

Module: R2: Intro to RISC-V Assembly

Section: CALL Task: RISC-V Function & Pointers

LAB 4 - <https://github.com/ImNomanCR7/fa21-lab-starter.git>

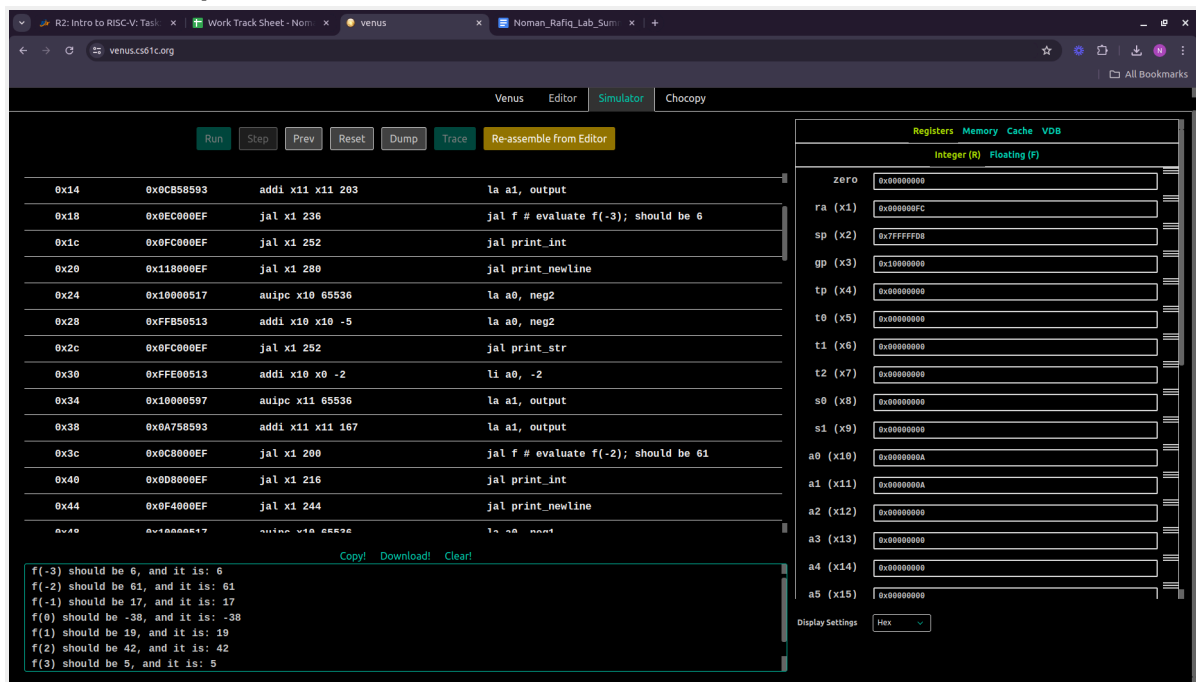
RISC-V Function & Pointers

Exercise 1: Function without Branches:

■ Code Snippet:

```
f:
# YOUR CODE GOES HERE!
addi a0, a0, 3      # add 3 to a0 to get the index
slli a0, a0, 2      # shift by 2 bits to calculate offset
add a1, a1, a0      # a1 = address of output[i]
lw a0, 0(a1)        # a0 = a[i]
jr ra               # Always remember to jr ra after your function!
```

Terminal Output:



