LAB REPORT 1
ECSE 324

Muhammad Anza Khan
260618490
Ayden Aamir Malik
260627064

STANDARD DEVIATION:

Start contains the address in memory of the list, initializes R2 with the size of the list, points R3 to the first element and loads the first element into R0.

LOOP goes through the loop to check for the maximum number.

DONE moves contents of R0 into R4.

MIN_LOOP finds the smallest number.

DONE_2 finds the standard deviation.

THE PROCESS

The code starts with the loading the address of the result location in register R4. By adding 8 to the address stored in R4 we reach the memory slot that holds the size of the list and store that address in register R2.

Again, adding 12 to the address held in R4 leads us to the address of the first element of the list.

Register R3 acts a pointer to an element in the list.

Finally, loading the contents of the address stored in the R3 in to register R0 ends the first subroutine.

Registers R0 and R1 are used to hold the numbers that are to be compared. With R0 holding the 'current largest' element.

Loop:

The subroutine starts with using the SUBS function which is basically a conditional subtraction function. We have used it as a loop counter which takes the size of the list from register R2 and subtracts it by 1 and updates the value of register R2. With that done, adding 4 to the address stored in register R3, makes it point to next element in the list meaning it has gone through one iteration.

Now the LOOP subroutine loads the contents of the address stored in R3 into R1 and using the CMP function it compares the contents of R0 with R1. If the contents of R1 are greater than those of R1 then replaces the two with each other using the MOV function.
The subroutine then goes back to the start of the loop with the B(branch) function.

Next, we load a memory address of the label RESULT_2 into register R6. This time R7 hold the size of the list of elements.

R9 points to the first element in the list.

Initially, R4 holds the first element of the list later it will hold the minimum element of the list through all the iterations of loop until we exit the loop and go down to the DONE_2 subroutine.
In DONE_2, we move the smallest value stored in the R4 into R11.
Next, we subtract the minimum from the maximum and store that value in R11 and use the arithmetic shift to the right by 2 bit which means we are dividing by 4, and ASR allows us to handle 2's complement integers.

CENTER CODE:

We use the BNE function for the sum loop because we are running a decrementing counter until the conditional flag is set to 0…we are subtracting 1 from the

START
This code starts with loading R3 with the N label i.e the address of the size of the list.
R1 holds the size of the list.
R0 points to the first element in the list.

SUM
Finds the sum of the elements of the list

CENTER
Finds whether the size of the list is an integer power of 2.

SUBRACTION
Subtracts the average from each element.

THE PROCESS

The code starts with R3 pointing to the N label which holds the size of the list.
R1 and R11 hold the size of the list.

The process starts with R0 pointing to the first element of the list.

The sum subroutine acts as a loop with R2 holding the first element of the list. We start summing the elements of the list while holding the it in the register R4.
Next, we subtract 1 from the list of the list held in R1 which acts as our loops decrement counter.
During the first iteration the branch equal tos' condition is not met, and we continue down to adding 4 to the pointer R0 which now points to the next second element. The 'branch not equal to' takes us back up to the SUM subroutine and the loop runs again 7 more times until the BEQ condition is met and the loop terminates.

Then, we initialize R6 with the number 0. This block of instructions divides the size of the list by 2 and counts the number of time this happens until the quotient hits 1. Register R6 acts as the counter here.

Going further down we move the value of R4 into R5 which the sum of the elements of the list, we divide the number stored in R5 using the ASR function.

Now that R5 holds the average, we start subtracting the average from each element in the list.
R7 holds the address of the list, R8 holds the size and R9 points to the first element of the list.

R10 is loaded with the first element in memory and using the subtract function we subtract the average from each element and store it back in the memory, update the counter until we traversed through the entire list and subtracted the average from each element.

BUBBLE SORT:

START
Like above, we give the code an address from where to retrieve data to bubble sort.
R0 holds the address to the first element in memory.
R1 holds the size of the list.
R4 holds the points to first element in memory.
Register R5 is the counter for our inner loop and is used as a condition flag in the CMP function to till the loop when to stop when it hit the size of the list.
R6 is the counter of the outer loop.

WHILE LOOP acts as an outer loop.

BUBBLE_SORT is called using the branch function. It acts as an 'inner loop'.

SWAP is called by the BUBBLE_SORT to swap two elements addresses.

CHECK retrieves contents of memory and loads then onto the registers for us to see whether the desired results are achieved.

THE PROCESS

It starts by comparing the size of the list stored in R1(8) with R5 which is initialized as 0.
We start by incrementing register R5 by 1.
We load our elements to be compared in registers R2 and R3 and use the CMP function to find whether the element stored in R2 is greater than or equal to that stored in R3, if so, we branch down to the SWAP subroutine where the elements are swapped by switching their respective addresses with each other and the pointer is incremented by 4 to point to the next element.

The above process shall continue until the element 8 is bubbled down to the end of the list and the SWAP subroutine exits with calling the BUBBLE_SORT subroutine where it realizes that R5 is at 8 currently, i.e the size of the list and exits the BUBBLE_SORT subroutine and goes up to the WHILE_LOOP where register R6, the outer loop counter is incremented by one again and the pointer are moved forward to point to the 7^{th} and the 6^{th} element of the list and register R5 is reset for the inner loop to go through the list once again.

The aforementioned process shall continue until register R6 hits 8 and exits the process and enters the CHECK subroutine.

In the CHECK subroutine we have hard coded the values that are expected be in their logically assigned registers by the end of the process.

IMPROVEMENTS

Overall, we could have made the following improvements:
1. Use less registers, would have been more optimized memory usage than.
2.Used more efficient looping, which ties in with our improvement of using less registers.
3. Using post and pre-incrementing addressing modes, this would have reduced the amount of instructions we used for running loops that access memory elements by almost 20% because we would not need separate instructions to load and increment a register to point to the next memory element in an array.
4.A case to check for array's of length 0 would be an additional feature as well to make our code more secure.