Computer Architecture Assignment-2

Course Name: EG 212, Computer Architecture

By: Kavya Gupta (IMT2023016), Pragya Rai (IMT2023529), Ananya Vundavalli (IMT2023537)

26 February 2024

1 Introduction

In this project we have designed a MIPS processor to implement the following programs:

- Sum of squares of digits of a number.
- Factorial of a number.
- Maximum element of an array.

2 Programs

2.1 Sum of squares of digits of a number

We take a number and find sum of the squares of the digits in the number. *Assumption: We assume the given number is a non-negative integer.*

2.2 Factorial of a number

We multiply all the numbers from 1 to the given number. *Assumption: We assume given number is a non-negative integer.*

2.3 Maximum element in an array

We take the maximum element to be 0 and compare it with every element of the array. If it is bigger than the max element then that number becomes the max element.In the end we get the maximum element of the array. *Assumption: The array can have a maximum of 10 integers.*

3 Assembly Code

3.1 Sum of squares of digits of a number

We initialize `$t1` ,i.e.,sum to 0 and `$t2` ,i.e., the divisor to 10. If `$t0` is 0, we exit the `loop`. Otherwise we divide `$t0` by 10. The remainder is stored in `$t3` and quotient in `$t0`. We square the value in `$t3`.We then add the value in `$t3` to `$t1`.

```
.data
    prompt:     .asciiz "Enter a number: "
    sum_msg:    .asciiz "Sum of squares of digits: "
    result:     .word 4    # Allocate 4 bytes for the result

.text
    main:
        lw $t0, 0($s0)
        # Find the sum of squares of digits
        addi $t1, $0, 0             # initialize sum to 0
        addi $t2, $0, 10            # initialize divisor to 10

    sum_of_squares_loop:
        beq $t0, $0, store_result   # if $t0 is 0, exit loop
        div $t0, $t2                # divide $t0 by 10
        mfhi $t3                    #remainder (last digit)
        mflo $t0                    # quotient (removed last digit)
        mul $t3, $t3, $t3           # square the remainder
        add $t1, $t1, $t3           # add the squared remainder to the sum
        j sum_of_squares_loop       # jump back to the beginning of the loop

    store_result:
        # Store the sum in the result memory location
        sw $t1, result              # store the sum in the result
        # Exit program
        addi $v0,$0, 10             # syscall code for exit
        syscall
```

3.2 Factorial of a number

In the `loop` we increment the value by 1. We have `$t1` ,i.e., the result,initialized to 1 and `$t2` ,i.e., loop counter to 1. If the loop counter is equal to user input+1, the exit `loop`. Otherwise multiply the result by the loop counter, then increment the loop counter by 1.

```
.data
    prompt:      .asciiz "Enter a non-negative integer: "
    result_msg:  .asciiz "Factorial is: "
    user_input:  .word 0  # space for storing user input

.text
    main:
        # Read user input (set user input to 5)
        addi $t0,$0, 6                          # set user input to 5
        sw $t0, user_input                 # store user input in memory
        lw $t0, user_input                 # load user input from memory into $t0
        addi $t0, $t0, 1                  # for comparison purposes

        # Calculate factorial
        addi $t1, $0, 1                   # initialize result to 1
        addi $t2, $0, 1                   # initialize loop counter to 1

    factorial_loop:
        slt $at, $t2, $t0
        beq $at, $0, end_factorial        # if loop counter equals user input, exit loop

        mul $t1, $t1, $t2                 # multiply result by loop counter
        addi $t2, $t2, 1                  # increment loop counter
        j factorial_loop                  # jump back to the beginning of the loop

    end_factorial:
        # Display the result
        addi $v0, $0, 4                   # syscall code for print_str
        la $a0, result_msg                # load address of result message string
        syscall

        addi $v0, $0, 1                   # syscall code for print_int
        add $a0, $t1, $0                   # load the result into $a0
        syscall

        # Exit program
        addi $v0, $0, 10                  # syscall code for exit
        syscall
```

3.3 Maximum element in an array

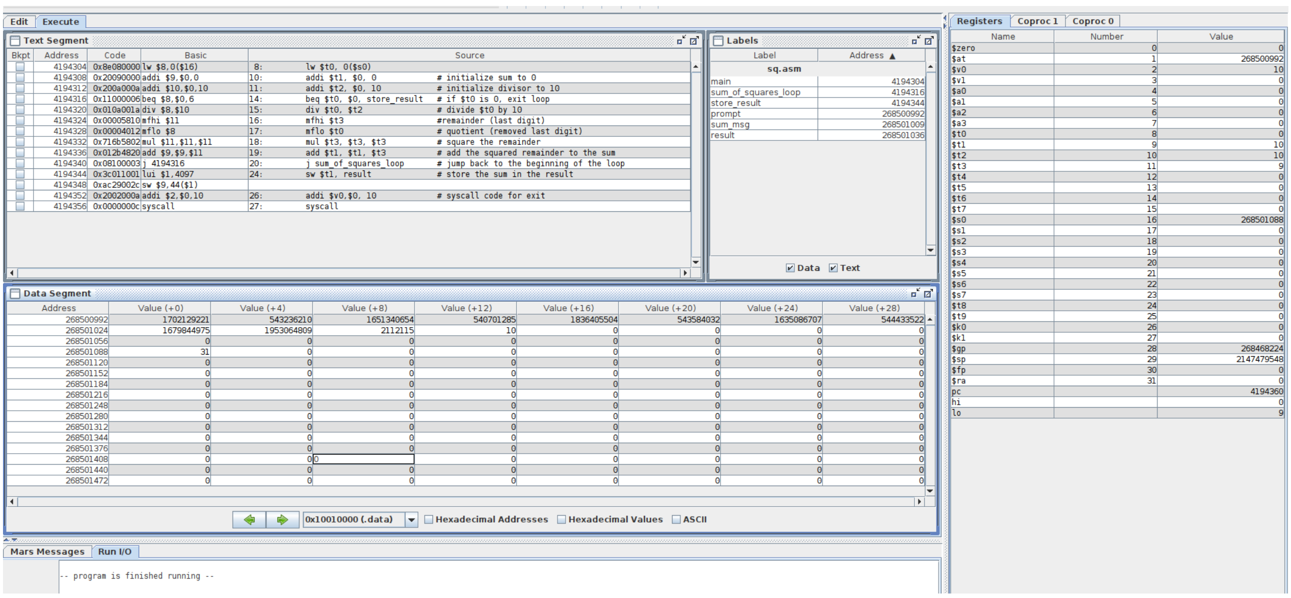
In `loop`, we load the word from array into `$t3`. Set `$t4` to 1 if value in `$t2 < $t3` , else set the value to 0. if `$t4` is 0, then `$t3` is not the max, so let it go to `not_max` label, then update max to `$t3`. In the `not_max` label, move to the next element, decrement the array size and if array size is not 0, the go to the `loop`. Otherwise, store max value in `$t2`.

```
.data
array: .word 5, 3, ,101, 1, 7, 9, 6, 2, 4, 10  # example array
array_size: .word 10  # size of the array
max: .word 0  # initialize max to the smallest possible integer

.text
main:

    la $t0, array  # load address of array into $t0
    lw $t1, array_size  # load array size into $t1
    lw $t2, max  # load max into $t2

    # loop through the array
loop:

    lw $t3, 0($t0)  # load word from array into $t3
    slt $t4, $t2, $t3   # set $t4 to 1 if $t2 < $t3, else 0
    beq $t4,$0, not_max  # if $t4 is 0, then $t3 is not the max
    add $t2, $t3, $0      # update max to $t3
not_max:

    addi $t0, $t0, 4  # move to next array element
    addi $t1, $t1, -1  # decrement array size
    bne $t1,$0, loop  # if array size is not zero, loop again

    sw $t2, max  # store the max value in max

    # print max value

    add $v0, $0, 1
    add $a0, $t2, $0
    syscall

    # exit program
    addi $v0, $0, 10
    syscall
```

4 Assembler

We have used MARS simulator as our assembler.

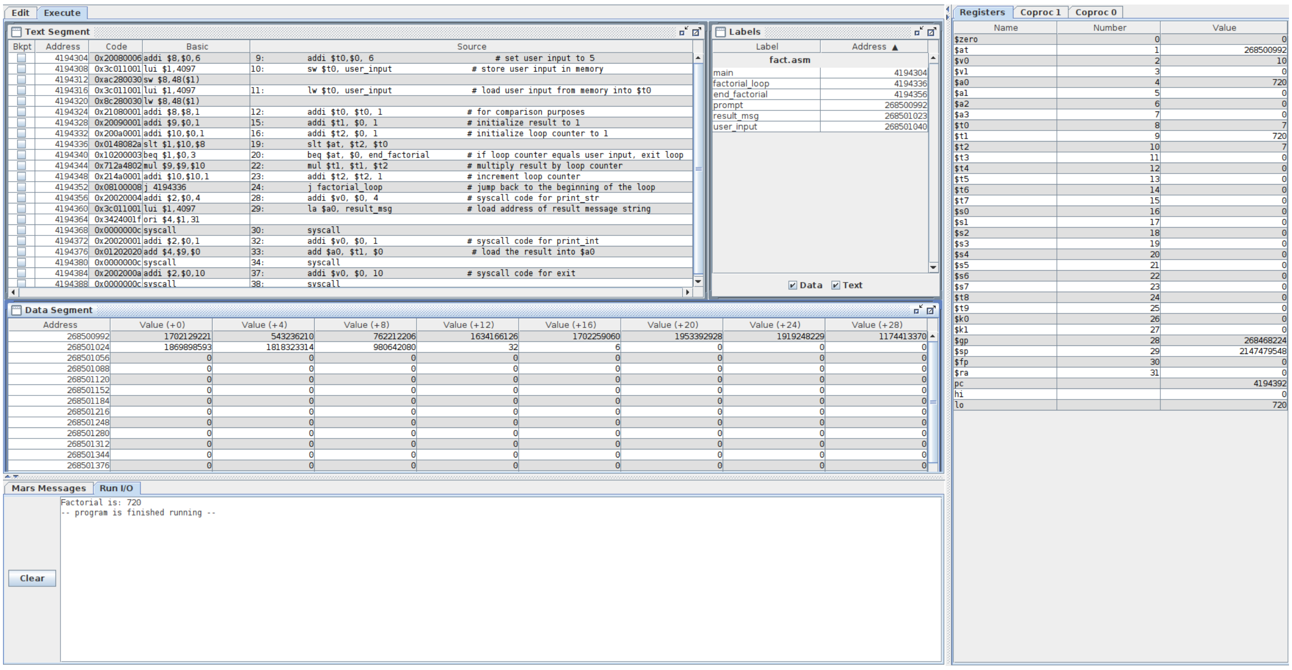
4.1 Sum of squares of digits of a number



