Cian Higgins
Rebekah Clarke
CS1021
Introduction to Computing
22 December 2017

Assignment 2 – Memory

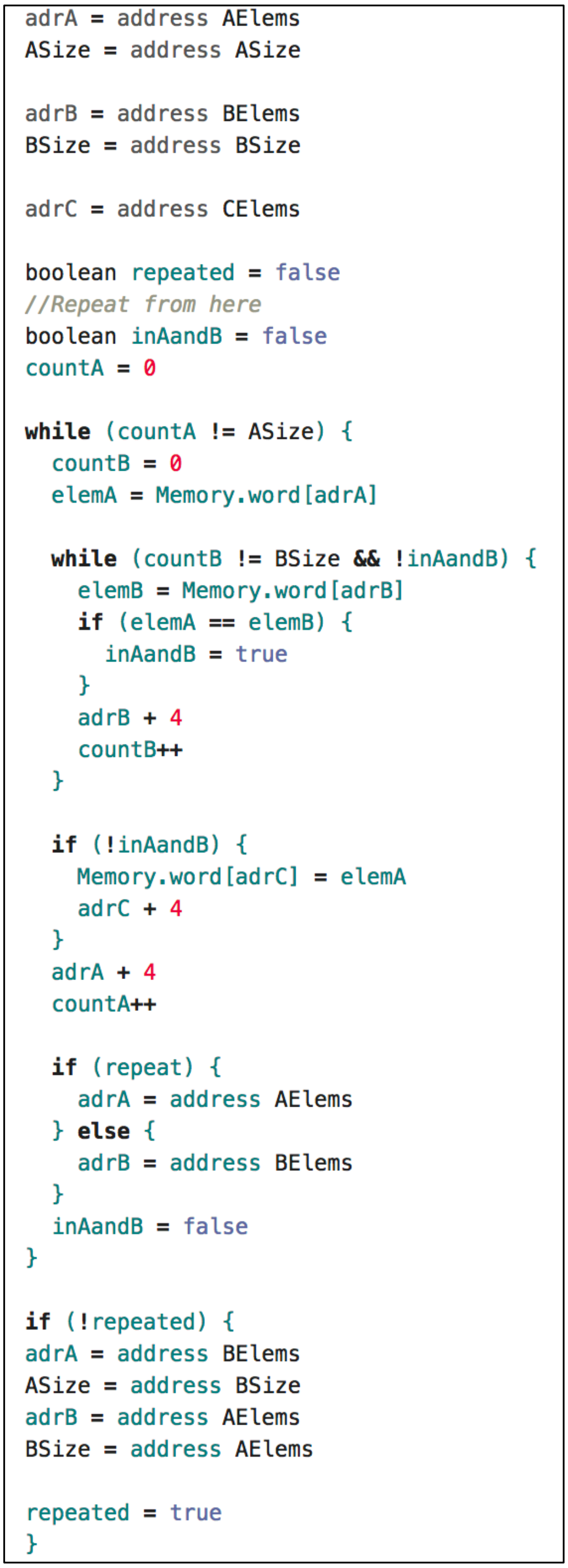
This report is an explanation and description of the three programs developed using ARM Assembly Language as part of Assignment 2. The assignment consists of three parts: (i) Sets – Symmetric Difference, (ii) Countdown Checker, and (iii) Lottery.
Part one required the program to determine the symmetric difference of two sets. Part two is a program used to check whether or not a word (or random string of characters) can be formed from a string of characters and finally part three is a program that checks a certain number of lottery tickets against the actual numbers drawn and counts how many tickets matched enough numbers to win prizes.

1 Sets – Symmetric Difference

Part one required a program to be written which would compare two sets and determine the symmetric difference of those two sets. The symmetric difference of two sets is the set of elements which are in either of the sets but not in the intersection of those two sets.
The program is required to create a third set, set C, by determining the symmetric difference of set A and set B and saving this difference into the new set. The sets A and B, along with the number of elements in each set is stored in memory. Space in memory is also allocated for set C.
The elements of the sets and the size of the sets are saved as 32-bit unsigned values (word-sized values). This means all addresses will need to be incremented by four to go to the next value and the ARM instructions LDR and STR will need to be used within the program.

