# Experiment 3: Introduction to RARS, RISC V Assembly Programming

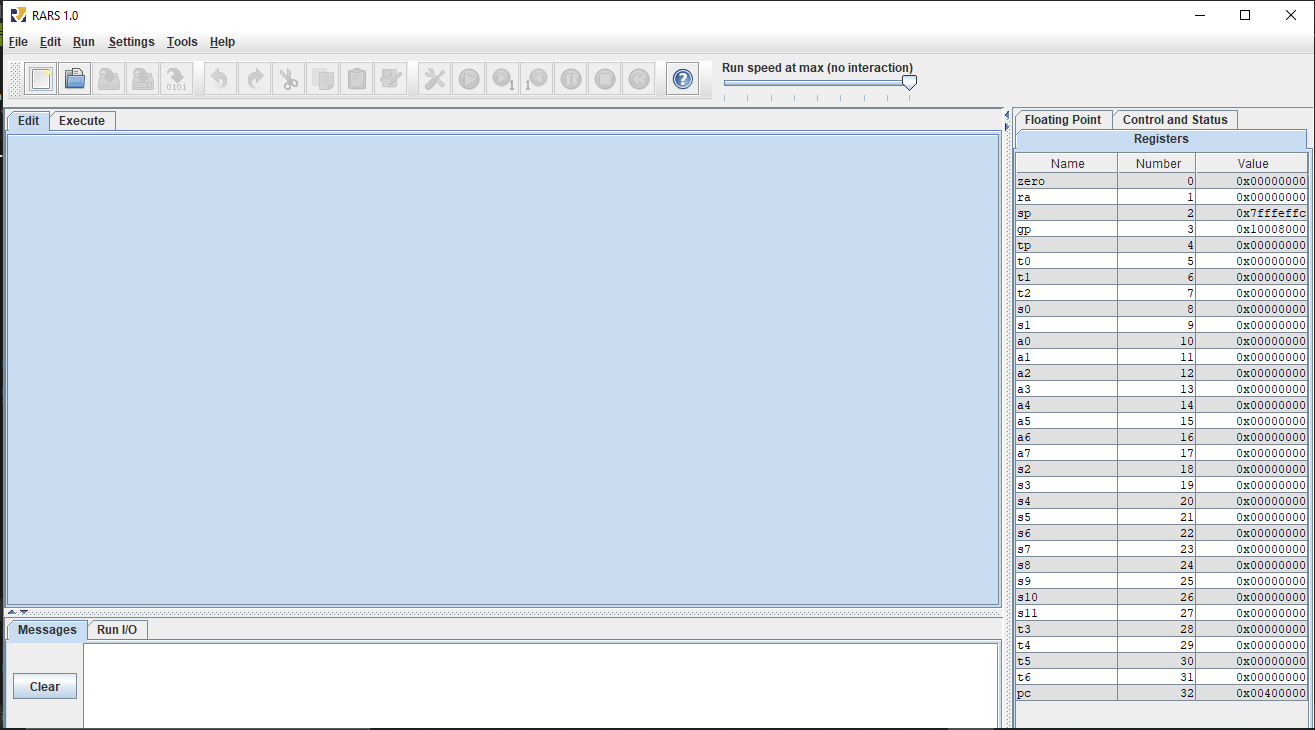
|  |  |  |
| --- | --- | --- |
| **Sl No** | **Name** | **ID No** |
| **1** | **Vishwas Vasuki Gautam** | **2019A3PS0443H** |

## RARS -- RISC-V Assembler and Runtime Simulator

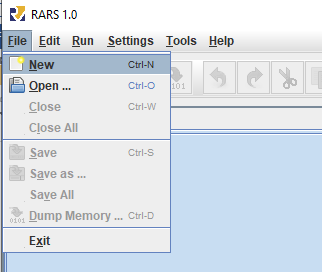
RARS, the RISC-V Assembler, Simulator, and Runtime, will assemble and simulate the execution of RISC-V assembly language programs. Its primary goal is to be an effective development environment for people getting started with RISC-V. Please refer to the steps below

Registers

Open RARS executable Jar file.