Machine Code:

```
For sum of squares:
 instruction_memory={
        4194304 : "10001110000010000000000000000000",
        4194308 : "00100000000010010000000000000000",
        4194312 : "00100000000010100000000000001010",
        4194316 : "00010001000000000000000000000110",
        4194320 : "00000001000010100000000000011010",
        4194324 : "00000000000000000101100000010000",
        4194328 : "00000000000000001000000000010010",
        4194332 : "01110001011010110101100000000010",
        4194336 : "00000001001010110100100000100000",
        4194340 : "00001000000100000000000000000011",
        4194344 : "00111100000000010001000000000001",
        4194348 : "10101100001010010000000000101100",}
```

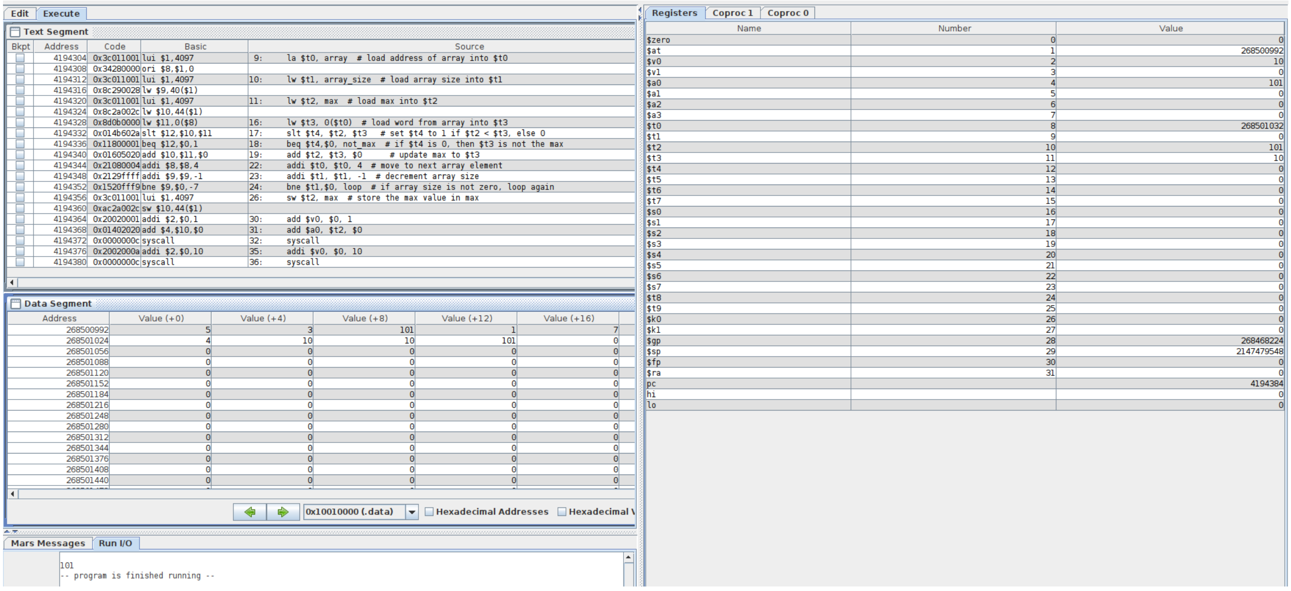
4.2 Factorial of a number



Machine Code:

```
For factorial:
    instruction_memory = {
        4194304: "00100000000010000000000000000101",
        4194308: "00111100000000010001000000000001",
        4194312: "10101100001010000000000000110000",
        4194316: "00111100000000010001000000000001",
        4194320: "10001100001010000000000000110000",
        4194324: "00100001000010000000000000000001",
        4194328: "00100000000010010000000000000001",
        4194332: "00100000000010100000000000000001",
        4194336: "00000001010010000000100000101010",
        4194340: "00010000001000000000000000000011",
        4194344: "01110001001010100100100000000010",
        4194348: "00100001010010100000000000000001",
        4194352: "00001000000100000000000000001000",}
```

4.3 Maximum element in an array



Machine Code:

```
For max element of array:
    instruction_memory={
    4194304 : "00111100000000010001000000000001",
    4194308 : "00110100001010000000000000000000",
    4194312 : "00111100000000010001000000000001",
    4194316 : "10001100001010010000000000101000",
    4194320 : "00111100000000010001000000000001",
    4194324 : "10001100001010100000000000101100",
    4194328 : "10001101000010110000000000000000",
    4194332 : "00000001010010110110000000101010",
    4194336 : "00010001100000000000000000000001",
    4194340 : "00000001011000000101000000100000",
    4194344 : "00100001000010000000000000000100",
    4194348 : "00100001001010011111111111111111",
    4194352 : "00010101001000001111111111111001",
    4194356 : "00111100000000010001000000000001",
    4194360 : "10101100001010100000000000101100",}
```

5 Processor

We have written the functions in our code as the different stages of the MIPS processor.

The first function is `Instruction Fetch` (IF) which fetches instruction at the current PC from `Instruction Memory` (IM).

The next function is `Instruction Decode` (ID) which decodes each instruction into different fields like opcode, rs, rt etc., and returns them.This calls the Control function to generate control signals for each instruction.

```python
#This function is used to fetch the instructions
def IF():
    global PC, instruction_memory
    inst = instruction_memory[PC]
    PC = PC + 4     #increment PC by 4
    return inst
    

#this function decodes each instruction into different fields
def ID(inst):
    global opcode, rs, rt, rd, shamt, funct, target, imm
    #declare the different fields as empty strings
    opcode = ""
    rs = ""
    rt = ""
    rd = ""
    shamt = ""
    funct = ""
    target = ""
    imm = ""
    opcode = inst[0:6]  #first 6 bits of instruction
    control(opcode)     #function call

    if opcode == "000010":  # jump instruction
        target = "0000" + inst[6:32] + "00"
        return opcode, rs, rt, rd, shamt, funct, target, imm

    elif opcode == "000000" or opcode == "011100":  # R-type instructions
        rs = inst[6:11]
        rt = inst[11:16]
        rd = inst[16:21]
        shamt = inst[21:26]
        funct = inst[26:32]
        return opcode, rs, rt, rd, shamt, funct, target, imm

    else:  # I-type instruction
        rs = inst[6:11]
        rt = inst[11:16]
        imm = inst[16:32]
        return opcode, rs, rt, rd, shamt, funct, target, imm
```

