Mid-Term Assignment - Connect 4 - Laboratory Documentation

15th March 2019

Name: John Keaney

Student ID: 18328855

10/31/2018

"I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write."

-John Keaney

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

Mid-Term Individual Assignment - Connect 4

Goal:

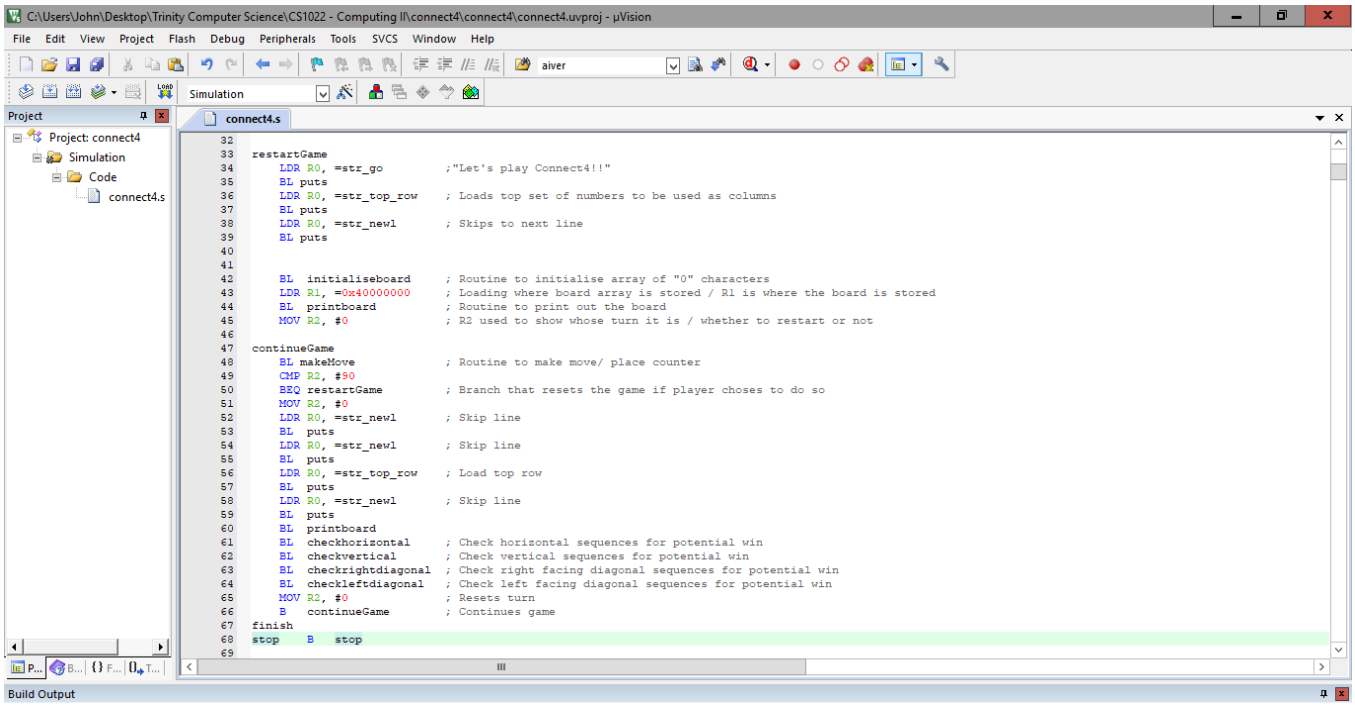
To create Connect 4 in Keil uVision5; a two-player puzzle game in which players take turns to drop discs into a vertically mounted board with seven columns and six rows. The program should check whether a player has won or not (i.e. 4 or more counters in a row). The program should also include an A.I. component that plays against the user.

Table of Contents:

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

```
// Main Pseudo Code

boolean finished = false;
boolean restartGame = false;

initialiseBoard (address, turn count)
// Miscellanious "puts"
printBoard (address, turn count)

if (restartGame == true)
{
        initialiseBoard (address, turn count)
        // Miscellanious "puts"
        printBoard (address, turn count)
        restartGame = false;
}

for (int i = 0; i < 42; i++)
{
        makeMove (address, turn count) / AIMove
        if (put == "q")
        {
                restartGame = true;
        }
        // Miscellanious "puts"
        printBoard (address, turn count)
        checkHorizontal (address, turn count)
        checkVertical (address, turn count)
        checkRightDiagonal (address, turn count)
        checkLeftDiagonal (address, turn count)
}
```

Explanation Of Main:



You can see from the pseudo-code that the main is simple enough. Essentially it can be boiled down to:

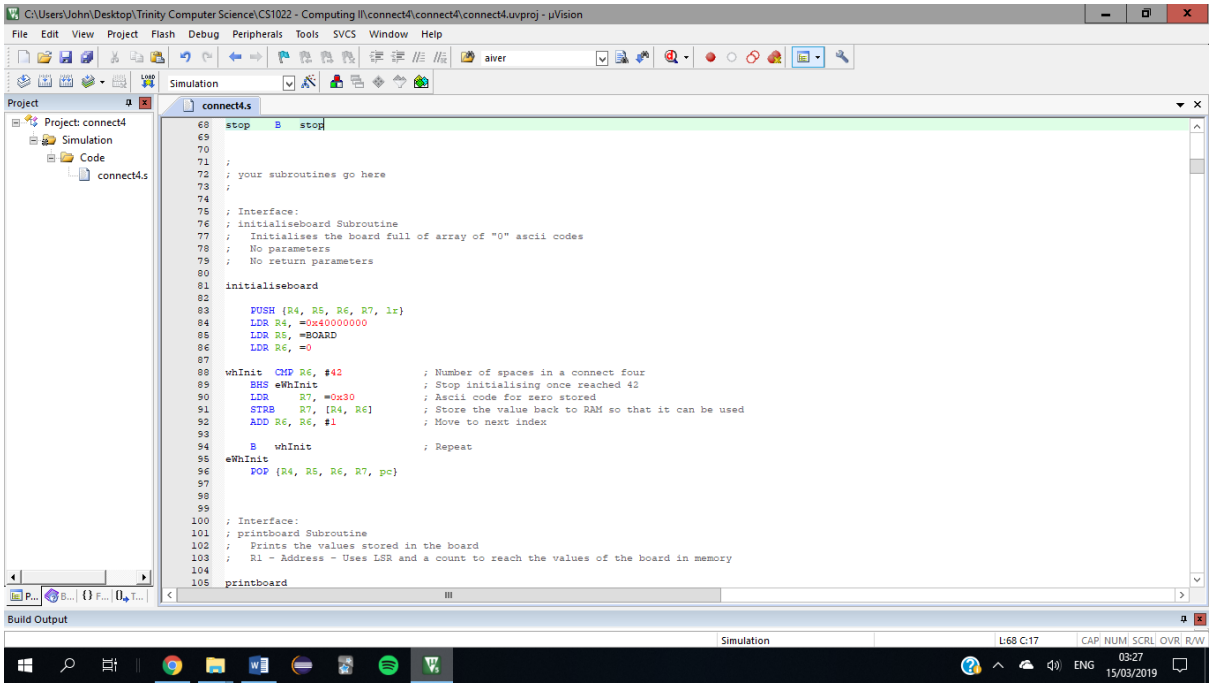
> “ A game is played 42 times, each time making one move, using the Branch Link instruction to perform various methods. The R1 and R2 registers are used to pass parameters. R1 = Array Address, R2 = Turn Count.”

