Sudoku Assignment

Documentation

Name - Krishanu Dey

Student Number - 17306518

Date – 10th March 2017

1 Introduction

This report describes the development of my Sudoku solving program in ARM Assembly Language. My Sudoku program has row = 1 and column = 1 as the first square box. The program contains the following functions –

1. **int compareGrids (AddressSudoku1, AddressSudoku2)** – Provided by the Instructor. The function checks the sudoku solved by the program with the correct sudoku solution. If they match, the function returns 1 in R0, and otherwise it returns 0 in R0.

2. **int getSquare(AddressSudoku, row, column)** – The following function returns the number stored in that particular square in the sudoku (found using the rows and column passed as parameter in R1 and R2 resp.).

3. **void setSquare(AddressSudoku, row, column, number)** – This functions sets the number passed through R3 in that particular square in the sudoku (found using the rows and column passed as parameter in R1 and R2 resp.)

4. **boolean isValid(AddressSudoku, row, column)** – This function checks if the number in that particular square is already present in that row, column or

that particular mini-grid. The function returns 0 in R3 if it is already present in that row, column or mini-grid and if not, it returns 1 in R3.

5. **boolean sudoku (addressGrid, row, column)** – This function is responsible for finding the solution for the sudoku whose address was passed through R0. The approach taken by this algorithm was brute force.

6. **Void printSudoku(addressGrid, row, column)** – This function prints the solved sudoku on the console.

The approach taken by all the functions have been explained later with the pseudo code

2 Function Explanations

1. getSquare –

Approach

This function takes the row and the column numbers as parameters and returns the number present in that square. As the row and the columns in the program start at 1 and 1, I first subtracted 1 from the row and the column value. Then the program multiplies the row number (which was decreased by 1) with 9, and adds the column number – this way I found out the 1D Index of that square box.

The 1D Index is added to the addressGrid provided and the value stored in that address spot is acquired using LDR. This value is returned using R3, and program control goes back to the next line from which the function was called.

Pseudo code

```
Int getSquare(addressGrid, row, column){
     row = row - 1;
     column = column - 1;
     tmpRow = row * 9
     Index1D = tmpRow + column
     number = addressGrid[Index1D]
    return number;
}
```

Table for testing various Inputs

Input		Output
ARM	**Pseudo Code**	
LDR R0, = testGridTwo LDR R1, =2 LDR R2, =1 BL getSquare	getSquare(testGridTwo, 2, 1)	3
LDR R0, = testGridTwo LDR R1, =5 LDR R2, =5 BL getSquare	getSquare(testGridTwo, 5, 5)	0
LDR R0, = testGridThree LDR R1, =2 LDR R2, =3 BL getSquare	getSquare(testGridThree, 2, 3)	3

2. setSquare –

Approach

The approach taken for this function is similar to the one for getSquare, but instead of LDR, I used STR here to store the passed number in the square.

This function takes the row and the column numbers as parameters and returns the number present in that square. As the row and the columns in the program start at 1 and 1, I first subtracted 1 from the row and the column value. Then the program multiplies the row number (which was decreased by 1) with 9, and adds the column number – this way I found out the 1D Index of that square box.

The 1D Index is added to the addressGrid provided and the number value passed as parameter through R3 is stored in that address spot using STR. Afterward, the control goes back to the next line from which the function was called.

Pseudo code

```
void setSquare(addressGrid, row, column, number){
    row = row - 1;
    column = column - 1;
    tmpRow = row * 9;
    Index1D = TmpRow + column;
    addressGrid[Index1D] = number;
}
```

Table for testing various Inputs

Input		Output
ARM	**Pseudo Code**	
LDR R0, = testGridTwo LDR R1, =5 LDR R2, =1 LDR R3, =7 BL setSquare	setSquare(testGridTwo, 5, 1, 7)	0 2 7 6 **5** 0 0 0 3 3 0 0 0 0 9 0 0 0 8 0 0 0 4 0 5 0 0 6 0 0 0 0 2 0 4 0 **7** 0 2 0 0 0 8 0 0 0 4 0 7 0 0 0 0 1 0 0 3 0 1 0 0 0 7 0 0 0 8 0 0 0 0 9 9 **7** 0 0 0 6 2 8 0
LDR R0, = testGridTwo LDR R1, =9 LDR R2, =2 LDR R3, =7 BL setSquare	setSquare(testGridTwo, 9, 2, 7)	
LDR R0, = testGridThree	setSquare(testGridTwo, 1, 5, 5)	

LDR R1, =1 LDR R2, =5 LDR R3, =5 BL setSquare		

3. **boolean isValid(AddressSudoku, row, column)** –

   Approach

   The function checks if the number in the index provided can be placed there or not by considering the rules of Sudoku in that particular row, column and mini-grid.