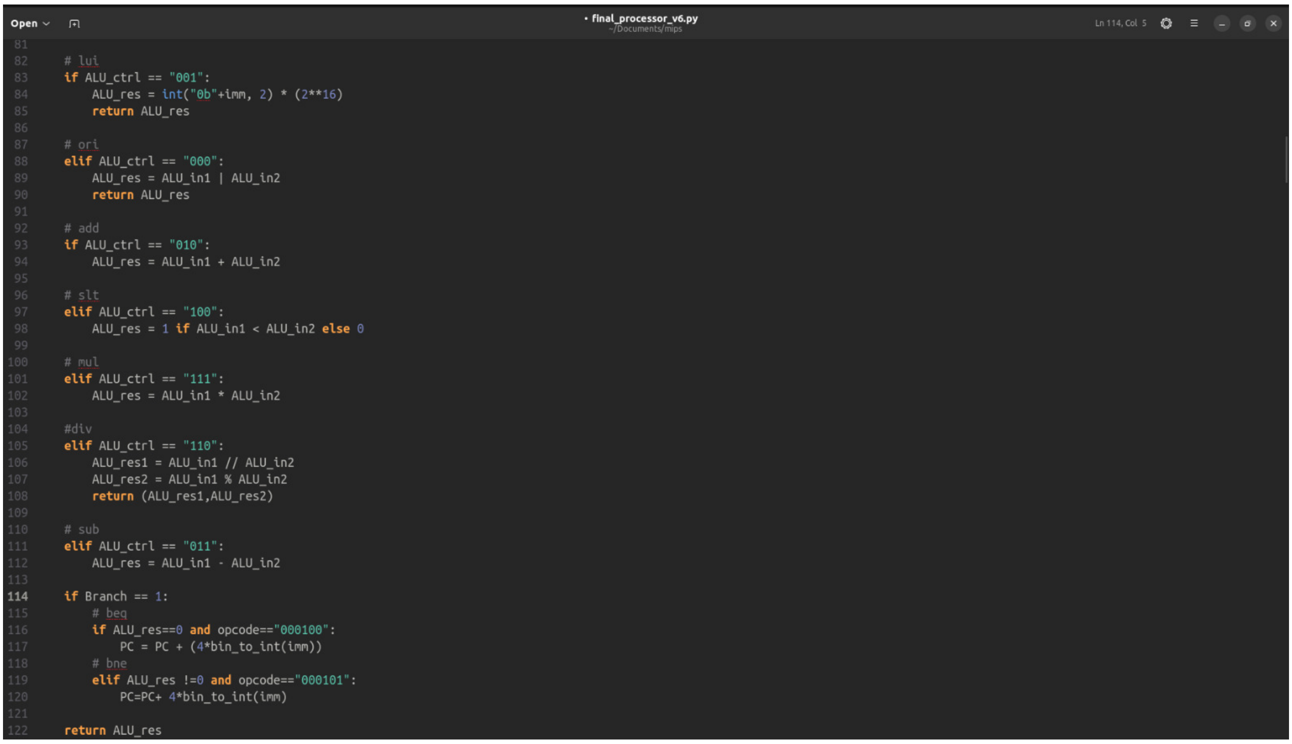
Figure 1: processor 1

`bin_to_int` function is used to convert binary string to a signed integer and also acts as the sign extend component in MIPS.

The `Execute` function(EX) is similar to ALU part to generate ALU output based on different ALU control signals generated by the `ALUControlUnit` function.

```python
#This function converts a binary string to a signed integer
def bin_to_int(bin_str):

    if bin_str[0] == '1':                              # For negative number
        if bin_str == '1' + '0' * (len(bin_str) - 1):   # Check if it's the minimum negative number
            return -2**(len(bin_str) - 1)
        bin_str = ''.join('1' if b == '0' else '0' for b in bin_str)  # Flip the bits
        return -1 * (int("0b"+bin_str, 2) + 1)         # Add 1 and then negate it

    else:                                              # For positive number
        return int("0b"+bin_str, 2)



#This function will perform the various operations based on ALUOp , funct
def EX(rs, rt, imm, funct, target, ALUOp):

    global ALUSrc , Branch, Jump, RegDst, reg, opcode,PC

    ALU_in1 = 0
    ALU_in2 = 0
    ALU_res = 0
    ALU_res1 = 0
    ALU_res2 = 0

    ALU_ctrl = ALU_controlUnit(ALUOp, funct) #function call

    # jump
    if Jump == 1:
        PC = int("0b"+target, 2)
        return 0
    if rs != "":
        ALU_in1 = reg[int("0b"+rs, 2)]  # input1 to ALU

    if ALUSrc == 0:
        ALU_in2 = reg[int("0b"+rt, 2)]  # input2 to ALU

    else:
        ALU_in2 = bin_to_int(imm)

```

Figure 2: processor 2



```
    # lui
    if ALU_ctrl == "001":
        ALU_res = int("0b"+imm, 2) * (2**16)
        return ALU_res

    # ori
    elif ALU_ctrl == "000":
        ALU_res = ALU_in1 | ALU_in2
        return ALU_res

    # add
    if ALU_ctrl == "010":
        ALU_res = ALU_in1 + ALU_in2

    # slt
    elif ALU_ctrl == "100":
        ALU_res = 1 if ALU_in1 < ALU_in2 else 0

    # mul
    elif ALU_ctrl == "111":
        ALU_res = ALU_in1 * ALU_in2

    #div
    elif ALU_ctrl == "110":
        ALU_res1 = ALU_in1 // ALU_in2
        ALU_res2 = ALU_in1 % ALU_in2
        return (ALU_res1,ALU_res2)

    # sub
    elif ALU_ctrl == "011":
        ALU_res = ALU_in1 - ALU_in2

    if Branch == 1:
        # beq
        if ALU_res==0 and opcode=="000100":
            PC = PC + (4*bin_to_int(imm))
        # bne
        elif ALU_res !=0 and opcode=="000101":
            PC=PC+ 4*bin_to_int(imm)

    return ALU_res
```

Figure 3: processor 3

`Memory` function (MEM) reads from the memory or writes into the memory as per control signals `memwr` and `mmemrd`. Also it has `memtoreg` control signal to write the data into the writeback phase. `WriteBack` function (WB) writes back the data returned by the MEM function into the register destination. *We have assumed regwr=2 as our div instruction ALU stage needs to return 2 outputs and read them into 2 special registers hi and lo.*

```python
#This function describes memory access
def MEM(ALU_res, rt):
    global MemRd, MemWr, MemtoReg, data_memory,hi,lo


    # if MemRd == 1 then read from the address produced by ALU
    if MemRd == 1:
        return data_memory[ALU_res]

    # if MemWr == 1 then write into the address produced by ALU
    elif MemWr == 1:
        data_memory[ALU_res] = reg[int("0b"+rt, 2)]


    if MemtoReg == 0:
        return ALU_res

    elif MemtoReg == 1:
        return data_memory[ALU_res]

#This function writes back to the reg
def WB(rd, rt, data):
    global reg,hi,lo, RegWr, RegDst

    if(RegWr == 2):      # if condition for division
        hi=data[1]
        print("hi",hi)
        lo=data[0]
        print("lo",lo)


    elif RegWr == 1:
        if RegDst == 1:
            if(funct=="010000"):
                reg[int("0b"+rd, 2)] = hi     #this hi is used in case of div,it stores remainder
            elif(funct=="010010"):
                reg[int("0b"+rd, 2)] = lo      #this lo is also used in case of div, it stores quotient
            else:
                reg[int("0b"+rd, 2)] = data
        else:
            reg[int("0b"+rt, 2)] = data
```

Figure 4: processor 4

The `control` function sets the control signals of `regwr, memwr, memrd` etc., according to the instruction. *Some of the instructions whose ALUOp is not well-known, we have assumed the ALUOps and ALUctrl.*

```python
#This function sets the control signal based on the opcodes
#Control signals RegDst, Branch, MemRd , MemtoReg , ALUOp, MemWr, ALUSrc , RegWr , Jump
def control(opcode):
    global RegDst, Branch, MemRd, MemtoReg, ALUOp, MemWr, ALUSrc, RegWr, Jump

    if opcode == "100011":  # lw instruction
        ALUOp = "00"
        ALUSrc = 1
        Branch = 0
        MemRd = 1
        MemtoReg = 1
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 1

    elif opcode == "101011":  # sw instruction
        ALUOp = "00"
        ALUSrc = 1
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 1
        Jump = 0
        RegDst = 0
        RegWr = 0

    elif opcode == "000000":  # R-type instruction
        ALUOp = "10"
        ALUSrc = 0
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 1
        RegWr = 1
```

Figure 5: processor 5

```python
    elif opcode == "011100":  # mul instruction
        ALUOp = "10"
        ALUSrc = 0
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 1
        RegWr = 1

    elif opcode == "001111":  # lui instruction
        ALUOp = "11"
        ALUSrc = 0
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 1

    elif opcode == "001101":  # ori instruction
        ALUOp = "1"
        ALUSrc = 1
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 1
```

Figure 6: processor 6

```python
    elif opcode == "001000":  # addi instruction
        ALUOp = "00"
        ALUSrc = 1
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 1

    elif opcode == "000100":  # beq instruction
        ALUOp = "01"
        ALUSrc = 0
        Branch = 1
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 0

    elif opcode == "000101":  # bne instruction
        ALUOp = "01"
        ALUSrc = 0
        Branch = 1
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 0
        RegDst = 0
        RegWr = 0
```

Figure 7: processor 7

```python
    elif opcode == "000010":  # j-type instruction
        ALUSrc = "0"
        Branch = 0
        MemRd = 0
        MemtoReg = 0
        MemWr = 0
        Jump = 1
        RegDst = 0
        RegWr = 0
```

