Name: _____

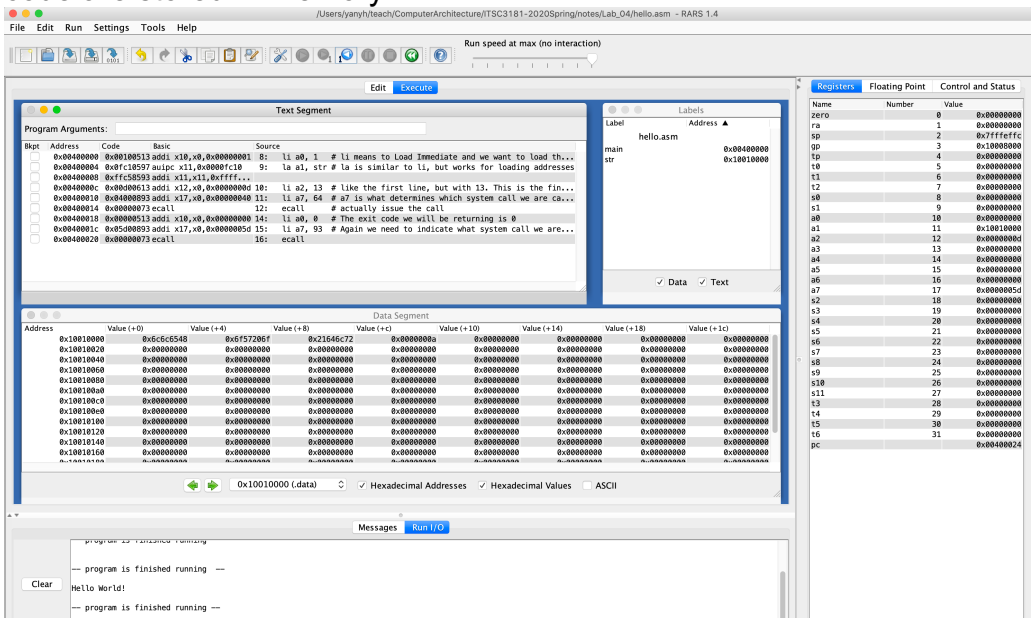
Lab for ITSC 3181, Introduction to Computer Architecture, Spring 2023

Lab #04 and #05: Converting C programs to RISC-V assembly and simulate their execution using RARS simulator. Due on 02/10 and count for 3% of the final grade percentage.

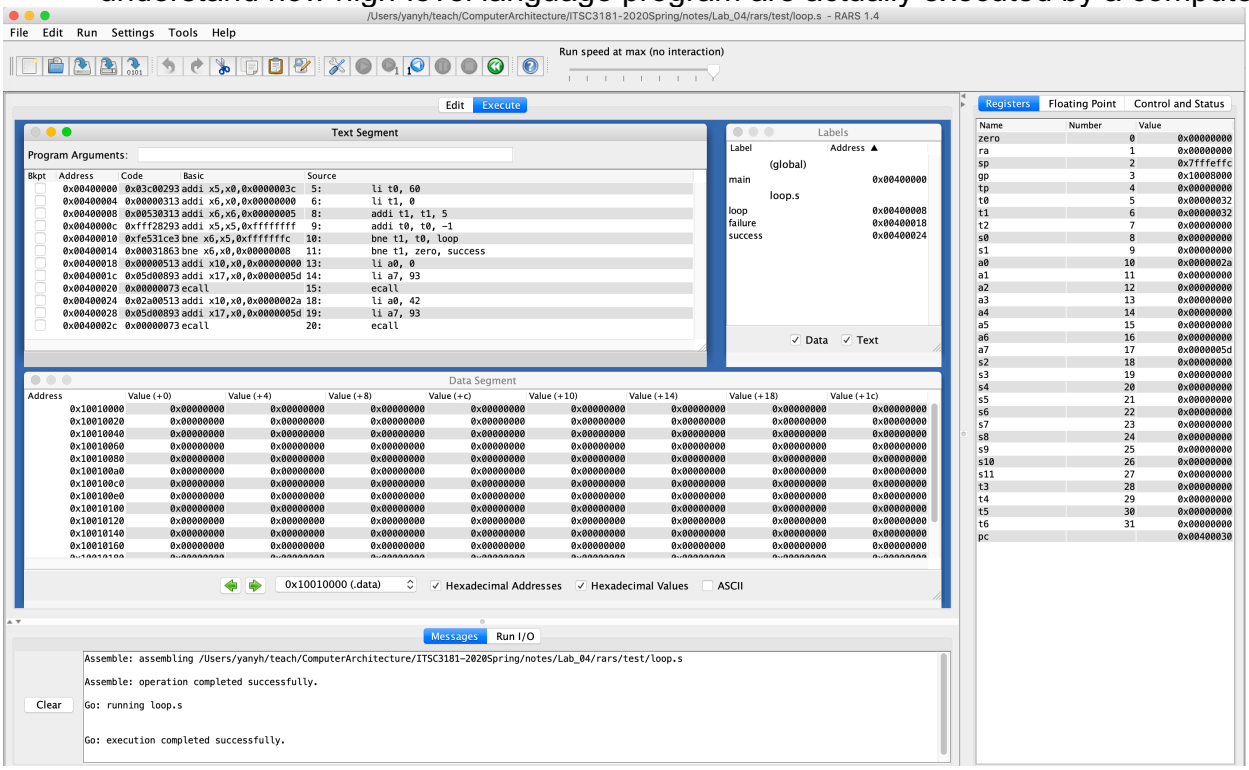
	Lab #04 and #05						
Steps	2	3	4	5	6	7 (bonus)	Total
Total Points	10	20	20	20	30	20	120
Your points							