Little is need to be said about test with regards to this. We will see whether this is working based on the other subroutines.

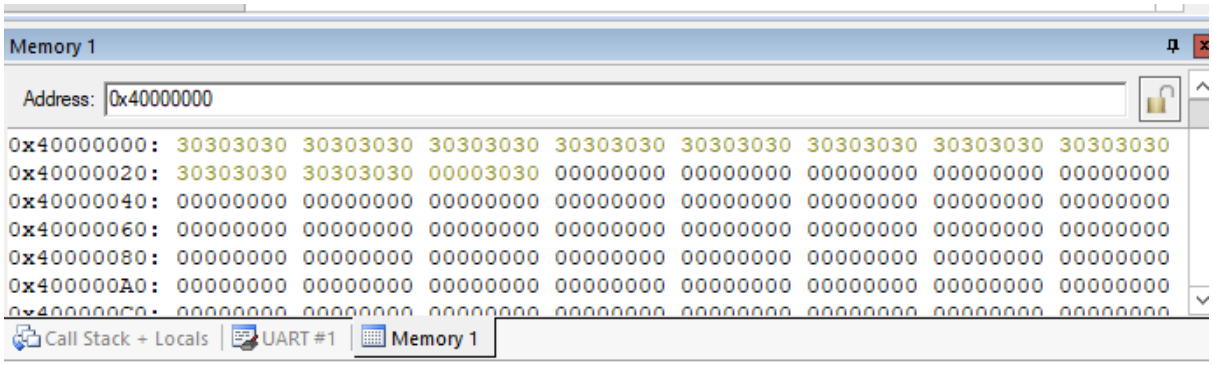
Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

```
// Initialise Board Pseudo Code

for (int i = 0; i < 42; i++)
{
        String 0 = "0"
        board [i] = 0;
}
```

Explanation Of Initialise Board Routine:



Like the main this routine is simple enough. The board array is stored at the address =0x40000000. Initialising the board fills it full of the ascii code for “0”; 0x30. As such we should see an array of the board beginning at the address full of “30”’s 42 times. R6 = The count. We do this by loading the #0x30 into R7 and then storing the byte of R7 into an assigned address. The assigned address is just the original address which is immediately offset by the count. This is seen in line 91 with the instruction “LDRB R7, [R4, R6]”. So after testing this routine we are left with:

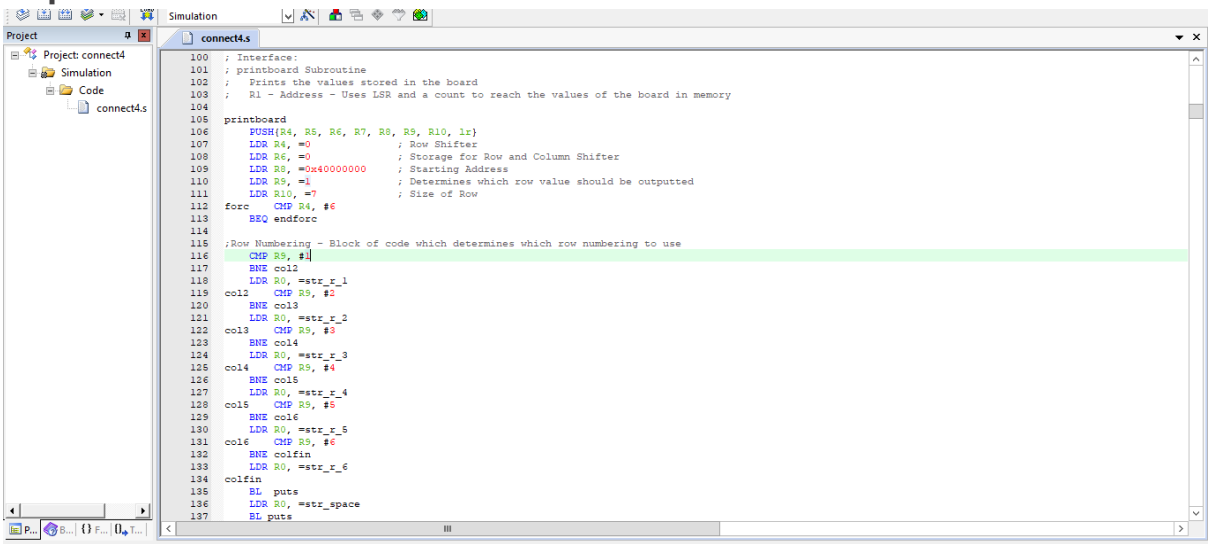


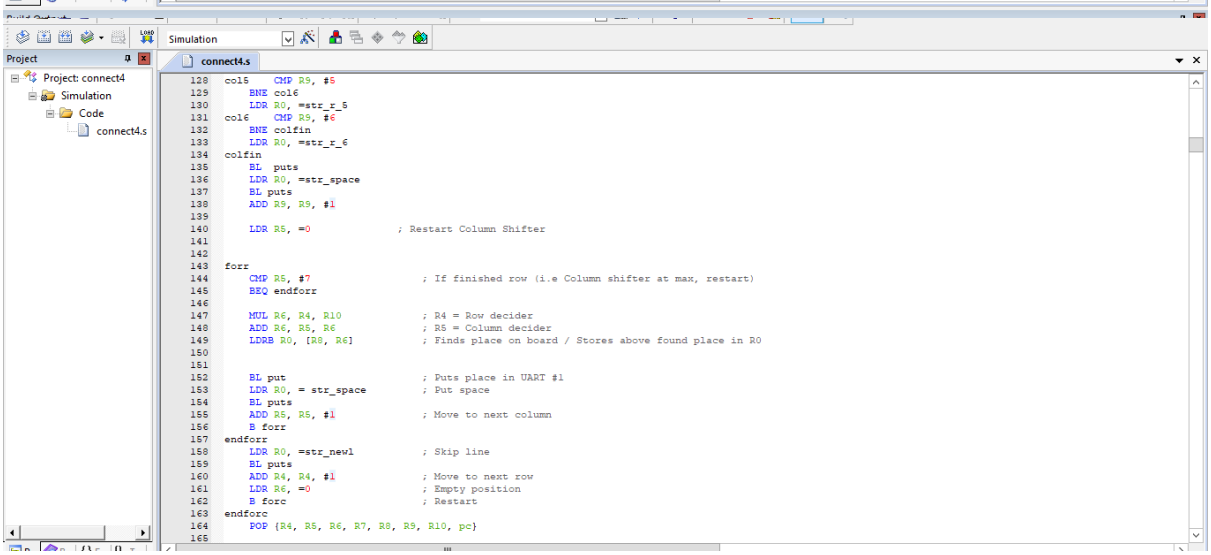
As such the routine can clearly be seen working.

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

```
// Print Board Pseudo Code

int columnNumber = 0; //R9
int rowShifter = 0; // R4
int columnShifter = 0; // R6
// R8 = Starting Address


for (int R4 = 0; R4 < 6; R4++)
{
        for (int R6 = 0; R6 < 6; R6++)
        {
                // Load position = board [R4][R6];
                System.out.print(position + " ");
                // Miscellaneous puts for skip row etc.
        }
        System.out.print(columnNumber + "  ");
        columnNumber++;
}
```

Explanation Of Print Board Routine:





From the first screenshot we see a block of code whose entire purpose is simply just to distance the row numbers to the left hand side using a count and puts operations.

The main aspect of this subroutine can be seen in the second block of code. Something which can be seen clearly in the pseudo code above is that the this routine consists of two for loops; the first representing the row number (or the forr/endforr label) , the second the column number (or the forc/endforc label).
To explain via assembly, both pieces of code use their indexes, or in this case R4/ R5 respectively to reach the next position of the array and load the byte stored at that position and then "put" it. They reach their positions by using a simple formula:

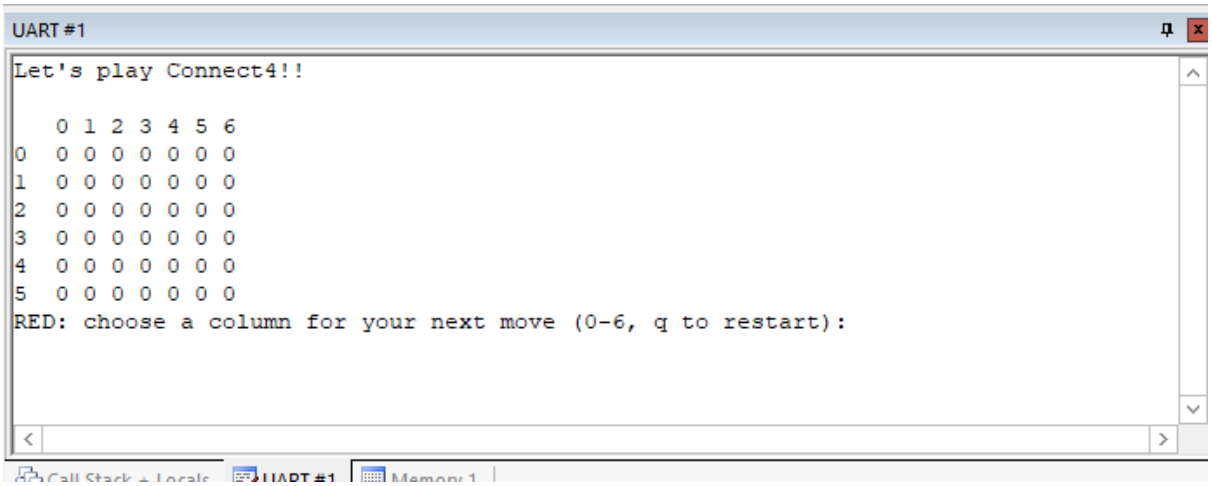
*"position = rowNumber * sizeOfRow"*
"position = position + columnNumber"
"load the byte of the (beginning of the array shifted by the position"

Or as I have written in the program:

MUL R6, R4, "Register with size of row"
ADD R6, R6, R5
LDRB R0, [beginning address, R6]

It is important to remember the above code as it is used plenty of times in the program and for convenience I have called it an array place finder.