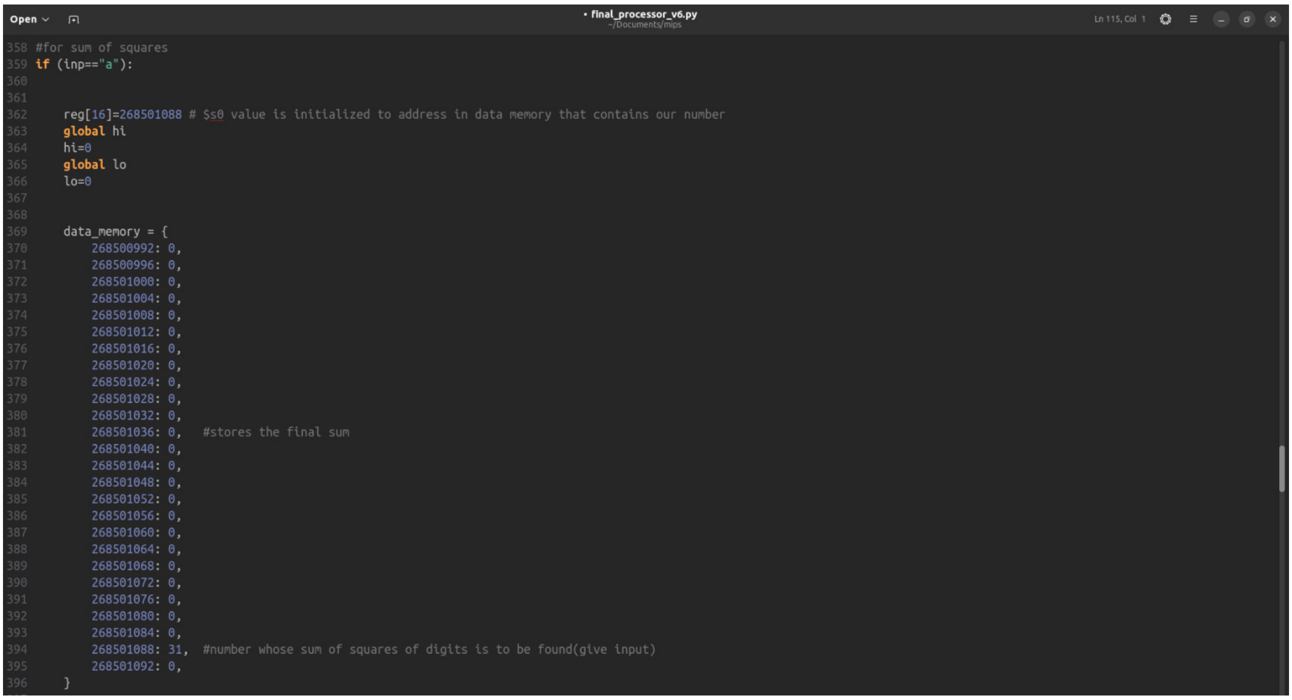
Figure 8: processor 8

```python
#This function generates ALU control signals which decide what operation is to be done by the ALU
def ALU_controlUnit(ALUOp, funct):
    global RegWr
    # lw, sw, addi
    if ALUOp == "00":
        return "010"

    # lui
    elif ALUOp == "11":
        return "001"

    elif ALUOp == "10":   # R format
        # add
        if funct == "100000":
            return "010"

        # slt
        elif funct == "101010":
            return "100"

        # mul ; alu control is the same as add(mul is repeated add)
        elif funct == "000010":
            return "111"

        #div
        elif funct=="011010":
            RegWr=2
            return "110"

        # mfhi - moves hi to rd
        elif funct=="010000":
            return "010"

        #mflo - moves lo to rd
        elif funct == "010010":
            return "010"

        # sub
        elif funct == "100010":
            return "011"

```

Figure 9: processor 9

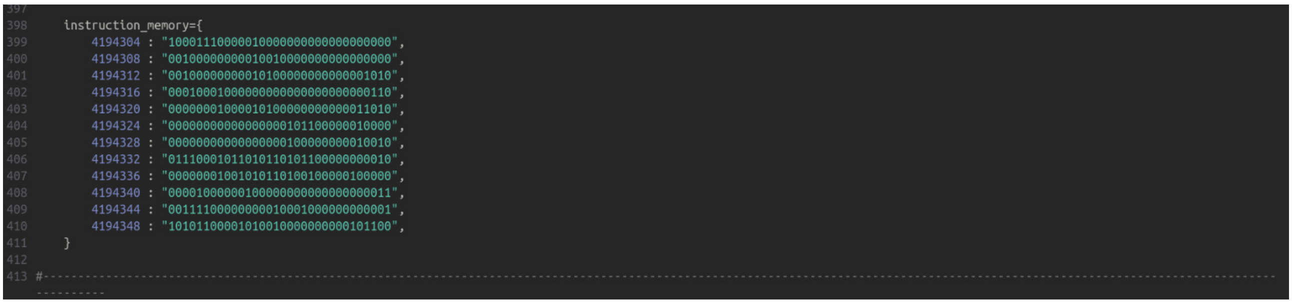
```python
    # beq,bne
    elif ALUOp == "01":
        return "011"

    #ori
    elif ALUOp == "1":
        return "000"

# Now we will initialize the state of the processor
#initialise PC to point to the first instruction
PC = 4194304

#initialise the various fields of instruction with empty strings
opcode = rs = rt = rd = shamt = funct = target = imm = ""

#initialise control signals of instructions with 0
RegDst = Branch = MemRd = MemtoReg = ALUOp = MemWr = ALUSrc = RegWr = Jump = 0

#initialise reg
reg = [0] * 32


#Controlling which program is to be run
inp=input('''Enter a letter from a,b,c which decides which program is to be run:
a.Sum of squares
b.Factorial
c.Max element of an array : ''')

print("\n")


#-----------------------------------------------------------------------------------------------------------------------------------------------------------
---------
```

Figure 10: processor 10



```
#for sum of squares
if (inp=="a"):


    reg[16]=268501088 # $s0 value is initialized to address in data memory that contains our number
    global hi
    hi=0
    global lo
    lo=0


    data_memory = {
        268500992: 0,
        268500996: 0,
        268501000: 0,
        268501004: 0,
        268501008: 0,
        268501012: 0,
        268501016: 0,
        268501020: 0,
        268501024: 0,
        268501028: 0,
        268501032: 0,
        268501036: 0,   #stores the final sum
        268501040: 0,
        268501044: 0,
        268501048: 0,
        268501052: 0,
        268501056: 0,
        268501060: 0,
        268501064: 0,
        268501068: 0,
        268501072: 0,
        268501076: 0,
        268501080: 0,
        268501084: 0,
        268501088: 31,  #number whose sum of squares of digits is to be found(give input)
        268501092: 0,
    }
```

Figure 11: processor 11



```
    instruction_memory={
        4194304 : "10001110000010000000000000000000",
        4194308 : "00100000000010010000000000000000",
        4194312 : "00100000000010100000000000001010",
        4194316 : "00010001000000000000000000000110",
        4194320 : "00000001000010100000000000011010",
        4194324 : "00000000000000000101100000010000",
        4194328 : "00000000000000001000000000010010",
        4194332 : "01110001011010110101100000000010",
        4194336 : "00000001001010110100100000100000",
        4194340 : "00001000000100000000000000000011",
        4194344 : "00111100000000010001000000000001",
        4194348 : "10101100001010010000000000101100",
    }

#-------------------------------------------------------------------------------------------------------------------------------------------
---------
```

Figure 12: processor 12

```
# for factorial
if (inp=="b"):
    data_memory = {
        268500992: 0,
        268500996: 0,
        268501000: 0,
        268501004: 0,
        268501008: 0,
        268501012: 0,
        268501016: 0,
        268501020: 0,
        268501024: 0,
        268501028: 0,
        268501032: 0,
        268501036: 0,
        268501040: 0,
        268501044: 0,
        268501048: 0,
        268501052: 0,
        268501056: 0,
        268501060: 0,
        268501064: 0,
        268501068: 0,
        268501072: 0,
        268501076: 0,
        268501080: 0,
        268501084: 0,
        268501088: 0,
        268501092: 0,
    }
```

Figure 13: processor 13

```
    instruction_memory = {
        4194304: "00100000000010000000000000000101",    #number whose factorial is to be found(give input)
        4194308: "00111100000000010001000000000001",
        4194312: "10101100001010000000000000110000",
        4194316: "00111100000000010001000000000001",
        4194320: "10001100001010000000000000110000",
        4194324: "00100001000010000000000000000001",
        4194328: "00100000000010010000000000000001",
        4194332: "00100000000010100000000000000001",
        4194336: "00000001010010000000100000101010",
        4194340: "00010000000100000000000000000011",
        4194344: "01110001001010100100100000000010",
        4194348: "00100001010010100000000000000001",
        4194352: "00001000000100000000000000001000",
    }

#--------------------------------------------------------------------------------------------------------------------------------------------------------
----------
```

Figure 14: processor 14

```
#max element of array(array can take upto 10 elements)
if (inp=="c"):
    data_memory = {
        268500992: 88,  #array elements (give input)
        268500996: 3,
        268501000: 66,
        268501004: 723,
        268501008: 9,
        268501012: 0,
        268501016: 0,
        268501020: 0,
        268501024: 0,
        268501028: 0,
        268501032: 5,  #size of array
        268501036: 0,  #stores the final result
        268501040: 0,
        268501044: 0,
        268501048: 0,
        268501052: 0,
        268501056: 0,
        268501060: 0,
        268501064: 0,
        268501068: 0,
        268501072: 0,
        268501076: 0,
        268501080: 0,
        268501084: 0,
        268501088: 0,
        268501092: 0,
    }
```

Figure 15: processor 15

```
    instruction_memory={
    4194304 : "00111100000000010001000000000001",
    4194308 : "00110100001010000000000000000000",
    4194312 : "00111100000000010001000000000001",
    4194316 : "10001100001010010000000000101000",
    4194320 : "00111100000000010001000000000001",
    4194324 : "10001100001010100000000000101100",
    4194328 : "10001101000010110000000000000000",
    4194332 : "00000001010010110110000000101010",
    4194336 : "00010001100000000000000000000001",
    4194340 : "00000001011000000101000000100000",
    4194344 : "00100001000010000000000000000100",
    4194348 : "00100001001010011111111111111111",
    4194352 : "00010101001000001111111111111001",
    4194356 : "00111100000000010001000000000001",
    4194360 : "10101100001010100000000000101100",
    }
```