We will use RISC-V Assembler and Runtime Simulator (RARS) for this lab, which is available from https://github.com/TheThirdOne/rars. **A video for introducing how to use RARS simulator is available from https://passlab.github.io/ITSC3181/resources/UsingRARS_ITSC3181.mp4 or from Canvas Medial Gallery.**

1. Download the latest RARS jar file from https://github.com/TheThirdOne/rars/releases/download/v1.5/rars1_5.jar on your computer. You should then be able to launch the jar file by either double clicking it or from launcher such as using open command in Mac OS X terminal.
2. Create and execute the hello world program following instructions from https://github.com/TheThirdOne/rars/wiki/Creating-Hello-World. Spend some time to read and understand each line of that page. Then in RARS, create the Hello World program, and then assembly and run the program by clicking the sub-menu items of the Run menu. Play with the example, menu items and the RARS interface to get familiar with the RARS tool. Check the address, binary code, instructions and source of the assembled code, and also check the register values and memory values (data segment part) of the program execution. After you run the program multiple times, you should run it step-by-step, i.e. instruction by instruction, and observe the change of values in registers and other locations. There are two sections, named .text section and .data section, in the program. The .text section is for the code and the .data section is for the data. There is a “main” label in the .text section and “str” label in the .data section. The two labels are symbols representing the addresses of the memory locations where corresponding data or code are stored in memory.



3. Convert and execute the loop.s file from RARS repo in the test folder.
    a. Download or cut-paste the raw file (https://raw.githubusercontent.com/TheThirdOne/rars/master/test/loop.s) to use it with RARS. Convert the loop.s program to C program that do the same as loop.s. You can execute the C program from https://repl.it/languages/c. In the loop.s file, check https://github.com/TheThirdOne/rars/wiki/Environment-Calls to understand the ecall. For this one, ecall is just a “return <code>” call in C.
    b. Assembly and run the loop.s program in RARS. See below screen shot. Check the address, binary code, instructions and source of the assembled code, and also check the register values and memory values (data segment part) of the program execution. After you run the program multiple times, you should run step-by-step, i.e. instruction by instruction and observe the change of values in registers and other locations. During the step-by-step simulation in RARS, do the simulation in your mind of the corresponding C program to understand how high-level language program are actually executed by a computer.



