CS1022 SUDOKU

Student name: Patrick Dillon Ryan

Student Number: 17340382

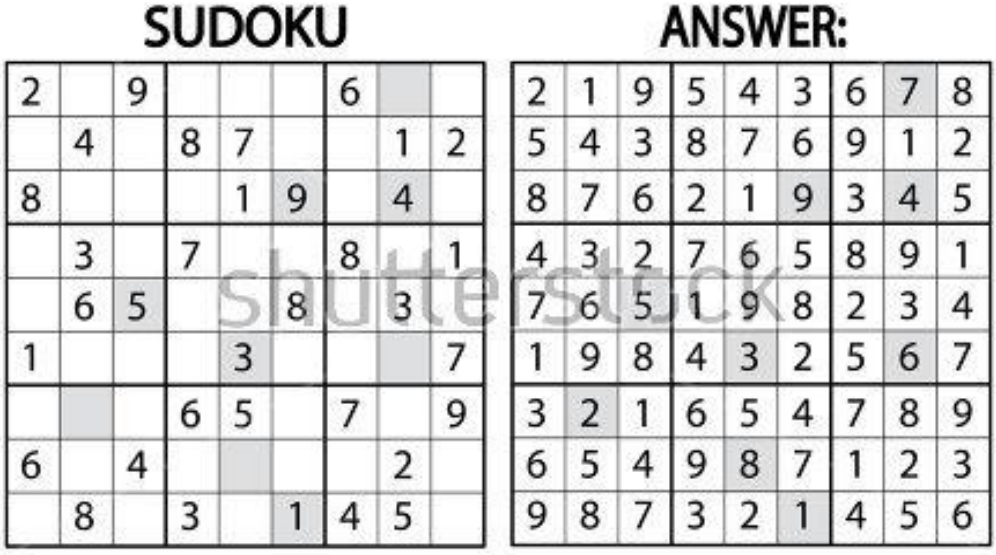
INTRODUCTION

The aim of this assignment was to build a Sudoku solver. The assignment was split into three sections which were three separate subroutines.

