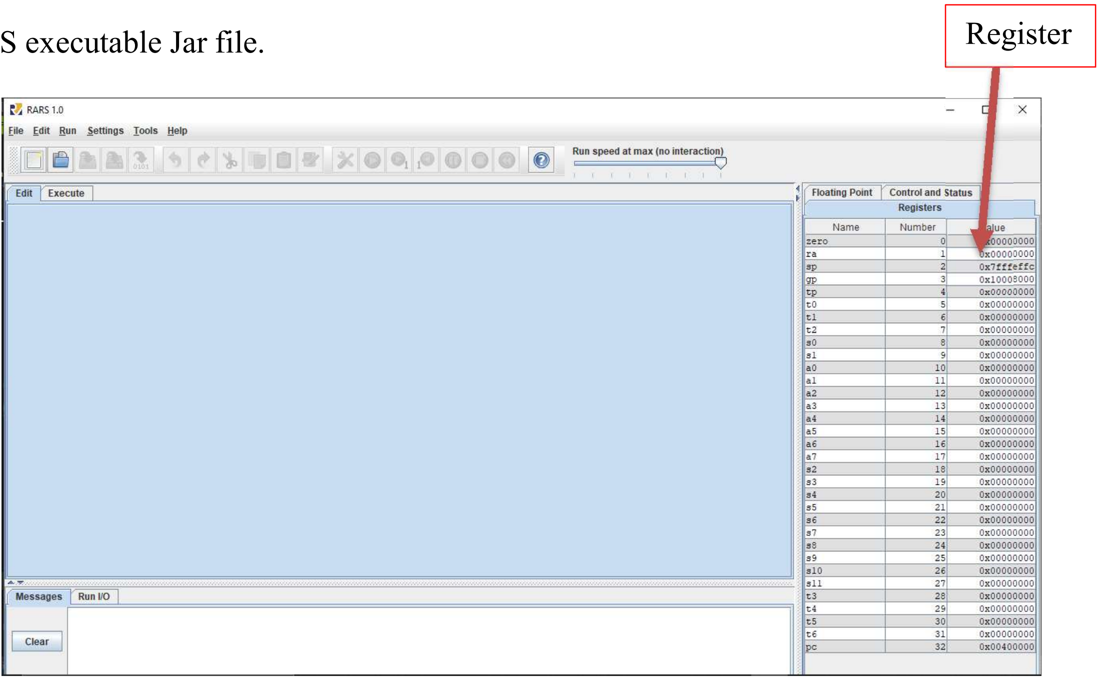
Experiment 3: Introduction to RARS, RISC V Assembly Programming

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

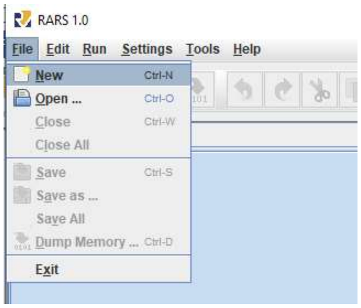
Open RARS executable Jar file.


Register

RARS 1.0
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Messages Run I/O
Clear
Floating Point Control and Status
Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
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	32	0x00400000

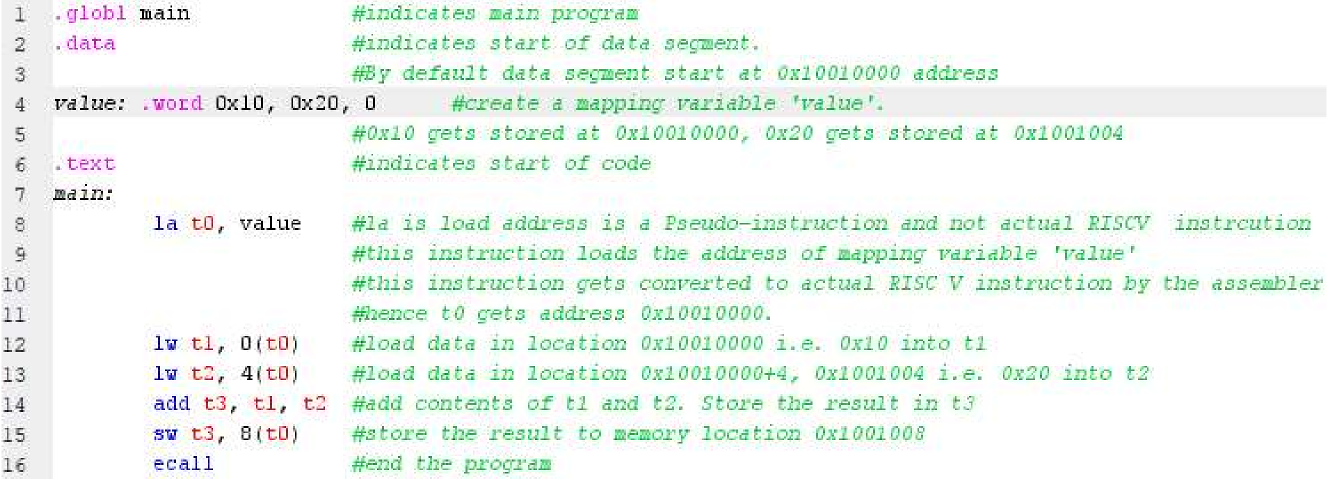
To create a new assembly program, click on File →New



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**

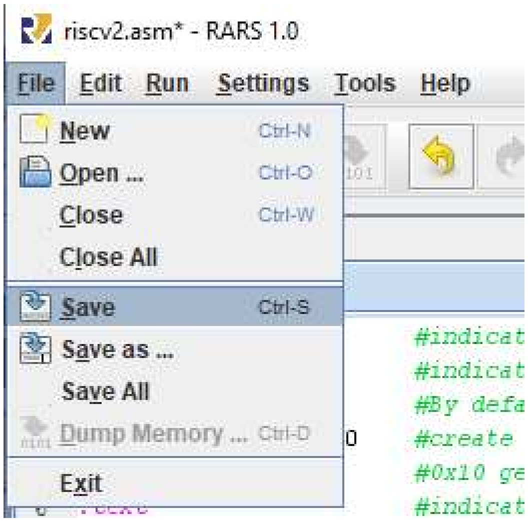
Below is the assembly code for this experiment. Please go through comments in the code for more details.



