# Laboratory Exercise 6

Intro to Basic RISC-V Assembly

Revision of October 24, 2025

In this lab, you will learn how to write basic programs in assembly. We will make use of the NIOS V processor, which employs the RISC-V instruction-set architecture. The NIOS V processor is integrated into the DE1-SOC computer system available in the labs' DE1-SOC boards. This system includes the NIOS V processor, memory for storing programs and data, and various I/O ports. Documentation for the DE1-SOC system with NIOS V can be found in the companion document titled *DE1-SoC Computer System with Nios V*, provided with this lab.

You will learn how to use the online simulator, **CPUlator** to work on programs for the DE1-SOC with NIOS V processor system. You will also have a chance to run your programs on the actual boards in lab using the **Monitor Program**.

An overview of the Nios V processor can be found in the tutorial document *Introduction to the Nios V Soft Processor*, which is also provided with Lab 6.

## 1 Introduction to CPUlator - Preparation

In this part of the lab exercise, you will watch a brief video provided by Prof. Brown that introduces the CPUlator simulator. CPUlator is a web-based tool that simulates the DE1-SOC computer, including the NIOS V processor, memory, and various input/output devices on the board.

CPUlator is a full-feature software development environment that allows you to compile (or more accurately, assemble) and debug code for the NIOS V processor in both assembly and C languages. This tool was developed by Dr. Henry Wong, a former TA for this course, who continues to maintain it voluntarily in his personal time. Thank you, Henry!

Access the introductory video using the following URL:

https://youtu.be/F4PDYijJX8U

The CPUlator website for the NIOS V DE1-SOC is available here:

https://cpulator.01xz.net/?sys=rv32-de1soc

## 2 Part I

In this section, you will write your own RISC-V assembly language program to perform two tasks: count the number of items in a list and calculate the sum of all the items in the list. The sum of the numbers should be stored in register `s10`, and the count of the numbers should be stored in register `s11`.

For instance, consider the list: 1, 2, 3, 5, 10. Your code should store the value 21 (0x15) in register `s10` and the value 5 in register `s11` when the program completes.

The list will be stored at the memory location labeled `LIST` and will consist of positive numbers (i.e., numbers>0) ending with a value of -1 (i.e: a -1 value indicates the end of the list).

An outline of the program for Part I is shown below.

```
.global _start
.text

_start:
    la s2, LIST
    addi s10, zero, 0
    addi s11, zero, 0

    # Write your code here

END: j END

.global LIST
.data
 LIST:
.word 1, 2, 3, 5, 0xA, -1
```

### What you should do

Follow the steps below to write and test your first assembly code. Complete Steps 1-3 below as your pre-lab preparation.

1. Open the CPUlator website for NIOS V DE1-SOC here: https://cpulator.01xz.net/?sys=rv32-de1soc
2. Write your RISC-V assembly program in CPUlator's Editor Pane.
3. Click on the `File` command near the top of the CPUlator window and the select `Save`

. . . . This will save a copy of your text file in your Download folder so you can re-use it for later. (You can use the `Load...` option under `File` to load a file into the editor; you should rename any file that is downloaded to an appropriate name).

4. Click on the $\boxed{\texttt{Compile and Load}}$ button to *assemble* your program and *load* it into the *memory* of the simulated computer system. You should see a "Compile succeeded" message in the `Messages` pane if you have no errors in the program code. If not, use the Editor pane to fix any errors that you encounter and recompile your program. Once the compilation is successful, the *CPUlator* window automatically displays the `Disassembly` pane.
5. Select the `Continue` command near the top of the *CPUlator* window. Take a look at the *CPUlator* `Register` pane located on the left hand side of the web page. Check to ensure values in `s10` and `s11` match your expected output. If not, use *CPUlator*'s breakpoint and `Step Into` features to debug and fix your code.
6. Test your code with other values of LIST.
7. Once your program works as expected and you have tested it fully, save your program as part1.s.
8. Transfer your file to the ug machines and make sure it passes the tester by running the `/cad2/ece253f/public/6/tester` command
9. Submit your file for marking. For your code to work correctly with the automarker, make sure you only write code where it says `# Write your code here`. If you change any of the code above or below this line, your code will fail the marker.

## Optional: In lab hardware testing of your code

Take your program, and add the following three lines of code just after the computation is complete (before the END: j END line):

```
.equ LEDs, 0xFF200000
la s2, LEDs
sw s10, 0(s2)
```

This code will display the sum of the list, in binary, on the LEDs on the board. We will be learning more about I/O devices in upcoming lectures; so for now, don't worry about why this bit of code works.

Following the instructions in this "Monitor Program for NIOS-V" tutorial video prepared for ECE352, run your program on the DE1-SoC board using the Monitor Program. Does it work? If not, debug it!

Show your TA that you have this program working.

## Submission

Submit your file as part1.s to the automarker. As your code will be tested by an automarker, you are required to adhere to the following submission guidelines.

1. The code containing the input data (i.e., starting with `.global LIST` and ending with `-1`) has to be at the end of the file.
2. The name of the list has to be `LIST`.
3. Make sure that you save the correct values to the appropriate registers, as specified in Part 1.

As a reminder, you can run the check submission command:

```
/cad2/ece253f/public/check_submission 6
```

to confirm that you have correctly submitted your file.

## 3 Part II

In this part, you are given a pre-written program to count the longest string of 1s in a word of data. You will not need to submit this part of the lab, but you will use this code in Lab 7 so you should take some time to understand how it works. You may be asked how this code works as part of you in-lab questions from the TA. Run this program in CPUlator and step through the code to make sure you can follow along. Then, try the program again for different values of LIST.

```
# Program that counts consecutive 1's.
.global _start
.text

_start:
        la s2, LIST         # Load the memory address into s2
        lw s3, 0(s2)
        addi s4, zero, 0    # Register s4 will hold the result

LOOP:
        beqz s3, END        # Loop until data contains no more 1's
        srli s2, s3, 1      # Perform SHIFT, followed by AND
        and s3, s3, s2
        addi s4, s4, 1      # Count the string lengths so far
        j LOOP


END: j END

.global LIST
.data
 LIST:
.word 0x103fe00f
```

## What you should do

Perform the following:

1. Create a new RISC-V assembly language program. Start by typing the RISC-V assembly language code shown below in CPUlator. This code uses an algorithm to count the longest string of 1s in a word of data. The algorithm uses shift and AND operations to find the required result. Make sure that you understand how this works.
2. Assemble the program. Fix any errors that you encounter (if you mistyped some of the code). Once the program is correctly assembled, single step through it to see how the program works. Try changing the word at the end of the program to see what happens.
3. Save your program as you will be asked to modify this code as part of Lab 7.