Project 1: **Calculator**

Instructor: Prof. Seokin Hong (seokin@knu.ac.kr)

Assigned: October 27, 2020

Due: 11:59pm November 15, 2020

1. Description

The goal of this project is to write a calculator program with RISC-V assembly language. The program will perform the **32-bit** addition, subtraction, multiplication and division with unsigned numbers. You will use a RISC-V emulator called **RARS** for this project.

- **RARS** (https://github.com/TheThirdOne/rars)
- The user manual of **RARS** is available in the LMS.

2. Requirements

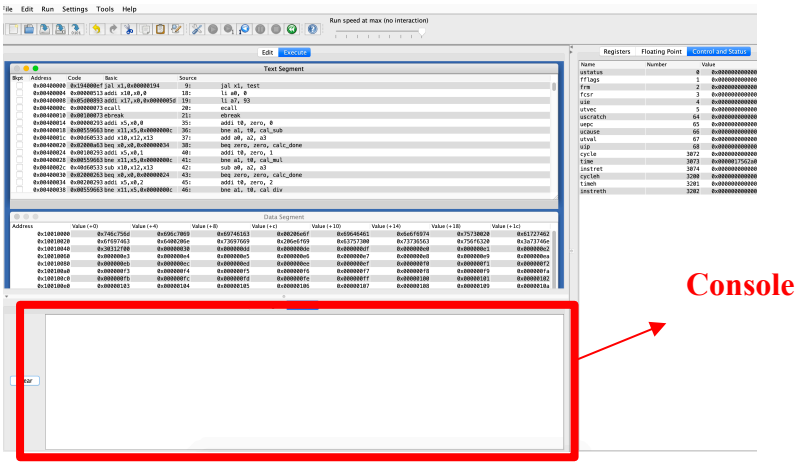
1) Write a RISC-V assembly language program that performs the **32-bit** addition, subtraction, multiplication, and division.

2) Inputs and results of the computation are 32-bit values.

3) **You cannot use "mul", "mulh", "mulhu", "mulhsu", "div", "divu", "rem", "remu", and "sub" instructions.**

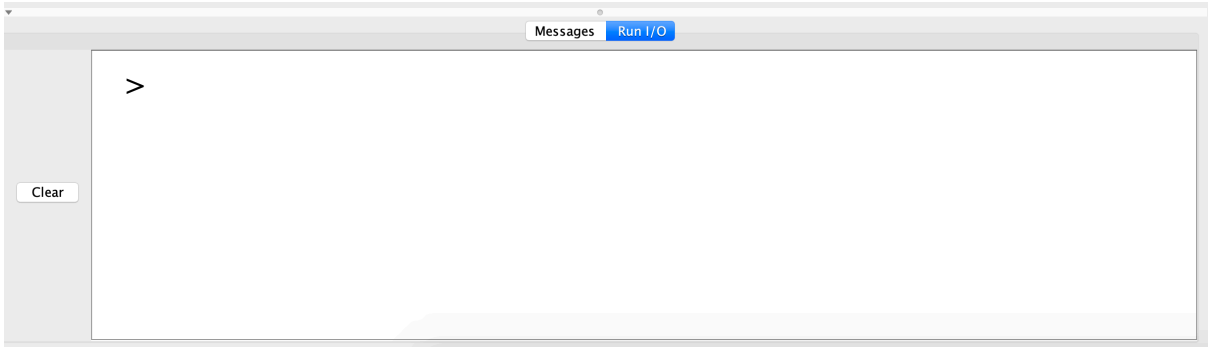
4) **You have to implement the algorithms (figure 1) we discussed in Chapter 3 for the multiplication and division.**

5) Your program should read a string from the console of the RARS emulator and print the computation result to the console.

