `git` commands.)