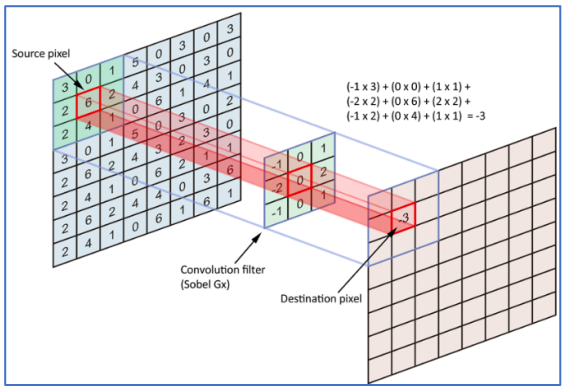
4. Implement a program to accumulate integer numbers from 1 to 100 using both C and RISC-V assembly, and simulate the assembly program execution using RARS.
    a. Write a main program in C from https://repl.it/languages/c to accumulate integers from 1 to 100 together using a for loop. The program should print the result (using C printf) and also retune the value of the accumulation (return). Execute the program from the browser and make sure it produces the expected output (5050 I believe).
    b. Using the loop.s as starting point, program 1-100 integer accumulation using RISC-V assembly. While the instructions we learned during the class should be sufficient to do the work, you can check RARS supported instructions (https://github.com/TheThirdOne/rars/wiki/Supported-Instructions) and use them. To print the result and return the result, your program should make environment call PrintInt, check https://github.com/TheThirdOne/rars/wiki/Environment-Calls.
5. Implement a program to find the average of 100 integers that are randomly generated, using both C and RISC-V assembly, and simulate the assembly program execution using RARS.

a. The C program MUST follow these steps: 1) declare an int array of 100 elements, and use a for loop to generate 100 integers and store them in the array; 2) use another for loop to accumulate those numbers by reading them from the array and adding up to a variable; 3) calculate the average by dividing the accumulated sum with 100, and 3) print the average and return the average. Your program should NOT do the number generation and accumulation in one loop. Write C program from https://repl.it/languages/c. The functions for generating random numbers can be found from Lab 03 1-D stencil code.
b. Converting the C program to RISC-V assembly and simulate its execution using RARS. To use array in assembly code, your code needs to reserve space in the data section. Check the memory.s file in RARS repo (https://github.com/TheThirdOne/rars/blob/master/test/memory.s) for using .space to reserve memory for an array identified by a symbol, and how to use la instruction to load the memory address (first element of the array) to a register.

6. Implement a program to do the simple 1-D stencil of 100 integers that are randomly generated, using both C and RISC-V assembly, and simulate the assembly program execution using RARS.
   a. For C program, convert the 1-D stencil program from lab_03 to use array reference (B[i]) to access array element instead of using pointers. The C program follows these steps: 1) declare two arrays, each has 100 elements, 2) use a for loop to randomly generate 100 integers and store them in one array; 2) use another for loop to do the 1-D stencil and store the result in the other array; Write a main program in C from https://repl.it/languages/c.
   b. Converting the C program to RISC-V assembly and simulate the program execution using RARS.
7. Implement a RISC-V assembly program for image convolution. Image convolution perform 3x3 filtering of each pixel of the source image and the result is written to the dest image. It is the operation widely used for many image processing algorithms, and convolution is one of the kernels of convolution neural network (CNN) for deep learning. The algorithm can be described using the following C code and figure. C array are stored in row-major and make sure you program calculates address of array elements (e.g. source[i][j]) correctly using what we studied during the class. Implement the program using RISC-V assembly and executes it on RARS.

```
int M = 16, N =16;
int filter[3][3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
int source[M][N], dest[M][N];
int i, j;
for (i=0; i<M; i++)
   for (j=0; j<N; j++)
      source [i][j] = rand( ); // init data with random

 /* convolution kernel */
for (i=1; i<M-1; i++)
   for (j=1; j<N-1; j++) {
      int tmp = 0;
      for (fi=0; fi<3; fi++)
         for (fj=0; fj<3; fj++) {
            tmp += source[i+fi-1][j+fj-1]*filter[fi][fj];
         }
      dest[i][j] = tmp;
   }
```



For submission: please submit a single PDF file that shows the execution screenshot of each program in RARS and C terminal required by this lab, and also submit all the source code (both C and assembly code) as well.