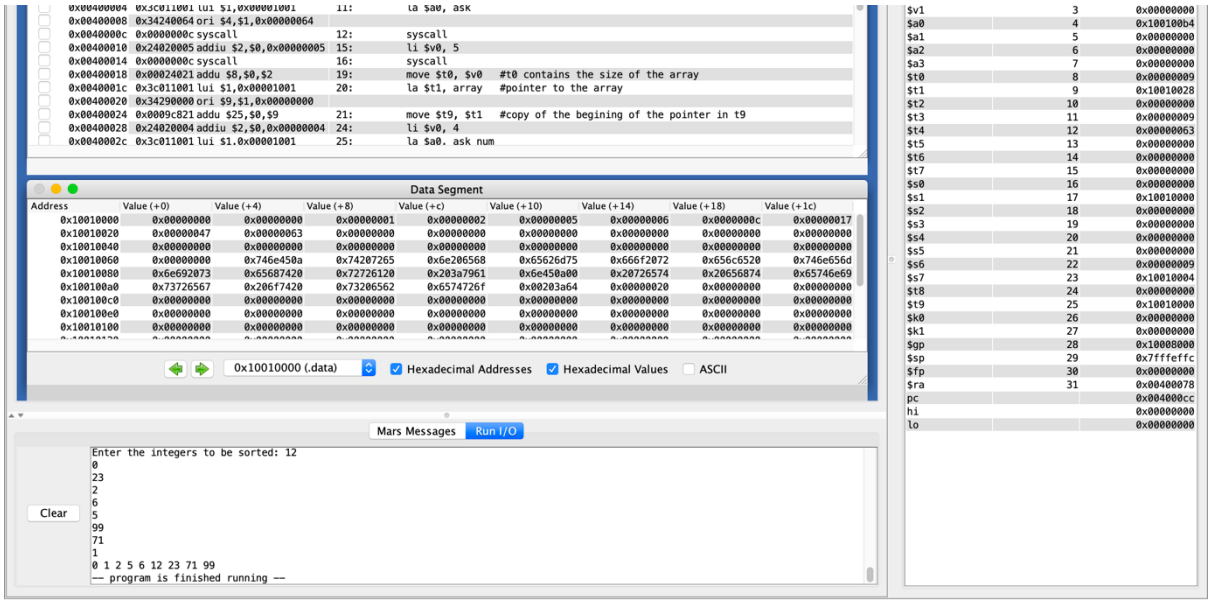
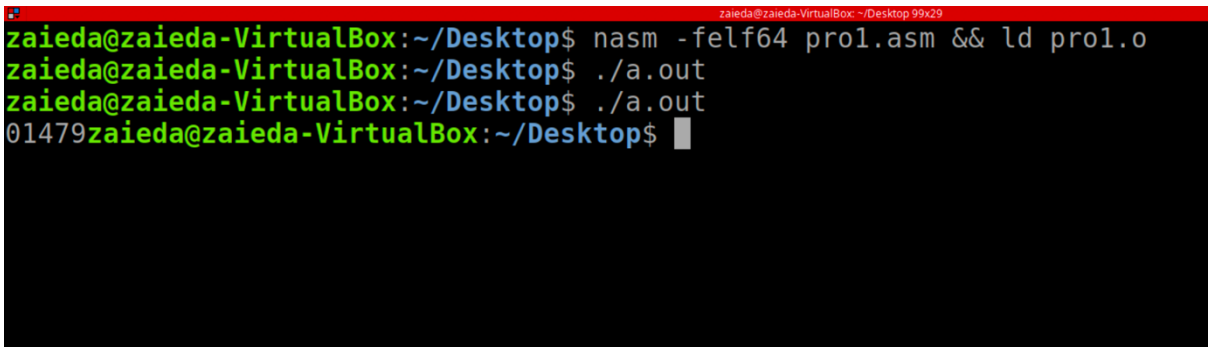
Problem 1:
In this problem we were asked to do a bubble sort on an array of integers. First, the program asks the user to for the size of the array and the items to be stored in the array in a loop. My design approach was to get a pointer at the end of the array and decrement it whenever the following loop is done once: comparing the first to elements and swapping if needed, then checking the next elements and so on until hitting the pointer of the last element which will be decremented every loop. For repeated items they stay the same and they are printed twice. The parameters were pushed on the stack to be received in the function itself.

In x86, I did the array static just to examine the logic in x86. I assumed that the user will not enter any numbers or anything just an output for it to change the program has to change. The parameters are sent to a label that pops the items from the stack to be used. The flow is generally the same as mips but the syntax is different. I do decrement the comparison loop every time I finish the loop, and all of that happens under 3 labels. Moreover, the printing of the array was clumsy at the end because there is no spacing however I made it such that it is only one digit, so there is a separation every single digit.

