ASHWIN ANAND - COMPUTER ARCHITECTURE LAB 4

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Course Name: Computer Architecture Lab

Course Number and Section: 14:332:333:01

Experiment: Lab # 4 – RISC-V Assembly

Lab Instructor: Mingbo Zhang

Date Performed: March 31st, 2021

Date Submitted: April 9th, 2021

Submitted by: Ashwin Anand - 192007894

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Computer Architecture and Assembly Lab
Spring 2021

Lab 4
RISC-V Assembly

Instructions

Please complete all the exercises below. Use the Venus RISC-V simulator to test your code. Note that the simulator is 32-bit, and we do not consider overflow here.
Be sure to comment on your code for clarity.
Once complete, upload your source code and PDF lab report on Sakai.

Exercises

1. [20 pts] Write a function in RISC-V that computes the following:
   - When given a positive integer **x** as an input, return **10x** using only **add** instructions.
   - When given a negative integer **y** as an input, compute the opposite value (i.e. **-y**) *without* using **mul** instructions.

Have a **main()** function check the value of the input, perform the appropriate computations, and print the output in the console. Note: you need to provide your own inputs and show screenshots of the outputs based on the given inputs. (Each part worth 10 pts.)

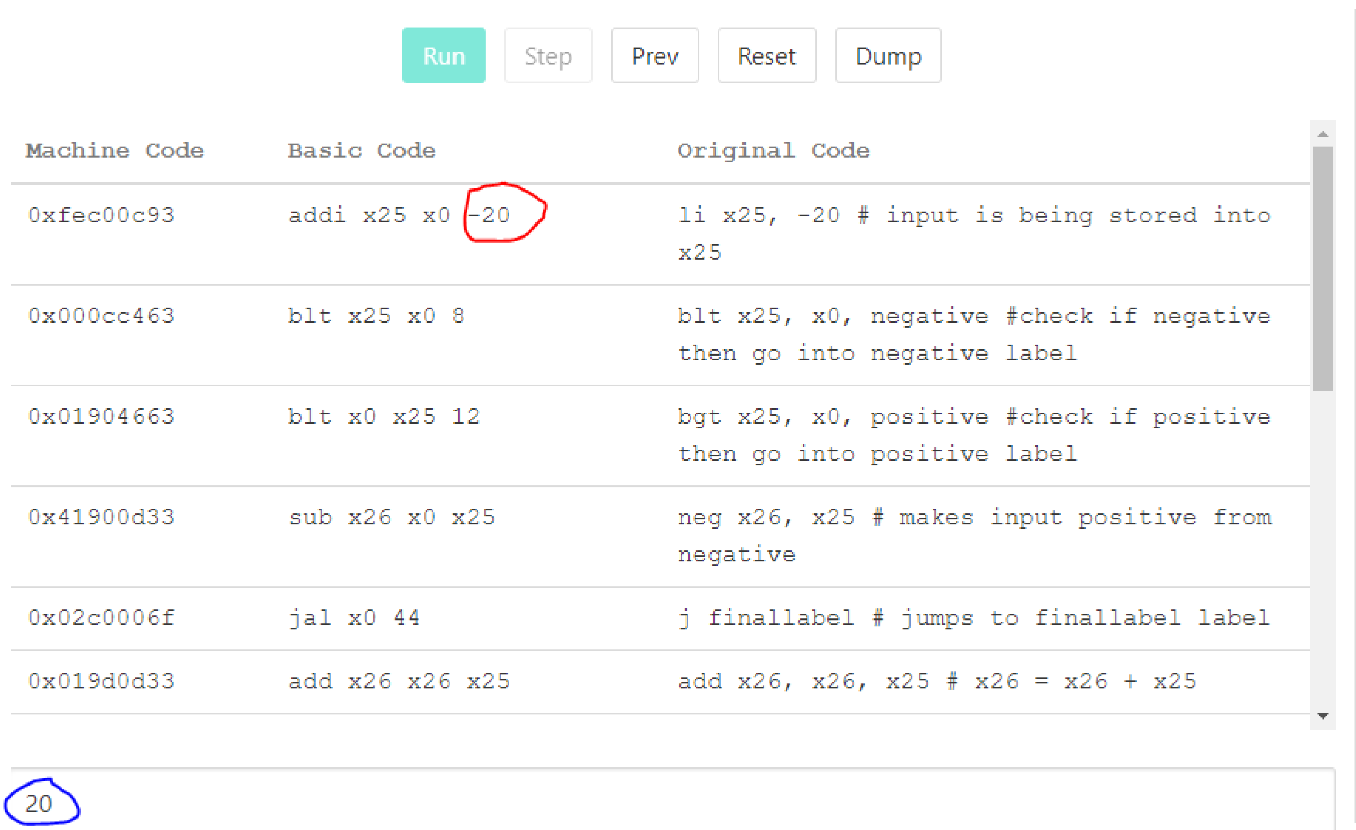
Code:

```
main:
li x25, -20 # input is being stored into x25
blt x25, x0, negative #check if negative then go into negative label
bgt x25, x0, positive #check if positive then go into positive label

negative: #negative label if input is negative
neg x26, x25   # makes input positive from negative
j finallabel # jumps to finallabel label

positive: #positive label if input if positive
#10 add lines for 10*input
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
```

```
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25
add x26, x26, x25 # x26 = x26 + x25

finallabel:
addi a0, x0, 1  # a0 = 0 + 1
add a1, x0, x26  # a1 = 0 + x26
ecall                    # print to console
```