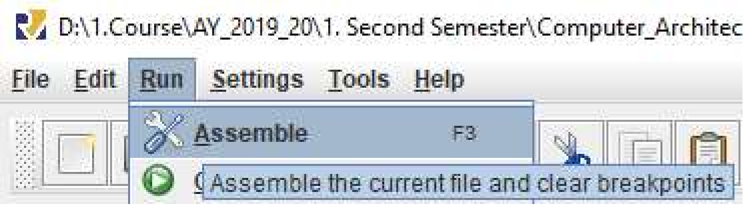
```
.globl main                 #indicates main program
.data                       #indicates start of data segment.
                            #By default data segment start at 0x10010000 address
value: .word 0x10, 0x20, 0      #create a mapping variable 'value'.
                            #0x10 gets stored at 0x10010000, 0x20 gets stored at 0x1001004
.text                       #indicates start of code
main:
        la t0, value        #la is load address is a Pseudo-instruction and not actual RISCV  instrcution
                            #this instruction loads the address of mapping variable 'value'
                            #this instruction gets converted to actual RISC V instruction by the assembler
                            #hence t0 gets address 0x10010000.
        lw t1, 0(t0)        #load data in location 0x10010000 i.e. 0x10 into t1
        lw t2, 4(t0)        #load data in location 0x10010000+4, 0x1001004 i.e. 0x20 into t2
        add t3, t1, t2      #add contents of t1 and t2. Store the result in t3
        sw t3, 8(t0)        #store the result to memory location 0x1001008
        ecall               #end the program
```

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.

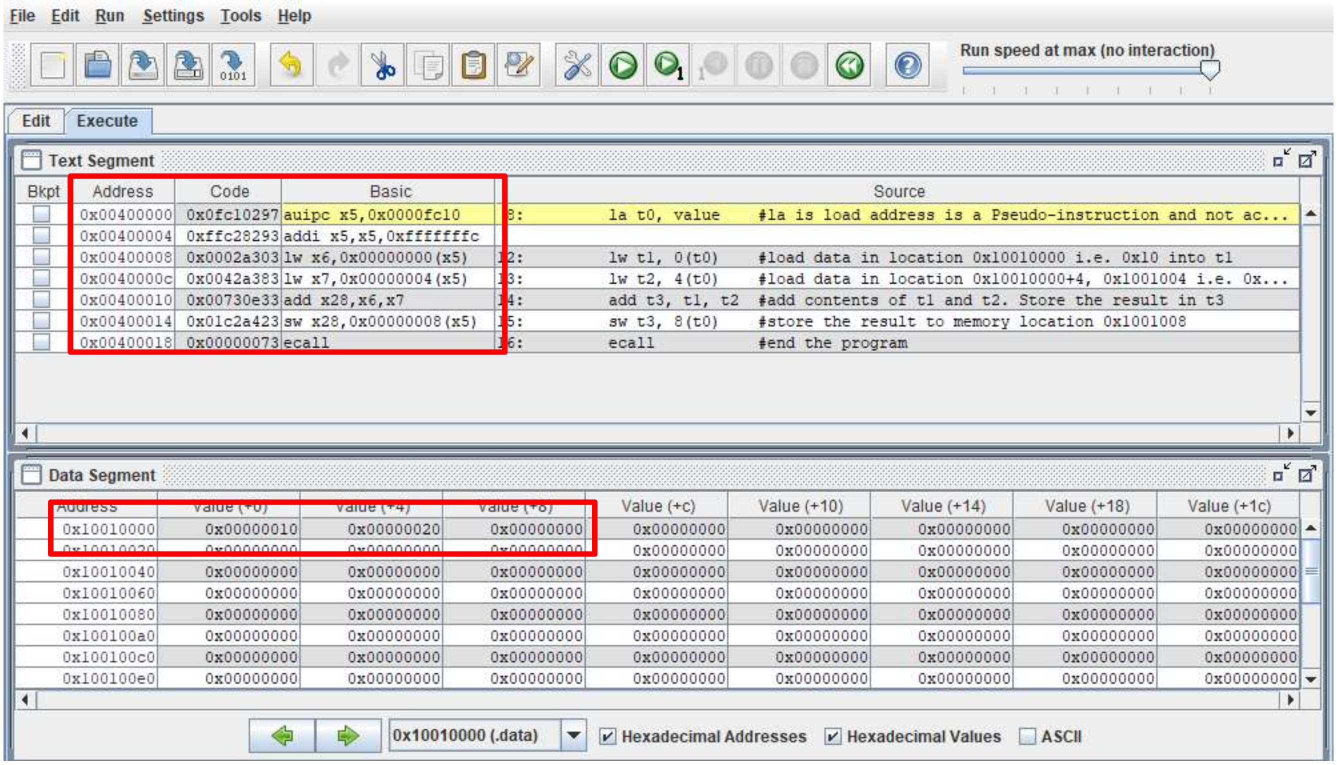
2. Save the assembly file with an extension *.asm* or *.s*.



3. After saving the code, assemble it by clicking **Run→Assemble**


D:\1.Course\AY_2019_20\1. Second Semester\Computer_Architec
File
Edit
Run
Settings
Tools
Help
Assemble
F3
Assemble the current file and clear breakpoints

4. 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.