1. Getting and setting digits
2. Validating solutions
3. Solving a Sudoku puzzle

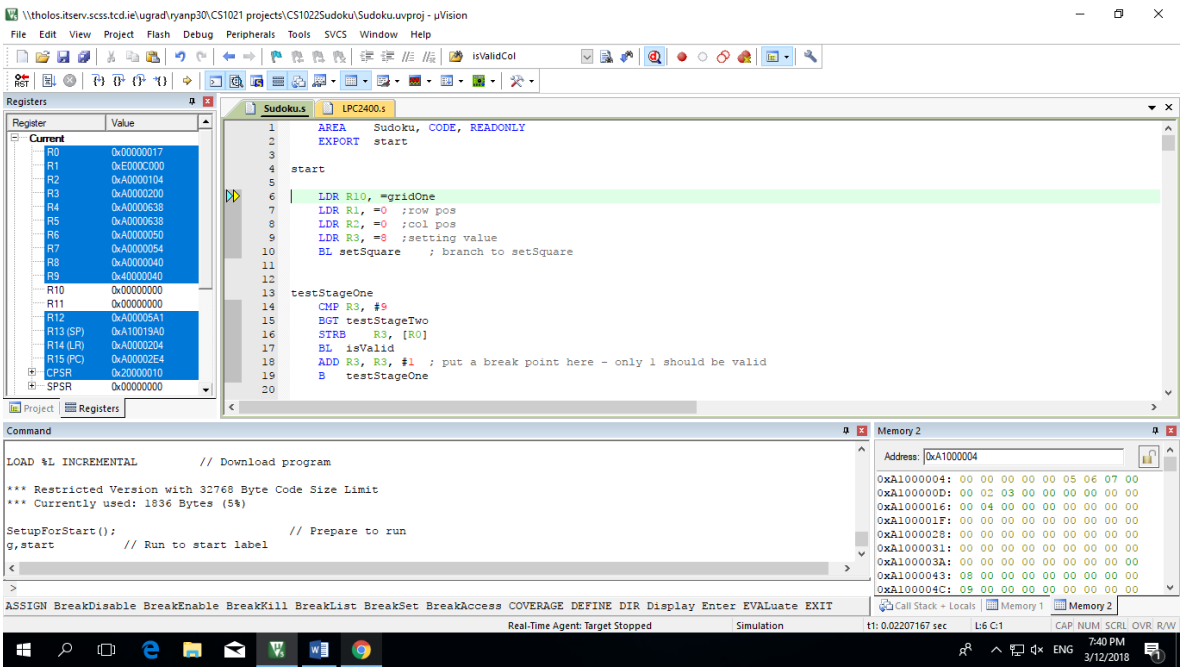
THE GAME OF SUDOKU

The game of sudoku is a logic based, combinational number-placement puzzle. The aim of the game is to fill in a 9 row by 9 column grid with digits so that each row, column and each of the 3X3 grids contain all of the digits from 1 to 9. The image below shows an example of a given 9X9 grid (unsolved) and then to the right of it the completed sudoku grid.

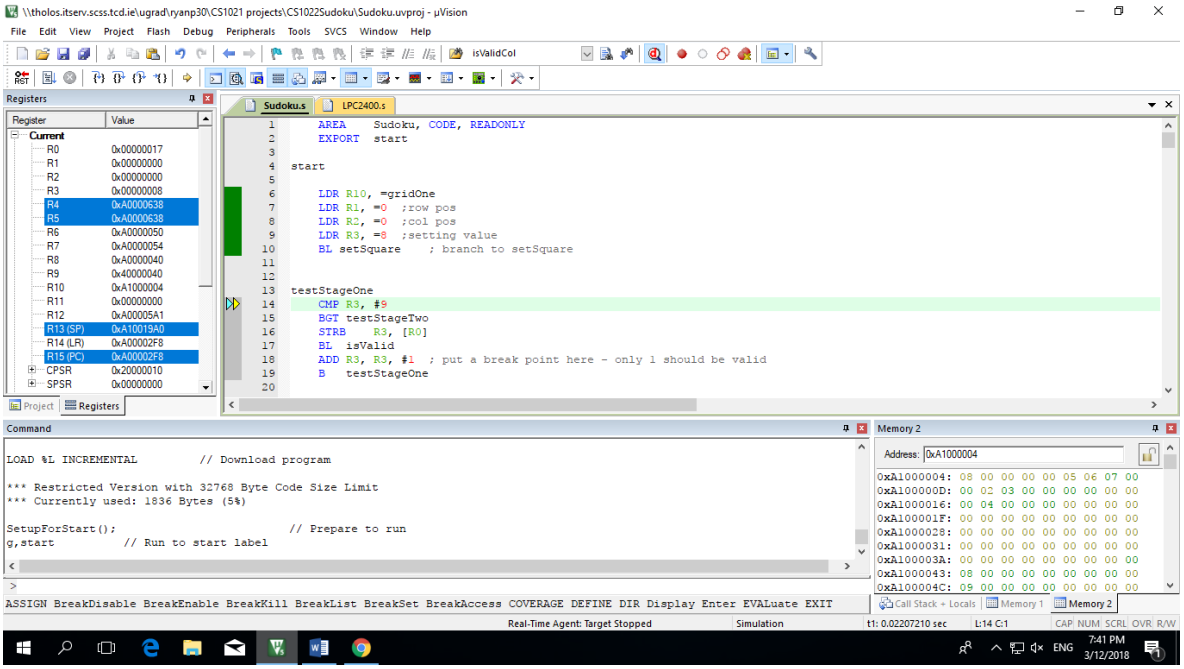


1. Getting and setting digits

The aim in this section is to write two subroutines, one subroutine to get the value of a digit in a given row and column and then a second subroutine which sets the value of a digit in the given row and column. The user input is to give a row and column position and the value which should go into that position. There were two subroutines wrote. The first one got the position where the value was to go. The second subroutine used that position and then placed the value in that position.



The 9 x 9 grid can be seen down the bottom right of the screen. The position that is selected by the user is 0,0 (Top left of the grid) the value that is to be passed into it is stored in R3 (8).



As seen in the top left corner of the 9 x 9 grid the value 8 has been passed into it.