Screenshots:

<u>Test Case 1:</u>
Input = - 20 as seen in red circle
Output = 20 as seen in blue circle



Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

<u>Test Case 2:</u>
Input = 20 as seen in red circle
Output = 200 as seen in blue circle

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x01400c93	addi x25 x0 20	li x25, 20 # input is being stored into x25
0x000cc463	blt x25 x0 8	blt x25, x0, negative #check if negative then go into negative label
0x01904663	blt x0 x25 12	bgt x25, x0, positive #check if positive then go into positive label
0x41900d33	sub x26 x0 x25	neg x26, x25 # makes input positive from negative
0x02c0006f	jal x0 44	j finallabel # jumps to finallabel label
0x019d0d33	add x26 x26 x25	add x26, x26, x25 # x26 = x26 + x25

200

<u>Test Case 3:</u>
Input = 5 as seen in red circle
Output = 50 as seen in blue circle

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x00500c93	addi x25 x0 5	li x25, 5 # input is being stored into x25
0x000cc463	blt x25 x0 8	blt x25, x0, negative #check if negative then go into negative label
0x01904663	blt x0 x25 12	bgt x25, x0, positive #check if positive then go into positive label
0x41900d33	sub x26 x0 x25	neg x26, x25 # makes input positive from negative
0x02c0006f	jal x0 44	j finallabel # jumps to finallabel label
0x019d0d33	add x26 x26 x25	add x26, x26, x25 # x26 = x26 + x25

50

<u>Test Case 4:</u>
Input = -11 as seen in red circle
Output = 11 as seen in blue circle

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0xff500c93	addi x25 x0 -11	li x25, -11 # input is being stored into x25
0x000cc463	blt x25 x0 8	blt x25, x0, negative #check if negative then go into negative label
0x01904663	blt x0 x25 12	bgt x25, x0, positive #check if positive then go into positive label
0x41900d33	sub x26 x0 x25	neg x26, x25 # makes input positive from negative
0x02c0006f	jal x0 44	j finallabel # jumps to finallabel label
0x019d0d33	add x26 x26 x25	add x26, x26, x25 # x26 = x26 + x25

11

<u>Test Case 5:</u>
Input = 520 as seen in red circle
Output = 5200 as seen in blue circle

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x20800c93	addi x25 x0 520	li x25, 520 # input is being stored into x25
0x000cc463	blt x25 x0 8	blt x25, x0, negative #check if negative then go into negative label
0x01904663	blt x0 x25 12	bgt x25, x0, positive #check if positive then go into positive label
0x41900d33	sub x26 x0 x25	neg x26, x25 # makes input positive from negative
0x02c0006f	jal x0 44	j finallabel # jumps to finallabel label
0x019d0d33	add x26 x26 x25	add x26, x26, x25 # x26 = x26 + x25

5200

<ins>Test Case 6:</ins>
Input = -520 as seen in red circle
Output = 520 as seen in blue circle

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0xdf800c93	addi x25 x0 -520	li x25, -520 # input is being stored into x25
0x000cc463	blt x25 x0 8	blt x25, x0, negative #check if negative then go into negative label
0x01904663	blt x0 x25 12	bgt x25, x0, positive #check if positive then go into positive label
0x41900d33	sub x26 x0 x25	neg x26, x25 # makes input positive from negative
0x02c0006f	jal x0 44	j finallabel # jumps to finallabel label
0x019d0d33	add x26 x26 x25	add x26, x26, x25 # x26 = x26 + x25

520

2. [30 pts] Write a function **exp()** in RISC-V that, when given a positive integer **x**, returns **x^5+6x^3+3x+4**. Your code should include the following:

- Write a **main()** function to call the **exp** function and print the output in the console. [5 pts]
- Function **exp()** will compute and return the expression. Function **exp** must call the following additional functions: [10 pts]
    - Call a **power()** function to calculate **x^n**. You may assume **n** will be positive. [10 pts]
    - Call a **times()** function to calculate **n*x**. [5 pts]

Note: you need to provide your own inputs and show screenshots of the outputs based on the given inputs.