To determine the symmetric difference and create a new set to store it in, the following approach is used by the program. The pseudocode for which can be viewed below this explanation.
The program utilises two while loops, the first loop takes an element from set A and starts a counter for the elements of set B. The second while loop is then entered but only when the B counter is not equal to the size of the set B, this means the loop will exit when the end of the set is reached. The second while loop must also have a Boolean set to false which determines if the current number is present as an element in both sets or not. The loop then takes an element from set B and compares it to the one take from set A. If they are equal, the loop is ended. If the element is present in both A and B it cannot be an element of the sets' symmetric difference. If the second loop reaches the end of set B and no matches are found, the number must be part of the symmetric difference, therefore it is added to set C. At this point the program will take the next element from set A and repeat the above process.
When the end of set A has been reached, the program switches the registers of anything belonging to set A with the registers of set B. The two loops are run again and any values

which are only present in set B will be added to the set of the symmetric difference. At this point the program will terminate as the symmetric difference has been fully determined.



```
adrA = address AElems
ASize = address ASize

adrB = address BElems
BSize = address BSize

adrC = address CElems

boolean repeated = false
//Repeat from here
boolean inAandB = false
countA = 0

while (countA != ASize) {
  countB = 0
  elemA = Memory.word[adrA]

  while (countB != BSize && !inAandB) {
    elemB = Memory.word[adrB]
    if (elemA == elemB) {
      inAandB = true
    }
    adrB + 4
    countB++
  }

  if (!inAandB) {
    Memory.word[adrC] = elemA
    adrC + 4
  }
  adrA + 4
  countA++

  if (repeat) {
    adrA = address AElems
  } else {
    adrB = address BElems
  }
  inAandB = false
}

if (!repeated) {
adrA = address BElems
ASize = address BSize
adrB = address AElems
BSize = address AElems

repeated = true
}
```

The program was tested, the results from the tests are summarised as follows:

(i)
Set A: {4, 6, 2, 13, 19, 7, 1, 3}
Set B: {13, 9, 1, 9, 5, 8}

Desired result: {4, 6, 2, 19, 7, 3, 9, 9, 5, 8}
Actual result: {4, 6, 2, 19, 7, 3, 9, 9, 5, 8}

(ii)
Set A: {1, 2, 3, 4, 5, 6, 7, 8, 9}
Set B: {5, 6, 7, 8, 9, 10, 11, 12}

Desired result: {1, 2, 3, 4, 10, 11, 12}
Actual result: {1, 2, 3, 4, 10, 11, 12}

(iii)
Set A: {8, 21, 99, 105}
Set B: {8, 12, 75, 98}

Desired result: {12, 21, 99, 105, 75, 98}
Actual result: {12, 21, 99, 105, 75, 98}

(iv)
Set A: {1, 2, 3, 4, 5}
Set B: {5, 4, 3, 2, 1}

Desired result: { }
Actual result: { }

The results obtained show that the program is functioning properly and returning the desired result in all tested cases. The program can correctly determine the symmetric difference of two sets and store it in a third set, C.

2 Countdown Checker

This part of the assignment required a program to be written in ARM Assembly that would check if a word could be made using a string of nine characters. The aim of the TV gameshow ‘Countdown’ is to find words using a set of nine letters. The program should store 1 in R0 if it is possible for the word to be made using the nine letters and it should store 0 in R0 in any other case. The approach taken for this part of the assignment is as follows:

The program uses a while loop that only runs if the current element from the word string is not equal to zero (the string is null terminated, so if it is equal to zero, the program has reached the end of the word) and if a Boolean is false. The Boolean will be set to true if it is impossible for the word to be made from the string of letters because a letter cannot be found. This terminates the program early and improves the efficiency of the program by a small bit. The while loop contains an if statement that checks the current letter from the string against the current letter of the word. If the letter is equal to zero, the not possible Boolean is set to true. If the letter is the same as the one in the word, the letter in the string of characters in replaced by ‘$’ so that the letter cannot be used again. Then the address of the word is increased by a byte and the letter address is reset. If the letter is not the same as the current letter in the word, only the letter address is increased, this way every available letter is checked against the letter from the word.