Exercise 2: Calling Convention Checker

■ Code Snippet:

```
# Author: Noman Rafiq
# Date: July 2, 2024
# Fixed CC Violations
```

```

.globl simple_fn naive_pow inc_arr

.data
failure_message: .asciiz "Test failed for some reason.\n"
success_message: .asciiz "Sanity checks passed! Make sure there are no
CC violations.\n"
array:
    .word 1 2 3 4 5
exp_inc_array_result:
    .word 2 3 4 5 6

.text
main:
    # We test our program by loading a bunch of random values
    # into a few saved registers - if any of these are modified
    # after these functions return, then we know calling
    # convention was broken by one of these functions
    li s0, 2623
    li s1, 2910
    # ... skipping middle registers so the file isn't too long
    # If we wanted to be rigorous, we would add checks for
    # s2-s10 as well
    li s11, 134
    # Now, we call some functions
    # simple_fn: should return 1
    jal simple_fn # Shorthand for "jal ra, simple_fn"
    li t0, 1
    bne a0, t0, failure
    # naive_pow: should return 2 ** 7 = 128
    li a0, 2
    li a1, 7
    jal naive_pow
    li t0, 128
    bne a0, t0, failure
    # inc_arr: increments "array" in place
    la a0, array
    li a1, 5
    jal inc_arr
    jal check_arr # Verifies inc_arr and jumps to "failure" on failure
    # Check the values in the saved registers for sanity
    li t0, 2623
    li t1, 2910

```

```

    li t2, 134
    bne s0, t0, failure
    bne s1, t1, failure
    bne s11, t2, failure
    # If none of those branches were hit, print a message and exit
normally
    li a0, 4
    la a1, success_message
    ecall
    li a0, 10
    ecall

# Just a simple function. Returns 1.
#
# FIXME Fix the reported error in this function (you can delete lines
# if necessary, as long as the function still returns 1 in a0).
simple_fn:
    li a0, 1
    ret

# Computes a0 to the power of a1.
# This is analogous to the following C pseudocode:
#
# uint32_t naive_pow(uint32_t a0, uint32_t a1) {
#     uint32_t s0 = 1;
#     while (a1 != 0) {
#         s0 *= a0;
#         a1 -= 1;
#     }
#     return s0;
# }
#
# FIXME There's a CC error with this function!
# The big all-caps comments should give you a hint about what's
# missing. Another hint: what does the "s" in "s0" stand for?
naive_pow:
    # BEGIN PROLOGUE
    addi sp, sp, -4
    sw s0, 0(sp)
    # END PROLOGUE
    li s0, 1
naive_pow_loop:

```

```

    beq a1, zero, naive_pow_end
    mul s0, s0, a0
    addi a1, a1, -1
    j naive_pow_loop
naive_pow_end:
    mv a0, s0
    # BEGIN EPILOGUE
    lw s0, 0(sp)
    addi sp, sp, 4
    # END EPILOGUE
    ret

```

```

# Increments the elements of an array in-place.
# a0 holds the address of the start of the array, and a1 holds
# the number of elements it contains.
#
# This function calls the "helper_fn" function, which takes in an
# address as argument and increments the 32-bit value stored there.
inc_arr:

```

```

    # BEGIN PROLOGUE
    # FIXME What other registers need to be saved? s0 & s1!
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)
    # END PROLOGUE
    mv s0, a0 # Copy start of array to saved register
    mv s1, a1 # Copy length of array to saved register
    li t0, 0 # Initialize counter to 0
inc_arr_loop:
    beq t0, s1, inc_arr_end
    slli t1, t0, 2 # Convert array index to byte offset
    add a0, s0, t1 # Add offset to start of array
    # Prepare to call helper_fn
    addi sp, sp, -4
    sw t0, 0(sp)
    # FIXME Add code to preserve the value in t0 before we call
    helper_fn
    # Hint: What does the "t" in "t0" stand for?
    # Also ask yourself this: why don't we need to preserve t1?
    #
    jal helper_fn

```

```

    # Finished call for helper_fn
    lw t0, 0(sp)
    addi sp, sp, 4
    addi t0, t0, 1 # Increment counter
    j inc_arr_loop
inc_arr_end:
    # BEGIN EPILOGUE
    lw ra, 0(sp)
    lw s0, 4(sp)
    lw s1, 8(sp)
    addi sp, sp, 12
    # END EPILOGUE
    ret

```

This helper function adds 1 to the value at the memory address in a0.

