

Assignment 7, Part A

Adrian Lozada

April 14, 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 the stack.

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 of
  elements).
2     selectionSort:
3         # s0 is i
4         # s1 is min_idx
5         # s2 is a0 or arr[]
6         # s3 is a1 or n
7         # s4 is n - 1
8
9         # Save the s and ra registers into the
         stack:
10        addi sp, sp, -24
11        sw s0, 0(sp)
12        sw s1, 4(sp)
13        sw s2, 8(sp)
14        sw s3, 12(sp)
15        sw s4, 16(sp)
16        sw x1, 20(sp)
17
18        # Store a0 and a1:
19        add s2, a0, x0
20        add s3, a1, x0
21
22        # Sort the array:
23        addi s4, a1, -1 # pass n - 1
24        add s0, x0, x0 # i = 0
25        for:
26            bge s0, s4, end_loop # i >= n - 1
                goto end_loop
27
28            # Call findMinimum function:
29            slli t0, s0, 2 # i * 4
30            add a0, s2, t0 # pass &array[i] to
                a0
31            sub a1, s3, s0 # pass n - i
32            jal x1, findMinimum
33
34            # Set min_idx to the value in a0:
35            add s1, a0, x0

```

```

36
37         if_swap:
38             beq s1, x0, continue # min_idx
                                   == 0 goto continue
39
40             # Call the swap function:
41             add t0, s1, s0 # t0 = min_idx
                                   + i
42             slli t0, t0, 2 # (min_idx + i)
                                   * 4
43             add a0, s2, t0 # a0 =
                                   &array[min_idx + i]
44             slli t0, s0, 2 # i * 4
45             add a1, s2, t0 # a1 = &array[i]
46             jal x1, swap # Call swap
47
48         continue:
49
50         # Increment i:
51         addi s0, s0, 1
52
53         # Jump to Loop:
54         jal x0, for
55
56     end_loop:
57
58     # Restore the s registers:
59     lw s0, 0(sp)
60     lw s1, 4(sp)
61     lw s2, 8(sp)
62     lw s3, 12(sp)
63     lw s4, 16(sp)
64     lw x1, 20(sp)
65
66     # Empty the stack:
67     addi sp, sp, 24
68
69     # Return:
70     jalr x0, x1, 0
71
72     # findMinimum takes a0 (base address) and a1 (number
       of elements).
73     findMinimum:
74         # t0 is min_idx
75         # t1 is min_E

```

```

76
77     # Initialize min_idx and min_E:
78     add t0, x0, x0 # min_idx = 0
79     slli t3, t0, 2 # min_idx * 4
80     add t2, a0, t3 # (a0 + min_idx * 4)
81     lw t1, 0(t2) # t1 = array[a0 + min_idx * 4]
82
83     # Loop through the array:
84
85     # t3 is i
86     addi t3, x0, 1
87
88     for_loop_2:
89         # Condition:
90         bge t3, a1, end_loop_2 # i >= a0 goto
91             end_loop_2
92         # Body:
93         slli t4, t3, 2
94         add t4, a0, t4 # t4 = &array[i]
95
96         # t5 = arr[i]
97         lw t5, 0(t4)
98
99         if_2:
100             bge t5, t1, continue_2 # array[i]
101                 >= min_E goto continue_2
102             add t0, t3, x0 # min_idx = i
103             slli t6, t0, 2 # min_idx * 4
104             add t6, a0, t6
105             lw t1, 0(t6) # t1 == array[i]
106
107         continue_2:
108
109         # Increment i:
110         addi t3, t3, 1
111
112         # Jump to Loop:
113         jal x0, for_loop_2
114
115     end_loop_2:
116     # Return min_idx:
117     add a0, t0, x0
118     # Return:
119     jalr x0, x1, 0

```

```

119      # swap takes a0 (address of first element) and a1
      (address of second element).
120      swap:
121          # t0 is temp
122          # t1 is temp_2
123
124          # Interchange the values:
125          lw t0, 0(a0) # temp = *a0
126          lw t1, 0(a1) # temp_2 = *a1
127          sw t1, 0(a0) # *a0 = temp_2
128          sw t0, 0(a1) # *a1 = temp
129
130          # Return:
131          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	s0	<i>i</i>	
0x7FFC	s1	<i>min_idx</i>	
0x7FF8	s2	<i>arr</i>	
0x7FF4	s3	<i>n</i>	
0x7FF0	s4	<i>n - 1</i>	
0x7FEC	x1	<i>ra</i>	<i>sp</i>