Lab 1: Introduction to AMR Programming

For ECSE 324: Computer Organization

Report Written by

Antoine Wang (260766084)

Alfred Wang (260771591)

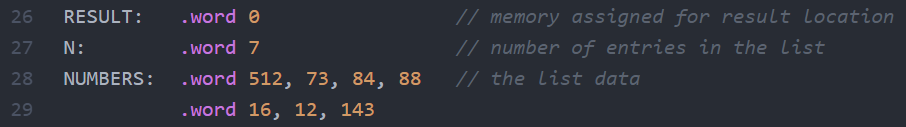
Due October 5th at 11:59PM

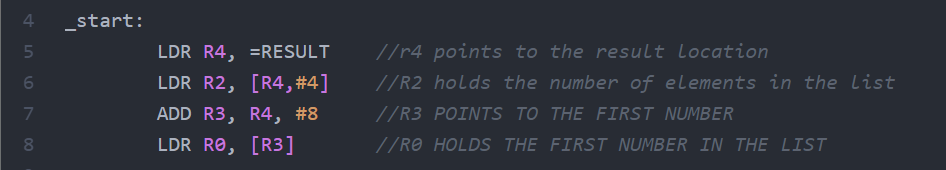
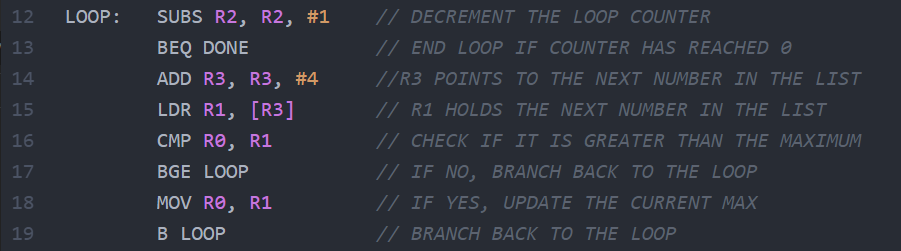
McGill University

1.1 Finding out the maximum value of a set of data

The purpose of the first section of this lab is to find out the maximum value of a set of data. The code has already been given at the beginning of this lab, but we still must fully understand the code for the sake of doing the following parts of the lab. The logic of the codes given is relatively straight forward as shown below.

1. Allocates memory for the result, counter, and the list of data.



1. Initialize registers that will be used for finding out the maximum value. 
2. Compare the current maximum value to the next number in the list. If the next number is greater, copy this number to the register storing the maximum value; otherwise, directly jump back to LOOP. Also, there is a counter at the beginning of the loop, so that all the numbers can be compared. 
3. When the maximum value is found, store this number back to the memory address that specified by R4 (Maximum value can both be found in register R0 and the memory address specified by R4). 

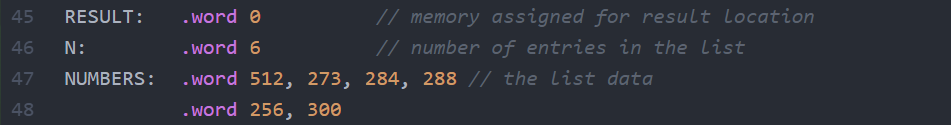
2.1 Finding out the range of a set of data

The purpose of this section of the lab is to find out the range of a give set of data. The basic logic to achieve this goal is to find out both the maximum and minimum values of the given set of data, and then subtract the maximum to the minimum to find out the range.

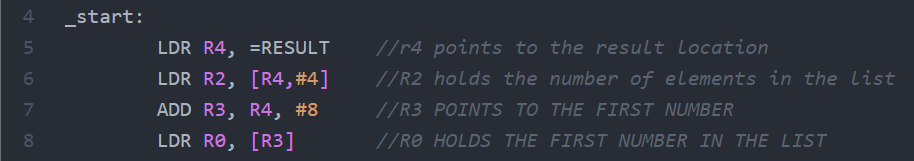
There are three main parts to the design of the code. The first part is to find out the maximum value of the given set of data, which is the same as the previous section of the lab. The second part is to find out the minimum value of the given set of data. Although it is finding the minimum instead of the maximum, but the logics are quite similar. We compare the current number and the next number; if the next number the next number is smaller than the current number, we copy this number to the register storing the minimum value; otherwise, jump back to the starting line of the loop. The last part is to calculate the range of the given set of data, which is simply subtracting the minimum value from the maximum value.

