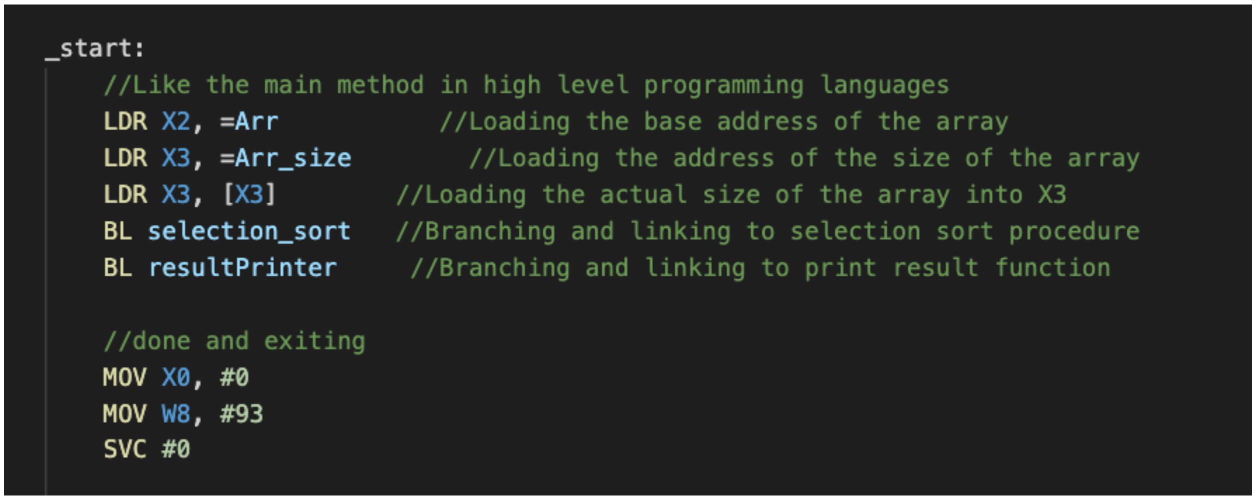
Burak Yesil
I pledge my honor that I have abided by the Stevens Honor System.

Step 1:

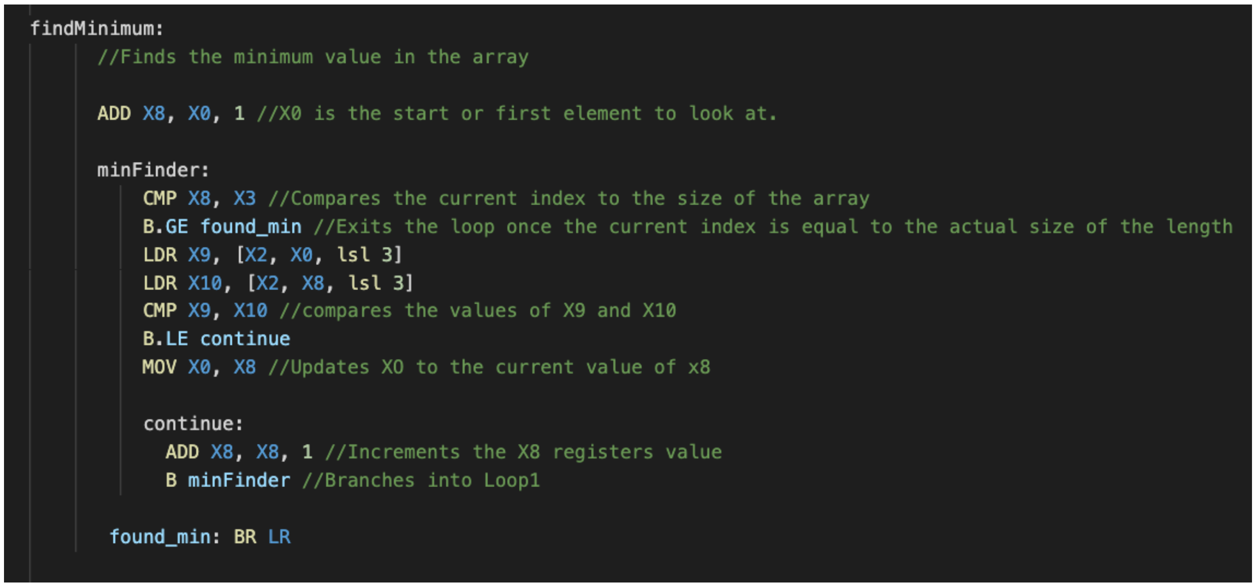


```
_start:
    //Like the main method in high level programming languages
    LDR X2, =Arr            //Loading the base address of the array
    LDR X3, =Arr_size         //Loading the address of the size of the array
    LDR X3, [X3]        //Loading the actual size of the array into X3
    BL selection_sort   //Branching and linking to selection sort procedure
    BL resultPrinter     //Branching and linking to print result function

    //done and exiting
    MOV X0, #0
    MOV W8, #93
    SVC #0
```

