# Week 9 Lab RISC-V Assembly language programming

## Objectives

Develop understanding and experience of:

1. Using labels in assembly language code as a way to indicate a memory address.
2. Using memory and outputting a string to the console.
3. Using jump and return instructions to create subroutines

Please note that some of these tasks are very similar to those required in the coursework.

## Part A Using memory and outputting strings

Which directive do we use to define the contents of memory?

Which directive do we use to write the program instructions?

n

What is the number for printing a string?

What does little endian mean?

The following exercises assume that you have the work available from last week. As these are practice exercises and not coursework, you are welcome to share code and/or work in small groups.

* You should have code from last week’s lab where the user entered an integer and the program added 1 and output the result (Week 8 lab part B exercise c).

For this exercise, take a copy of that program and amend it to output a message similar to “Please enter a whole number” before the user inputs their number and output a string and integers to give an equation.

For example, one complete run of the program (including user input) might be:

Please enter a whole number: 36

36 + 1 = 37

Expected result:

Screenshot of Run I/O:

* Note that this question asks you to add strings to a previous exercise and later in part B, you are asked to further extend it to add subroutines. If you are confident about your work, you should attempt both aspects together.

You should have some code from last week that stores three initial values in registers and calculates the sum of those numbers, outputting all the numbers with characters between each one.

You should change your program so that it gives output similar to the following:

Number 1: 5

Number 2: 12

Number 3: 137

Total: 154

Make sure that you use the assembler directives .data and .text. Set up suitable labels for the strings you want to output, use the assembler directive .string and put the string in double quotes. Use code provided on Moodle to find a block of code to output a string. Note that if you add strings only, your program will have a lot of repetition as you haven’t been asked to use a subroutine yet. You might want to look at the hex values stored in the data segment in RARS to see how the strings are stored.

Expected Result:

Screenshot of the Run I/O:

## Extension Exercises:

* Amend the extension exercises from last week’s lab B to output appropriate message prompts before and after the user inputs their values to give suitable instructions/information detailing what to do (e.g. for extension task 1, you could use ‘Enter a numerical character between 0 and 9’).