Code:

```
j Main

exp:
addi x2, x2, -16  # making 2 spots in stack
sw x7, 8(x2)      # storing x7 as second spot in stack
sw x1, 0(x2)      # storing x1 as first spot in stack
addi x5, x0, 5    # x5 = x0 + 5

jal x1, power # jumping to power function
lw x1, 0(x2)      # storing new value of x1 into stack
add x7, x8, x7    # x7 = x8 + x7
addi x5, x0, 0    # x5 = x0 + 0
addi x5, x5, 3    # x5 = x5 + 3

jal x1, power # jumping to power function
lw x1, 0(x2)      # storing new value of x1 into stack
add x13, x0, x8   # x13 = x0 + x8
addi x12, x0, 6   # x12 = x0 + 6

jal x1, times # jumping to times function
lw x1, 0(x2)      # storing new value of x1 into stack
add x7, x30, x7   # x7 = x30 + x7
addi x12, x0, 3   # x12 = x0 + 3
add x13, x0, x6   # x13 = x0 + x6

jal x1, times # jumping to times function
lw x1, 0(x2)      # storing new value of x1 into stack
add x7, x30, x7   # x7 = x30 + x7
addi x7, x7, 4    # x7 = x7 + 4
lw x1, 0(x2)      # storing new value of x1 into stack
addi x2, x2, 16   # x2 = x2 + 16
jalr x0, 0(x1)    # leaves to Main Function

power:
```

```
addi x2, x2,-8   # making 1 spot in a stack
sw x1, 0(x2)     # storing x1 in 1st spot in a stack
add x8, x0, x0   # x8 = x0 + x0
add x15, x0, x0  # x15 = x0 + x0
addi x8, x8, 1   # x8 = x8 + 1
add x22, x0, x0  # x22 = x0 + x0

PLoop:
bge x15, x5, ExitPLoop # checking if x15 is greater than or equal to x5 and heads to ExitPowerLoop
mul x8, x8, x6 # x8 = x8 * x6
addi x15, x15, 1 # x15 = x15 + 1
jal x0, PLoop # jumping to PowerLoop

ExitPLoop:
lw x1, 0(x2)   # storing new value of x1 into stack
addi x2, x2, 8 # x2 = x2 + 8
jalr x0, 0(x1) # leaves to Main Function

times:
addi x2, x2, -8   # x2 = x2 + (-8)
sw x1, 0(x2)      # storing x1 values from stack
mul x30, x13, x12 # x30 = x13 * x12
lw x1, 0(x2)      # storing new value of x1 into stack
addi x2, x2, 8    # x2 = x2 + 8
jalr x0, 0(x1)    # leaves to Main Function

Main:
addi x6, x0, 2  # x6 = input
jal x1, exp # jumping to Exp Function
addi x10, x0 1 # x10 = x0 + 1
add x11, x0, x7 # x11 = x0 + x7
ecall  # print to console
```

Screenshots:

Test Case 1:
Input = 2 as seen in red circle
Output = 90 as seen in blue circle

Run Step Prev Reset Dump

0x00012083	lw x1 0(x2)	lw x1, 0(x2) # storing new value of x1 into stack
0x00810113	addi x2 x2 8	addi x2, x2, 8 # x2 = x2 + 8
0x00008067	jalr x0 x1 0	jalr x0, 0(x1) # leaves to Main Function
0x00200313	addi x6 x0 2	addi x6, x0, 2 # x6 = input
0xf4dff0ef	jal x1 -180	jal x1, exp # jumping to Exp Function
0x00100513	addi x10 x0 1	addi x10, x0 1 # x10 = x0 + 1
0x007005b3	add x11 x0 x7	add x11, x0, x7 # x11 = x0 + x7
0x00000073	ecall	ecall # print to console

90

Test Case 2:
Input = 5 as seen in red circle
Output = 3894 as seen in blue circle