I started off by setting up the _start procedure. In this procedure, I started off by loading in the data variables from the data section. I then called the selection_sort procedure to sort the array whose base address is in register X2. Once the selection_sort procedure is done the _start procedure then go on to call the print_result procedure to print the new sorted array.

Step 2:

Next, I started off by implementing the findMinimum procedure. I started off with the findMinimum procedure instead of the selection_sort procedure because I wanted to make sure the foundational procedures worked before trying to solve the main problem.



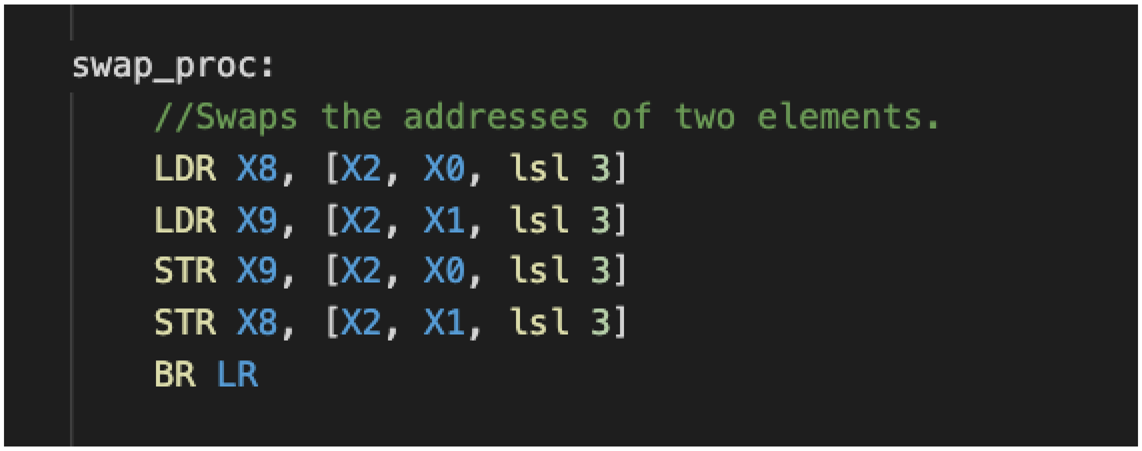
```
findMinimum:
      //Finds the minimum value in the array

      ADD X8, X0, 1 //X0 is the start or first element to look at.

      minFinder:
          CMP X8, X3 //Compares the current index to the size of the array
          B.GE found_min //Exits the loop once the current index is equal to the actual size of the length
          LDR X9, [X2, X0, lsl 3]
          LDR X10, [X2, X8, lsl 3]
          CMP X9, X10 //compares the values of X9 and X10
          B.LE continue
          MOV X0, X8 //Updates XO to the current value of x8

          continue:
            ADD X8, X8, 1 //Increments the X8 registers value
            B minFinder //Branches into Loop1

       found_min: BR LR
```

My approach to implementing the findMinimum procedure was to iterate through the list while comparing the current minimum with every element in the unsorted partition of the array, updating the current min if a new minimum is found.
I would also like to mention that the selection sort function calls the findMinimum procedure. Moreover, each time it calls the procedure, it splits the list into two partitions, the left half being the sorted partition and the right half being the unsorted partition. Also, through each iteration, the right partition shrinks and the left half of the array increases until the whole array is sorted.

Step 3:

Moving on, I worked on the swap procedure as the selection_sort procedure would need to call this function in order to update the array and move the minimum element of the unsorted partition to the left sorted partition of the array.

For example, If the array was [1,2 | 4,5,3,7], the swap function would change the array to become [1,2,3 | 5,4,7].