   From lines 131 to 151, the function checks if the number is already used in the row or not. It works using a for loop that goes through the elements in the row one by one. The function returns false if the number is already used in the row. If the number isn't already used, the program goes on to check the column and the grid.

   From lines 154 to 177, the function checks if the number is already used in the column or not. It goes through the elements of the column one by one. The function returns false if the number is already used in the column. If the number isn't already used, the program goes on to check the mini-grid.

   From lines 178 to 230, the function checks if the number is already used in the grid or not. This is done by finding the first element of that specific mini grid (using i = row – (row % 3) and j = column – (column % 3)). Then the program goes through each of the elements in the grid one by one. The function returns false if the number is already used in the mini-grid. If the number isn't already used, the program sets bool 'valid' to true and returns 'valid'.

Table for testing various Inputs

Input		Output
ARM	**Pseudo Code**	
LDR R0, = testGridTwo LDR R1, =1 LDR R2, =1 BL isValid	isValid(testGridTwo, 1, 1, 1)	**R0 = 1**
LDR R0, = testGridTwo LDR R1, =5 LDR R2, =1 BL isValid	isValid(testGridTwo, 1, 5, 7)	**R0 = 1**
LDR R0, = testGridTwo LDR R1, =1 LDR R2, =5 BL isValid	isValid(testGridTwo, 1, 5, 7)	**R0 = 0**

Pseudo code

```
boolean isValid(AddressSudoku, row, column){
   x = sudoku[row][column];

   //Is it already used in row.
   int jj=1
   for (; jj <= 9; ) {
     if( jj == column)
     {
        jj++
      }
      number = getSquare(i, jj)
      if (sudoku[i][jj] == x)
      {
         valid = false;
         return valid;
       }
     jj++
}

// Is 'x' used in column
int ii = 1
for (; ii <= 9; ) {
   if(ii == Row)
   {
       ii++
    }
    number = getSquare(i, jj)
    if (sudoku[ii][j] == x)
    {
       return false
    }
    ii++
}

;// Is 'x' used in sudoku 3x3 box.
int boxRow = i - i % 3;
int boxColumn = j - j % 3;

for ( ii=0 ; ii < 3; )
{
   for ( jj=0 ; jj < 3;)
   {
        tmp1 = boxRow + ii;
        tmp2 = boxColumn + jj;
        if(tmp1 == row && tmp2 == column)
            column++;
         number = sudoku[tmp1][tmp2];        // load the number
         if (number == x)
          {
              valid = false;
          }
         jj++;
        }
        ii++
    }
    valid = true;
    return valid;
}
```

4. boolean sudoku(addressGrid, row, column)

Approach

This function takes the brute force method to solve the Sudoku. It tries out each possibility using recursive function procedure. It sets the blank square with a number and if it is valid (checked using isValid), it goes on and sets the next blank number and if it is valid, it goes on to the next blank number, but if the number isn't valid, it will increment the number in the previous square(which was blank before). This goes on till the last blank square Sudoku function returns true.

Table for testing various Inputs

Input		Output
ARM	**Pseudo Code**	

LDR R0, = testGridTwo ;
LDR R1, =1
LDR R2, =1
BL printSudoku
	sudoku(**testGridTwo**, 1, 1)
	0 2 7 6 0 0 0 0 3
	3 0 0 0 0 9 0 0 0
	8 0 0 0 4 0 5 0 0
	6 0 0 0 0 2 0 4 0
	0 0 2 0 0 0 8 0 0
	0 4 0 7 0 0 0 0 1
	0 0 3 0 1 0 0 0 7
	0 0 0 8 0 0 0 0 9
	9 0 0 0 0 6 2 8 0
		1 2 7 6 5 8 4 9 3
		3 5 4 2 7 9 1 6 8
		8 9 6 3 4 1 5 7 2
		6 3 9 1 8 2 7 4 5
		7 1 2 4 9 5 8 3 6
		5 4 8 7 6 3 9 2 1
		2 8 3 9 1 4 6 5 7
		4 6 5 8 2 7 3 1 9
		8 7 1 5 3 6 2 8 4