0x00012083	lw x1 0(x2)	lw x1, 0(x2) # storing new value of x1 into stack
0x00810113	addi x2 x2 8	addi x2, x2, 8 # x2 = x2 + 8
0x00008067	jalr x0 x1 0	jalr x0, 0(x1) # leaves to Main Function
0x00500313	addi x6 x0 5	addi x6, x0, 5 # x6 = input
0xf4dff0ef	jal x1 -180	jal x1, exp # jumping to Exp Function
0x00100513	addi x10 x0 1	addi x10, x0 1 # x10 = x0 + 1
0x007005b3	add x11 x0 x7	add x11, x0, x7 # x11 = x0 + x7
0x00000073	ecall	ecall # print to console

3894

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

<u>Test Case 3:</u>
Input = 10 as seen in red circle
Output = 106034 as seen in blue circle

[illegible]	[illegible]	[illegible]
0x00012083	lw x1 0(x2)	lw x1, 0(x2) # storing new value of x1 into stack
0x00810113	addi x2 x2 8	addi x2, x2, 8 # x2 = x2 + 8
0x00008067	jalr x0 x1 0	jalr x0, 0(x1) # leaves to Main Function
0x00a00313	addi x6 x0 10	addi x6, x0, 10 # x6 = input
0xf4dff0ef	jal x1 -180	jal x1, exp # jumping to Exp Function
0x00100513	addi x10 x0 1	addi x10, x0 1 # x10 = x0 + 1
0x007005b3	add x11 x0 x7	add x11, x0, x7 # x11 = x0 + x7
0x00000073	ecall	ecall # print to console

106034

<u>Test Case 4:</u>
Input = 1 as seen in red circle
Output = 14 as seen in blue circle

[illegible]	[illegible]	[illegible]
0x00012083	lw x1 0(x2)	lw x1, 0(x2) # storing new value of x1 into stack
0x00810113	addi x2 x2 8	addi x2, x2, 8 # x2 = x2 + 8
0x00008067	jalr x0 x1 0	jalr x0, 0(x1) # leaves to Main Function
0x00100313	addi x6 x0 1	addi x6, x0, 1 # x6 = input
0xf4dff0ef	jal x1 -180	jal x1, exp # jumping to Exp Function
0x00100513	addi x10 x0 1	addi x10, x0 1 # x10 = x0 + 1
0x007005b3	add x11 x0 x7	add x11, x0, x7 # x11 = x0 + x7
0x00000073	ecall	ecall # print to console

14

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

3. [50 pts] Write a function **reorder** in RISC-V that, when given an array {3, 7, 45, 66, 80, 1}, returns the array in **reverse order** (i.e. the first number is now the last). Your code should include the following:

- Write a **main()** function to perform the following tasks:
  - Call an **input** function to write the given array values into a certain continuous memory starting from **0(0x0fffffe8)**. Your **input** function should work for any size of the array.
  - Call the **reorder** function to reorder the array.
    - Your function **reorder()** must call a **swap()** function to perform the necessary operations to reorder two elements of the array.
  - Call an **output()** function to print the reordered array values in the console. Your **output()** function should work for any array size.

Note: you need to show screenshots of the outputs based on the given inputs.
(There are 5 functions in total, and each function worth 10pts.)

