# Week 8 Lab B RISC-V Assembly language programming

## Objectives

* By the end of the session, you will be able to construct their own basic program using RISC-V Assembly Language.
* You will be able to write a program which will add, subtract and store numbers in different registers.
* You will be able to write a program which will output to the console in the software RARS.

## Part B Using environment calls in RISC-V assembly language

Again, you might want to use the concepts code from Moodle to get started. Now that your programs are getting longer, you should add comments (starting with a #) to explain your code. Remember to use the **Syscalls** tab from the **help** to remind you how to interact with the operating system.

What does **ecall** do?

Pass control over to the OS

What register do we store values in to instruct the operating system?

A7

What register stores additional information depending on the instruction?

A0

What value is used if you want to output an integer to the console?

1

The next two exercises build on each other, so you might want to work on both parts as one. Make sure that you separate code that does different actions and use comments in the code.

* Take a copy of your program that adds three numbers (from part A) and amend it:
  + Output the initial values from the three registers and the result to the console.

At this point you will be outputting just the numbers and no text.

Expected Result in the Run I/O:

Take a screenshot of **both** your code and the Run I/O:

* Now amend your program:
  + Output a single character between each number.
  + Use the service to output a character, please don’t use a string at this stage. The single character could be a comma, a space, a tab or a newline character according to your preference.

The expected output in the Run I/O is:

Paste a screenshot of your Run I/O here:

* Create a program:
  + The user inputs a whole number
  + The program stores that number and adds 1 to it,
  + Store the result in a different register, and outputs the result.

Expected Result:

Paste a screenshot of your code and Run I/O:

## Extension Exercises

Some of these tasks are likely to need some thought and so it might be good to work in small groups on the problem solving.

1. Write some code that will convert a single digit entered as a character to an integer. Look at the ASCII table to work out what you need to do to convert a character (0, 1, 2 … 9) to its numeric equivalent. You should assume that the input is always a digit. Remember that when you key input to a Java program (or other) it always starts out as a string of characters and is converted to an int in the program. When we are doing the environment call to read in an int, we are expecting the operating system to do that conversion.
2. Write a program that will convert a 4-digit binary number entered as four individual characters to its decimal equivalent, store that decimal equivalent in a register and output it as an integer. So, for the input 1101, it should output 13.

Assume that the characters that the user inputs are all 0 or 1 as expected and include the code you wrote for part a to convert from ASCII input to an integer. You can decide whether to store all the 4 input characters in registers or just work out the decimal conversion as you go along. You will find that your code has a lot of repetition as we haven’t covered any instructions to use for subroutines or loops. Make sure you test your program.

1. Write a program that converts a number of minutes input by a user to a number of hours and minutes. For example, film lengths are often written as a number of minutes, such as 108 which should be converted to 1 h 48 m.

For this exercise you will need to look at the RISC-V instructions that are available for integer division and remainder. You should assume that the input will be a valid integer.

Remember to test your program using numbers you have worked out separately.