ECSE 324 Computer Organization (Fall 2019)
Lab 1 Report

Group 91
Zheyu Liu 260784914
Sandra Deng 260770487

Submission date: 4th October

Part 1

Brief description of part 1

In the first part, we wrote an assembly code to find the largest number of a series of numbers. A list of data will be given, and the number of data in the list will also be given as well.

Approach Taken

For this part, we first used a register R4 for storing the result location. Then we used R0 to store the data of the address stored in R4 (the number of elements in the list). Afterwards,we used register R3 to point to the first number of the number lists, and R0 is used to hold the data in the number lists. After the initialization of the registers, we used a loop which iterates all numbers of the number list, and then checked if the current number in the number list was larger than the result (the register to store the maximum number). If the current number in the number list was larger than the result, we updated the result (the current maximum number in the list). Otherwise, we kept the result. After the iteration, we stored the result to the memory location.

Challenge we faced

This code was the first assembly code we wrote and the code was given in the description of the lab, so the main challenges we faced was the comprehension of the code. At the beginning of this lab we found the difference between LDR and MOV was blurry. As a consequence, we tried to replace MOV instruction by LDR instruction. However, we found it was not correct. By searching for the description of the MOV and LDR instruction, we found that the MOV instruction should be used to move data from one processor register to another processor register. The LDR instruction is used to move data from memory or immediate value to a processor register.

Part 2:

Brief description of rangecal:

In the second part, we wrote an assembly code to find the range of a list of data. A list of data will be given, and the number of data in the list will also be given as well.

Approach Taken

For finding the range of the data, we need to find the largest and the smallest number in the data list, then we calculate the difference between the largest and the smallest number to retrieve the range. For finding the largest data and the smallest data, the logic is similar to the part 1. We need to iterate the data list to update the current maximum or minimum number. We updated the maximum and the minimum in the same loop. If the current number is larger than the current maximum, update the current maximum. If it is not larger than the current maximum, the program would

proceed to check whether it is smaller than the current minimum and update the current minimum if it is smaller. The last step of this code was to calculate the difference between the maximum and the minimum by using SUBS command to subtract the minimum from the maximum and store the range in the memory.

Chanlenge we faced:

For this part, we first thought of using two loops by simply copying the code from part 1, and then replacing the instructions for updating the maximum to instructions for updating the minimum. Through our discussion, we found that the code in that way has larger cost and not brief enough, so we decided to update the maximum and the minimum in the same loop. When the current number was smaller than the maximum, it would skip the branch for updating the max, and then it would execute the branch to find the min for comparing the current number with the minimum. In this way, a single loop was enough to find the maximum and the minimum at the same time.

Part 3

Brief description of maxmin

In part 3 of this lab, we wrote an assembly program to compute the maximum value and the minimum value of an algebraic expression $f = (x_1 + x_2) * (y_1 + y_2)$. We are given 4 integers, which can be positive, negative or the same with other numbers.

Approach taken

We applied the following equation to simplify the calculation.

$$f = (x_1 + x_2) * (y_1 + y_2) = X * (S - X)$$

where $X = x_n + x_m$ and $S = x_1 + x_2 + x_3 + x_4$

There were several preparations we need to do before computing the f. All X(the sum of two numbers in the data list) was first calculated in our program, we considered all 6 conditions of combinations of X($x_1 + x_2$, $x_1 + x_3$, $x_1 + x_4$, $x_2 + x_3$, $x_2 + x_4$, $x_3 + x_4$). After that we stored all 6 possible X in the memory. The address of these values were consecutive, so we could access them in the later part of the program using loop and free some of the processor registers. We then calculated the S (sum of all four data in the data list) by looping through the input. The S was stored in a processor register for future calculation.

After the preparation, we looped through the X values and apply the same strategies we used in part2 to find out the maximum and the minimum of $X * (S - X)$. Once the loop was finished, the maximum and the minimum value are stored in the memory.

Challenge we faced

The major challenge we faced was to compute all the possible X values. The ideal way to solve this problem is to use nested loop. However, since it was indicated in the description that we only needed to consider the 4 numbers case, we decided to calculate the value directly by listing the equations of all the possible combinations. Besides the computation method, we also faced the challenge in the limited number of processor registers. In order to calculate all X values directly, we had to load all 4 numbers to the processor register and there are 6 combinations of X. For storing all numbers in the data list and all 6 X, we had already occupied 10 processor registers, and we only have less than 16 processor registers available. Additionally, we used several processor registers to store the number of data in the data list and some other data. As a consequence, we didn't have enough register for max and min calculation. To solve this problem, we stored the X value to the memory once it is calculated. The X values should be in consecutive memory address, so they can be accessed with loop in the max and min computation. With this strategy, we only needed 5 registers for the X calculation, and it was easier to manage all the data.