

Assignment 7, Part A

Adrian Lozada

April 11, 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.

```
1 # a0 is the base of the array.
2 # a1 is the number of elements in the array.
3 # a2 is the number that the function searches for.
4
5 # The function returns the index in a0 (if found), otherwise
6   -1.
7
8 find_int:
9     addi t0, x0, 1 # t0 is 1.
10    if:
11        blt a1, t0, return_1 # If size < 1 return -1
12    add t0, a0, x0 # t0 is a pointer to a0.
13    slli t3, a1, 2 # t3 is the size of the array in bytes.
14    add t1, a0, t3 # t1 is a pointer to the last element of
15                     the array.
16
17    loop:
18        blt t1, t0, return_1 # t1 < t0
19        lw t2, 0(t0) # t2 = *t0
20        if_equal:
21            beq t2, a2, return_found # If t2 == a2
22                                     return the index.
23        addi t0, t0, 4 # t0++
24        jal x0, loop
25
26    return_found:
27        sub a0, t0, a0 # Calculate the index.
28        srai a0, a0, 2
29        jalr x0, x1, 0
30
31    return_1:
32        addi a0, x0, -1 # Return -1.
33        jalr x0, x1, 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.** `func1` needs to store the values inside `a0`, `t1`, `s4-s10`, `ra` in

4

Implement the C code snippet in RISC-V assembly language. Use s0-s2 to hold the variables i, j, and min_idx in the function selectionSort. Be sure to handle the stack pointer appropriately. The array is stored in memory starting at address a0.

```

1      # selectionSort takes a0 (base address) and a1 (number
2      # of elements).
3      selectionSort:
4          # s0 is i
5          # s1 is j
6          # s2 is min_idx
7          # s4 is a0
8          # s5 is a1
9          # s6 is a1 - 1 (size)
10
11         # Save the s registers into the stack:
12         addi sp, sp, -28
13         sw s0, 0(sp)
14         sw s1, 4(sp)
15         sw s2, 8(sp)
16         sw s4, 12(sp)
17         sw s5, 16(sp)
18         sw s6, 20(sp)
19         sw x1, 24(sp)
20
21         # Store a0 and a1:
22         add s4, a0, x0
23         add s5, a1, x0
24
25         # Sort the array:
26         addi s6, a1, -1 # n - 1
27         add s0, x0, x0 # i = 0
28         for:
29             bge s0, s6, end_loop # i > n
30
31         # Call findMinimum function:
32         slli t0, s0, 2
33         add a0, s4, t0
34         sub a1, s5, s0
35         jal x1, findMinimum
36
37         # Set min_idx to the value in a0:
38         add s2, a0, x0

```

```

39
40         if_swap:
41             beq s2, s0, continue
42             # Call the swap function:
43             add t0, s2, s0
44             slli t0, t0, 2
45             add a0, s4, t0
46             slli t0, s0, 2
47             add a1, s4, t0
48             jal x1, swap
49
50         continue:
51
52         # Increment i:
53         addi s0, s0, 1
54
55         # Jump to Loop:
56         jal x0, for
57
58     end_loop:
59
60     # Restore the s registers:
61     lw s0, 0(sp)
62     lw s1, 4(sp)
63     lw s2, 8(sp)
64     lw s4, 12(sp)
65     lw s5, 16(sp)
66     lw s6, 20(sp)
67     lw x1, 24(sp)
68
69     # Empty the stack:
70     addi sp, sp, 28
71
72     # Return:
73     jalr x0, x1, 0
74
75
76     # findMinimum takes a0 (base address) and a1 (number
77     # of elements).
78     findMinimum:
79         # t0 is min_idx
80         # t1 is min_E
81
82         # Initialize min_idx and min_E:
83         add t0, x0, x0

```

```

83         add t2, a0, t0
84         lw t1, 0(t2)
85
86         # Loop through the array:
87         # t3 is i
88         addi t3, x0, 1
89         for_loop_2:
90             # Condition:
91             bge t3, a1, end_loop_2
92             # Body:
93             slli t4, t3, 2
94             add t4, a0, t4
95             # t5 = arr[i]
96             lw t5, 0(t4)
97             if_2:
98                 bge t5, t1, continue_2
99                 add t0, t3, x0
100                 slli t6, t0, 2
101                 add t6, a0, t6
102                 lw t1, 0(t6)
103
104             continue_2:
105
106             # Increment i:
107             addi t3, t3, 1
108
109             # Jump to Loop:
110             jal x0, for_loop_2
111
112         end_loop_2:
113         # Return min_idx:
114         add a0, t0, x0
115         # Return:
116         jalr x0, x1, 0
117
118     # swap takes a0 (address of first element) and a1
119     # (address of second element).
120     swap:
121         # t0 is temp
122         # t1 is a0
123
124         # Interchange the values:
125         lw t0, 0(a0) # temp = a0
126         lw t1, 0(a1) # a0 = a1
127         sw t1, 0(a0)

```

```
127         sw t0, 0(a1)
128
129     # Return:
130     jalr x0, x1, 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.

<i>Addresses</i>	<i>Registers</i>	<i>Variables</i>	<i>Stack_Pointer</i>
0x8000	<i>s0</i>	<i>i</i>	
0x7FFC	<i>s1</i>	<i>j</i>	
0x7FF8	<i>s2</i>	<i>min_idx</i>	
0x7FF0	<i>s4</i>	<i>arr</i>	
0x7FEC	<i>s5</i>	<i>n</i>	
0x7FE8	<i>s6</i>	<i>n - 1</i>	
0x7FE0	<i>x1</i>	<i>ra</i>	<i>sp</i>