Figure 16: processor 16

```
def print_result():

    print("Final result:\n")
    print("Reg : \n" , reg)
    print("\n")
    print("Data memory : \n" , data_memory)
    print("\n")

    if (inp=="a"):
        print("Sum of squares of digits:", data_memory[268501036])
    elif (inp=="b"):
        print("Factorial is ",reg[9])
    elif (inp=="c"):
        print("Max element of array is :",reg[10])

# Main simulation loop
while (1):
    #perform the 5 stages fetch , decode , execute , memory access and writeback
    #print value of PC at every step
    print("Current PC : " , PC)

    #instruction fetch
    inst = IF()

    #instruction decode
    opcode, rs, rt, rd, shamt, funct, target, imm = ID(inst)

    #instruction execute
    ALU_res = EX(rs, rt, imm, funct, target, ALUOp)  # Pass ALUOp to EX function

    #memory access
    mem_data = MEM(ALU_res, rt)  #function call

    #finally writeback
    WB(rd, rt, mem_data) #function call

    print("Reg : \n" , reg)
    print("\n")
    print("Data memory : \n" , data_memory)
    print("\n")
    print("----------------------------------------------------------------------------------------------------")
```

Figure 17: processor 17

```
    if (inp=="a"):
        if PC > 4194348:
            break
    elif (inp=="b"):
        if PC > 4194352:
            break
    elif (inp=="c"):
        if PC > 4194360:
            break

print("----------------------------------------------------------------------------------------------------")
print("\n")

print_result()
```

Figure 18: processor 18

6 Result

6.1 Sum of squares of digits of a number

Here the input number is 31.

```
Enter a letter from a,b,c which decides which program is to be run:
a.Sum of squares
b.Factorial
c.Max element of an array : a


Current PC :  4194304
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 0, 0, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194308
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 0, 0, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194312
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 10, 0, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194316
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 10, 0, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194320
hi 1
lo 3
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 10, 0, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 19: result 1

```
Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194324
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 31, 0, 10, 1, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194328
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 10, 1, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194332
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 10, 1, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 10, 1, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


-----------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 3, 1, 10, 1, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
```

Figure 20: result 2

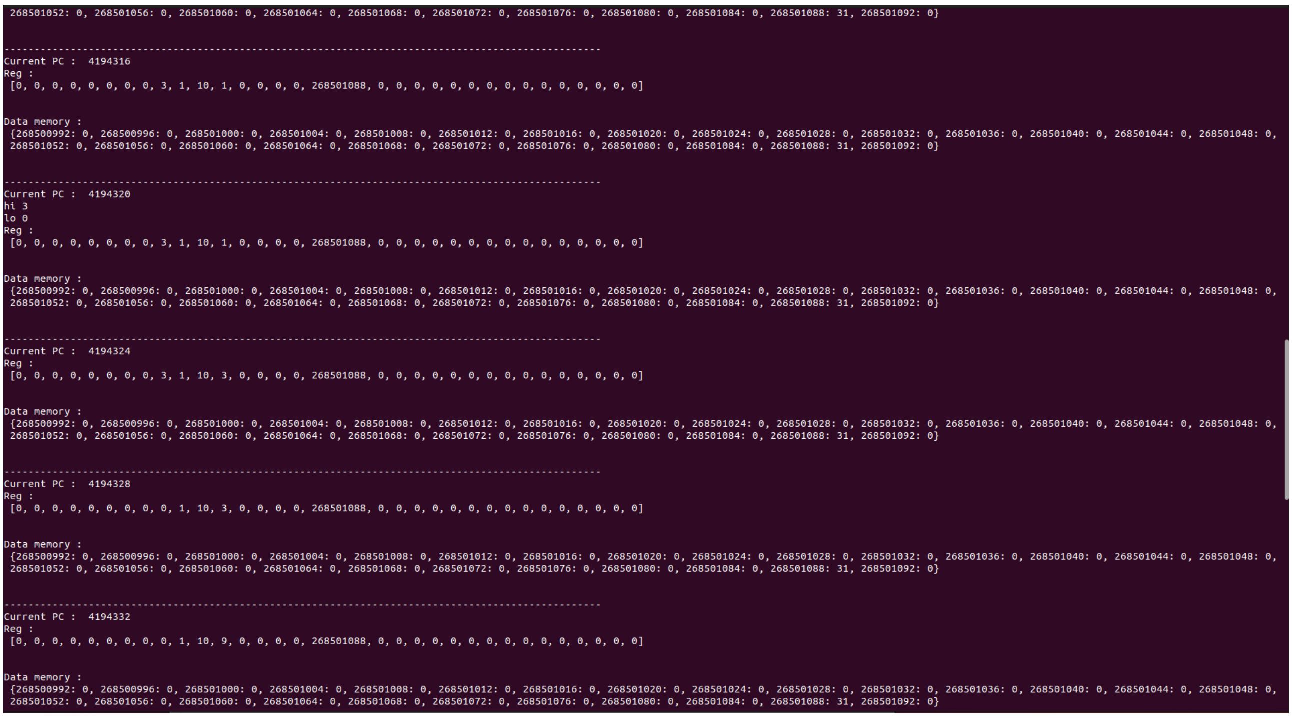


Figure 21: result 3

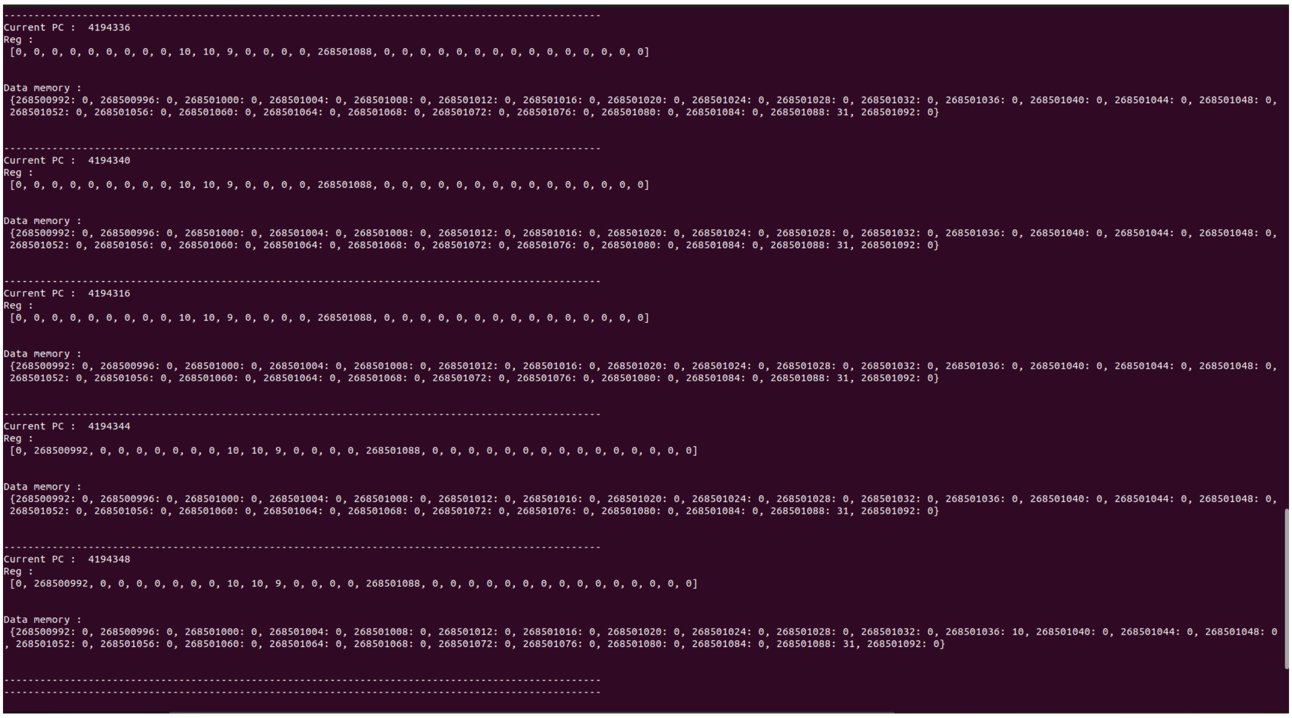
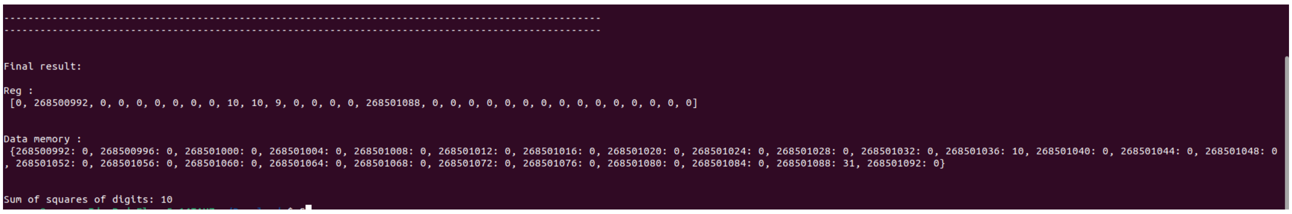


Figure 22: result 4

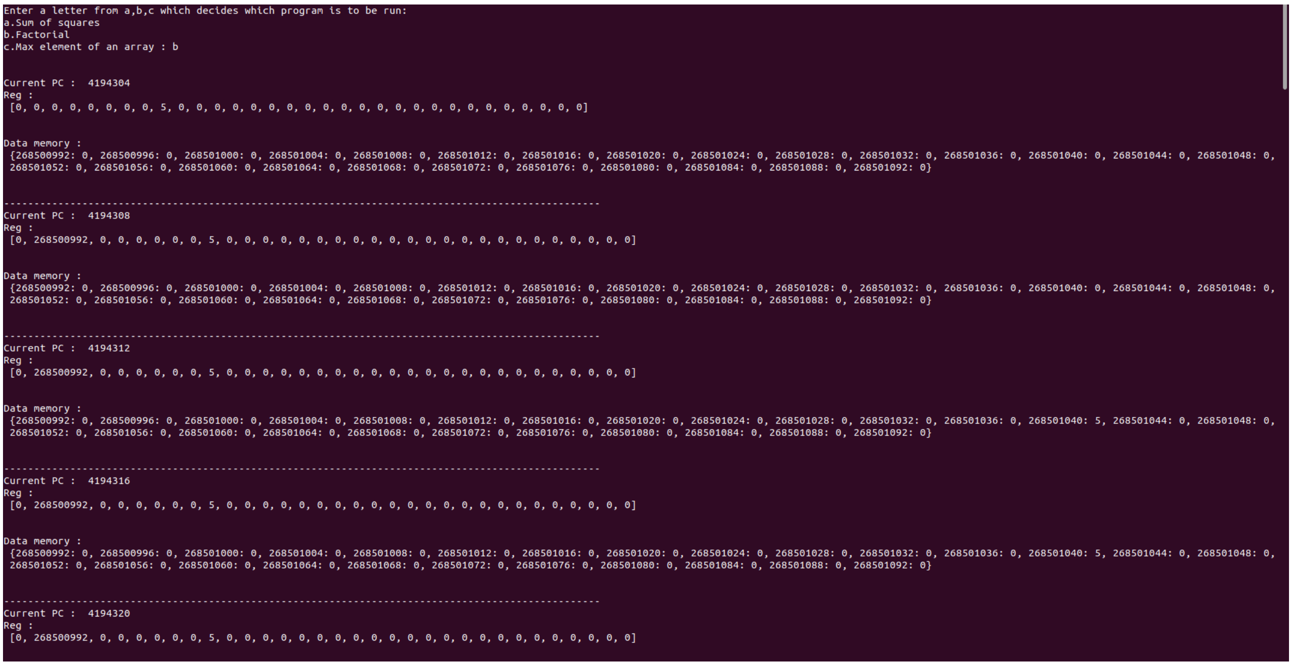