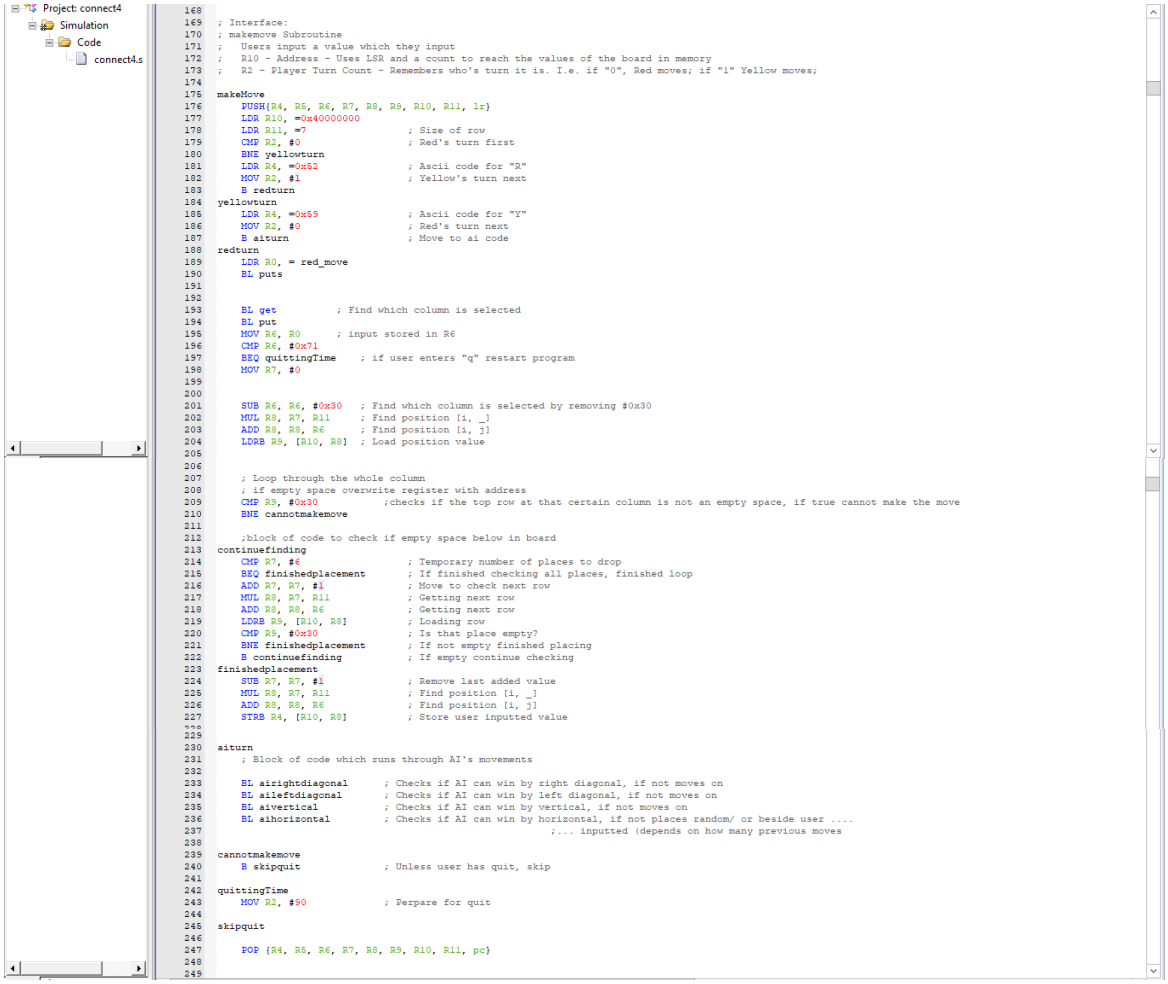
Using immediate offset the value is then placed in the R0 register and then using the "put" routine is placed into the UART#1 window. We can see that this successfully works from the screenshot**:**



Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

```
// Make Move Pseudo Code

int columnShifter = 0; // R6
// R10 = Starting Address
int rowSize = 7 // R11
boolean redTurn = true; // R2
String "R" = 0x52 // String "Y" = 0x59 // R4
boolean finished = false;
if (!finished)
{
        if (redTurn = true)
                String "R" = 0x52
        else
                String "Y" = 0x59
        // BL get input
        if(input = "q")
        finished = true;
        int index = BL get;
        // load byte of [address, board [index]]

        // Drop Down Pseudo Code
                for(int i = 0; i<6; i++)
                {
                if (board [index/ ("columnNumber")][i] == 0x30)
                        board [index/ ("columnNumber")][i] = "R"/"Y"
                        for(int i2 = i; i2>-1; i2--)
                        {
                                board [index/ ("columnNumber")][i2] = 0x30
                        }
                }
}
```

Explanation of makeMove/ dropDown Routines:




```
; Interface:
; makemove Subroutine
;   Users input a value which they input
;   R10 - Address - Uses LSR and a count to reach the values of the board in memory
;   R2 - Player Turn Count - Remembers who's turn it is. I.e. if "0", Red moves; if "1" Yellow moves;

makeMove
    PUSH{R4, R5, R6, R7, R8, R9, R10, R11, lr}
    LDR R10, =0x40000000
    LDR R11, =7                 ; Size of row
    CMP R2, #0                  ; Red's turn first
    BNE yellowturn
    LDR R4, =0x52               ; Ascii code for "R"
    MOV R2, #1                  ; Yellow's turn next
    B redturn
yellowturn
    LDR R4, =0x59               ; Ascii code for "Y"
    MOV R2, #0                  ; Red's turn next
    B aiturn                    ; Move to ai code
redturn
    LDR R0, = red_move
    BL puts


    BL get          ; Find which column is selected
    BL put
    MOV R6, R0      ; input stored in R6
    CMP R6, #0x71
    BEQ quittingTime    ; if user enters "q" restart program
    MOV R7, #0


    SUB R6, R6, #0x30   ; Find which column is selected by removing #0x30
    MUL R8, R7, R11     ; Find position [i, _]
    ADD R8, R8, R6      ; Find position [i, j]
    LDRB R9, [R10, R8]  ; Load position value


    ; Loop through the whole column
    ; if empty space overwrite register with address
    CMP R9, #0x30           ;checks if the top row at that certain column is not an empty space, if true cannot make the move
    BNE cannotmakemove

    ;block of code to check if empty space below in board
continuefinding
    CMP R7, #6              ; Temporary number of places to drop
    BEQ finishedplacement   ; If finished checking all places, finished loop
    ADD R7, R7, #1          ; Move to check next row
    MUL R8, R7, R11         ; Getting next row
    ADD R8, R8, R6          ; Getting next row
    LDRB R9, [R10, R8]      ; Loading row
    CMP R9, #0x30           ; Is that place empty?
    BNE finishedplacement   ; If not empty finished placing
    B continuefinding       ; If empty continue checking
finishedplacement
    SUB R7, R7, #1          ; Remove last added value
    MUL R8, R7, R11         ; Find position [i, _]
    ADD R8, R8, R6          ; Find position [i, j]
    STRB R4, [R10, R8]      ; Store user inputted value


aiturn
    ; Block of code which runs through AI's movements

    BL airightdiagonal      ; Checks if AI can win by right diagonal, if not moves on
    BL aileftdiagonal       ; Checks if AI can win by left diagonal, if not moves on
    BL aivertical           ; Checks if AI can win by vertical, if not moves on
    BL aihorizontal         ; Checks if AI can win by horizontal, if not places random/ or beside user ....
                                ;... inputted (depends on how many previous moves

cannotmakemove
    B skipquit              ; Unless user has quit, skip

quittingTime
    MOV R2, #90             ; Perpare for quit

skipquit

    POP {R4, R5, R6, R7, R8, R9, R10, R11, pc}
```

This routine consists of two separate methods; a makeAMove function and a dropDown function. The purpose of the first is to determine the player and then take the input. The drop

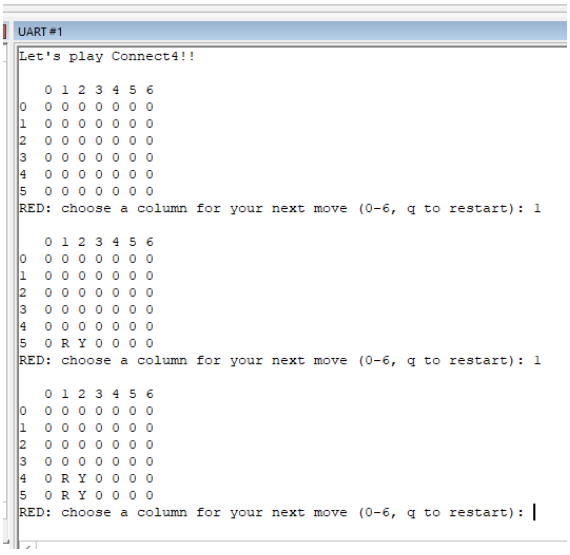
down then places the input in the correct column and continues down the rows until it reaches a non “0” and places above.

