Assignment 7, Part A

Adrian Lozada

April 16, 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, x0, 1 # t0 is 1.
        if:
            blt a1, t0, return_1 # If size < 1 return -1
        add t0, a0, x0 # 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 x0, loop

        return_found:
                    sub a0, t0, a0 # Calculate the index.
                    srai a0, a0, 2
                    jalr x0, x1, 0
        return_1:
                addi a0, x0, -1 # Return -1.
                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. 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-s2 to hold the variables i, j, and min$_idx in the fucntion selectionSort. Be sure to hangle the stack pointer appropriatly. The array is store
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 x1, 20(sp)

                        # Store a0 and a1:
                        add s2, a0, x0
                        add s3, a1, x0

                        # Sort the array:
                        addi s4, a1, -1 # pass n - 1
                        add s0, x0, x0 # 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 x1, findMinimum

                            # Set min_idx to the value in a0:
                            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
                               end_loop_2
91                          # Body:
92                          slli t4, t3, 2
93                          add t4, a0, t4 # t4 = &array[i]
94
95                          # t5 = arr[i]
96                          lw t5, 0(t4)
97
98                          if_2:
99                              bge t5, t1, continue_2  # array[i]
                                   >= min_E goto continue_2
100                             add t0, t3, x0 # min_idx = i
101                             slli t6, t0, 2 # min_idx * 4
102                             add t6, a0, t6
103                             lw t1, 0(t6) # t1 == array[i]
104
105                         continue_2:
106
107                         # Increment i:
108                         addi t3, t3, 1
109
110                         # Jump to Loop:
111                         jal x0, for_loop_2
112
113                     end_loop_2:
114                     # Return min_idx:
115                     add a0, t0, x0
116                     # Return:
117                     jalr x0, x1, 0
118
```

```
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.

Addresses	*Registers*	*Stack_Pointer*
0x8000	$s0$	
0x7FFC	$s1$	
0x7FF8	$s2$	
0x7FF4	$s3$	
0x7FF0	$s4$	
0x7FEC	$x1$	sp