Text Segment

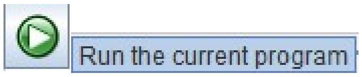
Bkpt	Address	Code	Basic		Source
□	0x00400000	0x0fc10297	auipc x5,0x0000fc10	8:	la t0, value #la is load address is a Pseudo-instruction and not ac...
□	0x00400004	0xffc28293	addi x5,x5,0xfffffffc		
□	0x00400008	0x0002a303	lw x6,0x00000000(x5)	12:	lw t1, 0(t0) #load data in location 0x10010000 i.e. 0x10 into t1
□	0x0040000c	0x0042a383	lw x7,0x00000004(x5)	13:	lw t2, 4(t0) #load data in location 0x10010000+4, 0x1001004 i.e. 0x...
□	0x00400010	0x00730e33	add x28,x6,x7	14:	add t3, t1, t2 #add contents of t1 and t2. Store the result in t3
□	0x00400014	0x01c2a423	sw x28,0x00000008(x5)	15:	sw t3, 8(t0) #store the result to memory location 0x1001008
□	0x00400018	0x00000073	ecall	16:	ecall #end the program

Data Segment

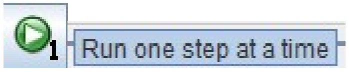
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000010	0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	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

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

5. 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.



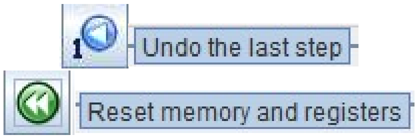
The second option, (which is preferred, to understand and debug the code) is to run the program one instruction at a time, by clicking "Run one step at a time" button.



6. 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

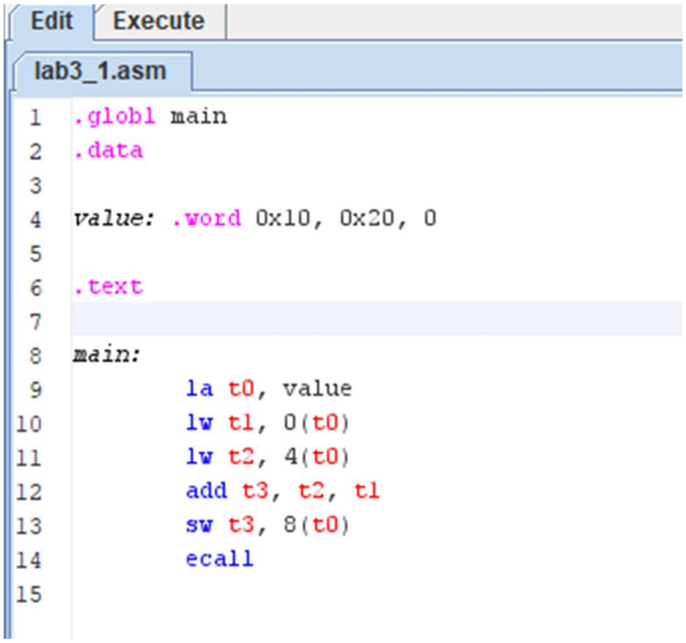
.text - indicates that following items are stored in the user text segment, typically instructions
.data - indicates that following data items are stored in the data segment
.globl sym – declare that symbol **sym** is global and can be referenced from other files
.word w1,….,wn – store n 32-bit quantities 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)

Q3.1. Copy image of assembly code for above exercise here. (In space below if you want to go to next line use “Shift-Enter”)

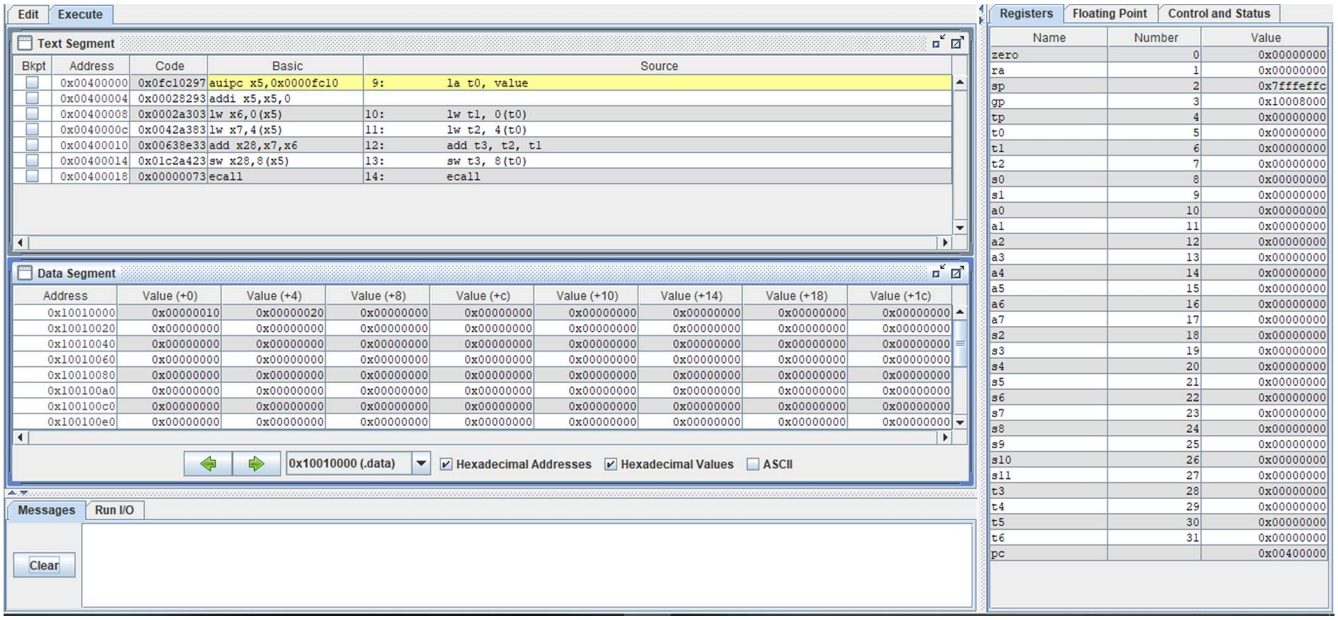


```
.globl main
.data

value: .word 0x10, 0x20, 0

.text

main:
        la t0, value
        lw t1, 0(t0)
        lw t2, 4(t0)
        add t3, t2, t1
        sw t3, 8(t0)
        ecall

```

Answer:

Q3.2. Copy the image of data segment before execution and after execution. Copy inputs and outputs of the program in your observation book.