The pseudocode for this program is as follows:

```
adrWord = address cdWord
adrLetters = address cdLetters

result = 1
currElemA = adrWord
notPoss = false

while (currElemA != 0 && notPoss == false) {
  currLetter = adrLetters
  if (currLetter == 0) {
      notPoss = true
  } else {
    if (currElemA == currLetter) {
    Memory.byte[adrLetters] = "$"
    adrWord + 1
    adrLetters = address cdLetters
    } else {
      adrLetters + 1
    }
  }
}
if (notPoss == true) {
  result = 0
}
```

Testing carried out on the program determined that it was functioning correctly and returning the correct value in R0 depending on whether a word could be made from the string of characters. The table below is a summary of the tests:

Word	**Letters**	**Desired result**	**Actual result**	**Pass / Fail**
“beets”	“daetebzsb”	R0 = 1	R0 = 1	Pass
“donkey”	"bconeydkz"	R0 = 1	R0 = 1	Pass
“donkry”	"bconeydkz"	R0 = 0	R0 = 0	Pass
“dog”	“bagkasido”	R0 = 1	R0 = 1	Pass
“house”	“tysajfhue”	R0 = 0	R0 = 0	Pass
“cat”	“catcatcat”	R0 = 1	R0 = 1	Pass
“a”	“zxcvbnkla”	R0 = 1	R0 = 1	Pass

The program passed all of the tests and is able to consistently determine if a word can be made from a string of characters or not.

3 Lottery

The final part of the assignment was to design and write a program that would check a number of imaginary lottery tickets against the actual numbers that were drawn for this lottery. The program has access to the number of lottery tickets and there is also three spaces made in memory for the program to save the number of tickets that matched four, five and six numbers with the draw, respectively.

The lottery numbers are stored as byte-sized values (they are not stored as ASCII characters) and the tickets are stored consecutively in memory, meaning a method of counting must be used to keep track of which ticket is currently being checked and which number on that ticket is being checked.

The program takes a simple approach and uses a number of while loops which check the draw numbers against the six numbers of the ticket. When a match is found, the match count is incremented. After all six numbers from the ticket have been checked with the six numbers of the draw, the program checks the match count and determines if that ticket matched four, five or six numbers, then it increments the appropriate register. At the end of the program, the numbers in the match four, five and six registers are saved to memory as word-sized values.

The program was tested in a number of different ways, for instance increasing the number of tickets, trying with no matches what so ever, trying with only matches in one register, and so on. The program performed well in all cases and reported the correct number of lottery tickets which matched with 4, 5 or 6 of the numbers from the draw.

The pseudocode for the lottery part of this assignment can be viewed on the next page.

```
adrTickets = adrTickets
adrCount = COUNT
adrDraw = DRAW
count = Memory.word[adrCount]

match4 = 0
match5 = 0
match6 = 0

ticketNumb = 0

while (ticketNumb < count) {
  matchCount = 0
  index = 0
  while (index < 6) {
    drawIndex = 0
    ticketElem = Memory.byte[adrTickets]
    while (drawIndex < 6) {
      drawNumb = Memory.byte[adrDraw]
      if (ticketElem == drawNumb) {
        matchCount++
      }
      adrDraw + 1
      drawIndex++
    }
    adrTickets + 1
    index++
    adrDraw - 6
  }
  if (matchCount == 4) {
    match4++
  } else if (matchCount == 5) {
    match5++
  } else if (matchCount == 6) {
    match6++
  }
  ticketNumb++
}

adrMATCH4 = MATCH4;
adrMATCH5 = MATCH5;
adrMATCH6 = MATCH6;

Memory.word[adrMATCH4] = match4;
Memory.word[adrMATCH5] = match5;
Memory.word[adrMATCH6] = match6;
```