LDR R0, = someInvalidGrid;
LDR R1, =1
LDR R2, =1
BL printSudoku
	sudoku(**someInvalidGrid** , 1, 1)
	3 2 7 6 4 0 0 0 3
	3 0 0 0 4 9 0 0 0
	8 0 0 0 4 0 5 0 0
	6 0 0 0 4 2 0 4 0
	0 0 2 0 0 0 8 0 0
	0 4 0 7 0 0 0 0 1
	0 0 3 0 1 0 0 0 7
	0 0 0 8 0 0 0 0 9
	9 0 0 0 0 6 2 8 0
		3 2 7 6 4 0 0 0 3
		3 0 0 0 4 9 0 0 0
		8 0 0 0 4 0 5 0 0
		6 0 0 0 4 2 0 4 0
		0 0 2 0 0 0 8 0 0
		0 4 0 7 0 0 0 0 1
		0 0 3 0 1 0 0 0 7
		0 0 0 8 0 0 0 0 9
		9 0 0 0 0 6 2 8 0

Pseudo code(provided by instructor)

```
bool sudoku ( addressgrid , wordrow , wordcol )
{
    boolresult = false;
    word nxtcol ;
    word nxtrow ;
    // Precompute next row and column
    nxtcol = col + 1 ;
    nxtrow = row ;
    if ( nxtcol > 8 ) {
       nxtcol = 0 ;
       nxtrow++;
    }
     if ( getSquare (grid , row , col) != 0 ) {
       // a pre−filled square
       if ( row == 8 && col == 8 ) {
          // last square −− success ! !
          return true ;
        }
        else {
           // nothing to do here – just move on to the next square
           result = sudoku ( grid , nxtrow , nxtcol ) ;
        }
```

```
            }
           else {
               // a blank square – try filling it with 1 . . . 9
              for ( byte try = 1 ; try <= 9 && ! result ; try++) {
                  setSquare ( grid , row , col , try ) ;
                  if ( isValid ( grid , row , col ) ) {
                     // putting the value here works so far . . .
                     if ( row == 8 && col == 8 ) {
                         // . . . lastsquare –– success! !
                         result = true ;
                     } else {
                        // . . . move on to the next square
                       result = sudoku ( grid , nxtrow , nxtcol ) ;
                     }
                  }
              }
              if ( !result ) {
                  // made an earlier mistake – back track by setting
                 // the current square back to zero / blank
                 setSquare ( grid , row , col , 0 ) ;
            }
       }
       return result;
   }
```

5. void printSudoku (addressGrid, row, column) – Extra Mile

Approach

This function prints the sudoku solved by the program on the console. It takes the row number and the column number as parameter in R1 and R2 respectively, and the address of the sudoku as parameter in R0.

Table for testing various Inputs

Input	Output

; printSudoku(testGridTwo,1,1)
LDR R0, = testGridTwo ;
LDR R1, =1
LDR R2, =1
BL printSudoku
	1 2 7 6 5 8 4 9 3
	3 5 4 2 7 9 1 6 8
	8 9 6 3 4 1 5 7 2
	6 3 9 1 8 2 7 4 5
	7 1 2 4 9 5 8 3 6
	5 4 8 7 6 3 9 2 1
	2 8 3 9 1 4 6 5 7
	4 6 5 8 2 7 3 1 9
	8 7 1 5 3 6 2 8 4

Pseudo code

```
void printSudoku(addressGrid, row, column)
{
    for(; row < 9; row++)
    {
        for(; column < 9; column++)
                systemOut.print(sudoku[row][column] + “ ”);
        systemOut.println();
    }
}
```

6. void printString (addressString) – Extra Mile

Approach

This function prints the string whose address has been passed as parameter in R1. It gets the ASCII value from the address provided and the for loop ends when the loop encounters null.

Table for testing various Inputs

Input		Output
ARM	Pseudo Code	
LDR R1, =Sudoku1String BL printString Sudoku1String DCB 83, 117, 100, 111, 107, 117, 32, 49, 0	System.out.println("Sudoku 1")	Sudoku 1
LDR R1, =Sudoku2String BL printString Sudoku2String DCB 83, 117, 100, 111, 107, 117, 32, 50, 0	System.out.println("Sudoku 2")	Sudoku 2

Pseudo code

```
void printString(addressString)
{
    for(int i=0; string[i]!=null; i++)
    {
            systemOut.print( string[i] );
    }
}
```

3 Extra Mile

The program prints out the solved the Sudoku using the function printSudoku(). It also titles the sudokus with “Sudoku” + sudoku number using the function printString().

1. printSudoku().

2. printString().

Please check function explanation for more details

4 Conclusion

This report explains what the program does, and then gives the explanation to the approach to the problem in a simplified manner.

The ‘Pseudo Code’ and the ‘Table for various Input and their Outputs’ are also provided.

All the functions have been tested with various inputs, and the outputs have been recorded in this report.