Answer: Before:



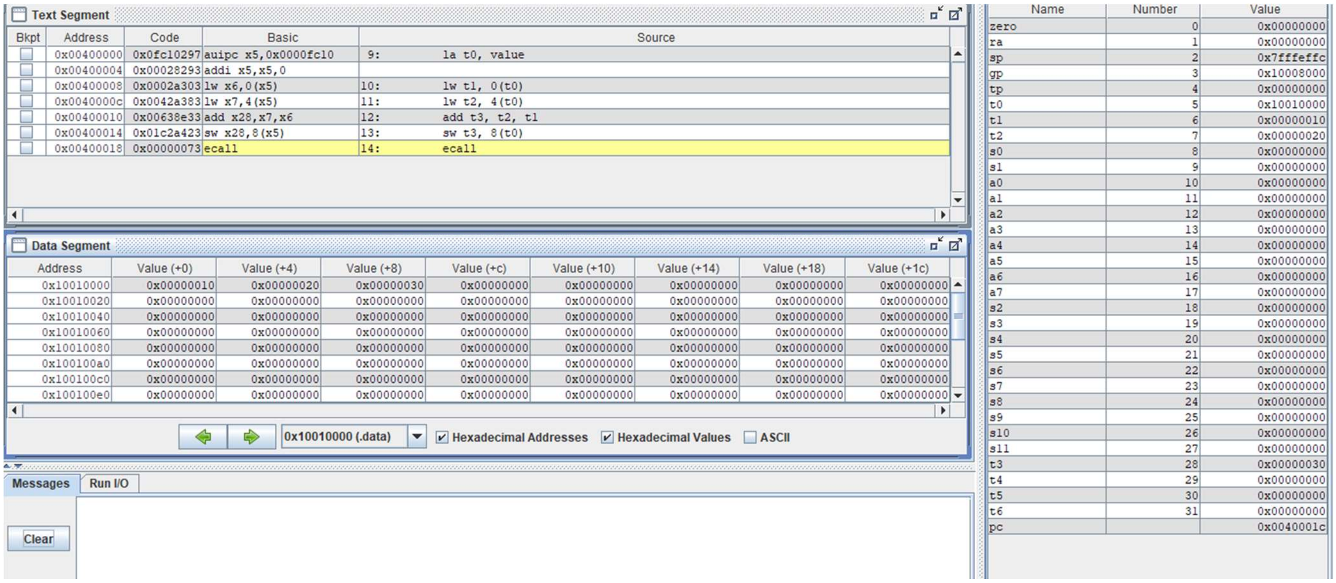
Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x0fc10297	auipc x5,0x0000fc10	9: la t0, value
☐	0x00400004	0x00028293	addi x5,x5,0	
☐	0x00400008	0x0002a303	lw x6,0(x5)	10: lw t1, 0(t0)
☐	0x0040000c	0x0042a383	lw x7,4(x5)	11: lw t2, 4(t0)
☐	0x00400010	0x00638e33	add x28,x7,x6	12: add t3, t2, t1
☐	0x00400014	0x01c2a423	sw x28,8(x5)	13: sw t3, 8(t0)
☐	0x00400018	0x00000073	ecall	14: ecall

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000010	0x00000020	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

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	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
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		0x00400000

After:



Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x0fc10297	auipc x5,0x0000fc10	9: la t0, value
☐	0x00400004	0x00028293	addi x5,x5,0	
☐	0x00400008	0x0002a303	lw x6,0(x5)	10: lw t1, 0(t0)
☐	0x0040000c	0x0042a383	lw x7,4(x5)	11: lw t2, 4(t0)
☐	0x00400010	0x00638e33	add x28,x7,x6	12: add t3, t2, t1
☐	0x00400014	0x01c2a423	sw x28,8(x5)	13: sw t3, 8(t0)
☐	0x00400018	0x00000073	ecall	14: ecall

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000010	0x00000020	0x00000030	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