Code:

```
.data
array: .word 3 7 45 66 80 1  # array input
sizeofarray: .word 6 # size of array
.text

j MAIN  # jumping to Main label

INPUT:
addi x2,x2,-16 # allocating 2 spots in stack
sw x20, 8(x2)  # assigning register x20 to second spot in stack
sw x27, 0(x2)  # assigning register x27 to second spot in stack
la x6, array     # psuedo instruction to place array into x6 register
lw x7, sizeofarray  #loading x7 with value inside variable n
add x20, x0, x0 # x20 = x0 + x0
add x28, x0, x5 # x28 = x0 + x5

LOOPINPUT:
bge x20, x7, INPUTDONE # checking to see if x20 is greater than or equal to
x7 if true then go to INPUTDONE label
lw x27, 0(x6)  # loading x27 with x6 value
sw x27, 0(x28) # storing value of x28 into x27
addi x6, x6, 4 # x6 = x6 + 4
```

```
addi x28, x28, 4 # x28 = x28 + 4
addi x20, x20, 1 # x20 = x20 + 1
jal x0, LOOPINPUT # jumping into LOOPINPUT

INPUTDONE:
lw x20, 0(x2) # loading x20 into first stack spot
lw x27, 0(x2) # loading x27 into second stack spot
jalr x0, 0(x1) # leaves to MAIN

REORDER:
addi x2, x2, -40 # x2 = x2 + (-40)
sw x27, 24(x2)   # x27 stores in 4th spot in stack
sw x8, 16(x2)    # x8 stores in 3rd spot in stack
sw x9, 8(x2)     # x9 stores in 2nd spot in stack
sw x1, 0(x2)     # x1 stores in 1st spot in stack
add x9, x9, x7   # x9 = x9 + x7
addi x8, x8, 2   # x8 = x8 + 2
div x9, x9, x8   # x9 = x9/x8
add x21, x0, x0  # x21 = x0 + x0
addi x31, x0, 4  # x31 = x0 + 4
add x27, x27, x5 # x27 = x27 + x5
sub x28, x28, x31 # x28 = x28 + x31

LOOPREORDER:
bge x21, x9, EXIT # checking to see if x21 is greater than or equal to x9 if true
then go to EXIT label
jal x1, SWAP # jumping to SWAP label
lw x1, 0(x2) # loading x1 into first stack spot
addi x21, x21, 1 # x21 = x21 + 1
addi x27, x27, 4 # x27 = x27 + 4
sub x28, x28, x31 # x28 = x28 - x31
jal x0, LOOPREORDER # jumping to LOOPREORDER label

EXIT:
lw x27, 24(x2) # loading x27 into 4th stack spot
lw x8, 16(x2)  # loading x8 into 3rd stack spot
lw x9, 8(x2)   # loading x9 into 2nd stack spot
lw x1, 0(x2)   # loading x1 into 1st stack spot
addi x2, x2, 8 # x2 = x2 + 8
jalr x0, 0(x1) # leaving to MAIN

SWAP:
addi x2, x2, -24 # x2 = x2 + (-24)
sw x14, 16(x2)   # x14 is stored in 3rd stack spot
sw x17, 8(x2)    # x17 is stored in 2nd stack spot
sw x1, 0(x2)     # x1 is stored in 1st stack spot
lw x14, 0(x27)   # loading x14 into x27
```

```
lw x17, 0(x28)   # loading x17 into x28
sw x14, 0(x28)   # storing x14 into x28
sw x17, 0(x27)   # storing x17 into x27
lw x14, 16(x2)   # loading x14 into 3rd stack spot
lw x17, 8(x2)    # loading x17 into 2nd stack spot
lw x1, 0(x2)     # loading x1 into 1st stack spot
addi x2, x2, 24  # x2 = x2 + 24
jalr x0, 0(x1)   # leaving to MAIN

OUTPUT:
addi x2, x2, -32 # x2 = x2 + (-32)
sw x20, 24(x2)   # x20 is stored in 4th stack spot
sw x10, 16(x2)   # x10 is stored in 3rd stack spot
sw x11, 8(x2)    # x11 is stored in 2nd stack spot
sw x18, 0(x2)    # x18 is stored in 1st stack spot
add x18, x5, x18 # x18 = x5 + x18
addi x10, x0, 1  # x10 = x0 + 1

LOOPOUTPUT:
bge x20, x7, DONEOUTPUT # checking to see if x20 is greater than or equal
to x7 if true then go to DONEOUTPUT label
addi x10, x0, 1  # x10 = x0 + 1
lw x11, 0(x18)   # loading x11 into x18
addi x18, x18, 4 # x18 = x18 + 4
ecall  # printing to console
addi x11, x0, 32 # x11 = x0 + 32
addi x10, x0, 11 # x10 = x0 + 11
ecall  # printing to console
addi x20, x20, 1 # x20 = x20 + 1
jal x0, LOOPOUTPUT # jumping to LOOPOUTPUT label

DONEOUTPUT:
lw x20, 24(x2) # x20 is stored in 4th stack spot
lw x10, 16(x2) # x10 is stored in 3rd stack spot
lw x11, 8(x2)  # x11 is stored in 2nd stack spot
lw x18, 0(x2)  # x18 is stored in 1st stack spot
jalr x0, 0(x1) # leaving to MAIN

MAIN:
li x5, 0x0fffffe8  # storing 0x0fffffe8 into x5 by pseudo instruction into memory
jal x1, INPUT      # jumping to INPUT label
jal x1, REORDER    # jumping to REORDER label
jal x1, OUTPUT     # jumping to OUTPUT label
```

Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey

Screenshots:

Test Case 1:
Input = 3 7 45 66 80 1 as seen in red circle
Output = 1 80 66 45 7 3 as seen in blue circle

```
.data
array: .word 3 7 45 66 80 1 # array input
sizeofarray: .word 6 # size of array
.text
```