It doesn't return anything.

C pseudocode for what it does: `"*a0 = *a0 + 1"`

#

FIXME This function also violates calling convention, but it might not

be reported by the Venus CC checker (try and figure out why).

You should fix the bug anyway by filling in the prologue and epilogue

as appropriate.

helper_fn:

```

    # BEGIN PROLOGUE
    addi sp, sp, -4
    sw s0, 0(sp)
    # END PROLOGUE
    lw t1, 0(a0)
    addi s0, t1, 1
    sw s0, 0(a0)
    # BEGIN EPILOGUE
    lw s0, 0(sp)
    addi sp, sp, 4
    # END EPILOGUE
    ret

```

YOU CAN IGNORE EVERYTHING BELOW THIS COMMENT

Checks the result of inc_arr, which should contain 2 3 4 5 6 after
one call.

You can safely ignore this function; it has no errors.

check_arr:

```

    la t0, exp_inc_array_result
    la t1, array
    addi t2, t1, 20 # Last element is 5*4 bytes off
check_arr_loop:
    beq t1, t2, check_arr_end
    lw t3, 0(t0)
    lw t4, 0(t1)
    bne t3, t4, failure
    addi t0, t0, 4
    addi t1, t1, 4
    j check_arr_loop
check_arr_end:
    ret

```

This isn't really a function - it just prints a message, then
terminates the program on failure. Think of it like an exception.
failure:

```

    li a0, 4 # String print ecall
    la a1, failure_message
    ecall
    li a0, 10 # Exit ecall
    ecall

```

- What caused the errors in `simple_fn`, `naive_pow`, and `inc_arr` that were reported by the Venus CC checker?
 - `simple_fn`: `t0` was used before it was initialized.

- **naive_pow**: Missing epilogue and prologues.
- **inc_arr**: Failure to save **s0**, **s1** in prologue/epilogue, and failure to save **t0** before calling **helper_fn**.
- **In RISC-V, we call functions by jumping to them and storing the return address in the ra register. Does calling convention apply to the jumps to the naive_pow_loop or naive_pow_end labels?**
 - No, since they're not functions, we don't need to return to the location the function was called from.
- **Why do we need to store ra in the prologue for inc_arr, but not in any other Function?**
 - inc_arr itself calls another function so it is a caller as well.
 - Since **ra** holds the address of the instruction to continue executing after returning, which is overwritten when we call another function. Therefore we need to save the **ra** before it is overwritten by another call.
- **Why wasn't the calling convention error in helper_fn reported by the CC checker?**
 - Because it is not declared **.globl**.

Exercise 3: Debugging:

■ Code Snippet:

```
.globl map

.data
arrays: .word 5, 6, 7, 8, 9
        .word 1, 2, 3, 4, 7
        .word 5, 2, 7, 4, 3
        .word 1, 6, 3, 8, 4
        .word 5, 2, 7, 8, 1

start_msg: .ascii "Lists before: \n"
end_msg:   .ascii "Lists after: \n"

.text
main:
    jal create_default_list
    mv s0, a0    # v0 = s0 is head of node list
```

```

    #print "lists before: "
    la a1, start_msg
    li a0, 4
    ecall

    #print the list
    add a0, s0, x0
    jal print_list

    # print a newline
    jal print_newline

    # issue the map call
    add a0, s0, x0      # load the address of the first node into a0
    la a1, mystery      # load the address of the function into a1

    jal map

    # print "lists after: "
    la a1, end_msg
    li a0, 4
    ecall

    # print the list
    add a0, s0, x0
    jal print_list

    li a0, 10
    ecall

map:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s1, 4(sp)
    sw s0, 8(sp)

    beq a0, x0, done    # if we were given a null pointer, we're done.

    add s0, a0, x0      # save address of this node in s0
    add s1, a1, x0      # save address of function in s1
    add t0, x0, x0      # t0 is a counter