Pseudocode

getSquare

- Push onto a full descending stack
- Row size = 9
- Index = row * row size
- Index = index + column
- R0 = grid position
- Pop from a full descending stack

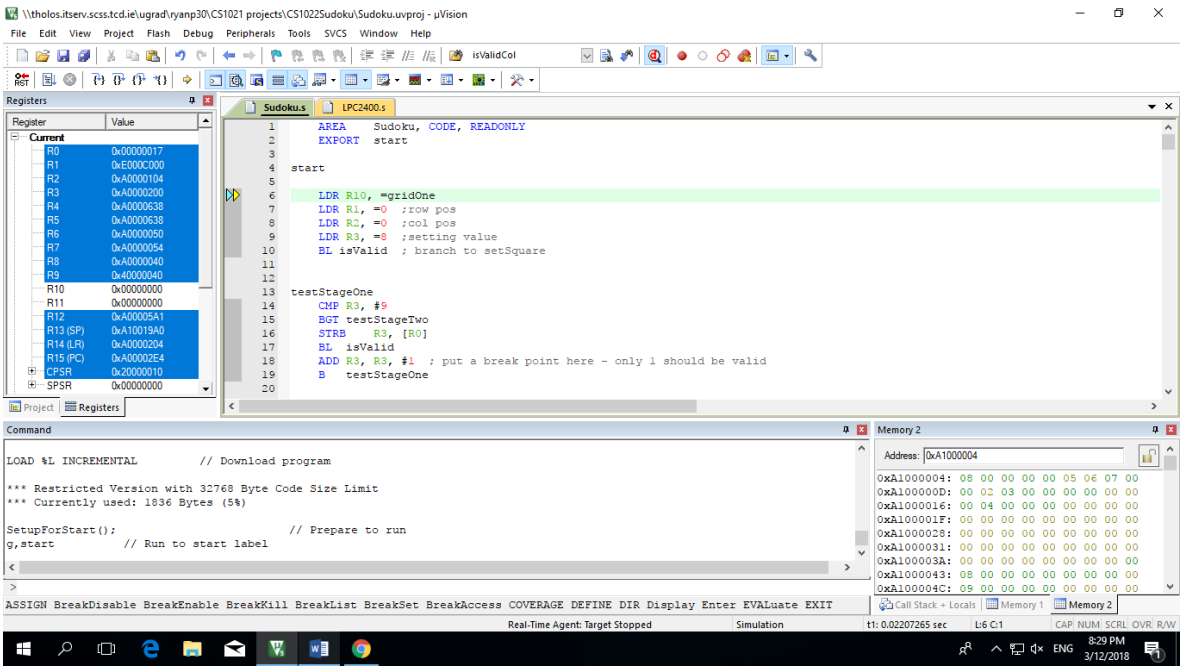
setSquare

- Push onto a full descending stack
- Row size = 9
- Index position = row * row size
- Index = index + column
- R3 = grid position
- Pop from a full descending stack