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

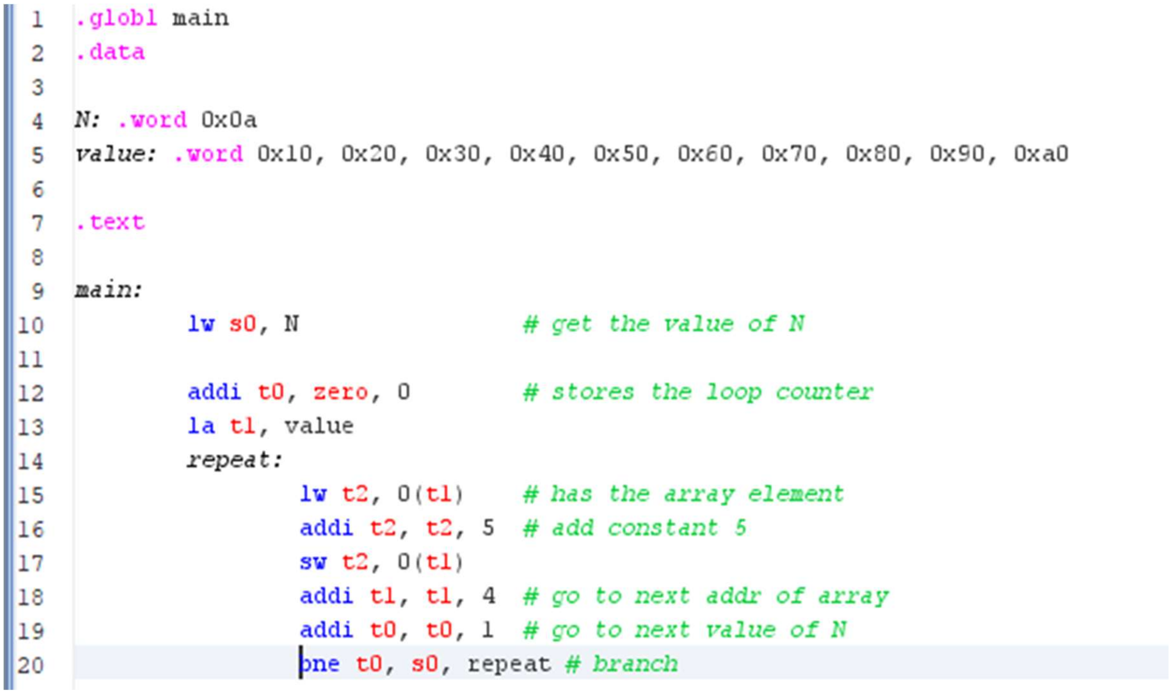
Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x10010000
t1	6	0x00000010
t2	7	0x00000020
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
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	0x00000030
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040001c

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 10_d)

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**)

Q3.3. Copy image of assembly code for above exercise here. Also write the code in your observation notebook.

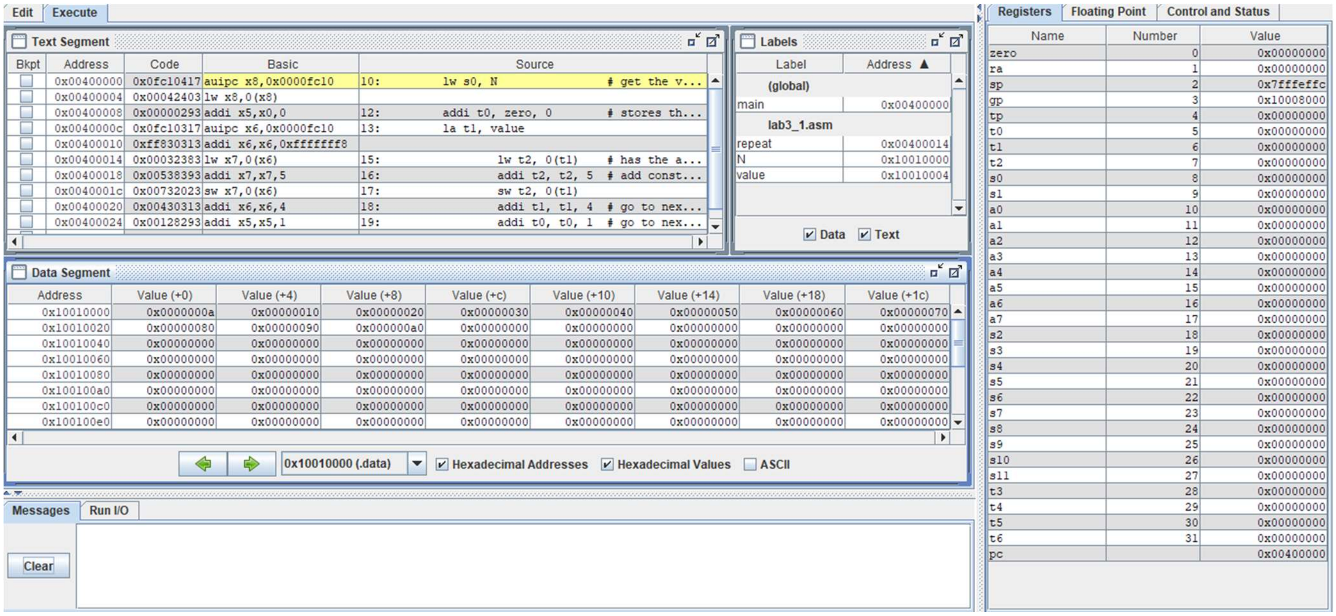
Answer:



```
.globl main
.data

N: .word 0x0a
value: .word 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0xa0

.text

main:
        lw s0, N                # get the value of N

        addi t0, zero, 0        # stores the loop counter
        la t1, value
        repeat:
                lw t2, 0(t1)    # has the array element
                addi t2, t2, 5  # add constant 5
                sw t2, 0(t1)
                addi t1, t1, 4  # go to next addr of array
                addi t0, t0, 1  # go to next value of N
                bne t0, s0, repeat # branch
```

Q3.4. Copy the image of data segment before execution and after execution for this program.

Answer: Before:



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x0fc10417	auipc x8,0x0000fc10	10: lw s0, N # get the v...
□	0x00400004	0x00042403	lw x8,0(x8)	
□	0x00400008	0x00000293	addi x5,x0,0	12: addi t0, zero, 0 # stores th...
□	0x0040000c	0x0fc10317	auipc x6,0x0000fc10	13: la t1, value
□	0x00400010	0xff830313	addi x6,x6,0xfffffff8	
□	0x00400014	0x00032383	lw x7,0(x6)	15: lw t2, 0(t1) # has the a...
□	0x00400018	0x00538393	addi x7,x7,5	16: addi t2, t2, 5 # add const...
□	0x0040001c	0x00732023	sw x7,0(x6)	17: sw t2, 0(t1)
□	0x00400020	0x00430313	addi x6,x6,4	18: addi t1, t1, 4 # go to nex...
□	0x00400024	0x00128293	addi x5,x5,1	19: addi t0, t0, 1 # go to nex...