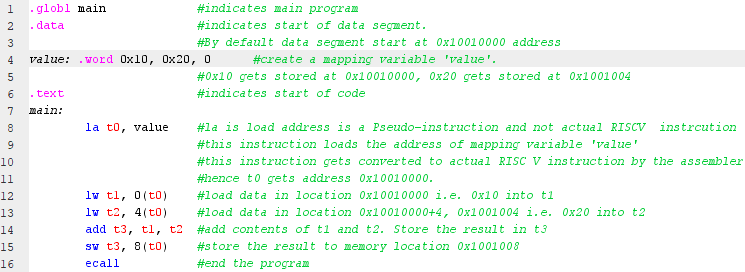
To create a new assembly program, click on File 🡪New



## Programming with RARs: An Example of adding two numbers in memory and storing the result back in memory

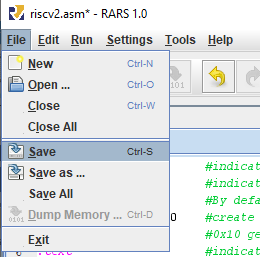
1. The assembly level program should be written using the editor and should be **saved as .asm or .s file**

Below is the assembly code for this experiment. Please go through comments in the code for more details.

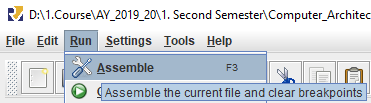


Here **.data** indicates the data segment. In data segment a location value is created and three words are stored. **.text** indicates the beginning of code segment where the main program is written. The program written here loads two words that are stored at location Value (i.e. Value[0] and Value[1]) and stores the result in third word location (i.e. Value[2]). In the above program **la t0, value** is NOT a valid RISC V instruction. This instruction is supported in RARS to make assembly programs simpler. There are many such instructions which are supported in RARS but are NOT valid MIPS instructions. Such instructions are called as **pseudo instructions**. These pseudo instructions will be converted to valid RISC V instructions by the assembler.

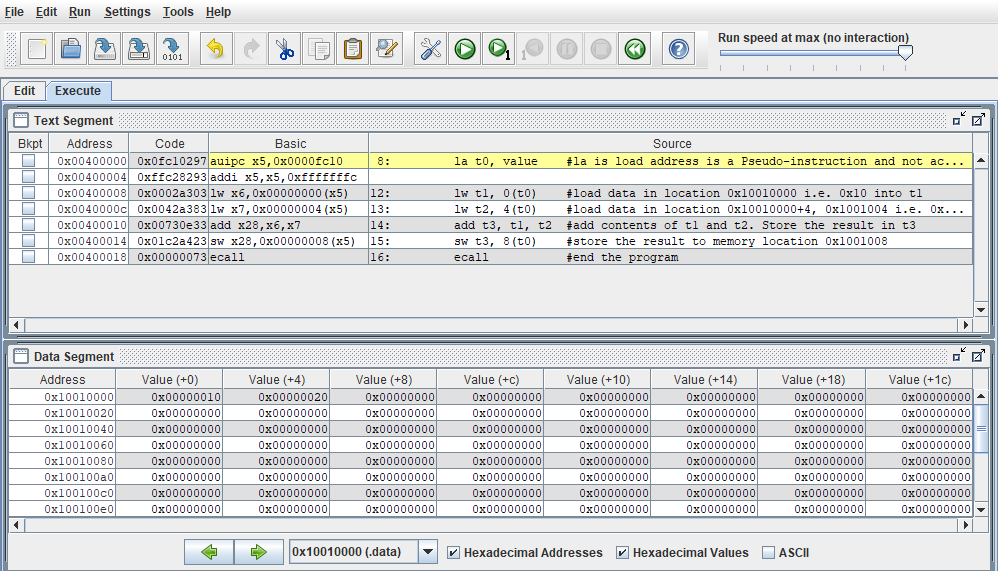
1. Save the assembly file with an extension *.asm* or .*s*.



1. After saving the code, assemble it by clicking **Run🡪Assemble**



1. After assembling the code successfully, execute window will open. Here two windows, one related to text segment (code) and the other related to data segment will be displayed. The highlighted section in text window show the actual RISC V instructions and instruction codes. Please note that by default program gets loaded from location 0x00400000. The highlighted section in data segment shows the values stored, i.e. 0x10, 0x20 and 0x00.



