Petit Samuel

***CS1021 / Introduction to Computing I***

# Assignment #2
# Using Memory

# Step 1 – Symmetric Difference

## Solution Description

The first step to solving this problem was to find out what was a symmetric difference of 2 sets of numbers, and it turns out to be "the set of elements which are in either of the sets and not in their intersection" (source: Wikipedia).

Once I understood clearly what was asked here was my approach step by step to the problem.

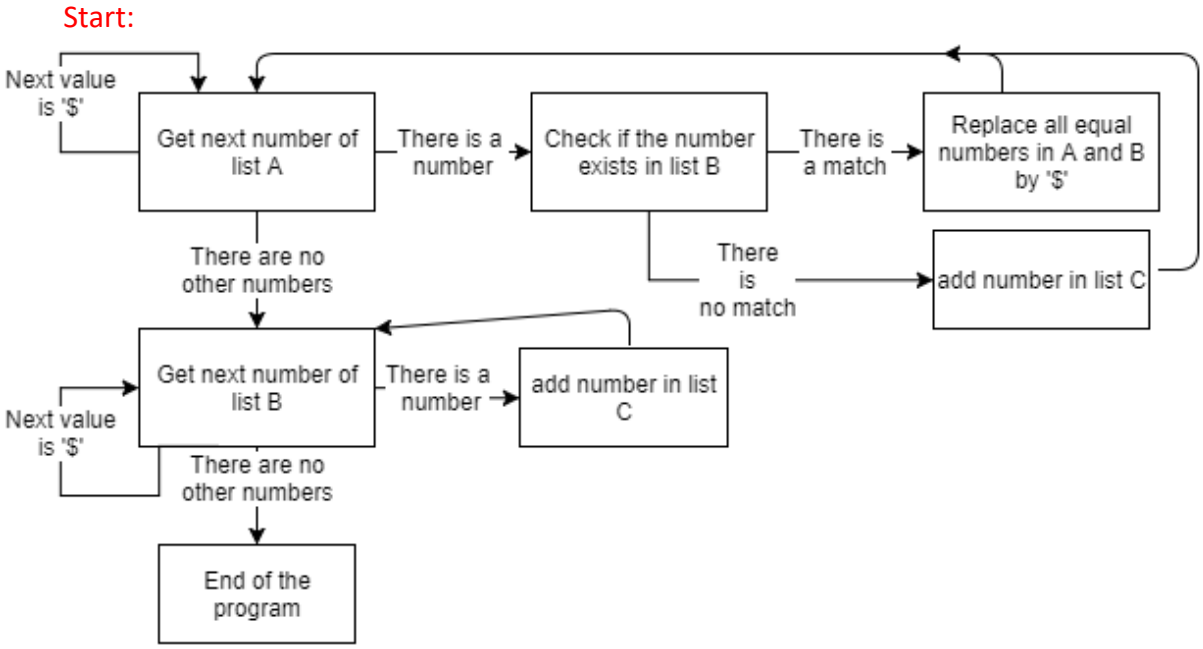


*Diagram showing my solution method for finding the symmetric difference of two sets A and B, storing the result inside the third set C*

A few things to note for this program: the values are 32 bits and unsigned, this means that my program needs to use LDR and STR to load and store values from and into registers. It also means that, to obtain the next value, the value of the address given needs to be added (or subtracted if needed) by 4.

## Testing Method

To test this program, I used a part of my program from the assignment #1. (the display part). I made a loop which went through all the values in the set C (the result set), and would proceed to print out that number in the console. Since that part of my program is already explained in my first documents for the first assignment I will not go into detail here.

I would also like to add that since I assumed you would use your own testing method and that it wasn't asked in the assignments sheet I did not leave that part of the program.

Finally, all the combination of sets I have tried have given correct outputs, therefore I assume that my solution is correct for most of the set combinations.

# Step 2 – Countdown Checker

## Solution Description

This program used the same first loop as the program above, this time, the program will have to go through each of the characters of a word stored in memory and find out whether the set of characters stored in the other set can form that word.

