ITSC 2181 Introduction to Computer Systems
Module 06 - Unit 3 Lab

Converting C programs with if-else, loops and array access to RISC-V assembly and simulating their execution using RARS simulator

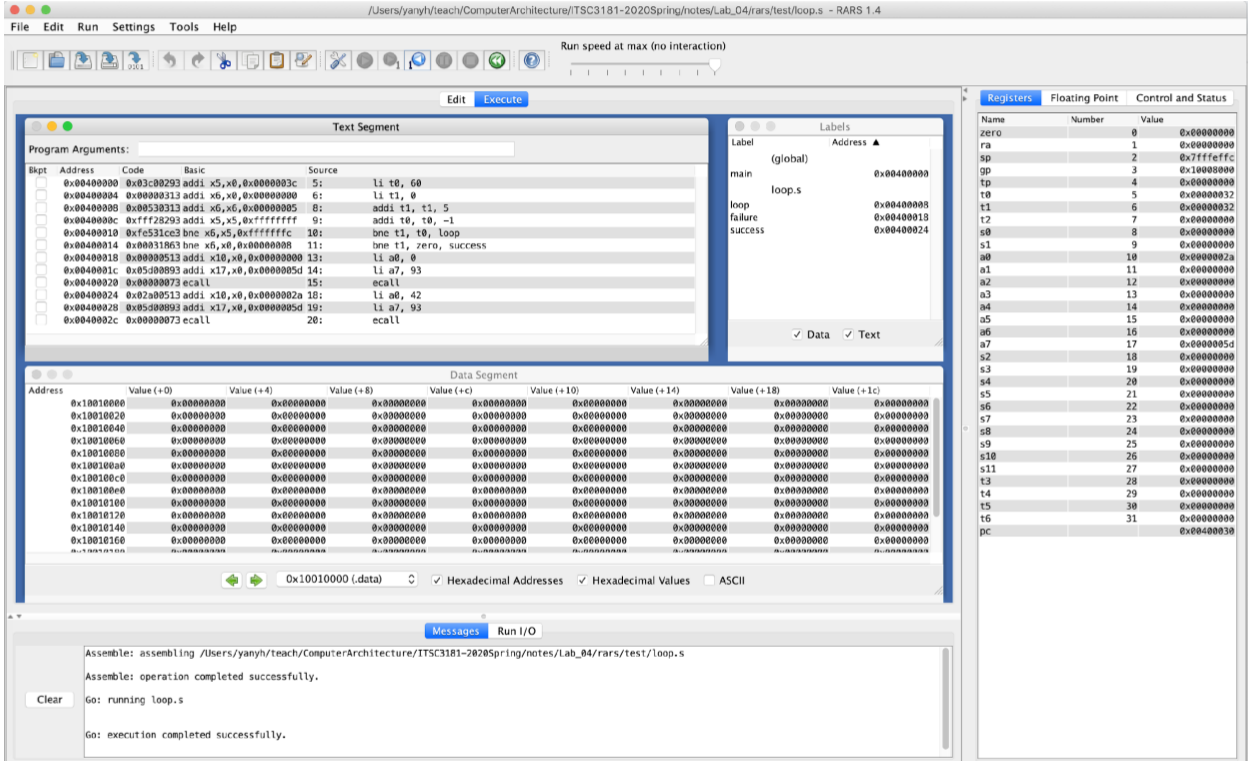
- In this lab, you will learn how to use branch instructions for `if-else` and `loop` statements.
- You will also practice using load and store and other instructions we studied in Unit 1 and 2 of this module.

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.**

Part 1:

Convert and execute the `loop.s` file from the RARS repo in the test folder.

1. Download or copy-paste the source file (https://raw.githubusercontent.com/TheThirdOne/rars/master/test/loop.s) and open it with RARS.
2. Convert the `loop.s` program to a C program that does the same as `loop.s`.
   a. You can execute the C program at https://repl.it/languages/c. or use the VM you configured earlier.
   b. Check https://github.com/TheThirdOne/rars/wiki/Environment-Calls to understand the ecall instruction used in the `loop.s` file. For this program, `ecall` is just a "return <code>" call in C.
3. Assemble and run the `loop.s` program in RARS. See the screenshot below.
   a. 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.
   b. 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.
   c. 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 programs are actually executed by a computer.
   d. To see the Labels window, go to the *Settings* menu and select "Show Labels Window (symbols table)."