1. To run the code, there are two options. First one is to run the complete program at once, by clicking on “Run the current program” button.



The second option, (which is preferred, to understand and debug the code) is to run the program one instruction at a time, by clicking “Run one step at a time” button.



1. The changes in the memory content or the register content after execution of each instruction, can be viewed in Data tab or registers tab respectively**.** In many programs it might be necessary to check the execution step by step. While running the single step execution you can constantly view that status of data memory and registers after every step for better understanding of working of any assembly program. Since most of the programs that you will be working on will be small use this option more frequently than “Run the current program” option.

You can also undo one step of execution or reset the memory and registers (to restart executing from the start) using the buttons shown below





Please refer to the link below for the list of supported instructions

<https://github.com/TheThirdOne/rars/wiki/Supported-Instructions>

Please refer to the link below for the assembler directives

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

### A few useful assembler directives

**.text**  - indicates that following items are stored in the user text segment, typically instructions

**.data** - indicates that following data items are stored in the data segment

**.globl** sym – declare that symbol **sym** is global and can be referenced from other files

**.word** w1,….,wn – store n 32-bit quantities in successive memory words

**.half** h1,…,hn – store n 16-bit quantities in successive memory words

**.byte** b1,…,bn – store n 8-bit quantities in successive memory bytes

**.ascii** str – store the string in memory but do not null terminate it. The strings are represented in double quotes “str”

**.asciiz** str – store the string in memory and null-terminate it

### Useful pseudo-instructions

**la s0, addr:** Load address into register s0

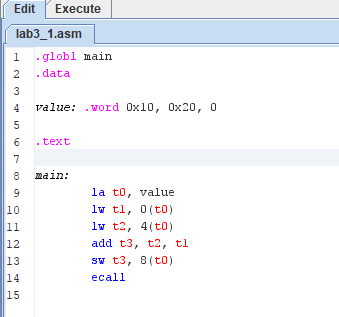
**lw t0, address:** Load a word at address into register t0

