Karl Doumar 260733160 February 9th, 2018

He Qian Wang 260688073

Group 32

Lab 1: Introduction to ARM Programming

Largest Number:

The largest number section wanted the user to find the largest number in each list. From analysing this problem from a coding perspective, we knew we needed a loop to go through each element, a counter, and the necessary registers to hold the elements we will be comparing. Now we loaded our result into the register R4, loaded the number of elements in the list to R2, Made R3 point to the first number, and gave R0 the contents of the first number in the list. Having the preliminary data to step into our loop, we simply updated our counter which was R2 every time we stepped into the loop.

Once inside the loop we raised a condition flag to check if the counter reached zero, from there we simply updated the pointer to check the next number in the list, used the compare instruction to see whether the first element was greater than the second, if yes store and overwrite whatever result was in R0 and branch back, if the first element was not greater than do not update the contents of R0 and branch back to the loop. This would continue until the counter reached zero. After attaining the maximum, our program was finished, and we stored the result into memory.

An area of improvement we saw after finishing the whole lab was the way we were utilizing the memory space available to us. For instance, storing values into memory should only be done for the final value and not the intermediate steps. When we did SUBS R2, R2, #1 we could have replaced SUBS with SUB and attained the same result without using that much memory.

Standard deviation idea:

For the Standard deviation part of the lab, the way we went about it was to follow a similar approach to the methodology used in the largest number section. First, we needed a maximum element, a minimum element, and the size of the list to undergo the fast-standard deviation computation. From those ideas, we simply elaborated and decided to do a Max Loop (which we call MLOOP) and a Minimum Loop (which we call LLOOP). After loading the result, number of elements, first element of the list, and a pointer to the first element of the list to their respective registers we were able to write out our loops.

Basing ourselves from the Max loop code we simply decremented our counter each time we entered the loop, raised a condition flag if the counter has reached zero, if not we continued with the program. We updated the pointer of the register we were using to point to the next element and possessing already the first element stored, we were able to use the CMP instruction to check if the first element is larger than the second, if not branch back to the MLOOP, if yes update the current maximum than branch back.

The minimum loop Followed the same logic as the Max loop, we simply switched our branching condition flag to BLE (less than) instead of BGE (greater than) that way if the element was less than the next element we update our current minimum. After doing all those computations we simply subtracted the register holding our maximum value from the register holding our minimum value and shifted the result by 2 bits, since division by 4 is simply 2 to the power of 2.

An area of improvement would probably have been to only store our result. We decided to store the intermediate steps leading up to the result, that is to say the maximum number and minimum number. However, after reviewing our methodology we realised that this was quite demanding in terms of memory.

Center an Array:

We know from the lab instructions that centering an array is to make it so that the sum of every element in the array becomes zero. There are a few constraints which are that the numbers of elements used must be a power of 2. Dividing becomes much simpler as we simply need to do a ASR by removing the number of power as bits. We would take the average and subtract every element of the array by the average to properly center it.

Our program has two sets of contents in memory which are N and numbers in the array. Our program initializes a few registers with the values of 0. Then we start loading values into the registers. We have a first loop which adds every element into a cumulative sum contained in the register R5. We stop once we are done looping through every element of the list.

The second loop is here to know what is the power of the numbers. Our program can constantly divide the number by 2 until we reach the value of 1. We will use this number to perform an arithmetic shift right by the power. The third is simply to loop back through the array and then subtract the value of the offset from the number stored in the array. Then we would store it back in memory using STR.

Sorting

Our sorting algorithm uses the same logic as the one given in the lab document. We have a nested loop that constantly checks for the Boolean SORTED. The Boolean is represented by a register R1 in our case. Our program consists of two loops. The first loop serves to initialize the parsing of the list of numbers and it passes the initial values to the registers. We are using register R5 to track the index of looping in the second part. The registers R2 and R3 serve as runners in the list. They will temporarily hold the values of the two numbers to be compared in the next steps.

The small loop will check every between each other. If the bigger one is in R6 instead of R7, we will have to swap the two values. We are using a temporary register R8 which will hold the value of the bigger element. Then we use the STR instruction to rewrite into memory.

Concerning Sorting, our algorithms are not register efficient. Especially after learning about stacks in ARM, we realized that we could make the program a lot of efficient than it currently is. Our takeaway is being able to be aware of the space usage.