```



```

# remember that each node is 12 bytes long:
# - 4 for the array pointer
# - 4 for the size of the array
# - 4 more for the pointer to the next node

# also keep in mind that we should not make ANY assumption on which
registers
# are modified by the callees, even when we know the content inside
the functions
# we call. this is to enforce the abstraction barrier of calling
convention.
mapLoop:
    lw t1, 0(s0)      # load the address of the array of current node
into t1
    lw t2, 4(s0)      # load the size of the node's array into t2

    slli t3, t0, 2
    add t3, t1, t3     # offset the array address by the count
    lw a0, 0(t3)      # load the value at that address into a0

    jalr s1           # call the function on that value.

    sw a0, 0(t3)      # store the returned value back into the array

    addi t0, t0, 1     # increment the count
    bne t0, t2, mapLoop # repeat if we haven't reached the array size
yet

    lw a0, 8(s0)      # load the address of the next node into a0
    mv a1, s1         # put the address of the function back into a1
to prepare for the recursion

    jal map           # recurse
done:
    lw s0, 8(sp)
    lw s1, 4(sp)
    lw ra, 0(sp)
    addi sp, sp, 16

print_newline:
    li a1, '\n'

```

```

    li a0, 11
    ecall
    jr ra

mystery:
    mul t1, a0, a0
    add a0, t1, a0
    jr ra

create_default_list:
    addi sp, sp, -24
    sw ra, 0(sp)
    sw s0, 4(sp)
    sw s1, 8(sp)
    sw s2, 12(sp)
    sw s3, 16(sp)
    sw s4, 20(sp)
    li s0, 0 # pointer to the last node we handled
    li s1, 0 # number of nodes handled
    li s2, 5 # size
    la s3, arrays
loop: #do...
    li a0, 12
    jal malloc # get memory for the next node
    mv s4, a0
    li a0, 20
    jal malloc # get memory for this array

    sw a0, 0(s4) # node->arr = malloc
    lw a0, 0(s4)
    mv a1, s3
    jal fillArray # copy ints over to node->arr

    sw s2, 4(s4) # node->size = size (4)
    sw s0, 8(s4) # node-> next = previously created node

    add s0, x0, s4 # last = node
    addi s1, s1, 1 # i++
    addi s3, s3, 20 # s3 points at next set of ints
    li t6 5
    bne s1, t6, loop # ... while i!= 5
    mv a0, s4

```

```

lw ra, 0(sp)
lw s0, 4(sp)
lw s1, 8(sp)
lw s2, 12(sp)
lw s3, 16(sp)
lw s4, 20(sp)
addi sp, sp, 24
jr ra

```

```

fillArray: lw t0, 0(a1) #t0 gets array element
           sw t0, 0(a0) #node->arr gets array element
           lw t0, 4(a1)
           sw t0, 4(a0)
           lw t0, 8(a1)
           sw t0, 8(a0)
           lw t0, 12(a1)
           sw t0, 12(a0)
           lw t0, 16(a1)
           sw t0, 16(a0)
           jr ra

```

```

print_list:
    bne a0, x0, printMeAndRecurse
    jr ra    # nothing to print
printMeAndRecurse:
    mv t0, a0 # t0 gets address of current node
    lw t3, 0(a0) # t3 gets array of current node
    li t1, 0    # t1 is index into array
printLoop:
    slli t2, t1, 2
    add t4, t3, t2
    lw a1, 0(t4) # a0 gets value in current node's array at index t1
    li a0, 1    # prepare for print integer ecall
    ecall
    li a1, ' '  # a0 gets address of string containing space
    li a0, 11   # prepare for print string ecall
    ecall
    addi t1, t1, 1
    li t6 5
    bne t1, t6, printLoop # ... while i!= 5
    li a1, '\n'
    li a0, 11