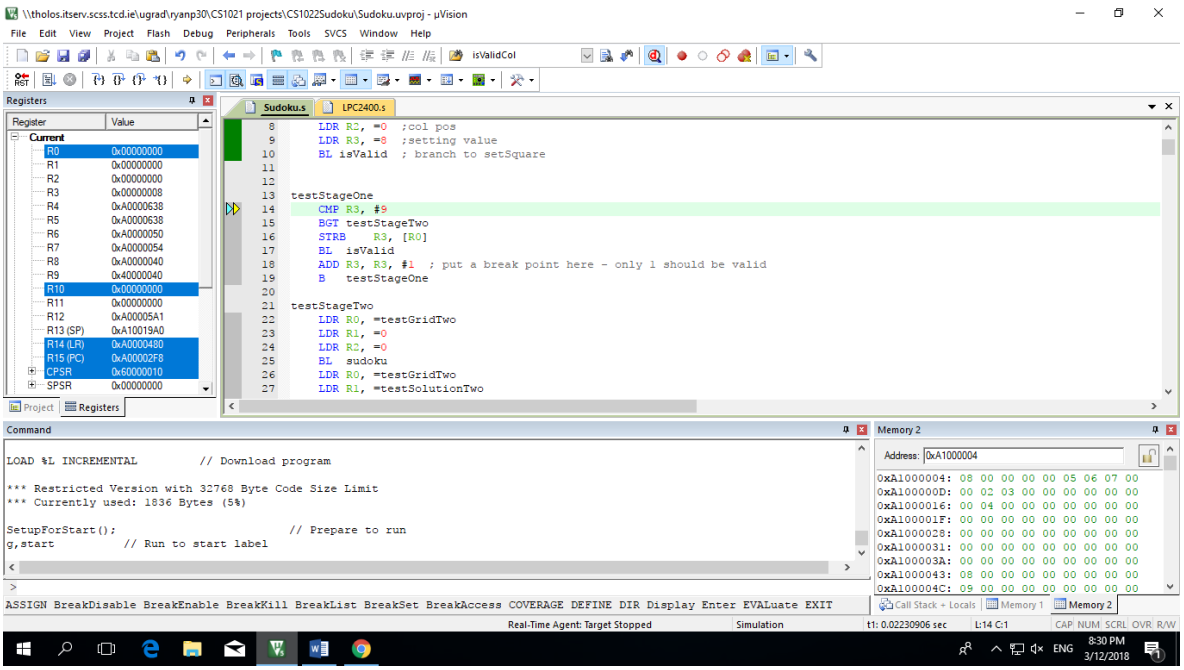
2. Validating solutions

The aim in this section is to write subroutines to check if a Sudoku grid represents a valid solution. The subroutine should return a Boolean result weather the solution is valid or not. The subroutine must also make use of the subroutine from part one to retrieve the digit from a given grid square.

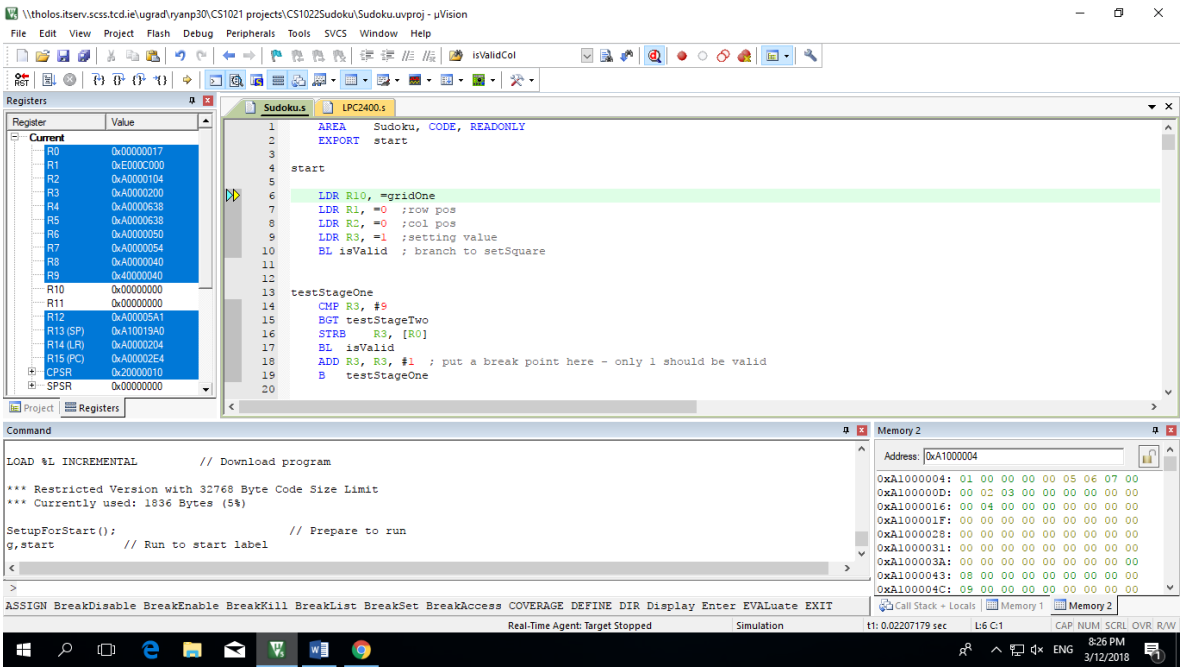
To test this code first I made R10 into a Boolean. 1 represented that the number is valid in that position and 0 represented that it was an invalid move.