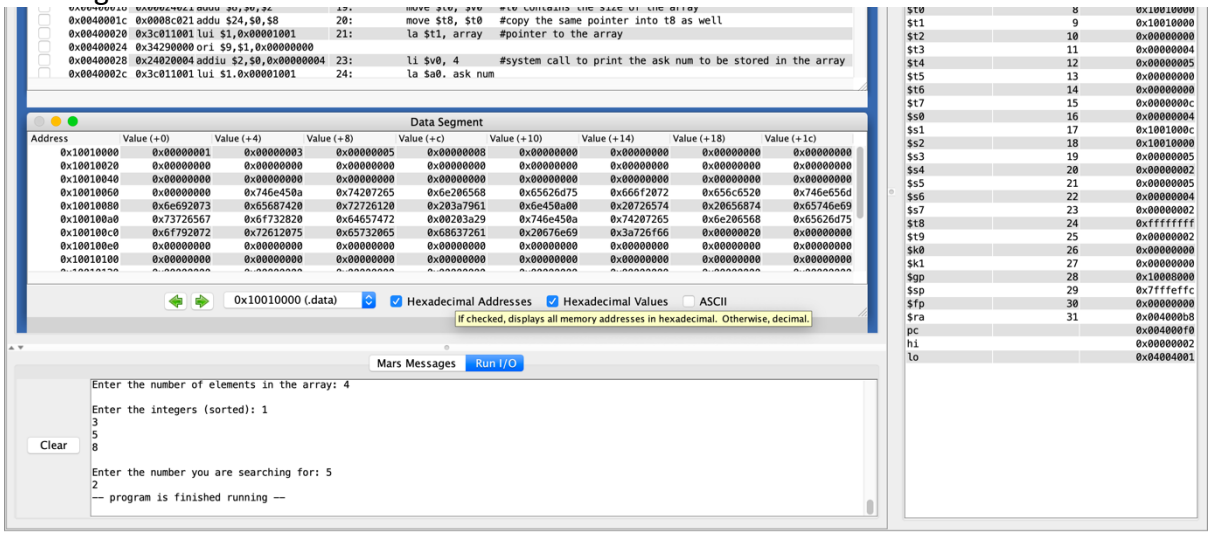


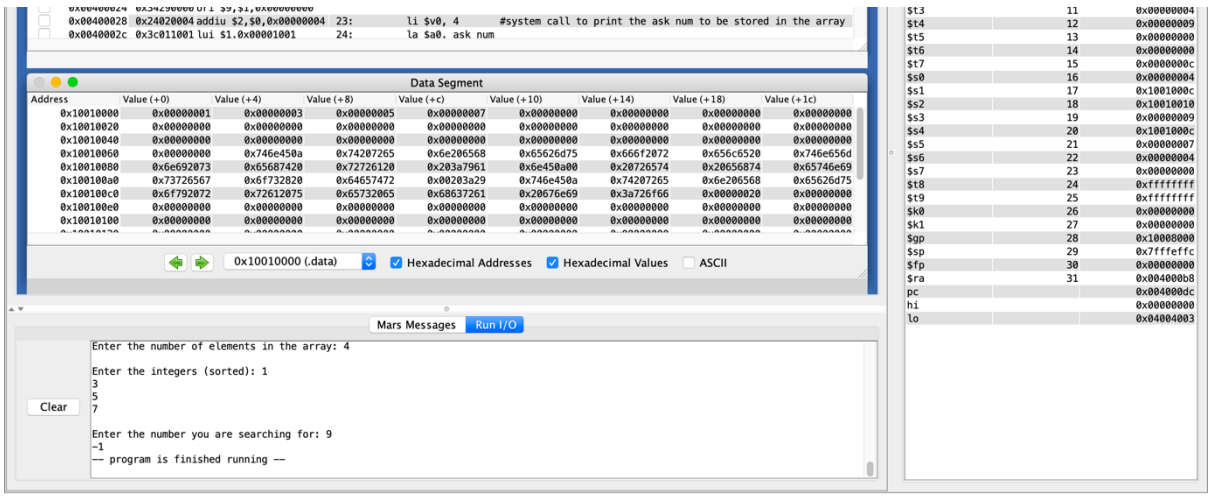


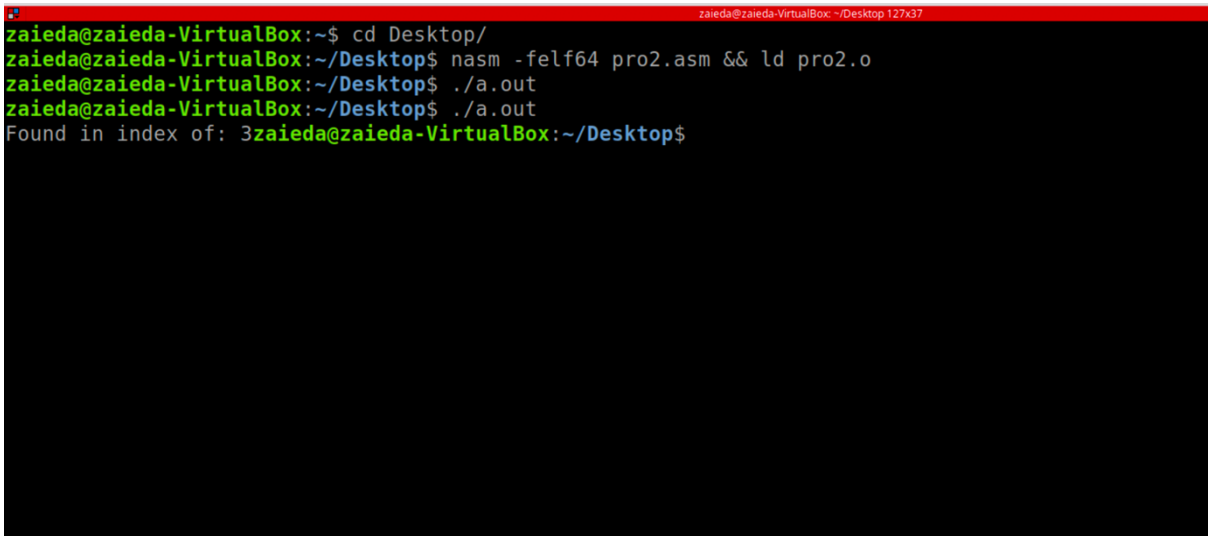
Problem 2:

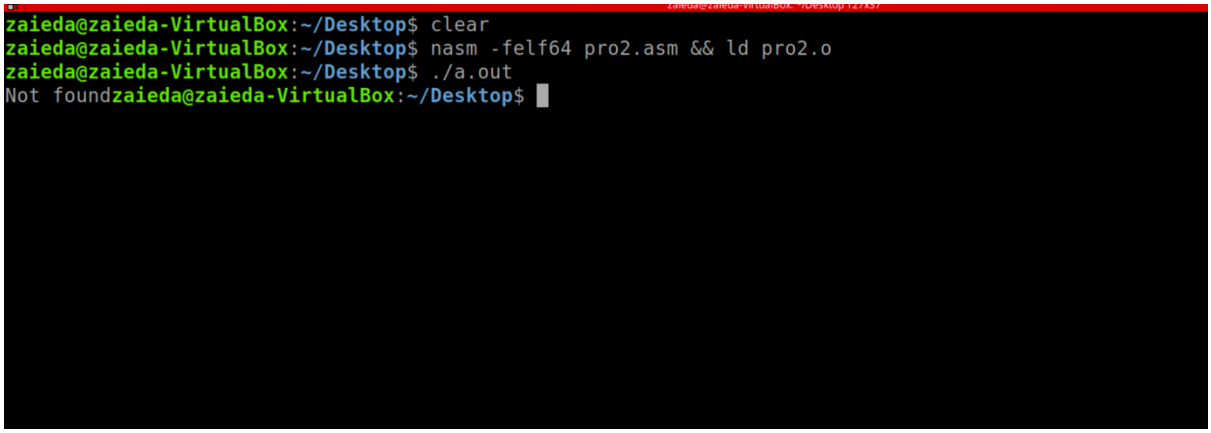
In this problem I have implemented the recursion on one stack frame in which I only update to the items in this stack frame and use it for the following call to reduce memory use. I have passed the return address of the first jal along with the right and left pointer, the size of the array and the element to be searching for. I started with the middle element in the array with some pointer fixation when dividing the end and start pointer by two. Then comparing the item with this middle element if it is equal (stopping case) then exit the loop. Check if less than or bigger than then update the stack frame and call the function again with these pointers updated. Until finding the element. If found I get the address and subtract it with the address of the first element and mod it by 4 to get its location.

In x86 I did the same logic of recursion where I only update the same stack frame such that I reduce the memory usage. I used call and retq to be able to manipulate the ra idea in mips. I made the most validations I can do with the current knowledge, assuming that if the user wants to change something, they will update the .asm file. I always update the stack using push and pop, and whilst Im doing that I call the recursion. Moreover the labels were done normally using jmp and other conditional instructions in x86_64, assuming that we are working with 64 bit.






zaieda@zaieda-VirtualBox: ~/Desktop 127x37
zaieda@zaieda-VirtualBox:~$ cd Desktop/
zaieda@zaieda-VirtualBox:~/Desktop$ nasm -felf64 pro2.asm && ld pro2.o
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out
Found in index of: 3zaieda@zaieda-VirtualBox:~/Desktop$


zaieda@zaieda-VirtualBox:~/Desktop$ clear
zaieda@zaieda-VirtualBox:~/Desktop$ nasm -felf64 pro2.asm && ld pro2.o
zaieda@zaieda-VirtualBox:~/Desktop$ ./a.out
Not foundzaieda@zaieda-VirtualBox:~/Desktop$