```

```

    ecall
    lw a0, 8(t0) # a0 gets address of next node
    j print_list # recurse. We don't have to use jal because we already
have where we want to return to in ra

```

malloc:

```

    mv a1, a0 # Move a0 into a1 so that we can do the syscall correctly
    li a0, 9
    ecall
    jr ra

```

Terminal Output:

The terminal output is displayed in two separate windows. The top window shows the state 'Lists before:' with five lines of numbers: '5 2 7 8 1', '1 6 3 8 4', '5 2 7 4 3', '1 2 3 4 7', and '5 6 7 8 9'. The bottom window shows the state 'Lists after:' with five lines of numbers: '30 6 56 72 2', '2 42 12 72 20', '30 6 56 20 12', '2 6 12 20 56', and '30 42 56 72 90'. Below the 'Lists after:' section, it says 'Found 0 warnings!'. Both windows have a dark background with light green text and buttons for 'Copy!', 'Download!', and 'Clear!' at the top.

Action Items:

- Find the six mistakes inside the map function in megalistmanips.s.
 - In the **map** label (line 2) the stack space allocated was of 12 bytes but we need actually 16 bytes since each node is of 12 bytes (- 4 for the array pointer - 4 for the size of the array - 4 more for the pointer to the next node) we need to save these 12 bytes plus the return address on the stack. Total of 16 bytes required.
 - In the **mapLoop** label we have to load the address of the array of current node into **t1**. So in order to load the address into **t1** we should use **lw** instead of add Instruction.
 - In the **mapLoop** label we have to offset the array address by the count but here we are just adding the count to the address and not adding the offset at all.
 - In the **mapLoop** label we have to load the address of next node in **a0** so instead of using the **la** here which takes the label as an input we should use **lw** instructions here.

- In line 84 we have to put the address back into **a1**. We should just use **mv** instructions to copy the address back into a1.
- In the **done** label the stack space deallocated should be of 16 bytes rather than 12 bytes and at the end the it should return back to function. So **ret** instructions should be added at the end.

After resolving the above bugs, the code execution using -cc to check for calling convention violations. Here's the output:

```

xe-user106@noman-10xengineers: ~/10x-Engineers/Remedia...
V/fa21-lab-starter$ java -jar tools/venus.jar lab03/ex2.s
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC
V/fa21-lab-starter$ java -jar tools/venus.jar -cc lab04/megalistmanips.s
Lists before:
5 2 7 8 1
1 6 3 8 4
5 2 7 4 3
1 2 3 4 7
5 6 7 8 9

Lists after:
30 6 56 72 2
2 42 12 72 20
30 6 56 20 12
2 6 12 20 56
30 42 56 72 90
Found 0 warnings!xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training
/R2-Intro-to-RISCV/fa21-lab-starter$

```

- For this exercise, we are requiring that you don't use any extra save registers in your implementation. While you normally can use the save registers to store values that you want to use after returning from a function (in this case, when we're calling f in map), we want you to use temporary registers instead and follow their caller/callee conventions. The provided map implementation only uses the s0 and s1 registers, so we'll require that you don't use s2-s11.
 - Yes! I have not used any saved register other than **s0** and **s1** but I have used a temporary register **t3** to preserve the value of **t1** when **jalr s1** was called.

- **Make an ordered list of each of the six mistakes in the megalistmanips_answers.txt file, and the corrections you made to fix them.**
 - Please refer to the attached **megalistmanips_answers.txt** file. Alternatively, you can find the file here:
https://github.com/ImNomanCR7/fa21-lab-starter/blob/main/lab04/megalistmanips_answers.txt
- **Save your corrected code in the megalistmanips.s file. Use the -cc flag to run a basic calling convention check on your code locally:**

```
xe-user106@noman-10xengineers: ~/10x-Engineers/Remedia...
V/fa21-lab-starter$ java -jar tools/venus.jar lab03/ex2.s
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC
V/fa21-lab-starter$ java -jar tools/venus.jar -cc lab04/megalistmanips.s
Lists before:
5 2 7 8 1
1 6 3 8 4
5 2 7 4 3
1 2 3 4 7
5 6 7 8 9

Lists after:
30 6 56 72 2
2 42 12 72 20
30 6 56 20 12
2 6 12 20 56
30 42 56 72 90
Found 0 warnings!xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training
/R2-Intro-to-RISCV/fa21-lab-starter$
```

