ARM Memory Assignment

Luke Hackett - 17324340
CS1021 Intro to Computing

Sets - Symmetric difference

Implementation:

The symmetric difference between two sets is the set of numbers that are exclusive to either set. To find this had a loop that started with the first element of set A. It compared each element in B to this element and if they matched the program changed the element in B to the '%' sign to mark that there had been a duplicate. In the event that the loop finished without finding a match, the A element would be added to a new set C and the size of C would be incremented by 1.

Once the program is finished looping through set A, it is a simple case of going through set B, comparing the elements to the '%' symbol. If the B element was not a '%' then that element was not a duplicate and therefore could be added to set C and the size of C was increased by 1. When the program was finished the symmetric difference of set A and set B was stored in a new set C in memory.

Testing:

For testing it was essential to ensure that the first element of each set was being checked properly, as well as the middle and end elements. So I used the following test sets:

A Elements	B Elements	C Elements
13,3,6,4,7	13,5,2,6,8	3,6,4,7,5,2,6,8
1,5,6,9,3	4,7,8,9,12	1,5,6,3,4,7,8,12
14,23,5,2,7	31,14,5,7,8	23,2,31,14,8

As we can see from these test results the program will work correctly for cases where the match is at the start, in the end or in the middle, as well as with multiple matches.

Countdown Checker

Implementation:

To see if one string can be formed from another, we first load into our registers the memory address of the string that we want to compare and the letters that are available to form this string. We start with the first letter of the string, and compare it to each of the letters in our list. If we find a match for our letter, we can replace the letter in the list with a '%' sign and move on to the next letter in our string. We store the result in R0, which we set to 1 each time we find a match and then after we finish comparing the letter to our list we check the value of R0. If we ever get to the end of the list of letters with no match when this check is performed R0 will be 0 and the program will end.

Testing:

For testing it was essential to ensure that the first letter in the string was being checked properly, as well as the middle and end elements. Also that the first, middle and end letters of the letter list were checked. So I used the following test sets:

cdWord	**cdLetters**	**R0**
beems	baetejmfj	0
dream	ademergth	1
atom	reqwipjld	0
attention	taenttoni	1

Lottery

Implementation:

First we load in the memory location of the tickets and the number if tickets into registers. We can then compare each ticket individually to the draw. Going through number by number we can add ro a count every time a match for a number is found. At the end of each ticket we could then compare the count of matched numbers to 4, 5 or 6 then add to the correct memory address MATCH4, MATCH5 or MATCH6. After the ticket is finished all counts reset and the memory address is shifted to the next ticket. After all tickets are finished the program will exit and the results of ticket matches will be stored.

Testing:

For testing we needed to see if the correct amount of numbers were being added and placed correctly in the match:

Tickets	**Draw**	**Matches**
1,2,3,4,5,6 3,4,5,6,7,8 2,3,4,5,6,7	1,2,3,4,5,6	Match 4: 1 Match 5: 1 Match 6: 1
1,2,3,4,5,6 3,4,5,6,7,8 2,3,4,5,6,7	8,2,4,9,7,6	Match 4: 0 Match 5: 1 Match 6: 1