CS 61C Fall 2023 — RISC-V Assembly, Functions — Discussion 4

1 Data Transfer

Using the given instructions and the sample memory array, what will happen when the RISC-V code is executed? For load instructions (`lw, lb, lh`), write out what each register will store. For store instructions (`sw, sh, sb`), update the memory array accordingly. Recall that RISC-V is little-endian and byte addressable.

```
li x5 0x00FF0000
lw x6 0(x5)
addi x5 x5 4
lhu x7 1(x5)
lh x8 1(x5)
lb x9 3(x6)
sh x8 2(x5)
```

0xFFFFFFFF	
	...
	0x00
	0xAC
	0x56
0x00FF0004	0x1C
	0x00
	0xAB
	0x01
0x00FF0000	0x24
	..
	0xDE
	0xAD
	0xBE
0x00AB0124	0xEF
	...
0x00000000	

2 Arrays in RISC-V

Comment what the following code block does. Assume that there is an array, `int arr[6] = {3, 1, 4, 1, 5, 9}`, which starts at memory address `0xBFFFFF00`. Let `s0` contain `arr`'s address `0xBFFFFF00`. You may assume integers and pointers are 4 bytes.

2.1

```
       add  t0, x0, x0
loop:  slti t1, t0, 6
       beq  t1, x0, end
       slli t2, t0, 2
       add  t3, s0, t2
       lw   t4, 0(t3)
       sub  t4, x0, t4
       sw   t4, 0(t3)
       addi t0, t0, 1
       jal  x0, loop
 end:
```

2.2 **Conceptual check:** Let `a0` point to the start of an array `x`. `lw s0, 4(a0)` will always load `x[1]` into `s0`.

3 Calling Convention Practice

Function `myfunc` takes in two arguments: a0, a1. The return value is stored in a0. In `myfunc`, `generate_random` is called. It takes in 0 arguments and stores its return value in `a0`.

```
myfunc:
    # Prologue (omitted)

    addi t0 x0 1
    slli t1 t0 2
    add t1 a0 t1
    add s0 a1 x0

    jal generate_random

    add t1 t1 a0
    add a0 t1 s0

    # Epilogue (omitted)
    ret
```

3.1 Which registers, if any, need to be saved on the stack in the prologue?

3.2 Which registers do we need to save on the stack before calling `generate_random`?

3.3 Which registers need to be recovered in the epilogue before returning?