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

CMPE012/L: COMPUTER SYSTEMS AND ASSEMBLY PROGRAMMING

LAB 4: DEADBEEF IN MIPS

MINIMUM SUBMISSION REQUIREMENTS:

- Lab4.asm in the lab4 folder
- README.txt in the lab4 folder
- Commit and Push your repo
- A Google form submitted with the Commit ID taken from your GITLAB web interface verifying that the above files are correctly named in their correct folders
- All of the above must be completed by the lab due date

LAB OBJECTIVE:

This lab is your introduction to the MIPS assembly language, and to the MARS IDE and simulator. You will learn how to implement the basic structures of programming (data types, loops, conditional statements, IO) using the MIPS language. You will also learn how to use the MARS IDE to compose, test, debug, and run your code, and how to use MARS to understand the relationship between code and hardware.

LAB PREPARATION:

Read this document in its entirety carefully. Read *Introduction to MIPS Assembly Language Programming* by Charles Kann, chapters 2, 3, and 7 (available for free online).

At http://courses.missouristate.edu/KenVollmar/MARS/tutorial.htm, you will find a downloadable tutorial. Read and walk through "Part 1: Basic Mars Use" in this tutorial.

LAB SPECIFICATIONS:

You will write a simple program in the MIPS32 language using the MARS IDE. This code will replicate the functionality of your DEADBEEF code from Lab 1 with one minor change: Instead of printing up to N = 1000, the code will prompt the user for an N, and print up to that value.

Note: No "Output.txt" file is required in this lab (nor should it be created by your MIPS code).

An example of the expected output is given below. Your code's output format should match this output format exactly:

```
Please enter a number N: 11
1
2
3
DEAD
5
6
7
DEAD
BEEF
10
11
```


Please note that part of our grading process is automated and any deviation from this format may cause our grading script to fail. This will waste our time and your points.

Your code should end cleanly without error (be sure to use the exit syscall). You may assume that N will be a positive number.

Your code should be readable:

- Include the required headers in your code (name, cruzid@ucsc.edu, etc.)
- Comment blocks of code. Many students place a comment on every line. This is acceptable, but mid- and high-level comments are essential for making code readable, so be sure to include those as well.
- Include comments to describe your register conventions (describe each register's "job" in your code).
- Use indentation to show the control structure of your code. Blocks of code inside of one loop should be indented once. Blocks of code inside a conditional should be indented once. Blocks of code inside a conditional inside of a loop should be indented twice. Etc.
- Select label names to reflect their purpose.

See chapter 7.7 of Kann for an example of very readable Assembly code. This is the quality of formatting that you should aspire to, and that we expect from you. Neatness here counts.

LAB OVERVIEW:

First, if there were any errors in your code from Lab 1, correct them now. You will use your code as a guide as you write your program in MIPS.

We suggest that you write your MIPS code incrementally. That is, code a little, test a little, code a little more, test a little more. You will almost never be able to sit down and write the full code at once and expect it to run the way you hope (and this is a terrible programming practice).

One incremental coding strategy would be, for example:

- First, produce "Hello World" in MIPS using MARS (i.e., print the prompt and exit).
- Then add code that takes input from the user.
- Next, make a simple loop that prints every integer from 1 to N.
- Add a conditional to that loop.
- Continue adding functionality until you've produced the desired DEADBEEF code.

Make sure you commit and push at each and every step along the way (and submit on the google form if you so desire).

Kann provides sections that explicitly describe how to build abstract structures like loops and conditionals in MIPS. Read these carefully.

INPUT/OUTPUT:

All Input/Output in a MIPS processor is handled using the syscall instruction. Any given MIPS processor will have some set of syscalls available to use. When the processor executes syscall, it checks the value in $v0 to determine which of its arsenal of syscalls it will perform.

In each lab, we will specify which syscalls you will use to complete the lab. This is done to ensure compatibility with the autograders. **In this lab, you MUST use syscall 5 for input**; you will find others useful for output. Refer to MARS' help document to learn more about these syscalls.

LAB WRITEUP:

Be sure your write-up contains:

- Appropriate headers
- Describe what you learned, what was surprising, what worked well and what did not
- Answer the following questions:
  1. In theory, how large can N be before your program fails? What determines this limit?
  2. The text of your prompt ("Please enter a number:") is stored in the processor's memory. After assembling your program, what is the range of addresses in which this string is stored?
  3. Of the instructions you used, which were pseudo-ops (instructions that are not part of the MIPS instruction language, that were translated into other instructions by the assembler)? How do the assembled instructions produce the appropriate result?
  4. How many registers did you use in writing this program? Could you have used fewer registers? Describe how, or explain why you cannot.

To alleviate file format issues we want lab reports in plain text. Feel free to use a word processor to type it up but please submit a plain text file and CHECK IT ON GITLAB to make sure it is readable.

BUMPS AND ROAD HAZARDS:

The difficulty of this lab is mostly in learning to use the MIPS language. MARS provides many tools to help you understand what MIPS is doing at the processor level. It is worth your time to play around in MARS and experiment with its various displays. Understanding the "Registers" pane and the memory panes ("Text Segment" and "Data Segment") are essential to understanding the behavior of the MIPS processor.

MARS comes with a very useful set of debug tools. Use them! Learn to use breakpoints to pause your code. Think carefully about how you expect the registers and output to look at a breakpoint, then check to see if your code matches this. Use the 1-step button to check through trouble spots.

You should experiment with code. For example, if you're not sure how the *div* instruction works, write a very short program to try various inputs and print their outputs.