Labels

Label	Address ▲
(global)	
main	0x00400000
lab3_1.asm	
repeat	0x00400014
N	0x10010000
value	0x10010004

☑ Data ☑ Text

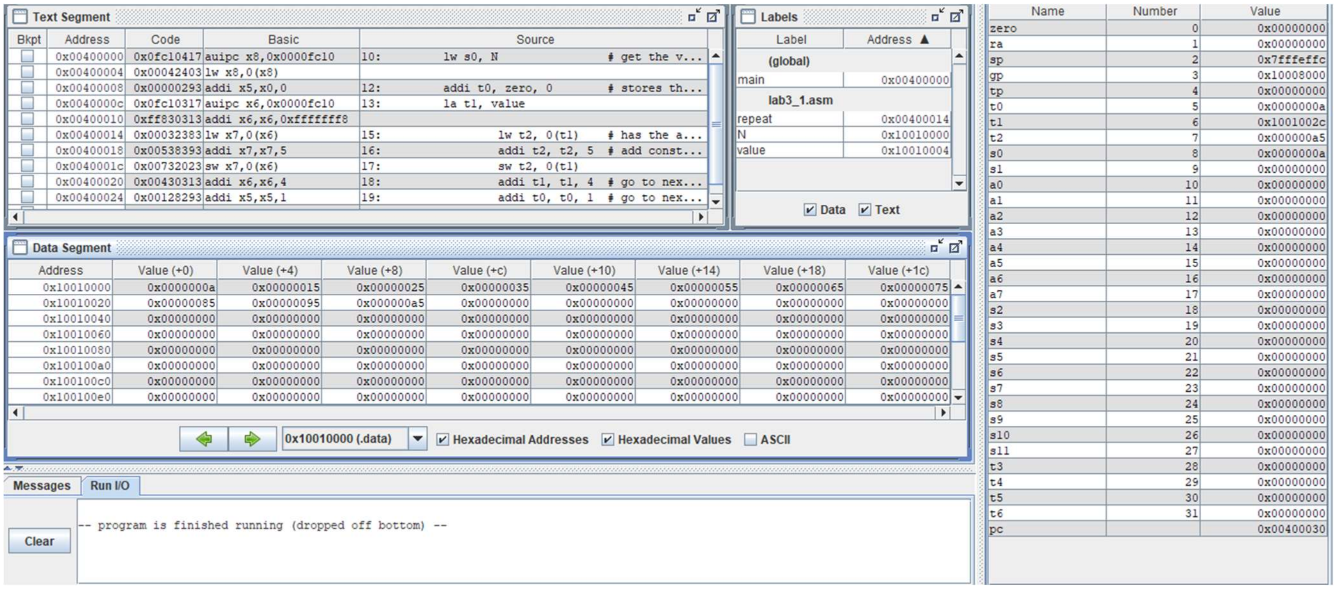
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000010	0x00000020	0x00000030	0x00000040	0x00000050	0x00000060	0x00000070
0x10010020	0x00000080	0x00000090	0x000000a0	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

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	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
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		0x00400000

After:



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x0fc10417	auipc x8,0x0000fc10	10: lw s0, N # get the v...
□	0x00400004	0x00042403	lw x8,0(x8)	
□	0x00400008	0x00000293	addi x5,x0,0	12: addi t0, zero, 0 # stores th...
□	0x0040000c	0x0fc10317	auipc x6,0x0000fc10	13: la t1, value
□	0x00400010	0xff830313	addi x6,x6,0xfffffff8	
□	0x00400014	0x00032383	lw x7,0(x6)	15: lw t2, 0(t1) # has the a...
□	0x00400018	0x00538393	addi x7,x7,5	16: addi t2, t2, 5 # add const...
□	0x0040001c	0x00732023	sw x7,0(x6)	17: sw t2, 0(t1)
□	0x00400020	0x00430313	addi x6,x6,4	18: addi t1, t1, 4 # go to nex...
□	0x00400024	0x00128293	addi x5,x5,1	19: addi t0, t0, 1 # go to nex...

Labels

Label	Address ▲
(global)	
main	0x00400000
lab3_1.asm	
repeat	0x00400014
N	0x10010000
value	0x10010004

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000015	0x00000025	0x00000035	0x00000045	0x00000055	0x00000065	0x00000075
0x10010020	0x00000085	0x00000095	0x000000a5	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