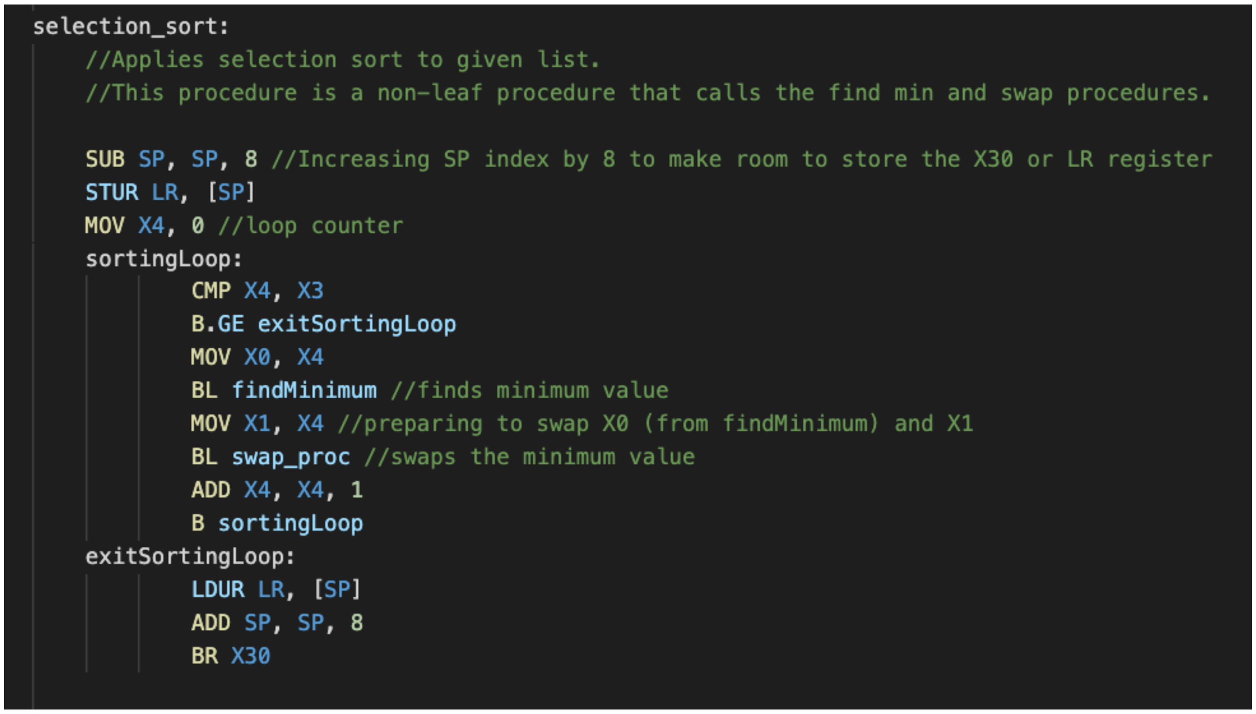


```
swap_proc:
    //Swaps the addresses of two elements.
    LDR X8, [X2, X0, lsl 3]
    LDR X9, [X2, X1, lsl 3]
    STR X9, [X2, X0, lsl 3]
    STR X8, [X2, X1, lsl 3]
    BR LR
```

All that is needed for this procedure is to load the addresses of the two elements into separate registers and then to store them in each other's original memory location, completing the swap.

Step 4:

Now that both the swap_proc and findMinimum procedure had been implemented, it was time to actually implement the selection sort algorithm.



```
selection_sort:
    //Applies selection sort to given list.
    //This procedure is a non-leaf procedure that calls the find min and swap procedures.

    SUB SP, SP, 8 //Increasing SP index by 8 to make room to store the X30 or LR register
    STUR LR, [SP]
    MOV X4, 0 //loop counter
    sortingLoop:
            CMP X4, X3
            B.GE exitSortingLoop
            MOV X0, X4
            BL findMinimum //finds minimum value
            MOV X1, X4 //preparing to swap X0 (from findMinimum) and X1
            BL swap_proc //swaps the minimum value
            ADD X4, X4, 1
            B sortingLoop
    exitSortingLoop:
            LDUR LR, [SP]
            ADD SP, SP, 8
            BR X30
```