In assembly, the code uses R2, based on either #0 or #1 whose turn it is. If R2 is #1 R is swapped for Y as the byte to be stored in the array.

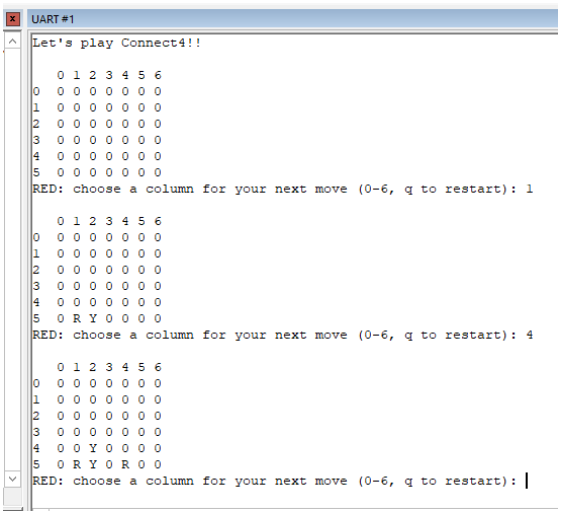
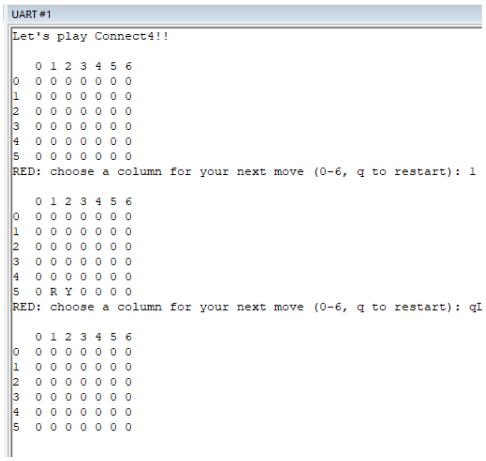
Then using BL get and BL put I grab the input. If the user inputs “q” / #0x71 the routine ends and R2 is stored with a value that will cancel the main and restart the program using a BEQ. Otherwise if a number is entered I then subtract #0x30 from the input to convert it to decimal. Then using the array position finder from above I find that position in the first row of the array and check if its empty. If it is it continues to check below until it either reaches the end or it reaches a value other than #0x30. This is performed by the dropDown section of my code. Which performs a for loop to continues to stay in the original column but increments by one row each time, ADD R7, R7, #1 at the end of each check.

Once the red operation is complete, a MOV R2, #1 is performed and then using a BEQ I perform the AI’s movement. However I’ll get to that later in the document. Once the AI’s routines are complete it’s red’s turn again.

To test my routine I preformed the following tests:

Inputs	Visual Examples
1, 1 This input demonstrates not only that the input works correctly, i.e. places in the correct column but also that the drop down works correctly. The two “R”’s can be seen placed on top of each other.	

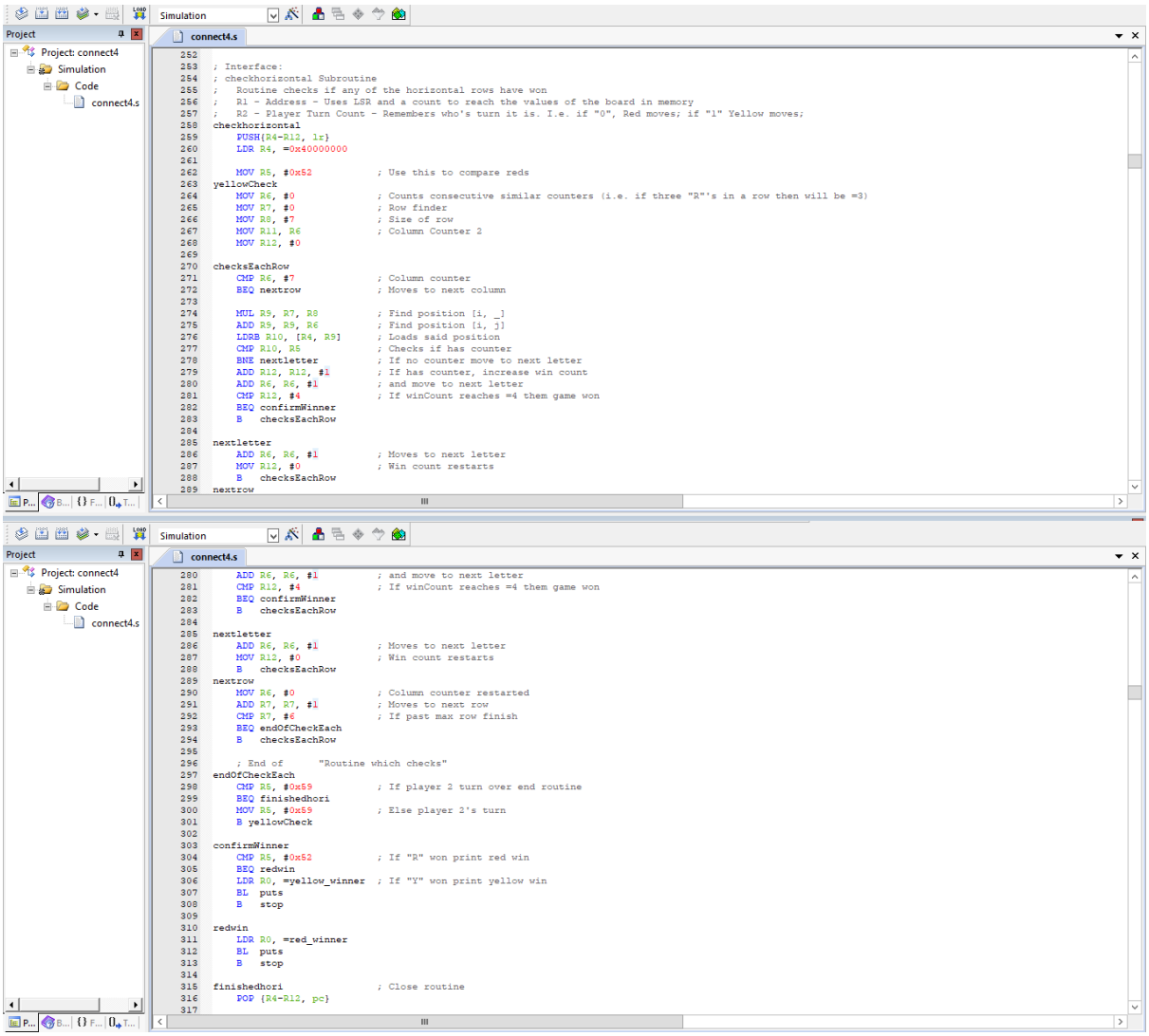
Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

