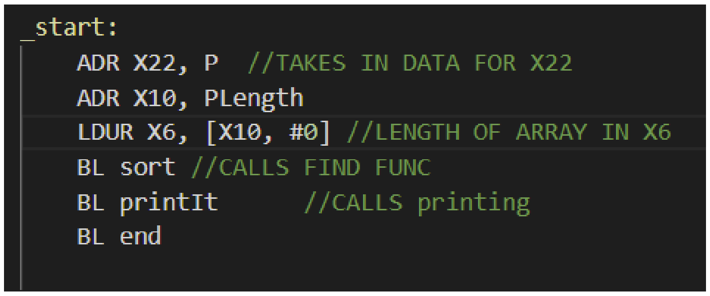
I pledge my honor that I have abided by the Stevens Honor System. - Adam Woo
CS 382 Project 1 Report   10/31/2021

My program outputs the sorted array in the first n+1 printed elements. However, it continues to output random values afterwards. You can see the sorted array in the beginning, but I was unable to stop it from printing.

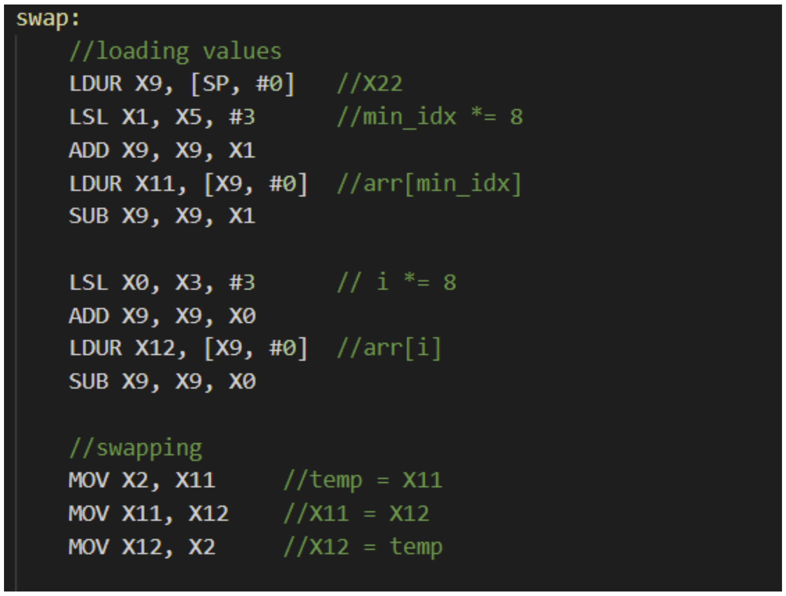
I used the following algorithm to complete the sorting:

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++) {
          if (arr[j] < arr[min_idx]) {
            min_idx = j;
          }
        }
        swap(&arr[min_idx], &arr[i]);
    }
}
```



```
_start:
    ADR X22, P  //TAKES IN DATA FOR X22
    ADR X10, PLength
    LDUR X6, [X10, #0] //LENGTH OF ARRAY IN X6
    BL sort //CALLS FIND FUNC
    BL printIt      //CALLS printing
    BL end
```

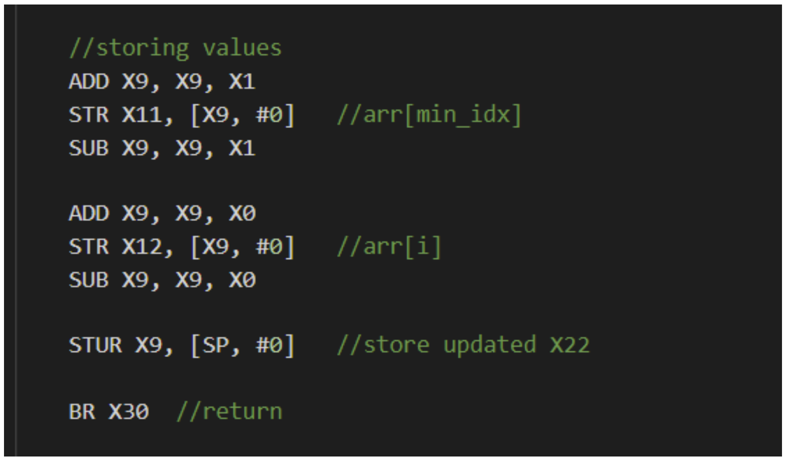
This is the start function that takes in the values from the .data area. It then calls Sort, then Print, then end.



