ECSE324 Lab 1 Group 47 – Nihal, Abdallah, Fouad

Standard Deviation

Description:

Aside from the size of the array and the array itself with numbers as inputs, we also declared two variables (called MAX, MIN) in memory initializing their value to 0 for both. These are meant to store the values of the min and max.

The logic behind the program is to loop through the list the first time and find the max, then loop through the same list a second time.

1. We began by first storing the values of the MIN and MAX variables in registers.
2. Loaded the size of the list into two different registers to account for both loops
3. Loaded the address of the first value of the array into two different registers again to account for both loops
4. Store the value of the first in array in two different registers
5. Begin loop to find max, ounce found go to loop to find min (finding the min was the same as the code in part1 of the lab except in the CMP line before the branch we swap the two registers
   - We do not store the max or min into the memory locations since we use different registers for each loop we kept them in the register until last part
6. Finally, we subtracted the values from one another and shifted the register by 2 bits and stored the value in the MAX variable.

Improvements:

In step 2 in the description section above we could have used one register to store the location of the size of the array, initialized only one register to store the size, then once the first loop was done reinitialize with the same value. This would have saved us a register. This goes the same for step 3 where we would only require one less register to hold the pointer to loop through. As well goes the same for the value of the first two registers. This goes to show we used 6 registers instead of 3 where in a larger program this type of code would not be efficient and we would certainly run out of space eventually.

We could have used one loop only and taken the first two values, compared them, given the higher value to MAX and lower one to MIN then looped through the list and asked first if the value was larger than the MAX if so replace it and continue to next value. This would have made the program twice as fast. Like this we also only require one register to store the size and one less even to store the location of the size of the array. However, to do this we would have had to store the value of the min and max into the memory locations (or a register if there is one available but it would be more efficient to use the registers for immediate calculations) MAX and MIN *unlike* we did in step 5 in the description above.

If we had for example 7 (0111 in binary) as the difference between the max and min then shifting it to the right twice would truncate to 0001 which is 1. This is of course a simplified version and we would need to implement a more sophisticated division functionality.

Center

Description:

We began by creating the variable ZERO to store the value of 0 in memory as we struggled to initialize a register to 0 as a counter without using memory. This would be later used to calculate the logarithm.

We then used 4 registers to all store the size of the array. The first one was used at the beginning of the program to check whether the size of the array was 1. This was another struggle we encountered as we did not figure it out until the third part of the lab that we could use CMP and compare it to the value #1. Instead we subtracted the value 1 from the size of the array and checked to see if it was equal to 0. The other registers were used to left shift until the value became 1 to find the logarithm, and the last two were used to loop through to a) find the average then b) subtract the average from each value. The flow of the program is as follows:

1. Find the logarithm of the size of the array to see how many times we must right shift the sum of the values to find the average. This is done by looping with each loop shifting the size of the array until it is equal to 1 where. Each loop increments the counter. The counter is the amount to shift the final value by.
2. Then we loop through the array and find the sum
3. Find the average by shifting and apply it to last element of list using same pointer used in the previous sum part and store it to the same place in memory.
4. Going backward with the same pointer we go apply the center to each element but keep track of when we end using a new counter defined at the beginning.

Improvements:

An improvement we made from the first part of the lab (standard deviation) was that we used the same pointer and instead went from back to front, this saved us a bit of time and space. We could have just like standard deviation used a pointer to the size of the list instead of using two different registers.

The same issue occurs here as with the division, if the size of the array was not a power of 2 then dividing would give a false average and an incorrect centering. Following from this we only checked to see if the size of the array was 1 and not if it was empty i.e. 0. Also, there does not seem to be a way of checking whether the values in the arrays are indeed numbers.

Sort

Description:

Instead of implementing the bubble sort algorithm we decided to use insertion sort as the code for finding the min was functioning and decided it would be quicker.

We use two counters, one to store the present size of the array and one to loop through the array. Every time we find the current minimum of the unsorted array we decrement the size of the array and set that value for the counter to loop through again and find the minimum in the next min loop. We use one register to store the location of the first element of the unsorted list and another register used as a pointer to loop through the array, once one minimum is found the first of the array gets updated and the pointer to loop receives the value to loop through again. Finally, we use one register to store the location of the smallest value in the current unsorted array, this is to switch between this and the first value. The program is structured as such:

1. We loop through the entire list at first and find the minimum, switch the first value of the array with the minimum in memory
2. We then update the length and pointers so that the first value is untouched and loop through the rest of the unsorted array again with the loop to find the minimum and do the same
3. Once the size of the unsorted array reaches 1 we have completed the sorting

Improvements:

Instead of having 2 different counters to save the start of the unsorted array and to loop through the array, we are unsure how to implement it but we would use multiplication to multiply the size of the current unsorted array by 4 to return to the start of the list. This would decrease the number of registers we require and thus make it easier for us to follow without having to go back each time and figure out what each register does and update them accordingly.

We struggled with a bug where we forgot to reset the location of the smallest element to the first of the list as sometimes the smallest was the first one in the list and it would take the location of the previous first.

Another aspect that complicates the readability of the code is that for all the lab parts we initialize the variables at the start, and improvement we could have made was write the initialization for example above the start of the min finding loop and instead of going back to the start we go back to just under the initialization yet still have all that code in one chunk under one name.