For this to be possible I used a first loop which goes through all the characters from a valid word contained in set A. For each letter, another while loop will look for that same character and if it finds it replace it with a symbol ('$') to mark that this letter has been already used (as you cannot re-use the same letter more than once). Then, if a letter from the word is not found in the other list (of random letters), the program stops and stores 0 in R0.

At the start of the program, R1 is set to 1 (which means it is assumed that the letters can form the word), as said earlier, if an exception is found the program will be stopped and 0 will be stored into R0. Therefore, if it finishes going through all the characters of the word then R1's value will not change (will remain of 1), which will mean that the letters can form the word.

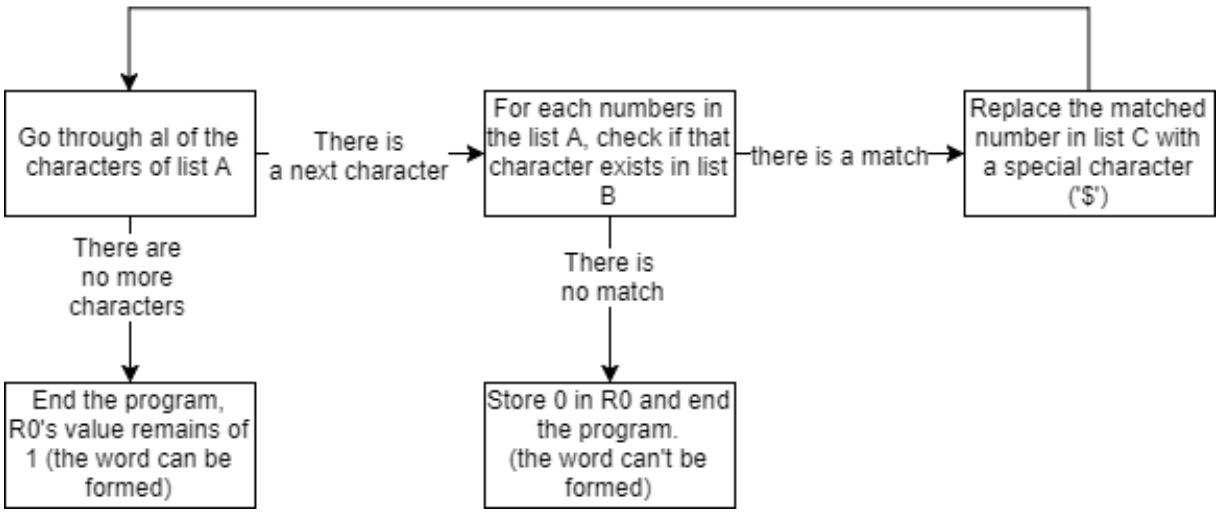


*Diagram showing my solution method for finding whether a word stored in a set A can be formed using characters in a set B. If it is possible, R0 will store 1, else it will store 0.*

Since this program deals with characters (letters), we know that ASCII values are 1 byte therefore this program will use STRB and LDRB to store and load values to and from memory. Then, to go from one letter to another, the value of the address will use an addition or subtraction of 1.

## Testing Method

This program was very easy to test as you can observe the value stored in R0, therefore it did not need any extra code to make it easy to test. My method of testing consisted of trying out different sets for the word as well as the list of letters and find out whether the output was correct or not.

# Step 3 – Countdown Checker

## Solution Description

This problem required (in my opinion) more thinking than the other two. It requires to compare a list of numbers with a set of 6 numbers, keeping the amount of 'tickets' that have 4 or more numbers matching.

Since the numbers for each ticket has been put together I first needed to figure out how to know when to reset the count of matching numbers for one ticket. I found that, since a ticket contains 6 numbers, I will reset the count of matching number for that ticket when:

Count % 6 == 0 (i.e. when the remainder of the count divided by 6 is 0)

To find out when to stop the loop going through all the values was quite simple: the value contained at the address 'COUNT' contained how many tickets there were to compare, therefore you must compare (x * 6) numbers (where x is the value contained in 'COUNT').