The logic and explanation of the code are shown below.

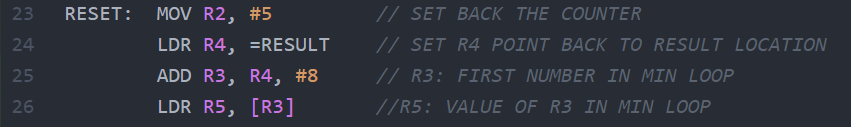
1. Allocate memory the result (for both the minimum and the maximum value), counter, and the list of data.



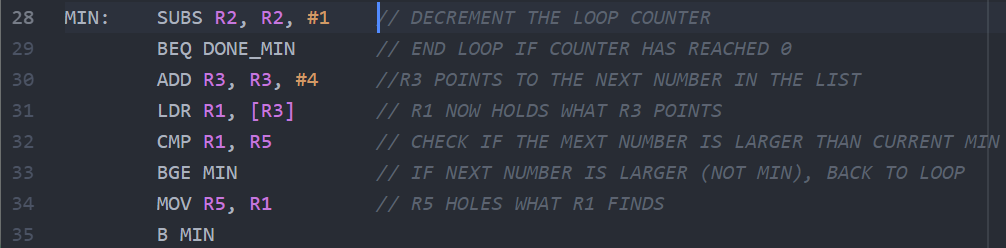
1. Initialize registers that will be used for finding out the range value.



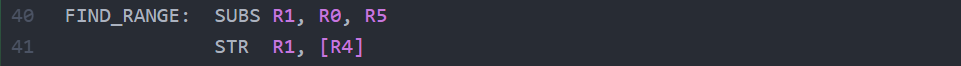
1. Find out the maximum value of the given set of data. The code and logic are the same as the previous section of the lab, except for the name of the loop and the register used.
2. This part of the code resets the counter and loads the first number of the list to R5, to get ready for finding out the minimum value of the list.



1. Find out the minimum value of the given set of data. As shown below, the logic is the same as finding the maximum value; however, when comparing the two numbers, the sequence of the two operands are revered compared to the code for maximum value. If current number is not greater than the next number, jump directly to the first line of the MIN loop; otherwise, copy the next number to the register that stores the minimum value.



1. After finding out the minimum and maximum values of the given set of data, we subtract the maximum value to the minimum value, and then store it back to the memory address specified in register R4 (Range value can both be found in register R1 and the memory address specified by R4).



Overall, the logic and code for this section of the lab is straight forward. There are not many improvements that can be made to the logic. However, it is highly possible that less registers can be used in the code while we are trying to implement our logic. Furthermore, there are other logic that can be tried to get the range of a given set of data. For instance, find out the subtraction results for all the combination of the numbers given, and then find out the largest subtraction result, which is also the range of the given set of data.

