UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE12/L: COMPUTER SYSTEMS AND ASSEMBLY LANGUAGE

Lab 3: Decimal Converter
Worth 70 points (65 lab + 5 report)

Lab Objectives:

Now that we are moving into programming, it is important for you to focus on good programming practices. When programming in assembly this is especially true as assembly is not a pretty or easily readable language. You will need to rely on clear and useful comments to make your code easy to read and understand.

Having a clear plan for your program is essential in any language, but this is especially true in a low-level language like assembly. Thus, for all programs in this class, plan out your program. *Write your flowchart or pseudo-code BEFORE your write your code!* Once a good plan is created, mapping your solution to assembly code is much, much easier.

This first lab is designed to get you acquainted with assembly and MARS simulator. Future labs will expect much more assembly code.

NOTE: There are syscalls that can make this lab substantially easier. Do NOT use them, you will receive 0 points for doing so. Specifically they are 5 and 35.

Part 1: Read and understand the program requirements

In this lab, you will design and write an MIPS program that implements the following:

- The MARS simulator will put the number to convert as a string into memory. The starter code given stores the address of that string in *$s0*.
  - The input will be valid integer that fits within a 32-bit 2's complement number.
- Your program will first print out the string of the number.
  - That line must start with the prefix "Input Number: ".
- It will then convert this number to binary allowing for negative numbers.
- After the number is converted it will print the number in binary.
  - That line must start with the prefix "Output Number: ".
- The program then exits.

Here is an example of what two executions of the program might look like in the console (yours can have a different welcome message but should have the same input/output prefixes):

```
Welcome to Conversion.
Input Number: 452
Output Number: 00000000000000000000000111000100
-- program is finished running --


Welcome to Conversion.
Input Number: -10389
Output Number: 11111111111111111101011101101011
-- program is finished running --
```

This code might seem simple for a high level language but you will have to handle it yourself in assembly. You will have to read in the number as ASCII and convert as an example.

Part 2: Design your Program

Now that you understand the specifications, it's time to design your program. We will not be giving you the specifics of how to do so but an overview of each task. Think about how you would perform each task and design your program to do so.

Printing Input String

You need to print out the input string, something the starter code already does for the welcome message.

Converting String to Integer

Your input integer is stored as an ASCII string with a NULL termination character. You will need to iterate over that string character by character and convert into a binary number as you go. Do not forget about the negative sign. It is easiest to set a flag and use the additive inverse after the string is converted to a positive integer. This is much easier than most input as we are not allowing invalid input for this lab.

Printing Integer in Binary

To print out the integer in binary you will need to use bitmasks. Review how bitwise "*AND*" works along with the shift operator to get an idea on how to do so. You will need to process each bit individually and print out a '0' or '1' for each. An example of using `syscall` to print out an individual character is shown below.

```
li $a0,0x4D  # syscall uses a0 for data, we load 'M' into a0
li $v0, 11 # 11 is print char for syscall
syscall # actually perform the syscall
```

Part 3: Implement your program in code:

Now that you have designed your program, it is time to implement it in assembly. Do not attempt to write the whole program at once and then attempt to debug it, it will not go well. Test each part well before moving on to the next.

We have given you BaseFile.asm. This file will load the string address into $*s0*, print a message, and exit the program. Use this as the basis for your lab, do not forget to rename it.

MARS Simulator

To use the IDE correctly you will need to ensure a few settings are correct.

- Settings -> Program arguments provided to the MIPS program must be set. This allows for the input of the string. A box at the top of the text segment will appear allowing you to enter a number.
- Settings -> Delayed Branching must be off. This should be the default setting but your program will not work if it gets turned on.

Tips for Readable Assembly code:

- Use comments to break down the elements of your code. While it is possible to have too many comments it is far more likely that there will be too few.
- Use whitespace (tabs and enters) to show your code's structure. Use labels that accurately describe the purpose of that label or variable.

Debugging Tips:

Learn to use the MARS simulator's debugging tools. In particular, spend some time playing with breakpoints. The tool bar shown below is very powerful:

Most important are the first three buttons. From left to right they are:

- Assembles the program and determines errors in the assembly itself, not the logic written.
- Runs the program. The bar to the right of the toolbar allows you to slow down the rate of instruction evaluation such that individual instructions can be observed.
- Single steps the program, evaluating one instruction. Note that only one instruction of pseudo-op will be evaluated.

When you debug, try to isolate bugs. Setting breakpoints will allow you to test if each segment of your code runs correctly. Testing as you go along instead of at the end will be the best way to ensure that your debugging process goes smoothly.

Lab Requirements

2 files required in lab3 folder in repository with commit id submitted to the google form:

- Lab3.asm
- README.txt

Check lab1 if you do not recall our expectations for lab write-ups. In addition:

- Discuss the algorithm(s) you designed. Were there any issues in implementing them?
- Discuss any assembly language techniques you developed or discovered.
- Several pseudo-ops were needed to complete this lab. Compare the code written with the assembled code and look at what the assembler did. How many more lines of assembly were written?

To alleviate file format issues we want lab reports in plain text. Feel free to use a word processor if you like to type it up but please submit a plain text file.

Collaboration: *You are allowed to discuss this lab with other students on this lab only from a high level, such as discussing program design, NOT by working on code together.*

Point Breakdown

5 pts: Print a custom greeting message

20 pts: Converts character decimal string into a 2SC number not using the inbuilt syscall

20 pts: Print as a binary string not using the inbuilt syscall

15 pts: Work with negative input not using the inbuilt syscall

5 pts: Have good comments in the code

5 pts: Write up