1,4 Like the input above, these inputs show that the input places the “R”/”Y” in the correct column. After entering 1 the R is placed in the the first column and continues to move down till it reaches the bottom. It does the same for the input of 4.	
1, “q” The board is initialised at the beginning and then an R and Y is placed. Then after entering “q” the board is cleared by re-initialising it.	

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

```
// Check Horizontal/Vertical/etc. Pseudo Code

// Switches between comparing R / Y by using CMP with R5

for (int i = 0; i < 5; i++)
{
        for (int i2 = 0; i2 < 6; i2++)
        {
                if (board [i][i2] == Y / R / R5
                {
                        winCount++;
                        if (winCount == 4)
                        {
                                System.out.print ("User wins"/"Comp wins")
                                B stop
                        }
                }
        }
        winCount = 0;
}
```

Explanations of Horizontal/Vertical/Left Diagonal/ Right Diagonal



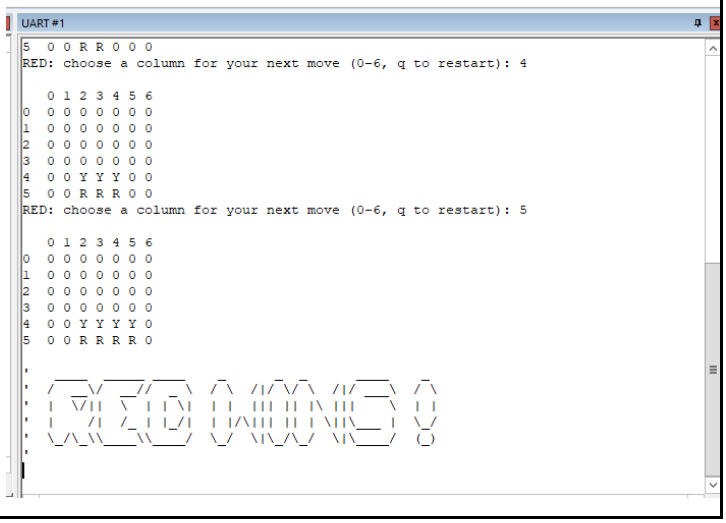
The first part of this routine is determines whether to check the “R” or “Y” first. It does this by using the R2 to determine whose turn it is and then stores the “R” / “Y” in R5.

Then using two for loops, one inside another, the program checks one byte at a time moving to the right checking for repeating R5’s. If the program finds a consecutive loop it increases a winCount. If the winCount reaches up to 4 (therefore 4 consecutive R5’s) it uses the R5 to determine who has won.

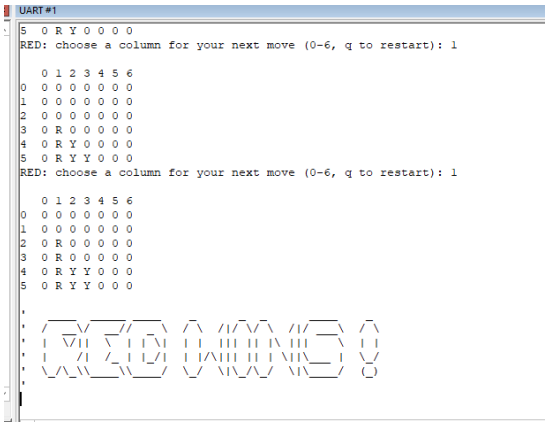
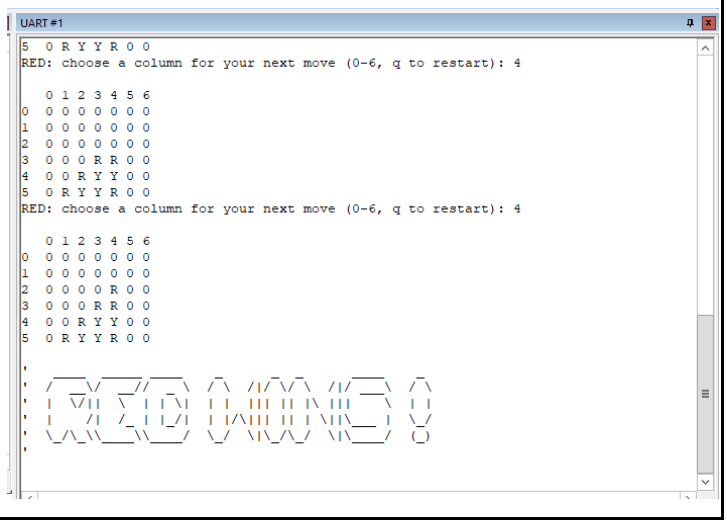
In assembly, what this is doing is using an ADD R/, R/, #1 to increase a counter and check each column of a row by keeping the row Register (in this case R7) the same until the column Register (in this case R6) which is affected by the counter reaches the end of it’s row (then B nextletter) . Once it has reached the end of the row:
(CMP R6, #6), it restarts the counter and increases the row until it reaches the end of the board (then B nextrow).
If the loaded byte loads a value the same as the hex code of our R5 register it increases the winCount by (in this case R12) 1. Once that R12 has reached #4, (4 in a row) it considers a winner and uses B stop to stop the program.

This routine is virtually the same for the checkVertical, checkRightDiagonal and checkLeftDiagonal operations. The only difference being that for the checkVertical it goes down each row one column at a time. Then for the diagonals it adjusts either forward one row and one column or back one row and back one column.

To demonstrate the success of this of these routines I have tested the following inputs:

Inputs	Visual Representation
2, 3, 4, 5 As you can see from the ascii art, once four consecutive “R” have been placed horizontally Red Wins !!!	