Exercise 4: Finding and Solving Bugs:

Action Items:

1. **Find the bugs in four of the five accumulators.**
 - **accumulatorone:** In this accumulator the **s0** register was not saved onto the stack before it was used and so the value of **s0** was not preserved.
 - **accumulatortwo:** The stack pointer is incremented by four in the prologue, and decremented by four in the epilogue. This breaks the stack of the caller.
 - **accumulatorthree:** This accumulator is correct there is no issue in it.
 - **accumulatorfour:** The function relies on **t2** being set to zero before running. Otherwise, it returns t2 plus the desired sum.

- **accumulatorfive:** The function doesn't check if the first element is zero. As such, it fails to provide the correct result if the first element is zero, and if the second element is nonzero.

2. For each broken accumulator, write a test that fails on the broken one, but passes the correct implementation.

➤ **Code Snippet:**

```
.import lotsofaccumulators.s

.data
inputarray: .word 1,2,3,4,5,6,7,0
inputarray1: .word 0,1,2,3,4,5,6,7,0

TestPassed: .ascii "Test Passed!"
TestFailed: .ascii "Test Failed!"

.text
# Tests if the given implementation of accumulate is
correct.
#Input: a0 contains a pointer to the version of accumulate
in question. See lotsofaccumulators.s for more details
#
#
#
#The main function currently runs a simple test that checks
if accumulator works on the given input array. All versions
of accumulate should pass this.
#Modify the test so that you can catch the bugs in four of
the five solutions!
main:

    la a0, inputarray    # for accumulators 1 to 4

    # ***** Uncomment below statement for Accumulator 5
    only ***** #
    la a0 inputarray1    # fail accumulatorfive

    li s0, 3             #fail accumulatorone
    li t2, 12            #fail accumulatortwo
    addi sp, sp, -4
```

```

    sw t2, 0(sp)

    jal accumulatorfive

    lw t1, 0(sp)
    addi sp, sp, 4
    li t3, 3
    bne t3, s0, Fail    #fail accumulatorone
    bne t2, t1, Fail    #fail accumulatortwo &
accumulatorfour
    beq a0 x0 Fail

    li t0, 28
    li t0, 0           # fail accumulatorfive (please
uncomment for accumulator 5)
    bne a0, t0, Fail
    j Pass

Fail:
    la a0 TestFailed
    jal print_string
    j End
Pass:
    la a0 TestPassed
    jal print_string
End:
    jal exit

print_int:
    mv a1 a0
    li a0 1
    ecall
    jr ra

print_string:
    mv a1 a0
    li a0 4
    ecall
    jr ra

exit:
    li a0 10

