ECSE 324 - Computer Organization
Lab 1 - Report

The goal of this first lab is to use the Intel FPGA Monitor Program to implement four simple algorithms to a given array using the Assembly Language: finding the largest integer, standard deviation calculation, centering and sorting (bubble sort).

Finding the Largest Integer

The goal of the largest integer algorithm is to find the largest integer in an array of integers. We are using a loop to determine this; however, before the loop starts, we store the number of elements and the first element of the array in two variables.

The main loop, named "LOOP" starts by assuming the first element is the largest element in the array. It compares this element (R0) with the next element in the array (CMP R0, R1). If the next element in the array is larger than the current maximum, the algorithm updates the current maximum element (MOV R0, R1)

The loop counter is set to the number of elements in the array, and it gets decremented by one at the end of every iteration to ensure that the algorithm covers the entire array.

The result is stored in R4, which is the location of RESULT.

Standard Deviation

The standard deviation is found using this equation:

$$= \sqrt{\frac{\Sigma(x_i - \mu)^2}{N}}$$

However, our algorithm calculates the standard deviation using this equation:

$$\sigma \simeq \frac{x_{max} - x_{min}}{4}$$

This equation is accurate enough for our project. This equation involves finding the maximum and minimum elements of the array, subtracting the minimum from the maximum and dividing the result by four.

The algorithm for finding the maximum element is explained above, and the algorithm for finding the minimum element is similar; except we assume that the first element is the minimum instead of maximum; and the algorithm updates the smallest element only if the current element is smaller than the smallest element (R0)

While we believe that both the minimum and the maximum element can be found using one loop, we decided to implement two loops; one for finding the maximum (named LOOP) and another for finding the minimum (named MINLOOP) as we were not given a time constraint. The maximum element is stored in R0 while the minimum is stored in R5.

Once MINLOOP is done, the program subtracts the minimum from the maximum (SUBS R9, R0, R5) and proceeds to divide the result by 4. As the quotient is a power of 2, we can shift the result right by 2 bits, which is equal to division by 4. (LSR R10, R9, #2) We then store the result in R4, which is the location of RESULT.

Centering Algorithm

Centering algorithm requires the program to find the average and to subtract it from all the elements in the array.

Finding the average involves the summation of all the elements in the array, which is done in a loop named ADDLOOP. This loop has a counter that starts at the number of elements and runs until the counter is equal to 0, and the counter is decreased by 1 at every iteration. This ensures that the program adds all the elements in the array. The sum is stored in R4.

Once "ADDLOOP" is done, the program branches to "AVGDONE", which acts as a while loop. As we are not able to simply divide two integers, the program will have to shift the sum stored in R4. The number of bits by which the program needs to shift the result is determined by $log_2(n)$, where n is the number of elements. This requires the number of elements to be equal to a power of 2.

"AVGDONE" loop calculates n in $log_2(n)$ by shifting the number of elements to right by one bit until it is equal to zero. Variable n is set as 0 (MOV R3, #0) and at every iteration, it gets increased by 1.

Once we have the correct n value, the program branches to the "DIV" loop, which simply shifts the result by n (R3) bits. (LSR, R0, R0, R3) The program then branches to the "SUBSLOOP" branch, where it subtracts the average from each number. (SUBS R8, R1, R0; where R1 is the current number, R0 is the average, and R8 is the temporary address to store the subtraction)

This loop runs until the subtraction is done for every value in the array. The final values are found at the same addresses as the initial NUMBERS elements.

Sorting Algorithm

In this project, we decided to use bubble sort to sort a given array of integers. The program has a variable for "sorted" which will be used as a boolean, and two loops to do the sorting process.

The first loop, called "FIRSTLOOP" is designed as a while loop where it will run as long as "*sorted = false"*. (CMP R5, #1, BEQ END) This loop then sets sorted as true (MOV R1, #1) which will be changed to false if two elements get swapped during the second loop. This is similar to the bubble sort algorithm in Java and C.

The program then branches to the second loop, which is a "for" loop that increments the counter by one at every iteration, and that runs until the counter is equal to the number of elements. The counter starts at 2 instead of 1, as the program holds the address for the first element before the loop starts for the very first time. At every iteration, it compares the element at position i+1 (of the array) with the element at position i (CMP R7, R6). If the element at position (i+1) is less than the element at position (i), the program swaps them by branching to a loop named REPLACE.

This loop sets *sorted* as *false* (MOV R5, #0); and swaps the elements in the NUMBERS array. This is done by storing the number value of the i-th element in the address of (i+1)th element in the array. This does not cause any problems or discrepancies in the array, as at every iteration, the program stores

the values of the elements it is comparing in separate addresses. This loop then branches back to the SECONDLOOP.

When SECONDLOOP exits, the program branches back to FIRSTLOOP. If there have been any changes when SECONDLOOP ran (e.g. if sorted = false), it will branch to SECONDLOOP again. However, if sorted is true; the loop will branch to END, which is an infinite loop by design.

Possible Improvements

It is possible to add a function to check if the given array is empty. This wasn't done in this project, as we were ensured that the given array would not be empty.

We could add another function to check if the given variable N is actually the number of elements we have. However, this would be a big challenge.

For centering, we could add a function to check if N is a power of 2, as otherwise, the result that we get wouldn't be correct.

For all four assignments, we could add a clause for when N (e.g. the number of elements) is 1. The maximum element where N is 1 is naturally the only element we have. Centering could be done; however, we would get 0 as a result.

This is especially important for sorting, as we start from n=2. The for loop condition wouldn't work, as it only checks for equivalence. It would be an unwanted infinite loop.