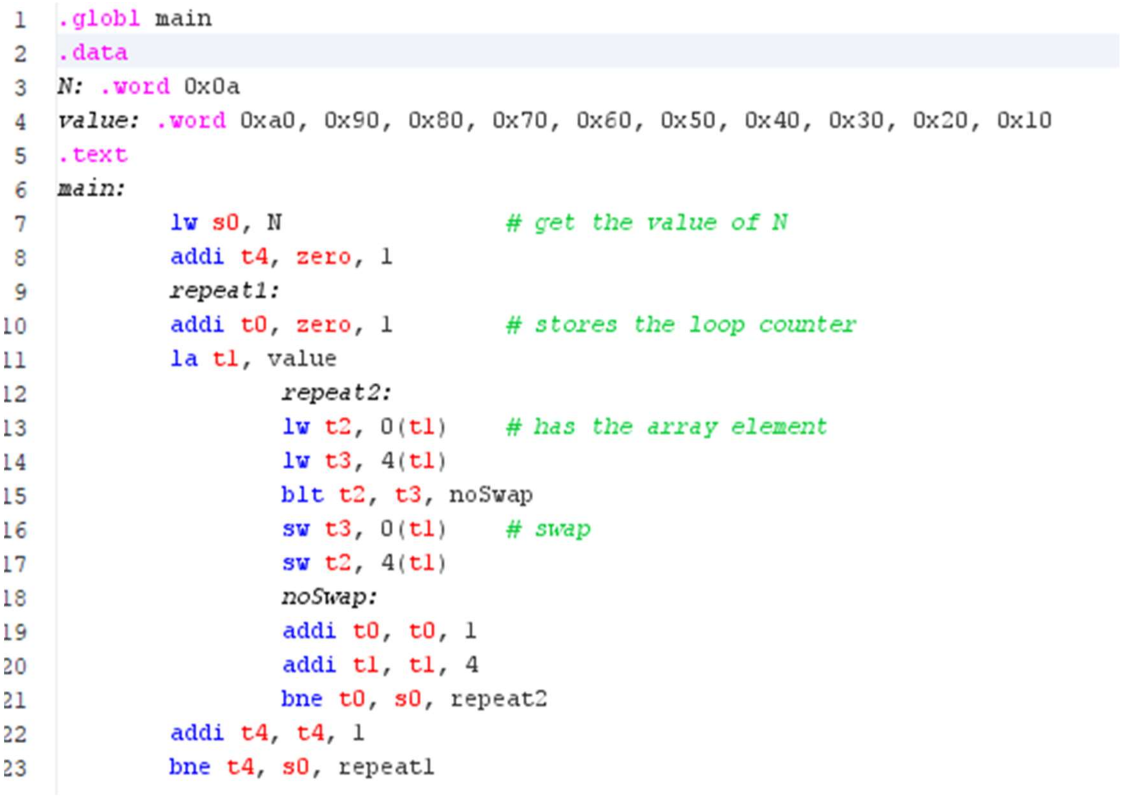
0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
-- program is finished running (dropped off bottom) --
```

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x0000000a
t1	6	0x1001002c
t2	7	0x000000a5
s0	8	0x0000000a
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
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

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

Q3.5. 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.

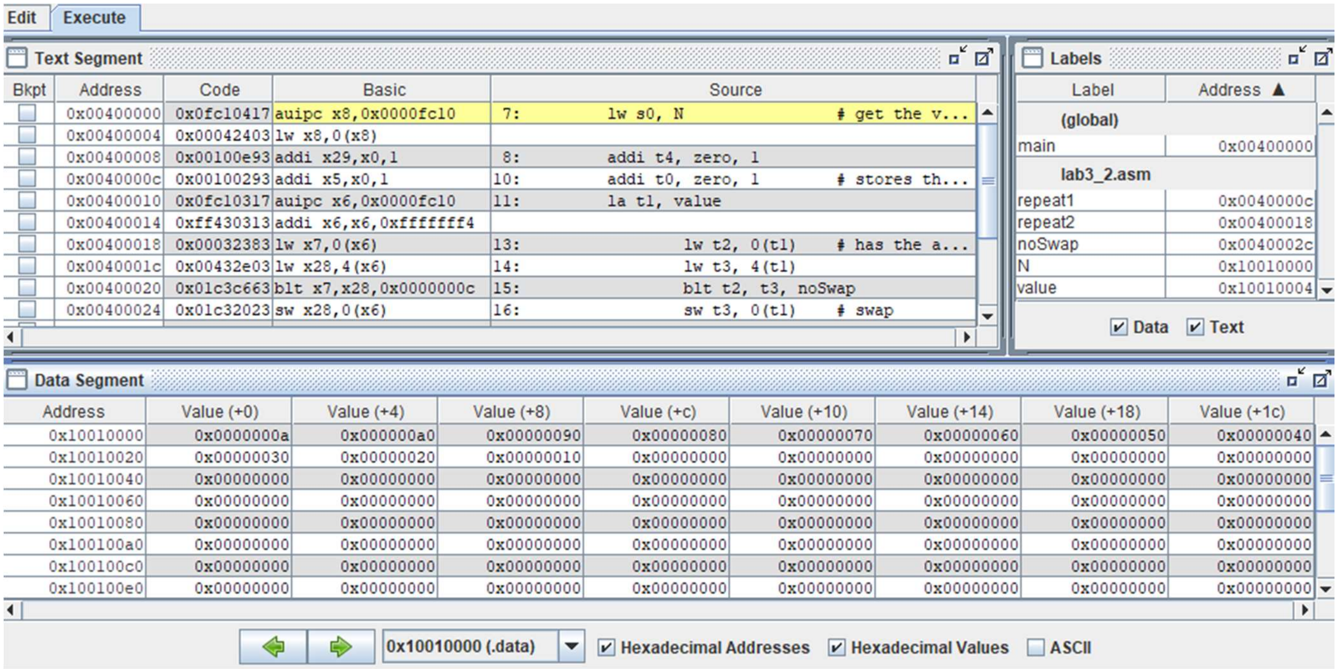