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

1, 1, 1, 1 The same as the above example once a row of consecutive “R”’s is created Red Wins.	
1, 2, 4, 3, 4, 4 This shows the creation of a row of “R”’s in a diagonal is seen from column 1 to column 4 resulting in a win for R.	

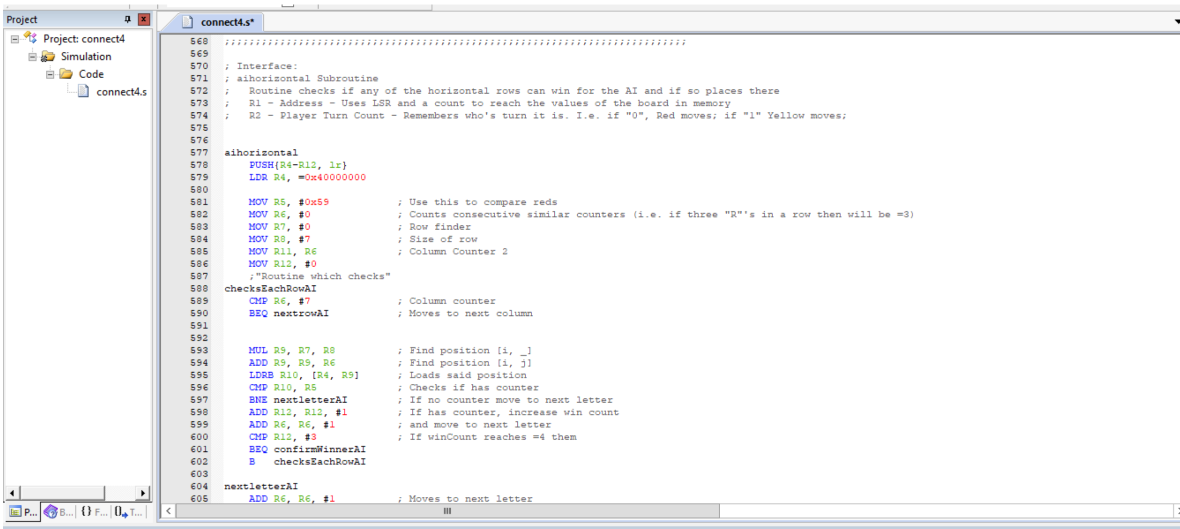
Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

A.I. Implementation

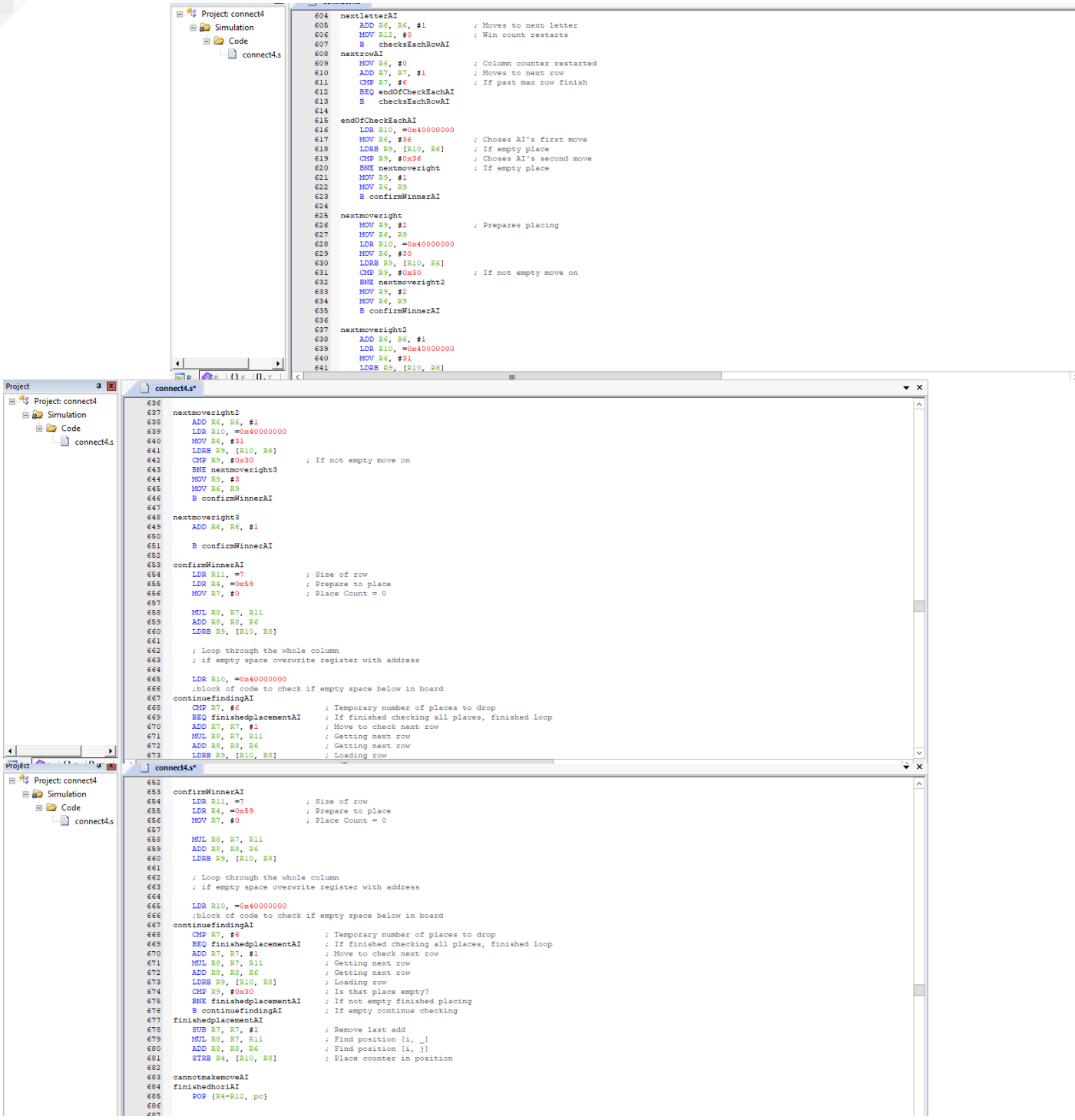
```
228
229
230  aiturn
231      ; Block of code which runs through AI's movements
232
233      BL airightdiagonal      ; Checks if AI can win by right diagonal, if not moves on
234      BL aileftdiagonal       ; Checks if AI can win by left diagonal, if not moves on
235      BL aivertical           ; Checks if AI can win by vertical, if not moves on
236      BL aihorizontal         ; Checks if AI can win by horizontal, if not places random/ or besid
237                                                          ;... inputted (depends on how many previ
238
```