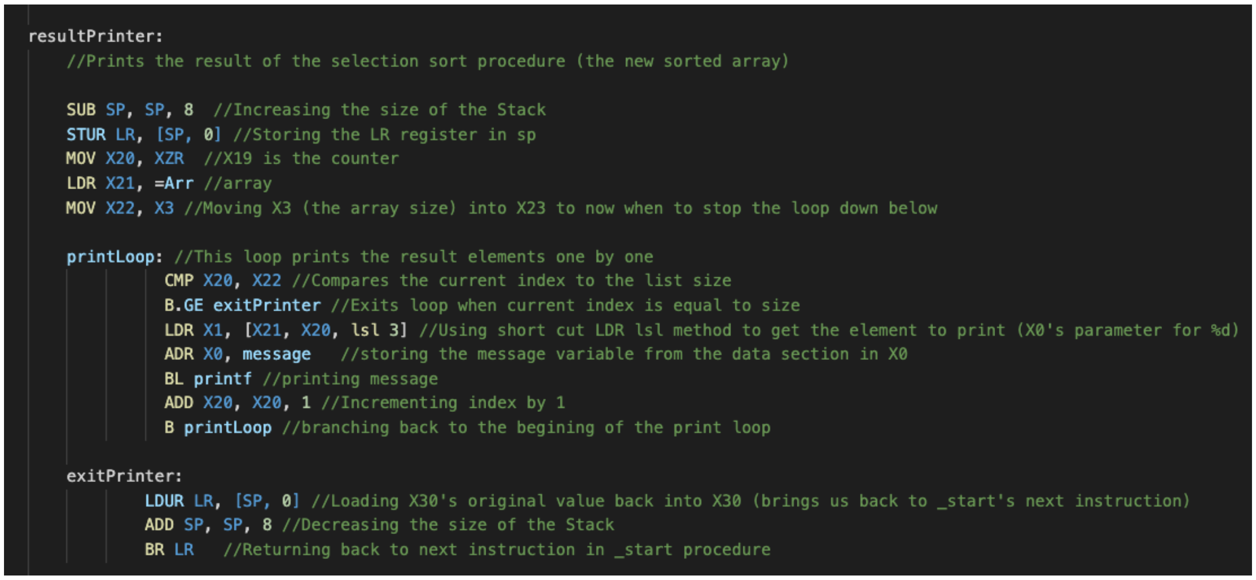
This was easy considering I finished the underlying procedures. I pretty much programmed a loop called sortingLoop that iterates until the counter, which starts at zero, is equal to the length of the array, in which it exits the loops since the array will be sorted.

Within each iteration, the loop calls findMinimum and swap_proc to update the array, while also incrementing the start index for the next iteration. As mentioned before, the incrementing is done to set up a sorted and unsorted partition in the array.

It is also important to mention that since this program requires non-leaf procedures, I had to store and load the X30 register in and from the stack using the stack pointer at different parts of the whole program.

Step 5:

Next I programmed the actual resultPrinter procedure for this procedure.