```
.globl main
.data
N: .word 0x0a
value: .word 0xa0, 0x90, 0x80, 0x70, 0x60, 0x50, 0x40, 0x30, 0x20, 0x10
.text
main:
        lw s0, N                # get the value of N
        addi t4, zero, 1
        repeat1:
        addi t0, zero, 1        # stores the loop counter
        la t1, value
                repeat2:
                lw t2, 0(t1)    # has the array element
                lw t3, 4(t1)
                blt t2, t3, noSwap
                sw t3, 0(t1)    # swap
                sw t2, 4(t1)
                noSwap:
                addi t0, t0, 1
                addi t1, t1, 4
                bne t0, s0, repeat2
        addi t4, t4, 1
        bne t4, s0, repeat1
```

Answer:

Q3.6. Copy the image of data segment before execution and after execution for this program.

Answer: Before:



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x0fc10417	auipc x8,0x0000fc10	7: lw s0, N # get the v...
□	0x00400004	0x00042403	lw x8,0(x8)	
□	0x00400008	0x00100e93	addi x29,x0,1	8: addi t4, zero, 1
□	0x0040000c	0x00100293	addi x5,x0,1	10: addi t0, zero, 1 # stores th...
□	0x00400010	0x0fc10317	auipc x6,0x0000fc10	11: la t1, value
□	0x00400014	0xff430313	addi x6,x6,0xfffffff4	
□	0x00400018	0x00032383	lw x7,0(x6)	13: lw t2, 0(t1) # has the a...
□	0x0040001c	0x00432e03	lw x28,4(x6)	14: lw t3, 4(t1)
□	0x00400020	0x01c3c663	blt x7,x28,0x0000000c	15: blt t2, t3, noSwap
□	0x00400024	0x01c32023	sw x28,0(x6)	16: sw t3, 0(t1) # swap

Labels

Label	Address ▲
(global)	
main	0x00400000
lab3_2.asm	
repeat1	0x0040000c
repeat2	0x00400018
noSwap	0x0040002c
N	0x10010000
value	0x10010004

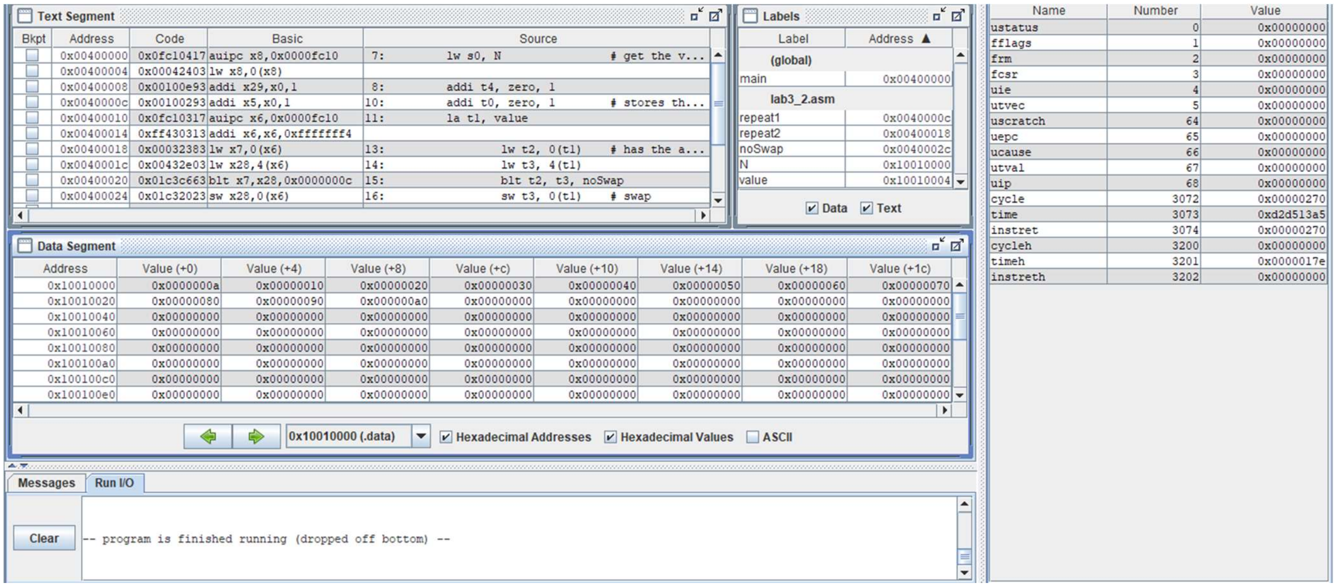
☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x000000a0	0x00000090	0x00000080	0x00000070	0x00000060	0x00000050	0x00000040
0x10010020	0x00000030	0x00000020	0x00000010	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

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

After:



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x0fc10417	auipc x8,0x0000fc10	7: lw s0, N # get the v...
□	0x00400004	0x00042403	lw x8,0(x8)	
□	0x00400008	0x00100e93	addi x29,x0,1	8: addi t4, zero, 1
□	0x0040000c	0x00100293	addi x5,x0,1	10: addi t0, zero, 1 # stores th...
□	0x00400010	0x0fc10317	auipc x6,0x0000fc10	11: la t1, value
□	0x00400014	0xff430313	addi x6,x6,0xfffffff4	
□	0x00400018	0x00032383	lw x7,0(x6)	13: lw t2, 0(t1) # has the a...
□	0x0040001c	0x00432e03	lw x28,4(x6)	14: lw t3, 4(t1)
□	0x00400020	0x01c3c663	blt x7,x28,0x0000000c	15: blt t2, t3, noSwap
□	0x00400024	0x01c32023	sw x28,0(x6)	16: sw t3, 0(t1) # swap

Labels

Label	Address ▲
(global)	
main	0x00400000
lab3_2.asm	
repeat1	0x0040000c
repeat2	0x00400018
noSwap	0x0040002c
N	0x10010000
value	0x10010004

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000010	0x00000020	0x00000030	0x00000040	0x00000050	0x00000060	0x00000070
0x10010020	0x00000080	0x00000090	0x000000a0	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

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

```
-- program is finished running (dropped off bottom) --
```

Name	Number	Value
ustatus	0	0x00000000
fflags	1	0x00000000
frm	2	0x00000000
fcsr	3	0x00000000
uie	4	0x00000000
utvec	5	0x00000000
uscratch	64	0x00000000
uepc	65	0x00000000
ucause	66	0x00000000
utval	67	0x00000000
uip	68	0x00000000
cycle	3072	0x00000270
time	3073	0xd2d513a5
instret	3074	0x00000270
cycleh	3200	0x00000000
timeh	3201	0x0000017e
instreth	3202	0x00000000

General questions

Q3.7. 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

Q3.8. 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

Q3.9. What is the default starting address of Data Memory and Text Memory?

Answer: data = 0x10010000, text = 0x00400000

Q3.10. List the concepts you learnt from this experiment.(Conlcusion/observations)

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