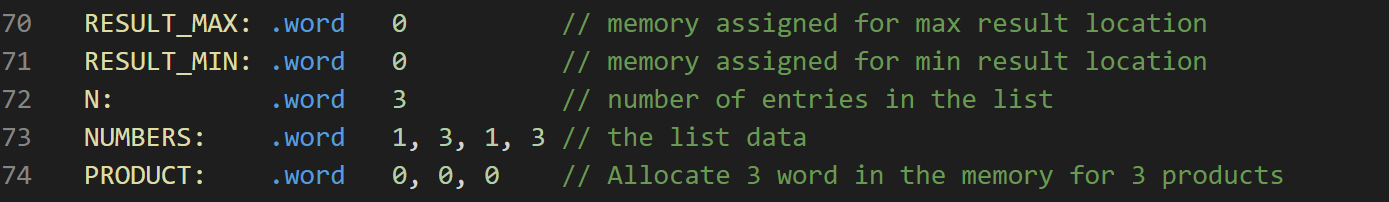
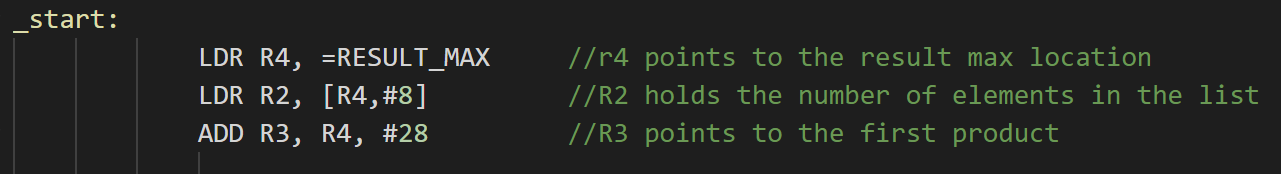
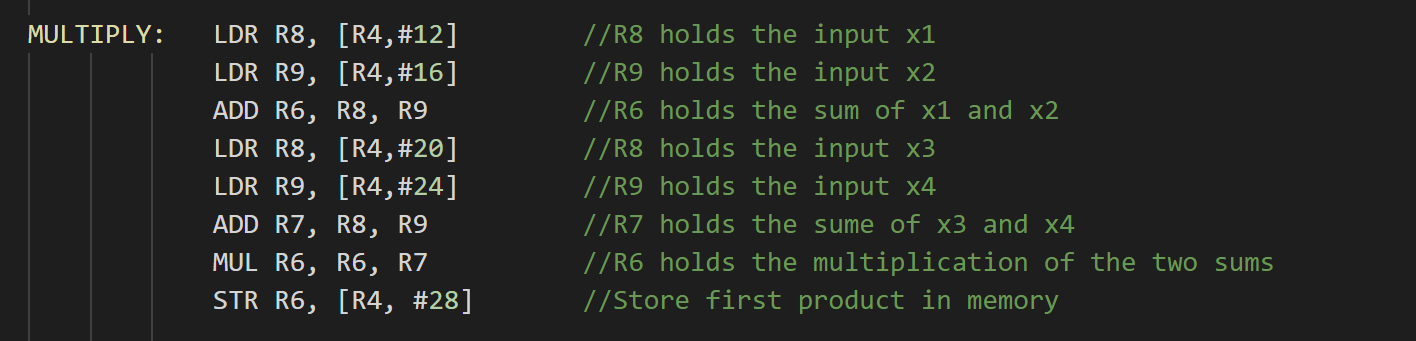
2.2 Maximum and minimum values of an algebraic expression

In this section, an algorithm is developed aiming to find the maximum and the minimum value of the products of two sums generated from the combination of 4 numbers.

The design of the code follows a series of abstractions and modulization. At the first place, the consideration of the code is purely mathematic. According to the calculation, 4 numbers, partitioned into two each, can only produce 3 distinctive results. All the possibilities are , and . Any further permutations will be redundant. Thus, with these 3 possible outcomes, the next step to consider is to search for maximum and minimum.

An important idea is that since from the previous questions we have already attained a functional algorithm, essentially a “for” loop which finds the largest value in a range of numbers. Later, that loop is adapted so that the minimum value can also be searched, which enables the solution to question 2.1, calculation for range. Thus, if those loops can apply directly, the two extreme values can be spotted immediately. There are a few things worth noticing about the reuse of the code. First, a plan for the memory should be taken care since the loop in the previous section only operates on CONSECUTIVE memory locations. The “next” pointer iterates by simply adding 4 bytes. Thus, after the three products are retrieved, they need to be allocated on three continuous spots right next to each other. Secondly, same as “rangecal” program, after going through the maximum loop, pointers and counter needed to be reset for the commence of the minimum loop.

Below is the overall logic of the “maxmin” program:

1. Allocates memory spot for two results, loop counter (value is the number of all products, thus 3), 4 arbitrary numbers and product spot.
2. Initialize the pointer.
3. Add two numbers, then add the other two, compute the product. After the calculation store the result back to the memory (location “PRODUCT”).
4. The procedure of finding maximum and minimum is identical to that in “rangecal.s”.
5. An example of storing the result. At the final output stage, the maximum can be found in both R0 register and the memory address specified by R4.

To sum up, for this task the logic is straightforward and thanks to the relatively simple conditions, the arithmetic procedure is not complex. However, the competency of the program fails if the product can be calculated among the sums of 3 numbers and 1 number, for example (X + Y + Z) \* W. The way the products are generated are mechanical.

Another interesting situation occur during the simplification of the code. Instead of load numbers into two separate registers, an attempt which is to merge the selection of the number and adding exists (see code in step 3). The failed code is like:

ADD R8, [R4, #12], [R4, #16]

The complier fails to interpret the command. A reasonable guess is that such a code with complexity is not supported by a RISC machine.