```
---------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------


Final result:

Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 0, 10, 10, 9, 0, 0, 0, 0, 268501088, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 10, 268501040: 0, 268501044: 0, 268501048: 0
, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 31, 268501092: 0}


Sum of squares of digits: 10
```

Figure 23: result 5

6.2 Factorial of a number

Here the input number is 5.



```
Enter a letter from a,b,c which decides which program is to be run:
a.Sum of squares
b.Factorial
c.Max element of an array : b


Current PC :  4194304
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------
Current PC :  4194308
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 0, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------
Current PC :  4194312
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------
Current PC :  4194316
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------
Current PC :  4194320
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 24: result 1

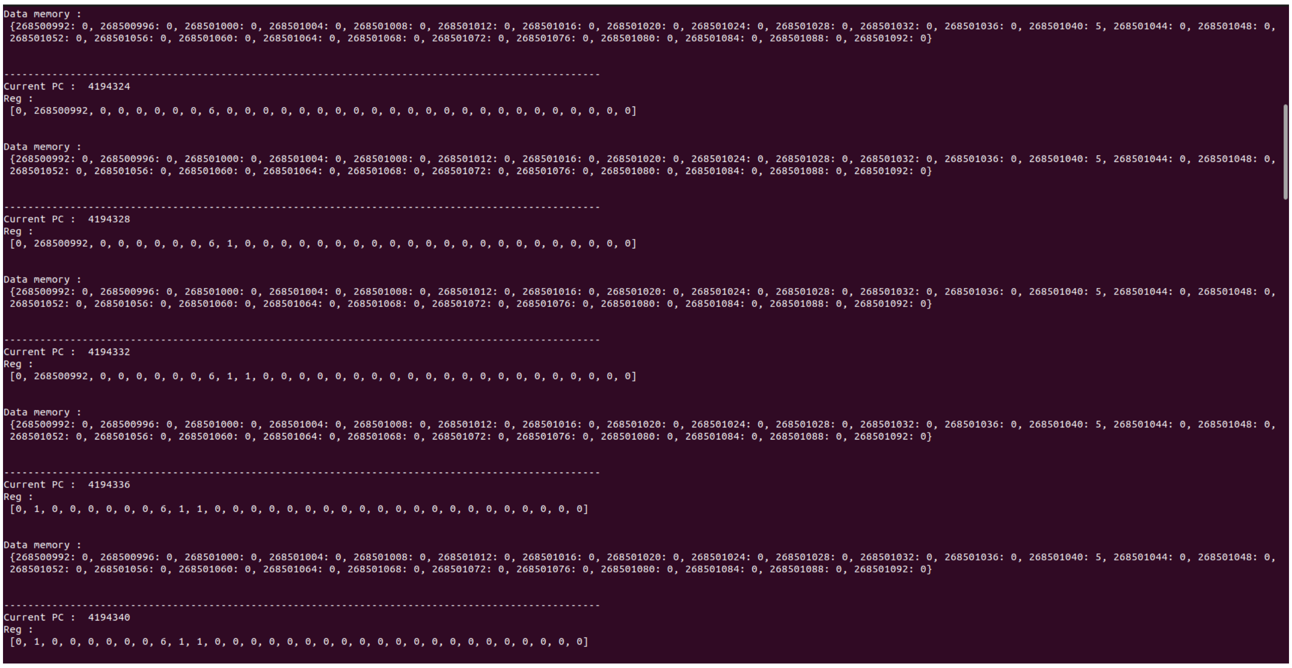


Figure 25: result 2

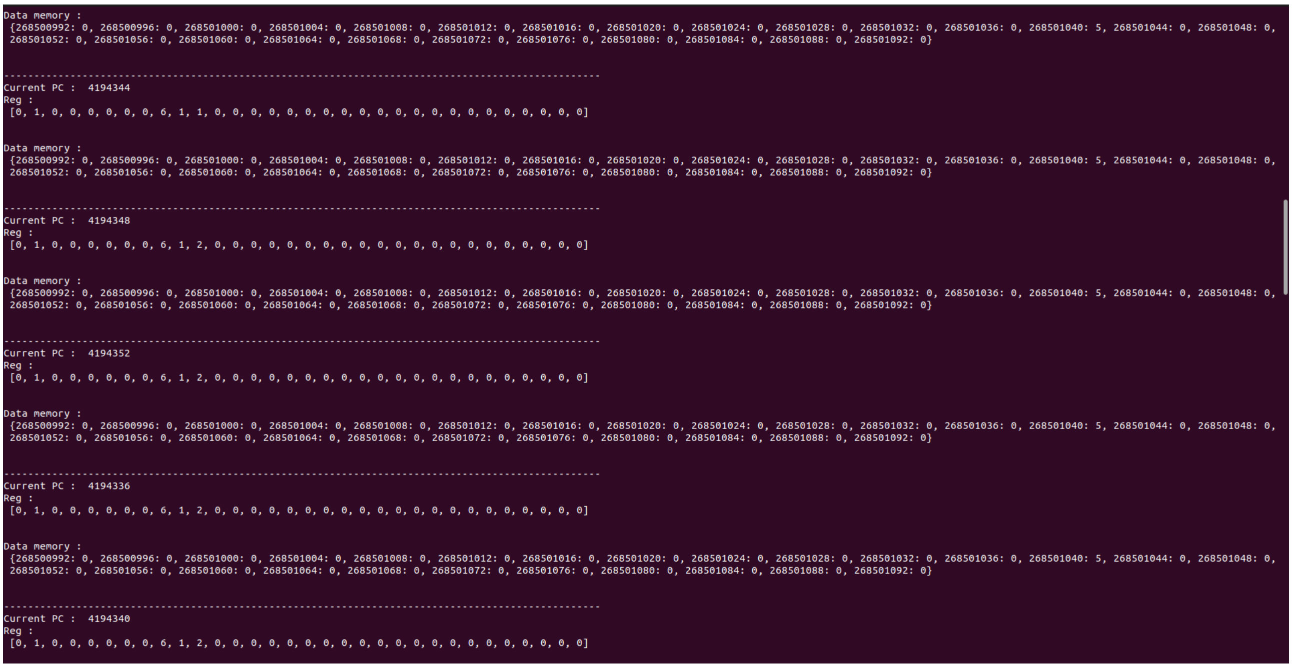
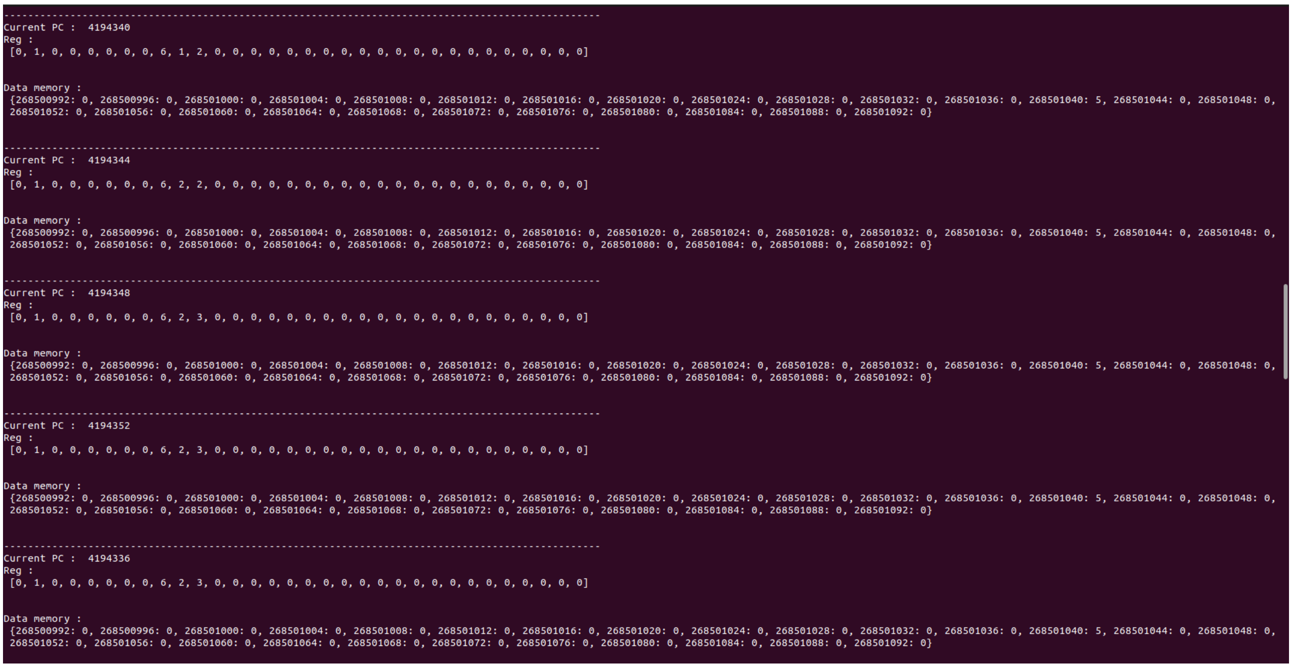
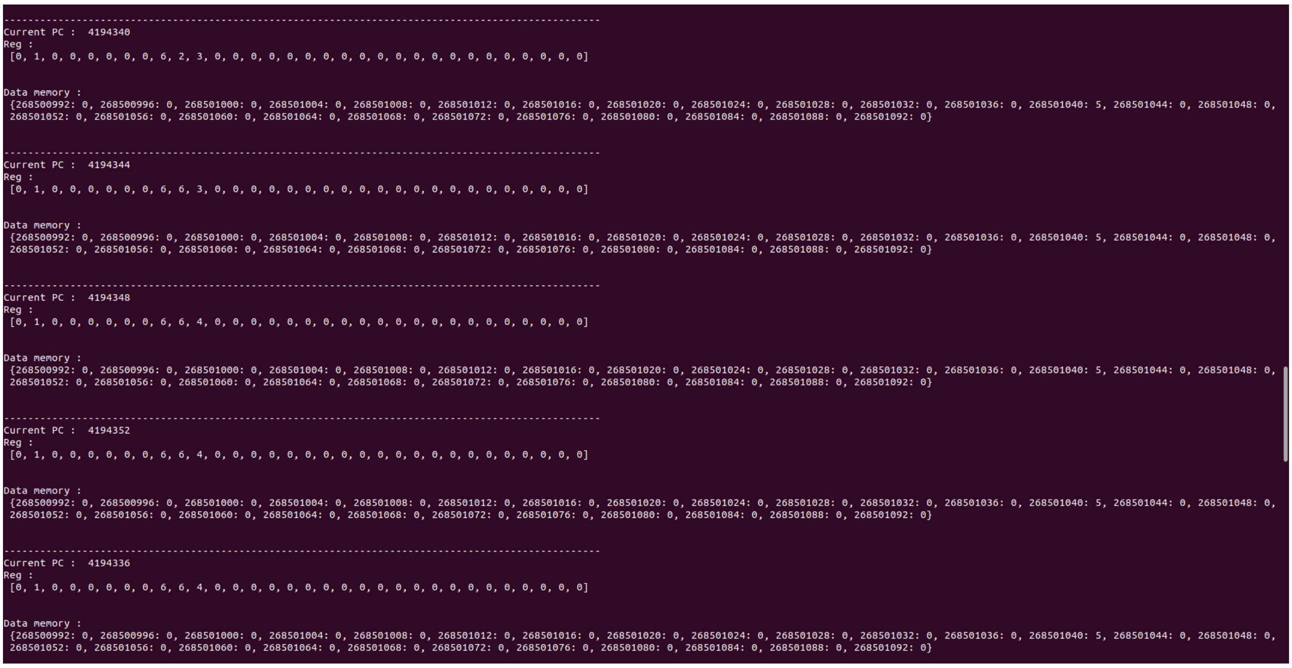


Figure 26: result 3