```
swap:
    //loading values
    LDUR X9, [SP, #0]   //X22
    LSL X1, X5, #3      //min_idx *= 8
    ADD X9, X9, X1
    LDUR X11, [X9, #0]  //arr[min_idx]
    SUB X9, X9, X1

    LSL X0, X3, #3      // i *= 8
    ADD X9, X9, X0
    LDUR X12, [X9, #0]  //arr[i]
    SUB X9, X9, X0

    //swapping
    MOV X2, X11     //temp = X11
    MOV X11, X12    //X11 = X12
    MOV X12, X2     //X12 = temp
```

This is the swap function.
This first part initializes and loads the values of array[min_idx] and array[i], so they can later be swapped. X11 stores array[min_idx] and X12 stores array[i].

Here, the two get swapped using a temporary register X2.

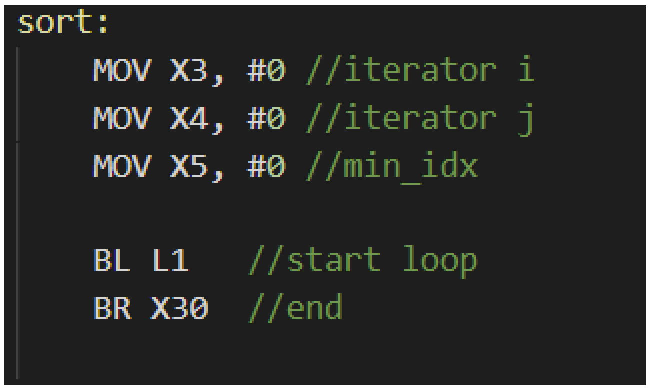


```
//storing values
ADD X9, X9, X1
STR X11, [X9, #0]   //arr[min_idx]
SUB X9, X9, X1

ADD X9, X9, X0
STR X12, [X9, #0]   //arr[i]
SUB X9, X9, X0

STUR X9, [SP, #0]   //store updated X22

BR X30  //return
```

Swap function contd.

The swapped values are then placed back into the array. Using STR.

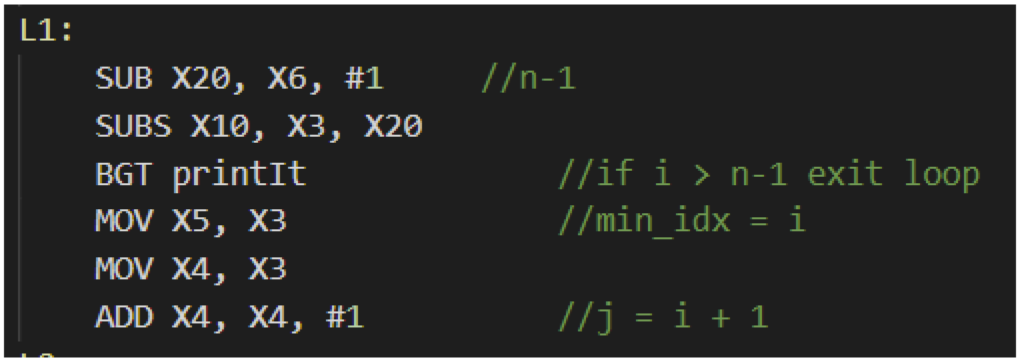
Here, the new array is sent back to the Stack Pointer and returned.



```
sort:
    MOV X3, #0 //iterator i
    MOV X4, #0 //iterator j
    MOV X5, #0 //min_idx

    BL L1   //start loop
    BR X30  //end
```

This is the main sorting function. The variables for the loops and min are initialized here.

It starts with calling the outer loop.

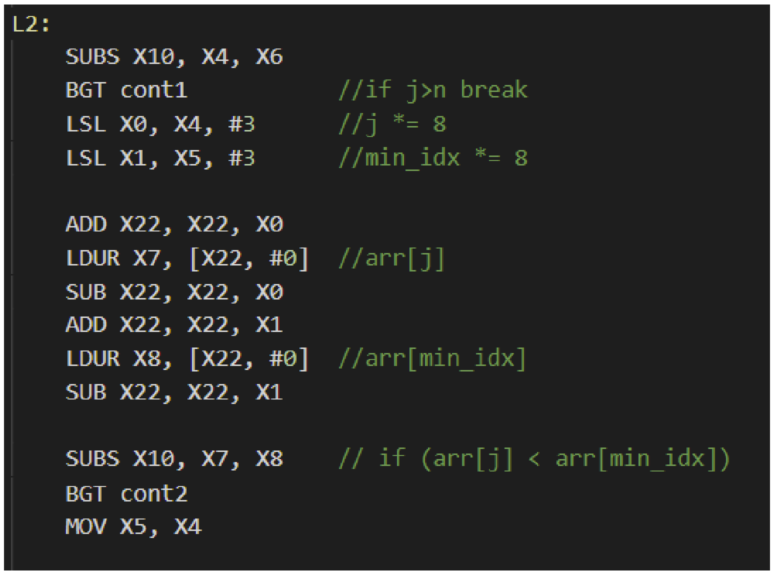