**li t0, value:** Load immediate value in to the register

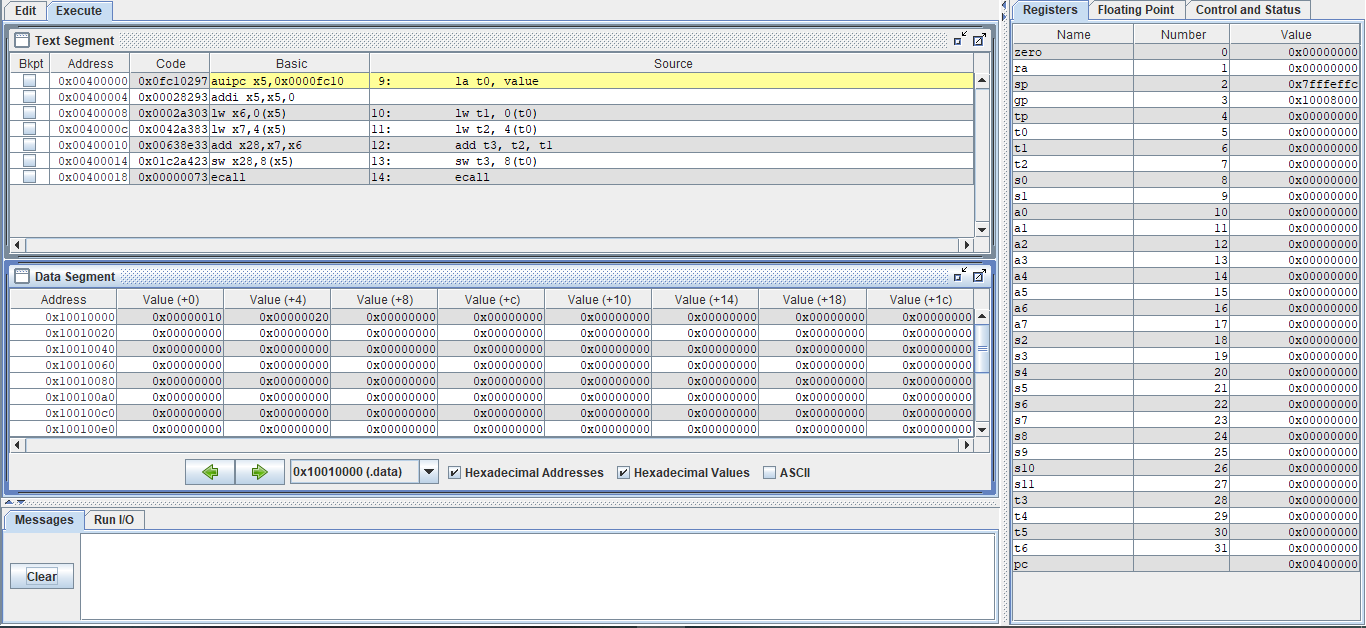
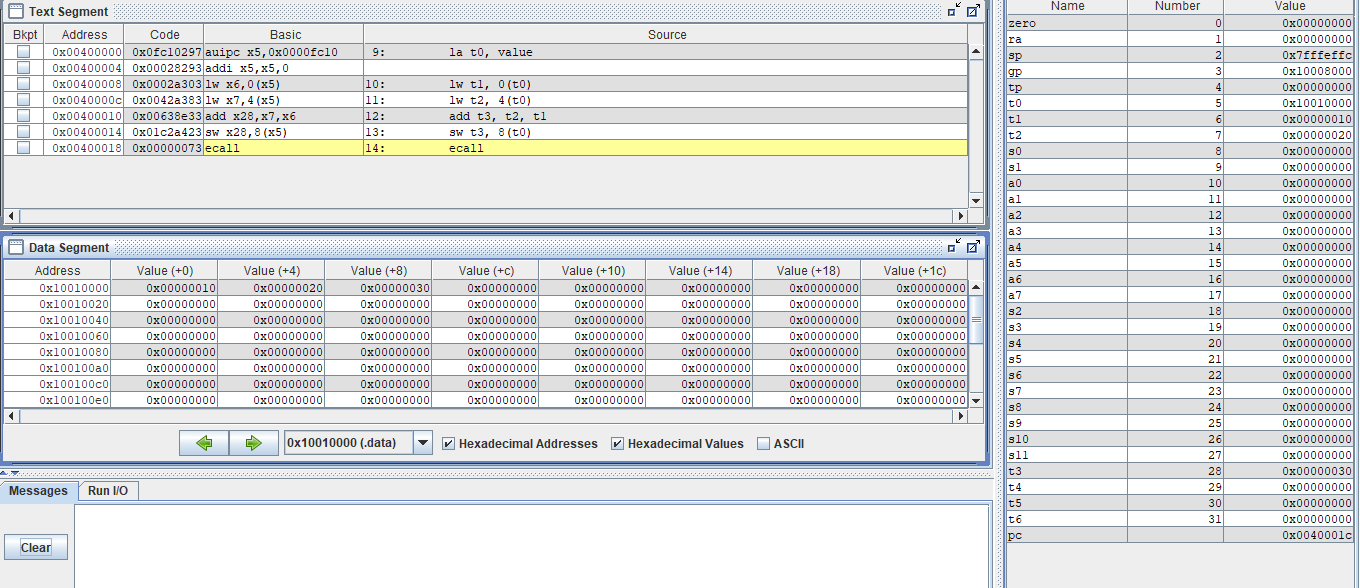
**move t0, t1**: move contents of register t1 to t0

**Exercise 3.1: Write RISC V assembly program to load two numbers from data segment, add them and store the result back to the data segment. (Code on page 2)**

1. **Copy image of assembly code for above exercise here. (In space below if you want to go to next line use “Shift-Enter”)**

Answer: 

1. **Copy the image of data segment before execution and after execution. Copy inputs and outputs of the program in your observation book.**

Answer: Before:   
  
  
After:   


**Exercise 3.2: Store N words in data segment (using .data) and write RISC V assembly code to add a constant value 5 to all the N words. The value of N is also stored in data segment. (Choose N value to be greater than or equal to 10d)**

