Computing Assignment #2

Memory

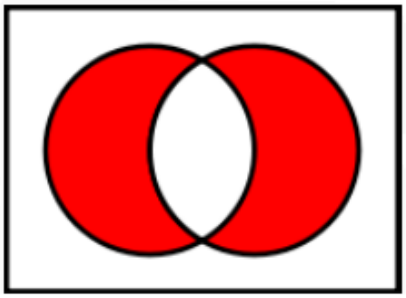
Documentation

Kamil Przepiórowski

Student no. 17327895

1 Sets – Symmetric Difference

Part one of the program creates a third set C, in main memory, which is the symmetric difference of sets A and B. In other words, the set C contains the areas highlighted in red, on the following diagram.



Symmetric Difference of Sets

$A \triangle B$

To do this, the program follows a pretty simple algorithm. In which it runs through sets A and B, and compares their values.

-First, it runs through set A, comparing each independent value of A to every value of B, checking for a match.

-If a match is not found, that value is stored in set C.

-After each value in A is compared to set B; the program runs through set B, comparing each independent value to every value in set A.

-If a match is not found, that value is stored in set C.

-If a match is found in either of the cases, the program skips over that value.

Set C is stored in main memory, and each time a value is added to it, a different value called CSize, is also increased, to represent the size of the set. The address is also incremented by 4 to make space for the next word value, and avoid overriding old ones.

The pseudo-code for the first part looks like the following.

```
while(ASize!=0)
{
areEqual=false
move BSize into a temp register, reset every loop

BElem=Memory.Word(adrB)

   while(BSizeTmp!=0)
   {
       if(AElem==BElem)
       {

       areEqual=true
       }

   BSizeTmp--
   adrB++
   BElem=Memory.Word(adrB)
   }

if(!areEqual)

store AElem in C
adrC++
CSize++


ASize--
adrA++

}
```

The pseudo-code for comparing set B to set A, is the same, but flipped around.

(A is swapped with B)

For example, if this is the input is:

```
ASize   DCD 8           ; Number of elements in A
AElems  DCD 4,6,2,13,19,7,1,3   ; Elements of A

BSize   DCD 6           ; Number of elements in B
BElems  DCD 13,9,1,9,5,8        ; Elements of B
```

Then the output would look like this.

```
CSize   DCD A           ; Number of elements in C
CElems  SPACE 4,6,2,19,7,3,9,9,5,8          ; Elements of C
```

```
0xA1000044: 0A 00 00 00 04 00 00 00 06 00 00 00 02 00 00 00 13 00 00 00 07 00 00 00 03 00 00 00 09 00 00 00 09 00 00 00 05 00 00 00 08 00 00 00
```

2 Countdown Checker

Part two of the program included a program based around the idea of the game 'Countdown'. The user is given a list of 9 letters, and has to create the longest possible word out of them. This program determines whether the word created by the user is legitimate. i.e. checks if every letter used is in the original list.

Firstly, the program checks whether there are actually 9 letters in the list, if not, then it stops. It follows this pseudo-code to achieve this. A simple loop and a counter which increments by 1 each time a number is found.

```
while(letter!=0)
{
counter++
adrB++
BElem=Memory.Word(adrB)
}

if(counter!=9)
end program
```

Next, the program runs through each letter of the word, and compares each individual letter with every letter in the list. If a match is found, both values get converted into a '!', (*to avoid cases when a letter from the list only appears once in the list but is used twice in the word*) and the program moves onto the next letter in the word.

```
while(letter of word != 0)
{
   Load start address of list
   R4 = first letter of list
   while(letter from list !=0)
   {


       if(letter from word = letter from list)
       {
       letter from word=!
       letter from list=!

       }
   adrB++
   BElem=Memory.Word(adrB)
   }

adrA++
AElem=Memory.Word(adrA)
}
```

So, for example:

Word	List of Letters
beets	daetebzs
!!!!!	da!!!!z!

Then, the program runs through the word again, and this time checks if every letter has been converted to a '!'. If that is the case, the boolean canBeFormed is set to true, otherwise, if the program detects a different character than '!', the boolean is set back to false and the program is stopped.

This is the pseudo code which the program follows to check whether the word can or cannot be formed from the given list of letters.

```
Load start address of word
R3 = letter of word

while(letter from word!=0)
{
   if(letter from word != '!')
   {

   R0 set to false
   program ends
   }
R0 set to true
adrA++
AElem=Memory.Word(adrA)
}
```

The boolean is stored in R0 and is set accordingly by the end of the program.

3 Lottery

In part three, the program checks how many people matched 4, 5 or 6 numbers in a lottery draw, and then stores those values in corresponding addresses in main memory.

After the program loads in all the register values required, it follows this pseudo-code in a loop, where it takes a number compares it with each number of the draw, if there is a match, a matchCounter is incremented by 1, if there isn't, it moves on to the next value in the ticket.

```
while(count of numbers on ticket < 6)
{
   if(count R0 == amount of tickets*numbers per ticket)
       end program

 R4 = first number of draw

   while(R4!=0)
   {
   if(number on ticket matches draw)
       {

       matchCounter++
       }

   adrB++
   BElem=Memory.Byte(adrB)
   }

adrA++
AElem=Memory.Byte(adrA)
count of numbers per ticket++
count for R0 ++
}
```

As the tickets are not separated by 0's in main memory, we need to keep track of how many numbers we checked per ticket (6 per ticket). Every time this counter reaches 6, the program checks how many values were matched 4/5/6 following this pseudo code.

```
if(numbers matched on ticket to draw count == 4)
{
   match4Count++
   reset count
}

if(numbers matched on ticket to draw count == 5)
{
   match5Count++
   reset count
}

if(numbers matched on ticket to draw count == 6)
{
   match6Count++
   reset count
}
else
   reset count
```

If the ticket hasn't matched 4/5/6 values, the counter is simply reset and the program moves onto the next ticket if R0 count is not finished.

Count for R0 is the total count of numbers on all tickets, i.e.

numberOfTickets*amountOfNumbersPerTicket

Once this counter reaches that value ^^, the program stops comparing the tickets and the draw values.

It moves on and loads match4Count, match5Count and match6Count into main memory as word sized values. The reason this happens at the end of the program, and not after every loop, is because it is much more efficient. Accessing registers is much faster than accessing main memory, and it is safe to assume ARM has enough space to contain all the required values, as it is likely less than ~8.7 billion tickets are sold.

For example, if these are the values;

```
COUNT   DCD 3                       ; Number of Tickets
TICKETS DCB 3, 8, 11, 21, 22, 31    ; Tickets                                   ;match 2
        DCB 7, 23, 25, 28, 29, 32                                               ;match 0
        DCB 10, 11, 12, 22, 26, 30                                              ;match 6


DRAW    DCB 10, 11, 12, 22, 26, 30  ; Lottery Draw
```

Then the following would be stored in main memory

(match4Count=0, match5Count=0, match6Count=1)

```
0xA1000020: 00 00 00 00 00 00 00 00 01 00 00 00
```