```
L1:
    SUB X20, X6, #1     //n-1
    SUBS X10, X3, X20
    BGT printIt             //if i > n-1 exit loop
    MOV X5, X3              //min_idx = i
    MOV X4, X3
    ADD X4, X4, #1          //j = i + 1
```

Here, L1 is the outer loop. It will iterate from 0 to n-1.

It also sets up j for the inner loop.

After its done running, it goes into the inner loop L2.



```
L2:
    SUBS X10, X4, X6
    BGT cont1           //if j>n break
    LSL X0, X4, #3      //j *= 8
    LSL X1, X5, #3      //min_idx *= 8

    ADD X22, X22, X0
    LDUR X7, [X22, #0]  //arr[j]
    SUB X22, X22, X0
    ADD X22, X22, X1
    LDUR X8, [X22, #0]  //arr[min_idx]
    SUB X22, X22, X1

    SUBS X10, X7, X8    // if (arr[j] < arr[min_idx])
    BGT cont2
    MOV X5, X4
```

This is the inner loop where most of the work is done.

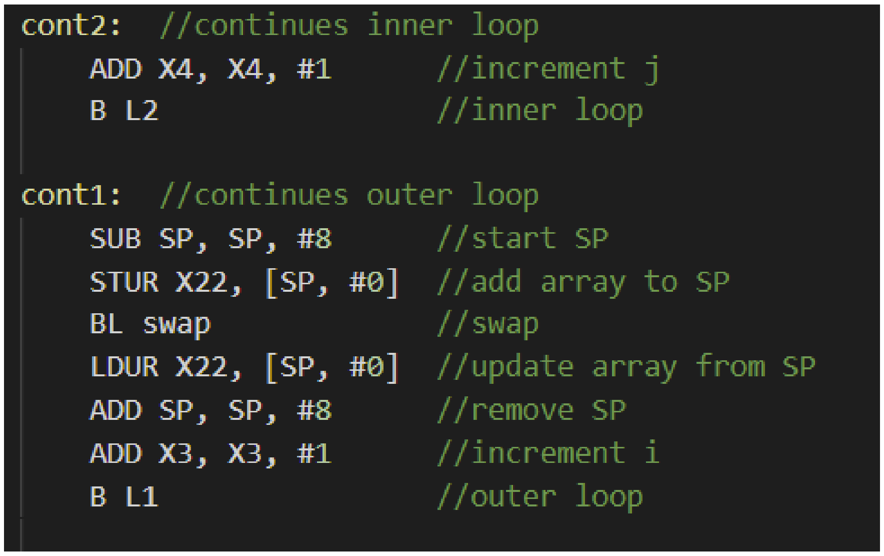
Here, we set up the indexes by multiplying by 8.

Here, we load the values from the array into new registers.

We then compare the registers to find the lowest.

If j > n, inner loop will end and outer loop will continue.

If arr[j] < arr[min_idx], inner loop will change min_idx to j. Otherwise if > it will go to cont2, skipping the change to min_idx.



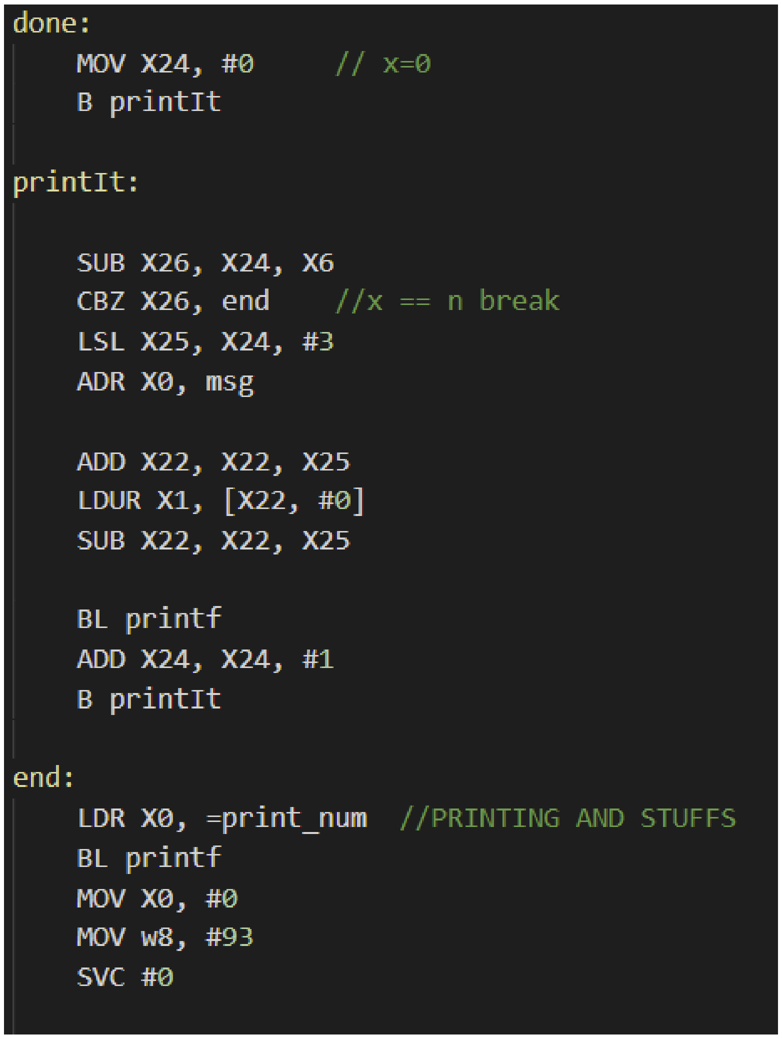
```
cont2:  //continues inner loop
    ADD X4, X4, #1      //increment j
    B L2                //inner loop