Bkpt	Address	Code	Basic	Source	
□	0x00400000	0x03c00293	addi x5,x0,0x0000003c	5:	li t0, 60
□	0x00400004	0x00000313	addi x6,x0,0x00000000	6:	li t1, 0
□	0x00400008	0x00530313	addi x6,x6,0x00000005	8:	addi t1, t1, 5
□	0x0040000c	0xfff28293	addi x5,x5,0xffffffff	9:	addi t0, t0, -1
□	0x00400010	0xfe531ce3	bne x6,x5,0xfffffffc	10:	bne t1, t0, loop
□	0x00400014	0x00031863	bne x6,x0,0x00000008	11:	bne t1, zero, success
□	0x00400018	0x00000513	addi x10,x0,0x00000000	13:	li a0, 0
□	0x0040001c	0x05d00893	addi x17,x0,0x0000005d	14:	li a7, 93
□	0x00400020	0x00000073	ecall	15:	ecall
□	0x00400024	0x02a00513	addi x10,x0,0x0000002a	18:	li a0, 42
□	0x00400028	0x05d00893	addi x17,x0,0x0000005d	19:	li a7, 93
□	0x0040002c	0x00000073	ecall	20:	ecall

Label	Address ▲
(global)	
main	0x00400000
loop.s	
loop	0x00400008
failure	0x00400018
success	0x00400024

☑ Data ☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000032
t1	6	0x00000032
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x0000002a
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x0000005d
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400030

```
Assemble: assembling /Users/yanyh/teach/ComputerArchitecture/ITSC3181-2020Spring/notes/Lab_04/rars/test/loop.s

Assemble: operation completed successfully.

Go: running loop.s


Go: execution completed successfully.
```

Part 2:

Implement a program to accumulate the integer numbers from 1 to 100 using RISC-V assembly, and simulate the assembly program execution using RARS. You should have already implemented a `C` program for this task in a previous lab.

1. Using the `loop.s` program as a 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.
2. To print the result and return the result, your program should make an environment call `PrintInt`, check https://github.com/TheThirdOne/rars/wiki/Environment-Calls.

Part 3:

Implement a program to find the average of 100 integers that are randomly generated using RISC-V assembly, and simulate the assembly program execution using RARS. You should have already implemented a `C` program for this task in the previous lab.

1. The program must follow the same steps as the C program you implemented before:
    a. Declare an int array of 100 elements, and use a for loop to generate 100 integers and store them in the array;
    b. Use another for loop to accumulate those numbers by reading them from the array and adding up to a variable.

   c. Calculate the average by dividing the accumulated sum with 100.
   d. Print the average and return the average. Your program should NOT do the number generation and accumulation in one loop**. You must use two separate loops.**
2. You can convert the C program to RISC-V assembly or write directly the RISC-V assembly. After that, simulate its execution using RARS.
   a. To use arrays in assembly code, your code needs to reserve space in the data section.
   b. Check the **`memory.s`** file in the RARS repo (https://github.com/TheThirdOne/rars/blob/master/test/memory.s) and the previous lab for using **`.space`** to reserve memory for an array identified by a symbol, and how to use the **`la`** instruction to load the base memory address (first element of the array) to a register.

Part 4:

Implement a program to find the maximum number in an array of 100 integers that are randomly generated, using both **`C`** and RISC-V assembly, and simulate the assembly program execution using RARS. The program should follow these steps:

1. Declare an array that has 100 elements.
2. Use a **`for`** loop to randomly generate 100 integers and store them in the array.
   a. You should use the **`RandIntRange`** system call.
   b. See https://github.com/TheThirdOne/rars/wiki/Environment-Calls
3. Use another **`for`** loop to find the max value in the array and return it.
4. Write a main program in C from https://repl.it/languages/c and make sure it executes correctly. You should use an algorithm similar to the one discussed in the lecture slides to find the minimum of an array.
5. Converting the C program to RISC-V assembly and simulating the program execution using RARS.

Submission:

1. Submit the source code of the C programs (**`.c`**) and assembly programs (**`.asm`**) you write in this lab.

2. Submit a single PDF file that shows the execution screenshot of the programs in RARS.