```


ecall

3. Terminal Outputs for the Accumulators.

○ accumulatorone:

```

xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter
xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$

17
18 TestPassed: .asciiz "Test Passed!"
19 TestFailed: .asciiz "Test Failed!"
20
21 .text
22 # Tests if the given implementation of accumulate is correct.
23 # Input: a0 contains a pointer to the version of accumulate to question. See listofaccumulators.s
24 # for more details.
25
26 # The main function currently runs a single test that checks if accumulator works on the given input array. All versions of accumulate should pass this.
27 # Modify the test so that you can catch the bugs in four of the five solutions!
28
29 main:
30     la a0, inputarray    # for accumulators 1 to 4
31     # ***** uncomment below statement for Accumulator 5 only ***** #
32     #la a0, inputarray5    # fail accumulatorfive
33
34     li s0, 3    #fail accumulatorone
35     li t2, 12    #fail accumulatortwo
36     addi sp, sp, -4
37     sw t2, 0(sp)
38
39     jal accumulatorone
40
41     lw t1, 0(sp)
42     addi sp, sp, 4
43     li t3, 0
44     bne t1, s0, fail    #fail accumulatorone
45     bne t2, t1, fail    #fail accumulatortwo & accumulatorfour
46     beq a0, a0, fail
47
48     li t0, 20
49     #fail accumulatorfive (please uncomment for accumulator 5)
50     bne a0, t0, fail
51     j pass
52
53 fail:
54     la a0, TestFailed
55     jal print_string
56     j end
57
58 pass:
59     la a0, TestPassed
60     jal print_string
61     j end
62
63 End:
64     jal exit
65
66 print_int:
67     mv a1, a0
68     lab04/accumulatortests.s 24,5 310
69     "lab04/accumulatortests.s" 69L, 1525B written 2024-07-03 10:17

```

○ accumulatortwo:

```

xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter
xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$
xe-user106$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10engineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$

24
25 TestPassed: .asciiz "Test Passed!"
26 TestFailed: .asciiz "Test Failed!"
27
28 .text
29 # Tests if the given implementation of accumulate is correct.
30 # Input: a0 contains a pointer to the version of accumulate to question. See listofaccumulators.s
31 # for more details.
32
33 # The main function currently runs a single test that checks if accumulator works on the given input array. All versions of accumulate should pass this.
34 # Modify the test so that you can catch the bugs in four of the five solutions!
35
36 main:
37     la a0, inputarray    # for accumulators 1 to 4
38     # ***** uncomment below statement for Accumulator 5 only ***** #
39     #la a0, inputarray5    # fail accumulatorfive
40
41     li s0, 3    #fail accumulatorone
42     li t2, 12    #fail accumulatortwo
43     addi sp, sp, -4
44     sw t2, 0(sp)
45
46     jal accumulatortwo
47
48     lw t1, 0(sp)
49     addi sp, sp, 4
50     li t3, 0
51     bne t1, s0, fail    #fail accumulatorone
52     bne t2, t1, fail    #fail accumulatortwo & accumulatorfour
53     beq a0, a0, fail
54
55     li t0, 20
56     #fail accumulatorfive (please uncomment for accumulator 5)
57     bne a0, t0, fail
58     j pass
59
60 fail:
61     la a0, TestFailed
62     jal print_string
63     j end
64
65 pass:
66     la a0, TestPassed
67     jal print_string
68     j end
69
70 End:
71     jal exit
72
73 print_int:
74     mv a1, a0
75     lab04/accumulatortests.s 31,22 311
76     "lab04/accumulatortests.s" 69L, 1525B written 2024-07-03 10:18

```

○ accumulatorthree:

```

xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Passed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$

24
25 TestPassed: .asciiz "Test Passed!"
26 TestFailed: .asciiz "Test Failed!"
27
28
29 .text
30 # Tests if the given implementation of accumulate is correct.
31 # Input: a0 contains a pointer to the version of accumulate in question. See lab04/accumulators.s
32 # for more details.
33
34 # The main function currently runs a single test that checks if accumulator works on the given input
35 # array. All versions of accumulate should pass this.
36 # Modify the test so that you can catch the bugs in four of the five solutions!
37
38 main:
39
40     la a0, inputarray    # for accumulators 1 to 4
41
42     # ***** Uncomment below statement for Accumulator 5 only ***** #
43     #la a0, inputarray    # fail accumulatorfive
44
45     li t0, 0              # fail accumulatorone
46     li t1, 0              # fail accumulatortwo
47     addi sp, sp, -4
48     sw t0, 0(sp)
49
50     jal accumulatorthree
51
52     lw t1, 0(sp)
53     addi sp, sp, 4
54     li t3, 0
55     bne t3, t0, fail      # fail accumulatorone
56     bne t3, t1, fail      # fail accumulatortwo & accumulatorfour
57     beq a0, a0, fail
58
59     li t0, 20
60
61     # fail accumulatorfive (please uncomment for accumulator 5)
62     bne a0, t0, fail
63     j Pass
64
65 fail:
66     la a0, TestFailed
67     jal print_string
68     j End
69
70 Pass:
71     la a0, TestPassed
72     jal print_string
73 End:
74     jal exit
75
76 print_int:
77     mv a1, a0
78
79 lab04/accumulatortests.s 31,24 311
80 lab04/accumulatortests.s" 69L, 1527B Written
81
82 2024-07-01 16:18