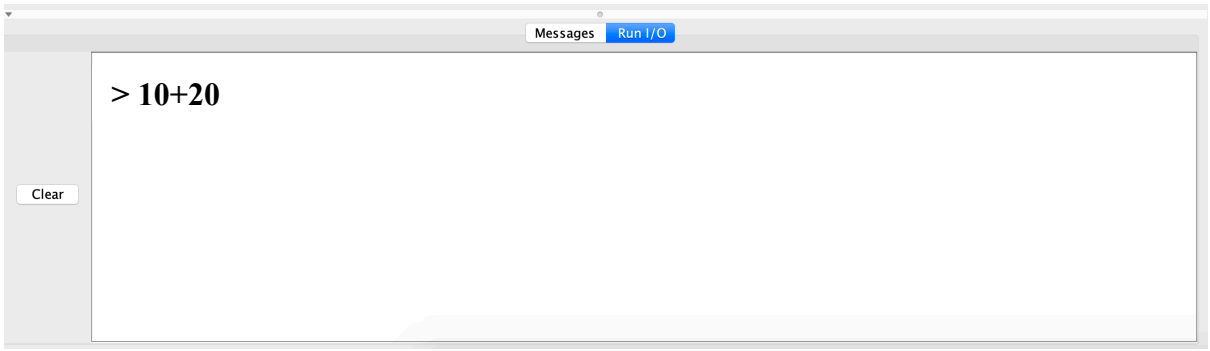


- **Execution example**

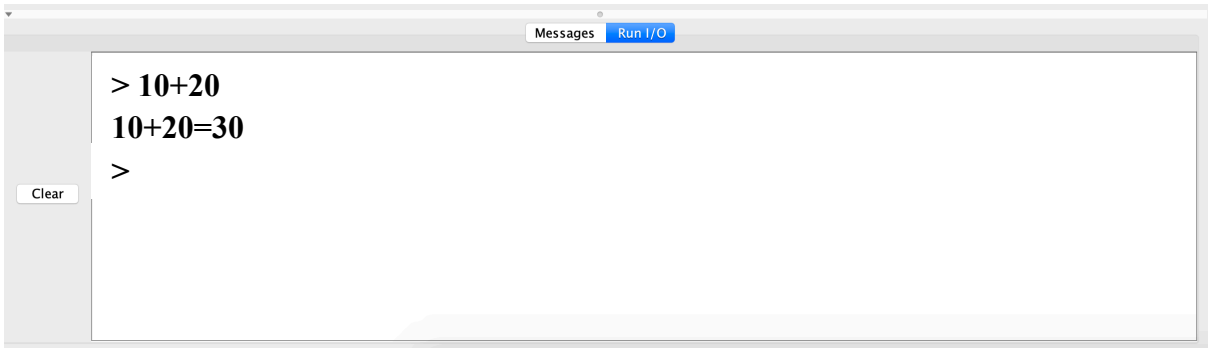
  a. Show prompt in the console

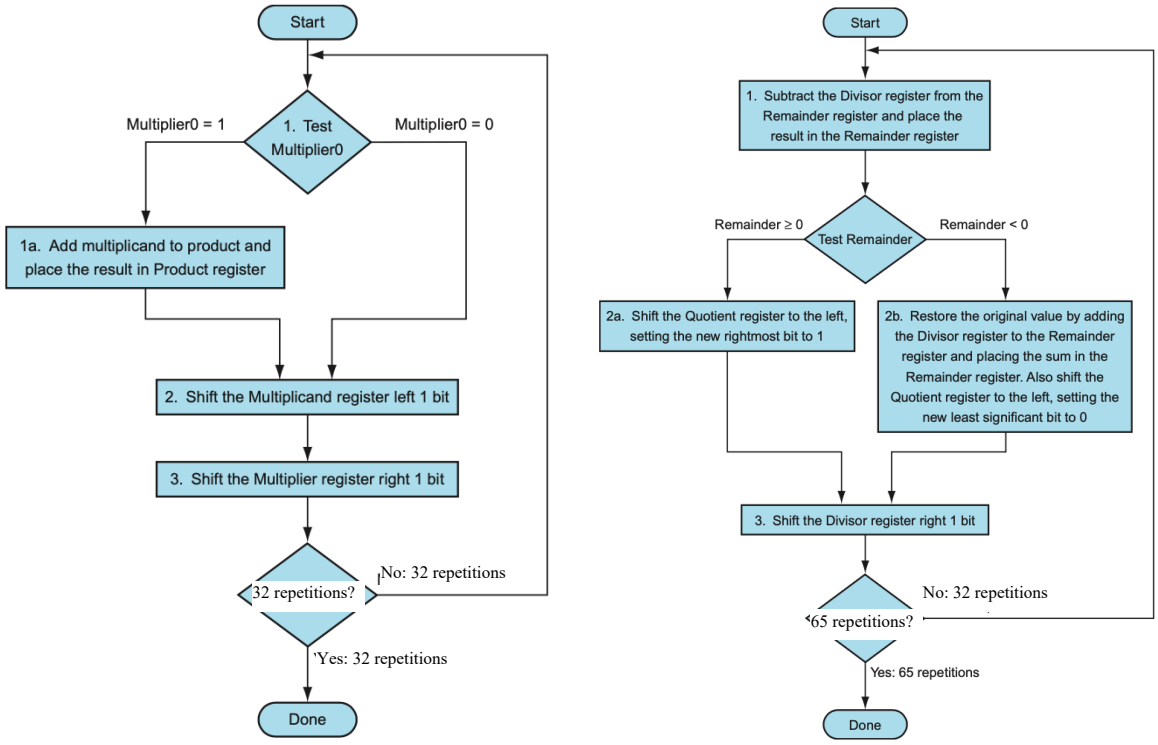


  b. **Type** an equation into the console



  c. Display a result





(a) Multiplication algorithm (b) Division algorithm

Figure 1. Multiplication and division algorithms

3. Procedure

1) **Step1:** Download RARS in the following link

https://github.com/TheThirdOne/rars/releases/download/continuous/rars_5f747b9.jar

2) **Step2:** Download "CA2020_PJ1.asm" and "common.asm" from the LMS.

3) **Step3:** Edit the CA2020_PJ1.asm to implement the calculator program.
   - Complete main() procedure
   - Complete calc() procedure
   - You can add your own procedures in the CA2020_PJ1.asm file.

4) **Step4:** Assemble and test your code
   - main() procedure already includes a basic testing facility. So, you can test the functionality of your code as soon as you complete the calc() procedure.
   - **Your code should meet the following requirements on the argument passing for the calc() procedure.**
     - **x11** should contain the type of the arithmetic operations (0: addition, 1: subtraction, 2: multiplication, 3: division)
     - **x12** should contain the first operand (the dividend for division)
     - **x13** should contain the second operand (the divisor for division)
     - **x10** should be used to return a computation result to caller

- **x14** should be used to return the remainder of a division operation to caller

5) **Step5:** Submit your assignment to the LMS. You should only submit the following files:
   - CA2020_PJ1.asm

4. Grading

Your submission will be graded based on the following criteria.