```
------------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

Figure 27: result 4



```
------------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 6, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 6, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 6, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

Figure 28: result 5

```
------------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 24, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 120, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 120, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 1, 0, 0, 0, 0, 0, 0, 6, 120, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 6, 120, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

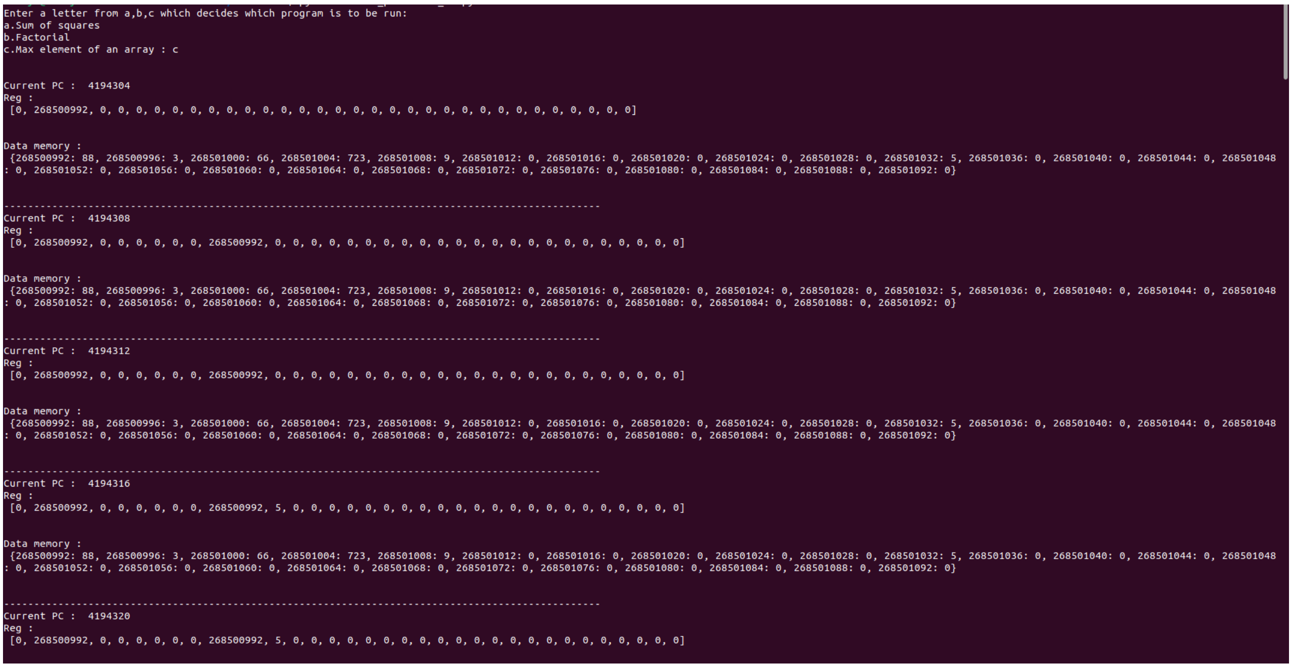
Figure 29: result 6

```
------------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 6, 120, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------


Final result:

Reg :
 [0, 0, 0, 0, 0, 0, 0, 0, 6, 120, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 0, 268500996: 0, 268501000: 0, 268501004: 0, 268501008: 0, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 0, 268501036: 0, 268501040: 5, 268501044: 0, 268501048: 0,
 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