```

○ **accumulatorfour:**

```

xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Failed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$ java -jar tools/venus.jar lab04/accumulatortests.s
Test Passed:xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISC-V/faz1-lab-starter$

24
25 TestPassed: .asciiz "Test Passed!"
26 TestFailed: .asciiz "Test Failed!"
27
28
29 .text
30 # Tests if the given implementation of accumulate is correct.
31 # Input: a0 contains a pointer to the version of accumulate in question. See lab04/accumulators.s
32 # for more details.
33
34 # The main function currently runs a single test that checks if accumulator works on the given input
35 # array. All versions of accumulate should pass this.
36 # Modify the test so that you can catch the bugs in four of the five solutions!
37
38 main:
39
40     la a0, inputarray    # for accumulators 1 to 4
41
42     # ***** Uncomment below statement for Accumulator 5 only ***** #
43     #la a0, inputarray    # fail accumulatorfive
44
45     li t0, 0              # fail accumulatorone
46     li t1, 0              # fail accumulatortwo
47     addi sp, sp, -4
48     sw t0, 0(sp)
49
50     jal accumulatorfour
51
52     lw t1, 0(sp)
53     addi sp, sp, 4
54     li t3, 0
55     bne t3, t0, fail      # fail accumulatorone
56     bne t3, t1, fail      # fail accumulatortwo & accumulatorfour
57     beq a0, a0, fail
58
59     li t0, 20
60
61     # fail accumulatorfive (please uncomment for accumulator 5)
62     bne a0, t0, fail
63     j Pass
64
65 fail:
66     la a0, TestFailed
67     jal print_string
68     j End
69
70 Pass:
71     la a0, TestPassed
72     jal print_string
73 End:
74     jal exit
75
76 print_int:
77     mv a1, a0
78
79 lab04/accumulatortests.s 31,23 311
80 lab04/accumulatortests.s" 69L, 1526B Written
81
82 2024-07-01 16:20

```

○ **accumulatorfive:**

```

xe-user106@noman-10xengineers: ~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$ j
xe -jar tools/venus.jar Lab04/accumulatorTests.s
Test Failed: xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$ java -jar tools/venus.jar Lab04/accumulatorTests.s
Test Failed: xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$ java -jar tools/venus.jar Lab04/accumulatorTests.s
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$ java -jar tools/venus.jar Lab04/accumulatorTests.s
Test Failed: xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$ java -jar tools/venus.jar Lab04/accumulatorTests.s
Test Failed: xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R2-Intro-to-RISCv/Faz1-lab-starter$

34
35 TestPassed: acExit "Test Passed"
36 TestFailed: acExit "Test Failed"
37
38
39 text
40
41 # P Tests if the given implementation of accumulate is correct.
42 # Input: a0 contains a pointer to the version of accumulate in question. See lab04/accumulators
43 # for more details.
44
45
46
47
48
49
50
51 # The main function correctly runs a simple test that checks if accumulator works on the given test
52 # array. All versions of accumulate should pass this.
53 Modify the test so that you can catch the bugs in four of the five solutions!
54 main:
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
```