Machine Code	Basic Code	Original Code
0x1400006f	jal x0 320	j MAIN # jumping to Main label
0xff010113	addi x2 x2 -16	addi x2,x2,-16 # allocating 2 spots in stack
0x01412423	sw x20 8(x2)	sw x20, 8(x2) # assigning register x20 to second spot in stack
0x01b12023	sw x27 0(x2)	sw x27, 0(x2) # assigning register x27 to second spot in stack
0x10000317	auipc x6 65536	la x6, array # psuedo instruction to place array into x6 register
0xff030313	addi x6 x6 -16	la x6, array # psuedo instruction to place array into x6 register

```
1 80 66 45 7 3
```

<u>Test Case 2:</u>
Input = 1 80 66 45 7 3 as seen in red circle
Output = 3 7 45 66 80 1 as seen in blue circle

```
.data
array: .word 1 80 66 45 7 3 # array input
sizeofarray: .word 6 # size of array
.text
```

Machine Code	Basic Code	Original Code
0x1400006f	jal x0 320	j MAIN # jumping to Main label
0xff010113	addi x2 x2 -16	addi x2,x2,-16 # allocating 2 spots in stack
0x01412423	sw x20 8(x2)	sw x20, 8(x2) # assigning register x20 to second spot in stack
0x01b12023	sw x27 0(x2)	sw x27, 0(x2) # assigning register x27 to second spot in stack
0x10000317	auipc x6 65536	la x6, array # psuedo instruction to place array into x6 register
0xff030313	addi x6 x6 -16	la x6, array # psuedo instruction to place array into x6 register

3 7 45 66 80 1

<u>Test Case 3:</u>
Input = 29 80 53 49 64 90 as seen in red circle
Output = 90 64 49 53 80 29 as seen in blue circle

```
.data
array: .word 29 80 53 49 64 90 # array input
sizeofarray: .word 6 # size of array
.text
```

Machine Code	Basic Code	Original Code
0x1400006f	jal x0 320	j MAIN # jumping to Main label
0xff010113	addi x2 x2 -16	addi x2,x2,-16 # allocating 2 spots in stack
0x01412423	sw x20 8(x2)	sw x20, 8(x2) # assigning register x20 to second spot in stack
0x01b12023	sw x27 0(x2)	sw x27, 0(x2) # assigning register x27 to second spot in stack
0x10000317	auipc x6 65536	la x6, array # psuedo instruction to place array into x6 register
0xff030313	addi x6 x6 -16	la x6, array # psuedo instruction to place array into x6 register

90 64 49 53 80 29

<u>Test Case 4:</u>
Input = 7 4 0 9 as seen in red circle
Output = 9 0 4 7 as seen in blue circle

```
.data
array: .word 7 4 0 9 # array input
sizeofarray: .word 4 # size of array
.text
```

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x1400006f	jal x0 320	j MAIN # jumping to Main label
0xff010113	addi x2 x2 -16	addi x2,x2,-16 # allocating 2 spots in stack
0x01412423	sw x20 8(x2)	sw x20, 8(x2) # assigning register x20 to second spot in stack
0x01b12023	sw x27 0(x2)	sw x27, 0(x2) # assigning register x27 to second spot in stack
0x10000317	auipc x6 65536	la x6, array # psuedo instruction to place array into x6 register
0xff030313	addi x6 x6 -16	la x6, array # psuedo instruction to place array into x6 register

9047

Test Case 5:
Input = 8 -3 9 7 2 -5 1 4 as seen in red circle
Output = 4 1 -5 2 7 9 -3 8 as seen in blue circle

```
.data
array: .word 8 -3 9 7 2 -5 1 4 # array input
sizeofarray: .word 8 # size of array
.text
```

Machine Code	Basic Code	Original Code
0x1400006f	jal x0 320	j MAIN # jumping to Main label
0xff010113	addi x2 x2 -16	addi x2,x2,-16 # allocating 2 spots in stack
0x01412423	sw x20 8(x2)	sw x20, 8(x2) # assigning register x20 to second spot in stack
0x01b12023	sw x27 0(x2)	sw x27, 0(x2) # assigning register x27 to second spot in stack
0x10000317	auipc x6 65536	la x6, array # psuedo instruction to place array into x6 register
0xff030313	addi x6 x6 -16	la x6, array # psuedo instruction to place array into x6 register

```
4 1 -5 2 7 9 -3 8
```