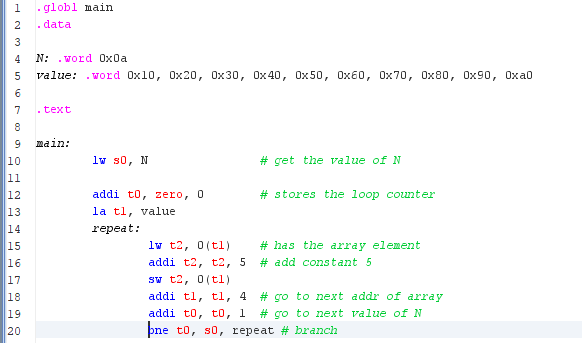
For example: .data

N: .word 0x0a

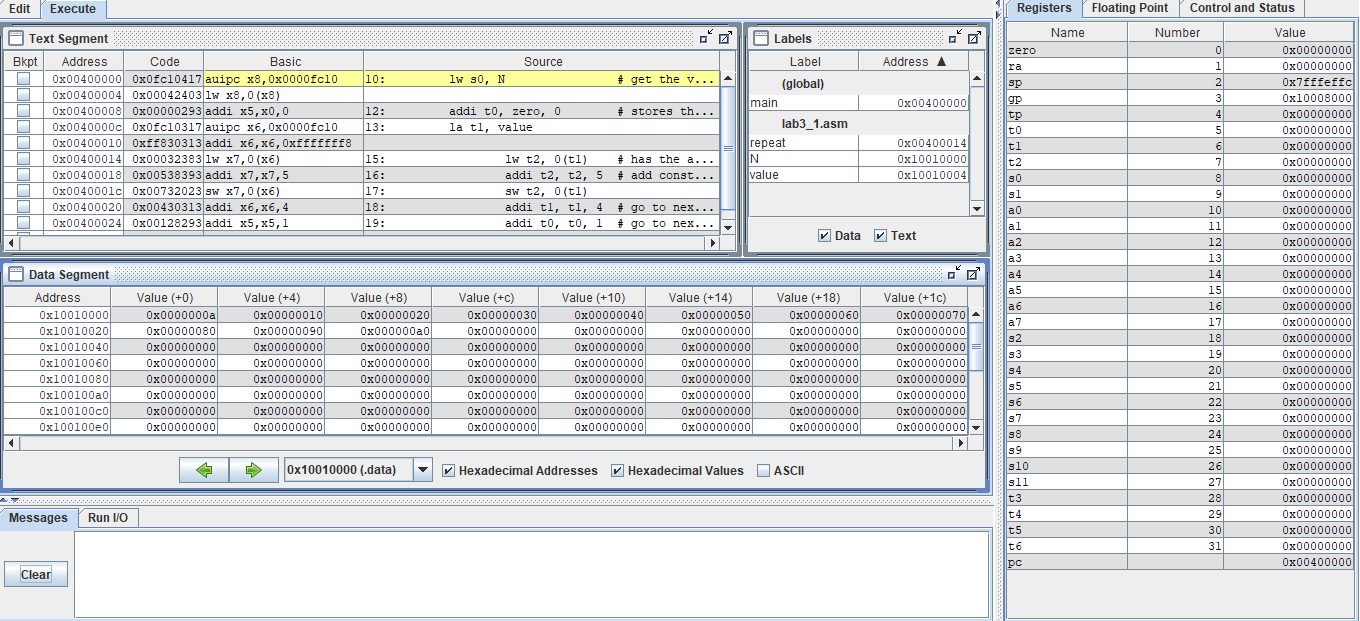
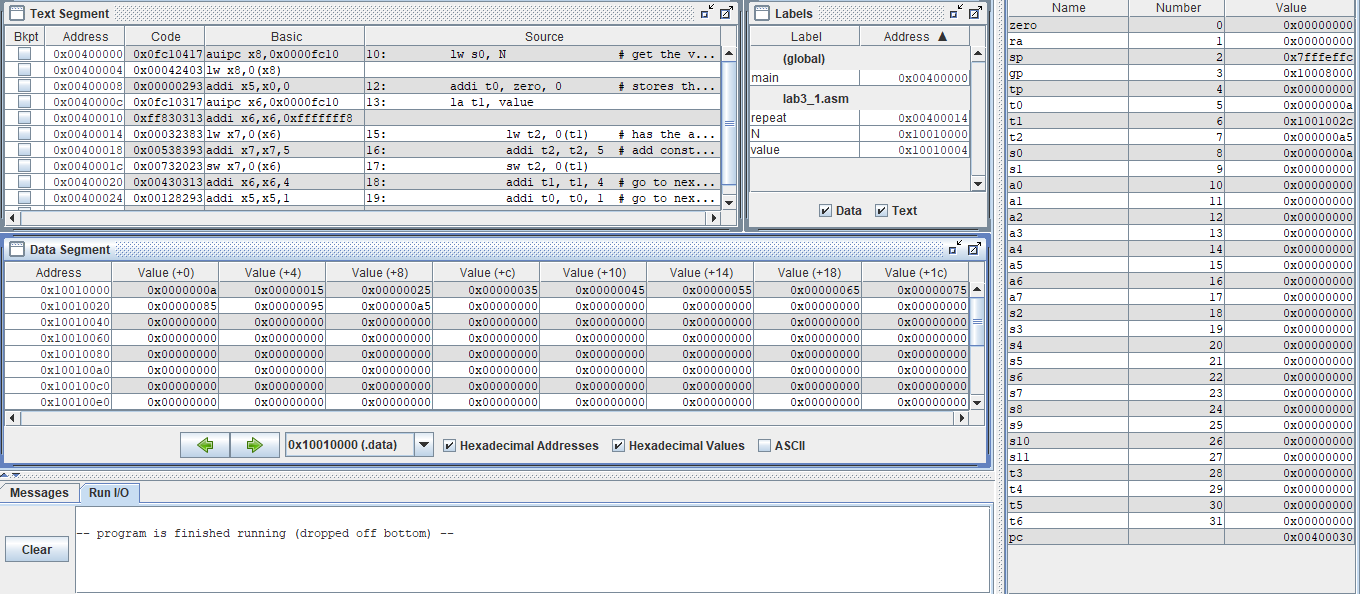
Value: .word 0x32, 0x20, 0x12, 0x45, 0x56, 0x21, 0x67, 0x10, 0x67, 0x90