Factorial is  120
```

Figure 30: result 7

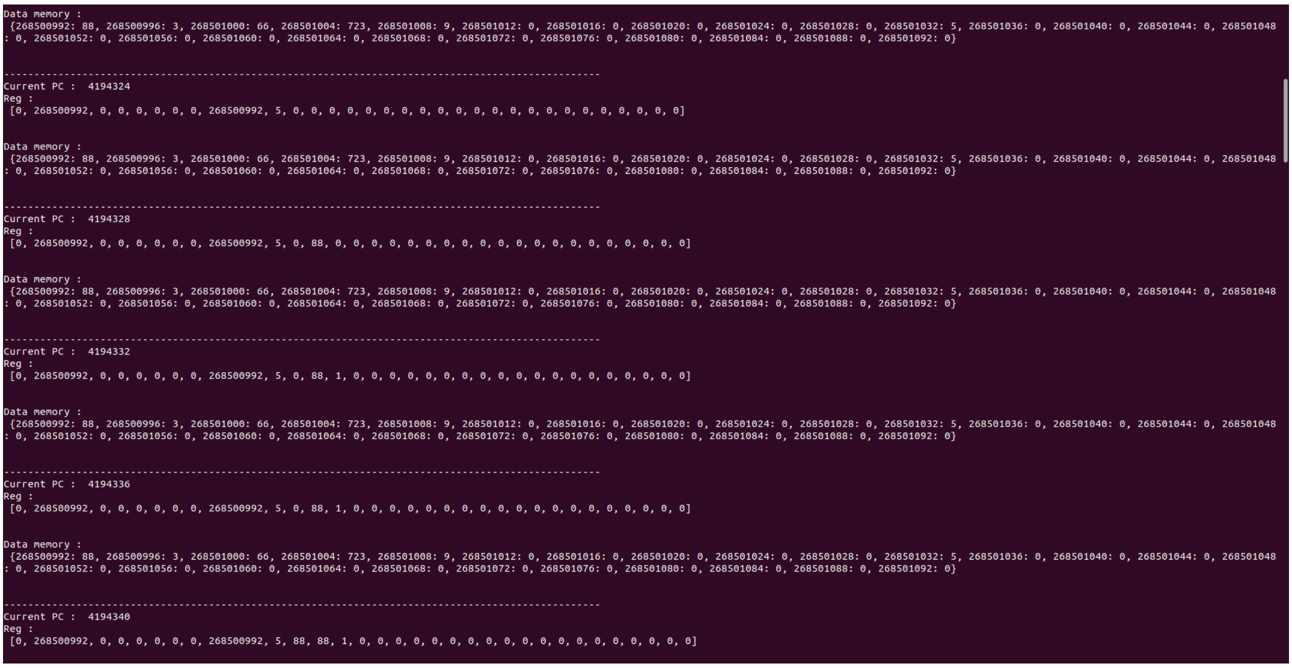
6.3 Maximum element in an array

Here the input array is [88,3,66,723,9].



```
Enter a letter from a,b,c which decides which program is to be run:
a.Sum of squares
b.Factorial
c.Max element of an array : c


Current PC :  4194304
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194308
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194312
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194316
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194320
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 31: result 1



```
Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194324
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194328
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 88, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194332
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 0, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


---------------------------------------------------------------------------------------------------
Current PC :  4194340
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500992, 5, 88, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 32: result 2

```
Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 5, 88, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 4, 88, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 4, 88, 88, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194328
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 4, 88, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194332
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 4, 88, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 33: result 3

```
Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268500996, 4, 88, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501000, 4, 88, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501000, 3, 88, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501000, 3, 88, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


----------------------------------------------------------------------------------------
Current PC :  4194328
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501000, 3, 88, 66, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 34: result 4

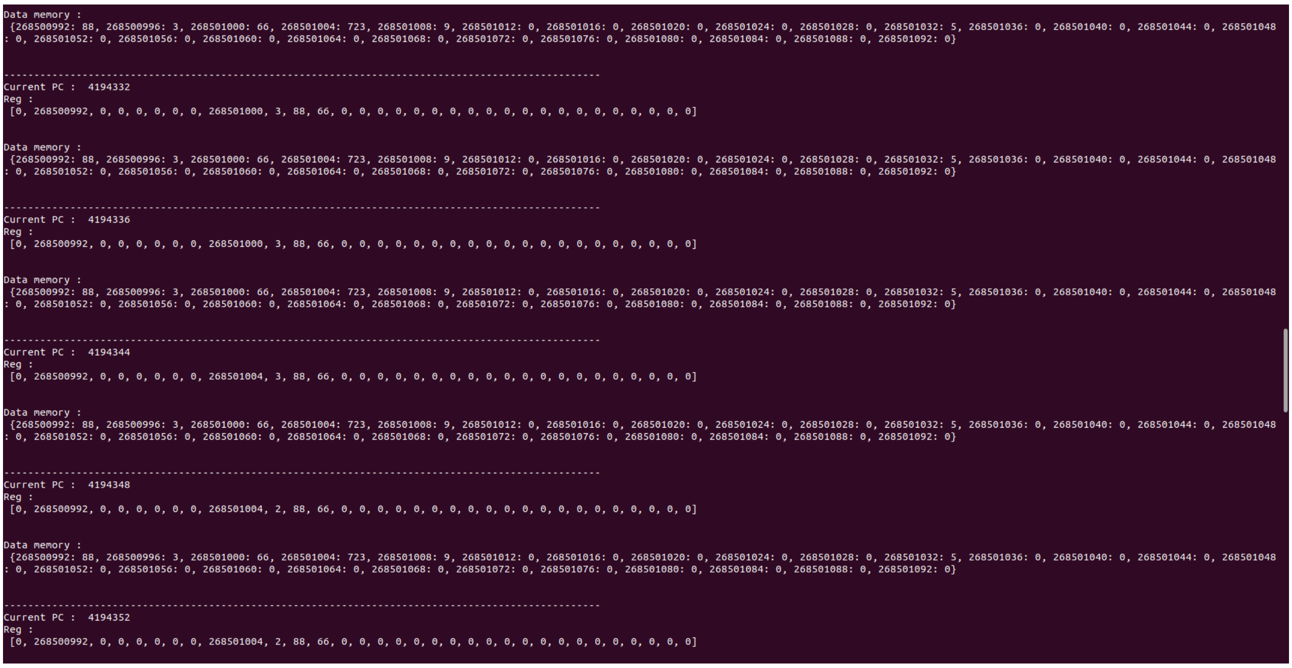


Figure 35: result 5

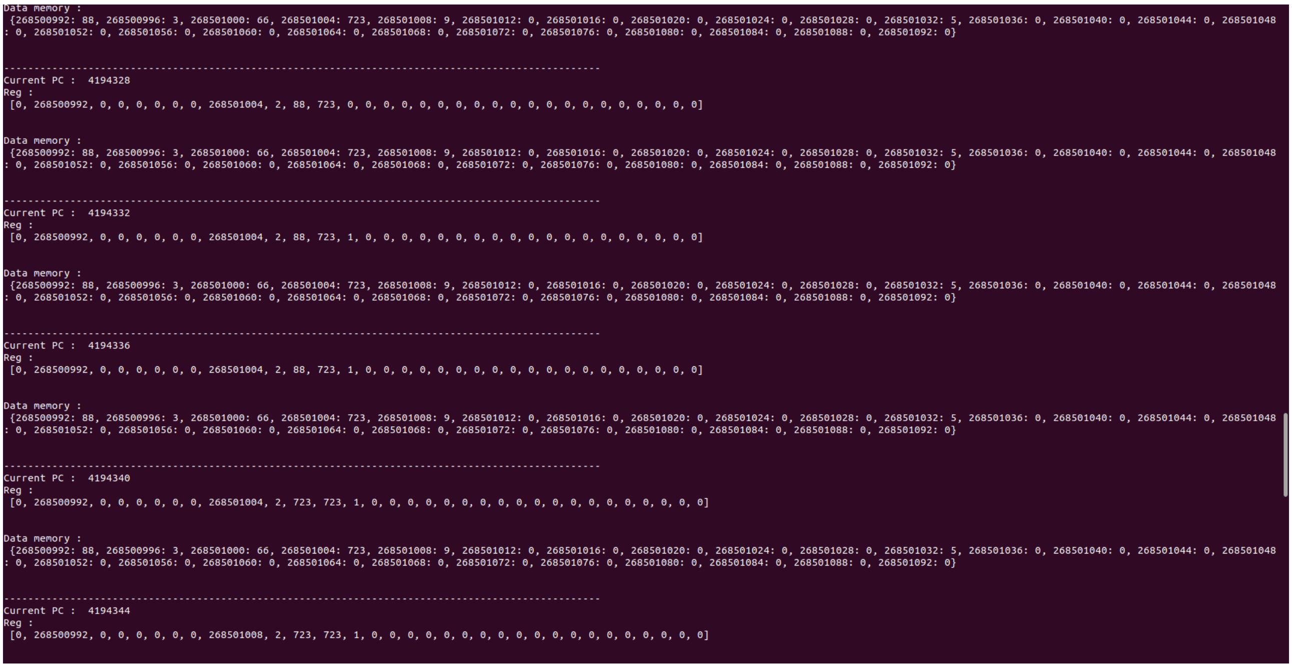


Figure 36: result 6

```
------------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501008, 1, 723, 723, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501008, 1, 723, 723, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194328
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501008, 1, 723, 9, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194332
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501008, 1, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194336
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501008, 1, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

Figure 37: result 7

```
------------------------------------------------------------------------------------------
Current PC :  4194344
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 1, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194348
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 0, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194352
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 0, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194356
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 0, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 0, 268501040: 0, 268501044: 0, 268501048
: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}


------------------------------------------------------------------------------------------
Current PC :  4194360
Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 0, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 723, 268501040: 0, 268501044: 0, 2685010
48: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

Figure 38: result 8

```
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------


Final result:

Reg :
 [0, 268500992, 0, 0, 0, 0, 0, 0, 268501012, 0, 723, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Data memory :
 {268500992: 88, 268500996: 3, 268501000: 66, 268501004: 723, 268501008: 9, 268501012: 0, 268501016: 0, 268501020: 0, 268501024: 0, 268501028: 0, 268501032: 5, 268501036: 723, 268501040: 0, 268501044: 0, 2685010
48: 0, 268501052: 0, 268501056: 0, 268501060: 0, 268501064: 0, 268501068: 0, 268501072: 0, 268501076: 0, 268501080: 0, 268501084: 0, 268501088: 0, 268501092: 0}
```

Figure 39: result 9