Assignment 7, Part A

Adrian Lozada

April 17, 2023

1

Independent

2

Write a RISC-V assembly function to search a specified integer in an integer array. The function should take the base address of the array, the number of elements in the array, and the specified integer as function arguments. The function should return the index number of the first array entry that holds the specified value. If no array element is the specified value, it should return the value -1.

```
# a0 is is the base of the array.
# a1 is the number of elements in the array.
# a2 is the number that the function searches for.

# The function returns the index in a0 (if found), otherwise -1.

find_int:
        addi t0, zero, 1 # t0 is 1.
        if:
            blt a1, t0, return_1 # If size < 1 return -1
        add t0, a0, zero # t0 is a pointer to a0.
        slli t3, a1, 2 # t3 is the size of the array in bytes.
        add t1 a0, t3 # t1 is a pointer to the last element of the array.

        loop:
            blt t1, t0, return_1 # t1 < t0
            lw t2, 0(t0) # t2 = *t0
            if_equal:
                    beq t2, a2, return_found # If t2 == a2 return the index.
            addi t0, t0, 4 # t0++
            jal zero, loop

        return_found:
                    sub a0, t0, a0 # Calculate the index.
                    srai a0, a0, 2
                    jalr zero, ra, 0
        return_1:
                addi a0, zero, -1 # Return -1.
                jalr zero, ra, 0
```

3

Consider a RISC-V assembly function func1. func1 has three passing arguments stored in registers a0, a1 and a2, uses temporary registers t0-t3 and saved registers s4-s10. func1 needs to call func2 and other functions may call func1 also. func2 has two passing arguments stored in registers a0 and a1, respectively. In func1, after the program returns to func1 from func2, the code needs the original values stored in registers t1 and a0 before it calls func2.

a. Ten words need to be stored in the stack.
b. Funct1 needs to store the values inside a0, t1, s4-s10, ra in the stack.

4

Implement the C code snippet in RISC-V assembly language. Use s0 and s1 to hold the variable i, and min_idx in the function selectionSort. Be sure to handle the stack pointer appropriately. Clearly comment on your code.

a)

```
# selectionSort takes a0 (base address) and a1 (number of elements).
      selectionSort:
                        # s0 is i
                        # s1 is min_idx
                        # s2 is a0 or arr[]
                        # s3 is a1 or n
                        # s4 is n - 1

                        # Save the s and ra registers into the stack:
                        addi sp, sp, -24
                        sw s0, 0(sp)
                        sw s1, 4(sp)
                        sw s2, 8(sp)
                        sw s3, 12(sp)
                        sw s4, 16(sp)
                        sw ra, 20(sp)

                        # Store a0 and a1:
                        add s2, a0, zero
                        add s3, a1, zero

                        # Sort the array:
                        addi s4, a1, -1 # pass n - 1
                        add s0, zero, zero # i = 0
                        for:
                            bge s0, s4, end_loop # i >= n - 1 goto end_loop

                            # Call findMinimum function:
                            slli t0, s0, 2 # i * 4
                            add a0, s2, t0 # pass &array[i] to a0
                            sub a1, s3, s0 # pass n - i
                            jal ra, findMinimum

                            # Set min_idx to the value in a0:
```

```
                add s1, a0, zero

                if_swap:
                    beq s1, zero, continue # min_idx == 0 goto continue

                    # Call the swap function:
                    add t0, s1, s0 # t0 = min_idx + i
                    slli t0, t0, 2 # (min_idx + i) * 4
                    add a0, s2, t0 # a0 = &array[min_idx + i]
                    slli t0, s0, 2 # i * 4
                    add a1, s2, t0 # a1 = &array[i]
                    jal zero, swap # Call swap

                continue:

                # Increment i:
                addi s0, s0, 1

                # Jump to Loop:
                jal zero, for

            end_loop:

            # Restore the s registers:
            lw s0, 0(sp)
            lw s1, 4(sp)
            lw s2, 8(sp)
            lw s3, 12(sp)
            lw s4, 16(sp)
            lw ra, 20(sp)

            # Empty the stack:
            addi sp, sp, 24

            # Return:
            jalr zero, ra, 0

    # findMinimum takes a0 (base address) and a1 (number of elements).
    findMinimum:
                # t0 is min_idx
```

```
                    # t1 is min_E

                    # Initialize min_idx and min_E:
                    add t0, zero, zero # min_idx = 0
                    add t2, a0, t0 # (a0 + min_idx)
                    lw t1, 0(t2) # t1 = array[a0 + min_idx * 4]

                    # Loop through the array:

                    # t3 is i
                    addi t3, zero, 1

                    for_loop_2:
                        # Condition:
                        bge t3, a1, end_loop_2 # i >= a0 goto
                           end_loop_2
                        # Body:
                        slli t4, t3, 2
                        add t4, a0, t4 # t4 = &array[i]

                        # t5 = arr[i]
                        lw t5, 0(t4)

                        if_2:
                            bge t5, t1, continue_2  # array[i]
                               >= min_E goto continue_2
                            add t0, t3, zero # min_idx = i
                            slli t6, t0, 2 # min_idx * 4
                            add t6, a0, t6
                            lw t1, 0(t6) # t1 == array[i]

                        continue_2:

                        # Increment i:
                        addi t3, t3, 1

                        # Jump to Loop:
                        jal zero, for_loop_2

                    end_loop_2:
                    # Return min_idx:
                    add a0, t0, zero
                    # Return:
                    jalr zero, ra, 0
```

```
118         # swap takes a0 (address of first element) and a1
               (address of second element).
119         swap:
120                 # t0 is temp
121                 # t1 is temp_2
122
123                 # Interchange the values:
124                 lw t0, 0(a0) # temp = *a0
125                 lw t1, 0(a1) # temp_2 = *a1
126                 sw t1, 0(a0) # *a0 = temp_2
127                 sw t0, 0(a1) # *a1 = temp
128
129                 # Return:
130                 jalr zero, ra, 0
```

b)

Assume that the selectionSort is the function called. Draw the status of the stack before calling selectionSort and during each function call. Indicate stack addresses and names of registers and variables stored on the stack; mark the location of sp; and clearly mark each stack frame. Assume the sp starts at 0x8000.

Before calling selectionSort:

Addresses	*Registers*	*Stack_Pointer*	*Variables*	*Stack_Frame*
0x8000	???	*SP*	???	*

During the first call to selectionSort. The stack remains the same for each iteration in the loop. Hence, for findMinimum and swap, the stack remains the same.

Addresses	*Registers*	*Stack_Pointer*	*Variables*	*Stack_Frame*
0x8000	???		???	
0x7FFC	*ra*		???	*selectionSort*
0x7FF8	*s*4		???	*selectionSort*
0x7FF4	*s*3		???	*selectionSort*
0x7FF0	*s*2		???	*selectionSort*
0x7FEC	*s*1		???	*selectionSort*
0x7FE8	*s*0	*SP*	???	*selectionSort*

After calling selectionSort:

Addresses	*Registers*	*Stack_Pointer*	*Variables*	*Stack_Frame*
0x8000	???	*SP*	???	*