Lab 2

Instructions

- Complete each task and demonstrate the working program to your tutor. Tasks should be demonstrated using AVR Studio's simulator. This lab must be marked by the end of the lab in week 4.
- Read the lab sheet carefully before you come to the week 3 lab. Utilise this slot to interact with tutors to solve your doubts.
- Any other doubts can be posted in the course forum indicating your lab time so that the corresponding tutors can answer those.
- Reserve the slot on week 4 exclusively for demonstration. Please have all the exercises completed when you come to this lab.

Hints

- Be organised and neat – it will heavily reduce the debugging time
- Put comments
- Have a printed sheet of the AVR instruction set (page 1,10-15 of http://www.cse.unsw.edu.au/~cs2121/AVR/AVR-Instruction-Set.pdf)

Part A – Reverse String (3 Marks)

Implement a program that loads a null-terminated string from program memory and pushes it onto the stack, then writes out the reversed string into data memory. The stack pointer must be initialized before use.

Eg: "abc",0 will be stored in RAM as "cba",0

Part B – Function calls (3 marks)

Make *in-place array sorting* you implemented in part D of lab 1 into a function named *insert_request*:

- address of the sorted array (array in the **data memory**), the number of elements in the array and the next value to be inserted are the arguments to the function.
- the return value is the number of elements after the insertion.

Make sure you save the conflict registers correctly. Now outside the function:

- load the sorted array (1, 2, 5, 7, 8, 12, 20) from program memory to data memory
- iterate through the queue of numbers (0, 1, 10, 25, 6) in the program memory while calling *insert_request* for each element.

Part C – Basic lift controller (4 marks)

Extend *insert_request* function above to a basic lift controller (simulation based on data structures and algorithms) that can service the requests for the lift. Lift should be like in a real-life scenario (see the prologue in lab 1). Note that the lift does not move yet, and you should only simulate the ordering of the subsequent service requests.

The sorted array now represents the remaining requests which the lift should service. The next value to be inserted represents the next request call entered by a user. Now, the sorting of the array should be performed based on the current floor and the travel direction of the lift - which you can assume to be arguments. Two examples are given below:

- The lift is currently on floor 3 and it is moving upwards. It is supposed to service the requests for floors (5,7,8,1) respectively – this is the "sorted" array now. A user on floor 10 requests for the lift. When this request is inserted the "sorted" array looks like (5,7,8,10,1). Then another request comes, this time from floor 6 and the array looks like (5,6,7,8,10,1), The next request is from floor 7, but it should not be inserted as 7 is already in the array.
- The lift is on floor 8 and moving down. The list of subsequent floors to visit are (5,3,2,10,11,15). If the button for floor 13 is pressed, it will be inserted into the list such that the new list is (5,3,2,10,11,13,15). If the button for floor 4 is then pressed, the new list will be (5,4,3,2,10,11,13,15).

Demonstrate the simulation by changing the "sorted" array and the queue of subsequent requests in the program memory.