(Hint for programming: You can use pseudo instructions **lw t0, N; la s0, Value**)

1. **Copy image of assembly code for above exercise here. Also write the code in your observation notebook.**

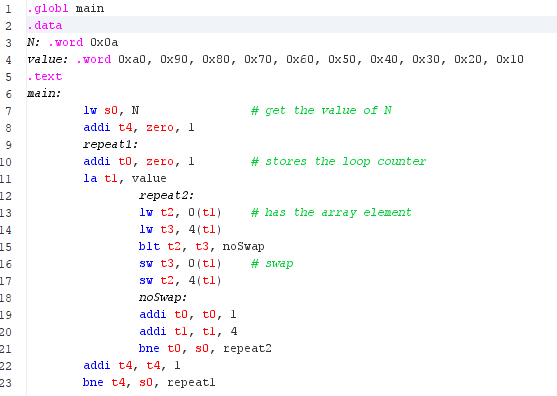
Answer: 

1. **Copy the image of data segment before execution and after execution for this program.**

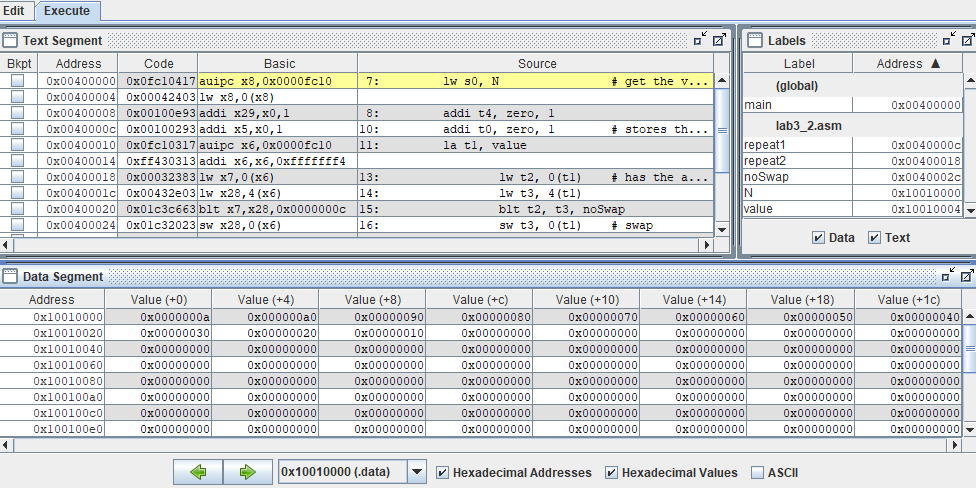
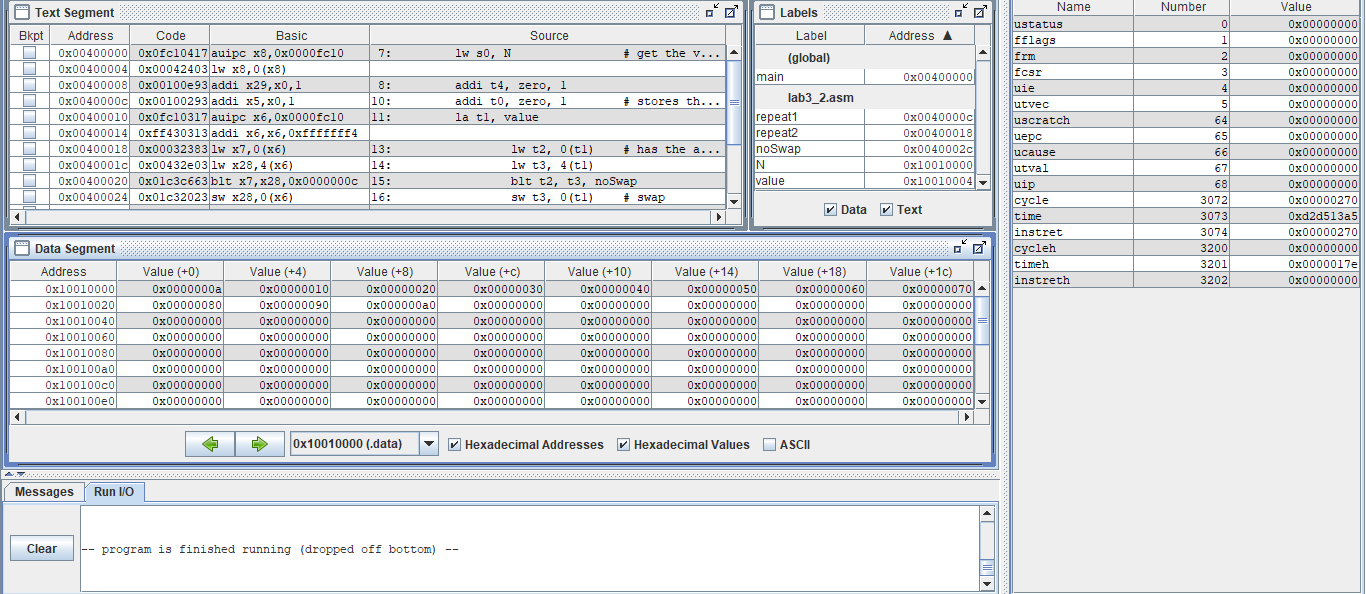
Answer: Before:   
  
After:   


**Exercise 4.3: Store N words in data segment (using .data) and write RISC V assembly code to arrange them in ascending order.**

1. **Copy image of your assembly code for above exercise here. Show the output of this program to the Lab instructor. Also write the code in your observation notebook.**

Answer: ****

1. **Copy the image of data segment before execution and after execution for this program.**

Answer: Before:  
  
  
After:   


**General questions**

1. **Test all the instructions discussed in class using the RARs tool. Also verify the corresponding instruction codes. List the instructions, their instruction codes and brief working of all the instructions that you have tested,**

Answer: auipc = 0fc10417 = add upper immediate to pc  
 lw = 00042403 = load word from memory  
sw = 01c32023 = store word in the memory   
addi = 00100e93= add constant to register   
blt = 01c3c663 = branch if less than   
bne = fe8292e3 = branch is not equal

1. **RISC V can be classified into which kind of architecture?**

**A. Stack based**

**B. Memory base**

**C. Load-Store**

**D. Register-Memory**

Answer: Load-Store

1. **What is the default starting address of Data Memory and Text Memory?**

Answer: data = 0x10010000, text = 0x00400000

1. **List the concepts you learnt from this experiment.(Conlcusion/observations)**

Answer: Learnt assembly programming and how to use the RARs tool