cont1:  //continues outer loop
    SUB SP, SP, #8      //start SP
    STUR X22, [SP, #0]  //add array to SP
    BL swap             //swap
    LDUR X22, [SP, #0]  //update array from SP
    ADD SP, SP, #8      //remove SP
    ADD X3, X3, #1      //increment i
    B L1                //outer loop
```

These functions are used to continue the loops and make sure they actually loop. Cont2 will continue the inner loop and increment i. This is called after the comparison between elements in the array.

Cont1 will be performed after the inner loop is done running. This is when we swap. To swap, we need to setup the Stack Pointer. AFter setting it up, we call the swap.

After the swap, we update our main array X22. and we also remove the Stack Pointer. We then loop back to the outer loop.

The outer loop will branch to done, once i > n-1



```
done:
    MOV X24, #0     // x=0
    B printIt

printIt:

    SUB X26, X24, X6
    CBZ X26, end    //x == n break
    LSL X25, X24, #3
    ADR X0, msg

    ADD X22, X22, X25
    LDUR X1, [X22, #0]
    SUB X22, X22, X25

    BL printf
    ADD X24, X24, #1
    B printIt

end:
    LDR X0, =print_num  //PRINTING AND STUFFS
    BL printf
    MOV X0, #0
    MOV w8, #93
    SVC #0
```

Done sets up x for the printing loop.

Here we do all of the printing and stuff.

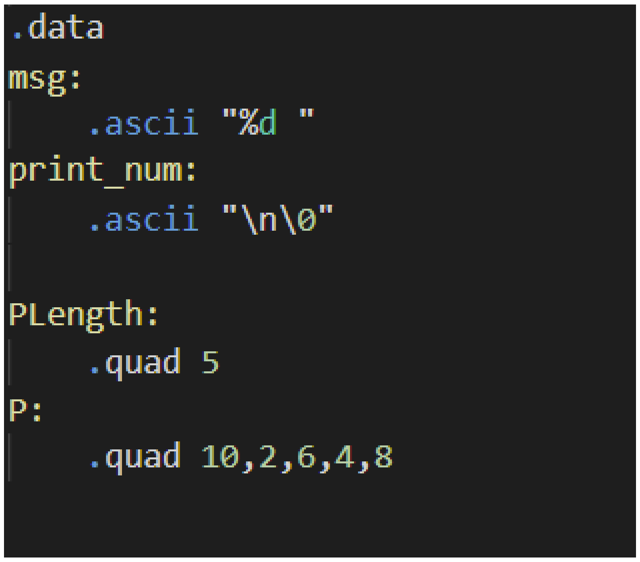
I'm pretty sure this is where my code is breaking.

We want to loop until we've gone through the entire array, so we compare the iterator (X24) with the length of the array (X6). Once the iterator and length are equal we break from the loop.

Then we go through with printing out the current value of array[x24].

We loop back here.

This is more printing syntax stuff



```
.data
msg:
    .ascii "%d "
print_num:
    .ascii "\n\0"

PLength:
    .quad 5
P:
    .quad 10,2,6,4,8
```

Here is all of the printing strings and stuff

Here is the length of the array

Here is the actual array.