1) **Basic functionality test : 80%**
   - **Addition :** 10%
   - **Subtraction :** 10%
   - **Multiplication :** 30%
   - **Division :** 30%

2) **Test with the console : 20%**
   - **Addition :** 5%
   - **Subtraction :** 5%
   - **Multiplication :** 5%
   - **Division :** 5%

Late Day Policy

Submission is due at 11:59pm on the due date. A grading penalty will be applied to late assignments. Any assignment turned in late will be penalized 25% per late day.

Plagiarism

No plagiarism will be tolerated. If the assignment is to be worked on your own, please respect it. If the instructor determines that there are substantial similarities exceeding the likelihood of such an event, he will call the two (or more) students to explain them and possibly to take an immediate test (or assignment, at the discretion of the instructor) to determine the student's abilities related to the offending work.

Appendix

1. How to use the system calls to read a string from the console and to write a string to the console?

   https://github.com/TheThirdOne/rars/wiki/Environment-Calls

2. Assembler directives

   https://github.com/TheThirdOne/rars/wiki/Assembler-Directives

3. RISC-V calling convention register usage

Register	ABI Name	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5–7	t0–2	Temporaries
x8	s0/fp	Saved register/frame pointer
x9	s1	Saved register
x10–11	a0–1	Function arguments/return values
x12–17	a2–7	Function arguments
x18–27	s2–11	Saved registers
x28–31	t3–6	Temporaries
f0–7	ft0–7	FP temporaries
f8–9	fs0–1	FP saved registers
f10–11	fa0–1	FP arguments/return values
f12–17	fa2–7	FP arguments
f18–27	fs2–11	FP saved registers
f28–31	ft8–11	FP temporaries