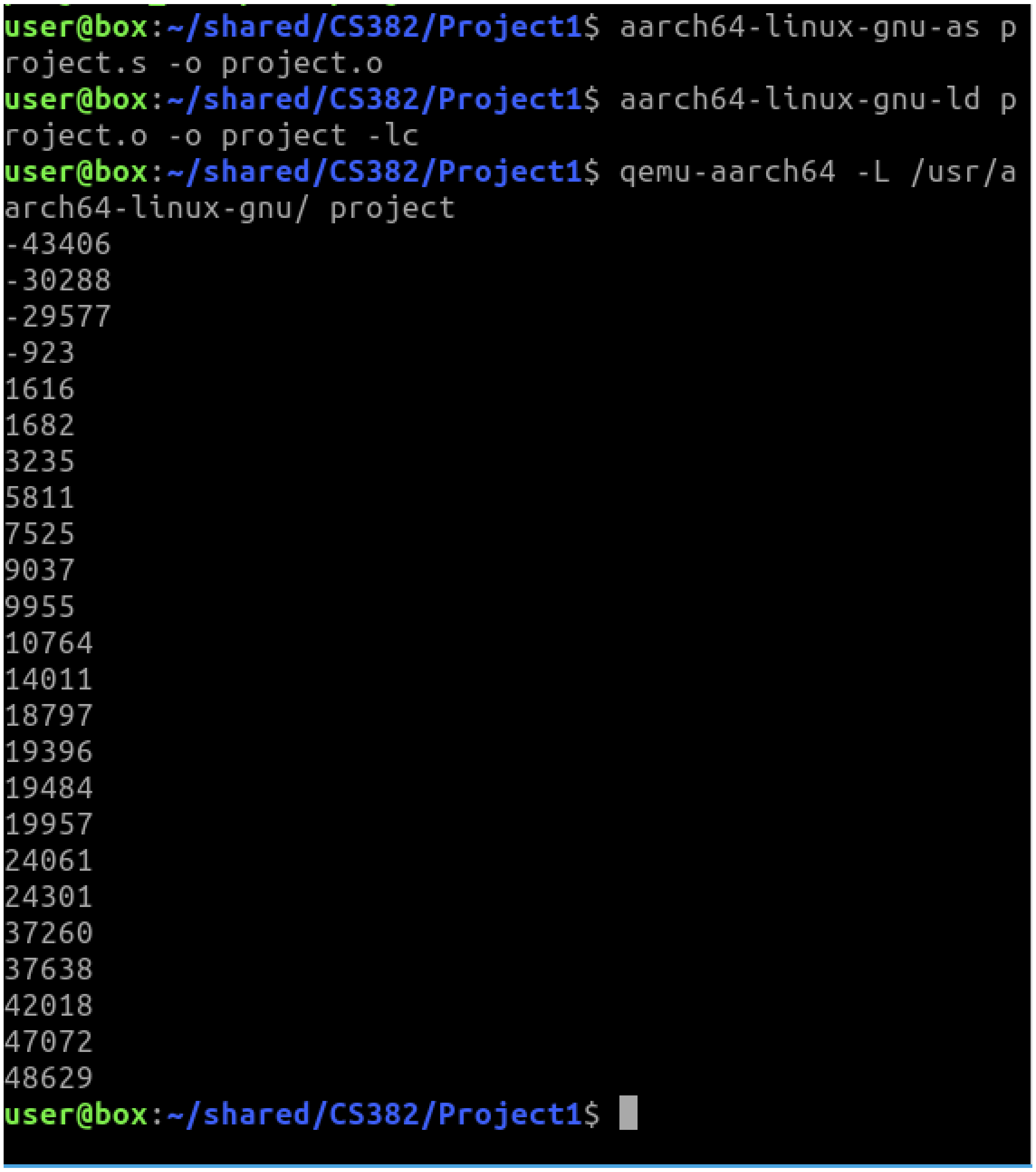


```
resultPrinter:
    //Prints the result of the selection sort procedure (the new sorted array)

    SUB SP, SP, 8  //Increasing the size of the Stack
    STUR LR, [SP, 0] //Storing the LR register in sp
    MOV X20, XZR  //X19 is the counter
    LDR X21, =Arr //array
    MOV X22, X3 //Moving X3 (the array size) into X23 to now when to stop the loop down below

    printLoop: //This loop prints the result elements one by one
              CMP X20, X22 //Compares the current index to the list size
              B.GE exitPrinter //Exits loop when current index is equal to size
              LDR X1, [X21, X20, lsl 3] //Using short cut LDR lsl method to get the element to print (X0's parameter for %d)
              ADR X0, message   //storing the message variable from the data section in X0
              BL printf //printing message
              ADD X20, X20, 1 //Incrementing index by 1
              B printLoop //branching back to the begining of the print loop

    exitPrinter:
            LDUR LR, [SP, 0] //Loading X30's original value back into X30 (brings us back to _start's next instruction)
            ADD SP, SP, 8 //Decreasing the size of the Stack
            BR LR   //Returning back to next instruction in _start procedure
```

For this procedure I used a loop called printLoop that prints each element individually. It was pretty straightforward. However, there was an issue at my first attempt when using the parameters X2, X7. Those values were constantly changing, preventing my function from properly working. Therefore, I had to use the Callee-Saved registers.

Step 6:

Finally to test my program, I used a random unordered set generator to create a set of 24 elements. I then used those numbers as my test array. After running my program, I got the following result which showed that my selection sort program works. I also wanted to make sure it works with a 100 number array and it worked. (Dont be thrown off by me using project.s instead of selection.s. I originally had my selection.s file named project.s.



```
user@box:~/shared/CS382/Project1$ aarch64-linux-gnu-as p
roject.s -o project.o
user@box:~/shared/CS382/Project1$ aarch64-linux-gnu-ld p
roject.o -o project -lc
user@box:~/shared/CS382/Project1$ qemu-aarch64 -L /usr/a
arch64-linux-gnu/ project
-43406
-30288
-29577
-923
1616
1682
3235
5811
7525
9037
9955
10764
14011
18797
19396
19484
19957
24061
24301
37260
37638
42018
47072
48629
user@box:~/shared/CS382/Project1$
```