After having a first loop that goes through all the numbers contained in each ticket, you need another loop to compare those values with the winning numbers, if there is a match then the count of matched numbers increases by one. Then, as said above, once the count of numbers compared (be careful not to mistake this for the count of matching numbers the ticket has with the winning ticket) has a remainder of 0 when divided by 6, that means it has finished comparing one ticket, then the count of matching numbers is compared with 4, 5 and 6. If that count is equal to one of those numbers, the appropriate value stored in memory containing the amount of winners with X matching value is added one and the matching numbers count is set to 0. If it does not match any of those 3 numbers, it is only reset and the value it had is then discarded.

Here is a simple diagram illustrating how my solution works:

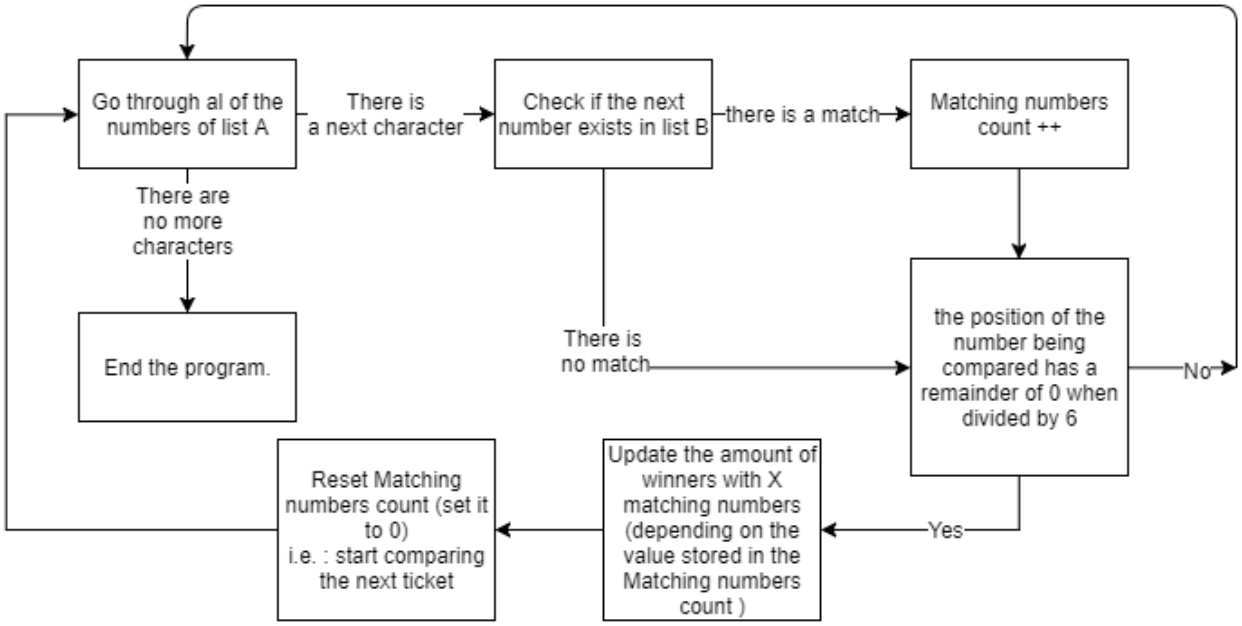


*Diagram showing my solution method for finding how many tickets have 4, 5 or 6 matching number to a winning ticket, storing the result in memory*

Finally, all the numbers have a range from 1 to 32, therefore they can be stored within a byte. This program will then use LDRB to get values from the memory places which contain the numbers. However, there may be more winners than what a byte could store, therefore I will use STR/LDR when it comes to using the amounts of winners.

Note : at the end of the program I store the amount of winners for 6, 5 and 4 matching numbers in registers R0, R1 and R2 (in this same order), this assumes that there can only be a maximum of winners that can be stored in a 32bit number (a maximum of 4,294,967,295 for an unsigned number and 2,147,483,647 for a signed number), this is more than fair as it is very unlikely that as many people would enter in this lottery, even less likely to have that many winners.

## Testing Method

Testing this solution was similar to the 2nd program, at the end of all computations I stored the results in R0, R1 and R2 (as said above). I only then had to compare the answers to check if they were correct for the different numbers stored in memory I have tried.