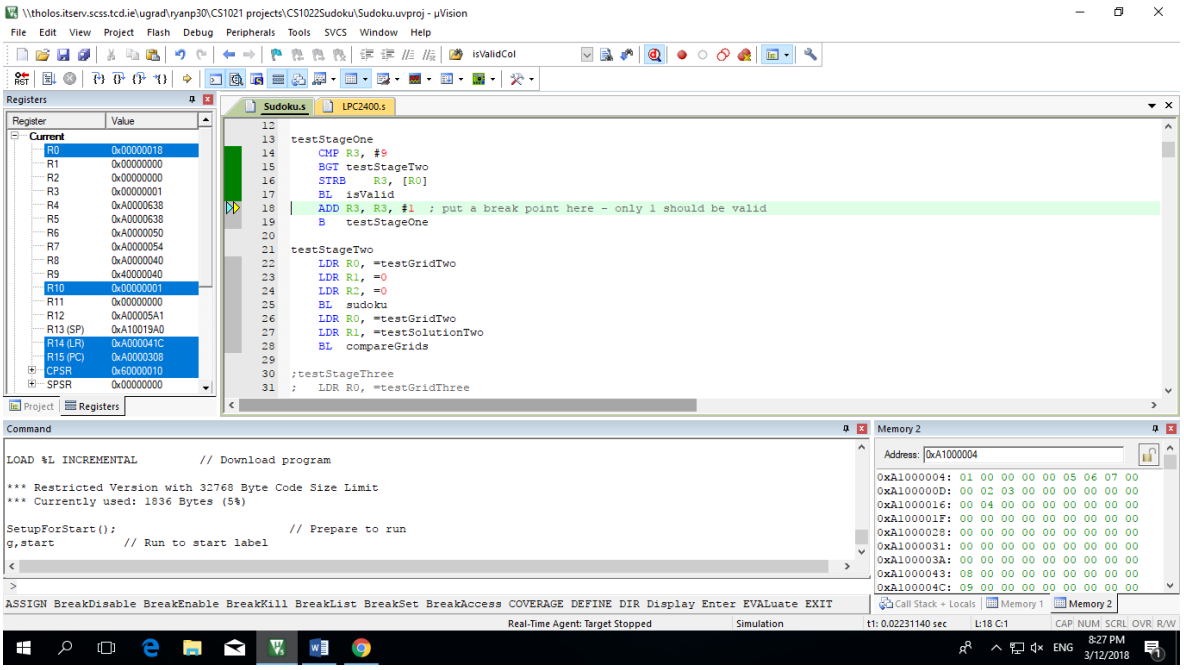
To test the code, the value 8 was passed into R3. If the code worked correctly it should pass the value 0 into R10 to show that it is an invalid move as the value 8 is already in the row.



As seen in R10 the value is 0 which represents an invalid move.



The program was retested with the value 1 stored in R3. The value 1 is a valid entry and should return the value 1 in R10 to show it's a valid input



Once the program has run the value in R10 turns to 1, this shows that 1 is a valid entry.

Pseudocode

```
isValid
        Push onto a full descending stack
        if(true)
        Pop from a full descending stack
invalid
        Pop from a full descending stack
validRow
        Push onto a full descending stack
        row=0
        9 for for loops
column=0
row
bhs
        set column back to 0
column
get row column R1, R2
        move row column into R7
        if value = 0
        columnCheck = column
        columnCheck = column + 1
columnCheck
        move columncheck into R2
        columnCheck = 9
        BHS forColumn
        get row columnCheck R1, R2
        move row columnCheck into R0
        Compare row column to row columnCheck
        if equal return 1 for false
        else increment columnCheck
forColumn
        column++
forRow
        row++
forRowEnd
        valid = true
        Pop from a full descending stack
endColumn
        valid = false
        Pop from a full descending stack
validColumn
        Push onto a full descending stack
        row=0
        9 for for loops
        column=0
row1
        BHS
        set column back to 0



column1
        get row column R1, R2
        move row column into R7
        if value = 0
        colCheck = column
        colCheck = column + 1
columnCheck1
        move columncheck into R2
        columnCheck = 9
get row columnCheck R1, R2
```

```
        move row columnCheck into R0
        Compare row column to row columnCheck
        if equal return 1 for false
        else columnCheck++

forColumn1
        column++
forRow1
        row++
forRowEnd1
        valid = true
        Pop from a full descending stack
endColumn1
        valid = false
        Pop from a full descending stack
```

3. Solving a Sudoku puzzle

The aim in this subroutine is to use pseudocode given to us to try and make a sudoku puzzle solver. A “brute force” approach can be adopted to solve the puzzle but iterating through the digits from one to nine.

The pseudocode which was given to us was translated into arm assembly language but I was unable to get it to work correctly. It wouldn’t solve the puzzle. I tried many different approaches but was unsuccessful in all my attempts.