Here we can see the A.I. block I talked about earlier in the makeAMove routine. This block of code; the A.I. turn block uses branch links to run routines to check if the yellow could win by placing a counter in a certain position it places it there. Otherwise it places it in a random spot or/ next to the player.

```
// AI Moveset Pseudo Code

AI Turn

        checkVerAI
        checkRightDiaAI
        checkLeftDiaAI
        checkHorAI
        // Otherwise places random position


        E.g.:
        checkHorAI
        // same routine as the check for win, if winCount =3
        // use the makeAMove subroutine to place a counter next to the previous
        // counter
        // If no win choice, random position chosen
```


Project
Project: connect4
Simulation
Code
connect4.s
connect4.s*
;;
; Interface:
; aihorizontal Subroutine
;   Routine checks if any of the horizontal rows can win for the AI and if so places there
;   R1 - Address - Uses LSR and a count to reach the values of the board in memory
;   R2 - Player Turn Count - Remembers who's turn it is. I.e. if "0", Red moves; if "1" Yellow moves;
aihorizontal
PUSH{R4-R12, lr}
LDR R4, =0x40000000
MOV R5, #0x59 ; Use this to compare reds
MOV R6, #0 ; Counts consecutive similar counters (i.e. if three "R"'s in a row then will be =3)
MOV R7, #0 ; Row finder
MOV R8, #7 ; Size of row
MOV R11, R6 ; Column Counter 2
MOV R12, #0
;"Routine which checks"
checksEachRowAI
CMP R6, #7 ; Column counter
BEQ nextrowAI ; Moves to next column
MUL R9, R7, R8 ; Find position [i, _]
ADD R9, R9, R6 ; Find position [i, j]
LDRB R10, [R4, R9] ; Loads said position
CMP R10, R5 ; Checks if has counter
BNE nextletterAI ; If no counter move to next letter
ADD R12, R12, #1 ; If has counter, increase win count
ADD R6, R6, #1 ; and move to next letter
CMP R12, #3 ; If winCount reaches =4 them
BEQ confirmWinnerAI
B checksEachRowAI
nextletterAI
ADD R6, R6, #1 ; Moves to next letter

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

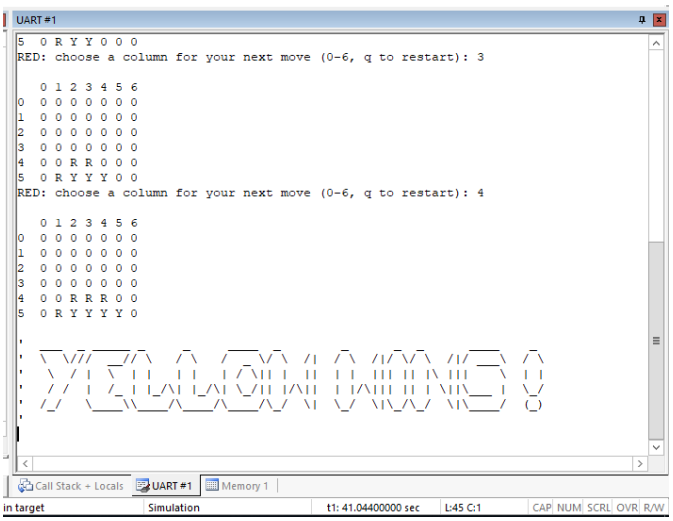


Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

This was undoubtedly the longest routine of the program mainly because it incorporates two routines; a check method and then a move method, and then multiplies that by four for each individual type of check.

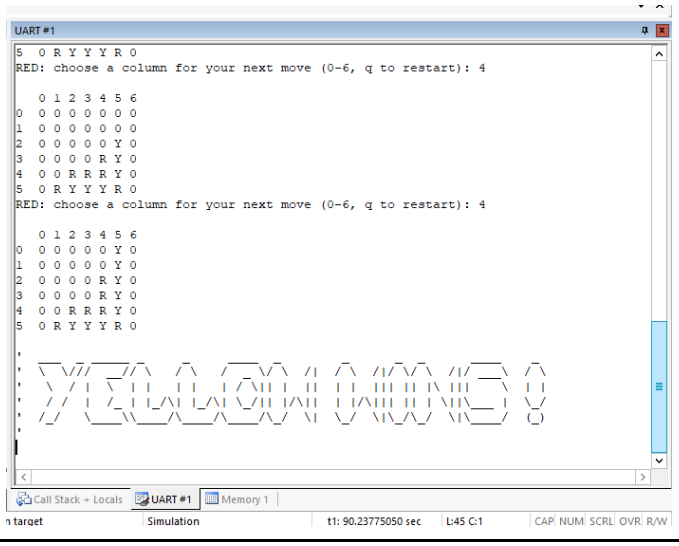
In assembly, it starts with the checkVerticalAI routine and using the array position finder method explain earlier checks each individual byte by using an ADD R/, R/, #1 to increase the shift by #1 each check. If the check CMP more than 3 cases of consecutive hex codes of the same value it chooses to place another counter on those hex codes in the same column to win. It places this counter by using the make a move method where it check each byte of a column below and determines if it can move down or not. Once it reaches the end or reaches a BNE R/, #0x59 , the program places the counter in the current position.

This is virtually the same for the other Branch Links/Routines in the A.I. turn block of code, the only difference being the check routines are swapped with their respective routines.

Inputs	Visual Representation
1, 2, 3, 3 The AI will continually move to the right of my input except at the last move because it knows that moving to the right one more time would guarantee a win	

Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

1, 2, 3, 5, 4, 4, 4
The AI moves three to the left by then I block it from getting a fourth. Knowing this, instead it moves up using the vertical branch and continues until it wins. Showing that it corrects itself if the win is no longer available.

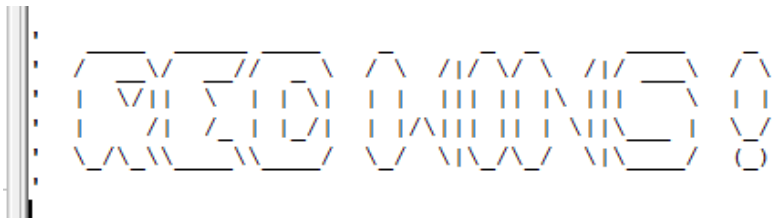


Name: John Keaney
Student ID: 18328855
15th March 2018
Lecturer/Instructor: Jonathan Dukes

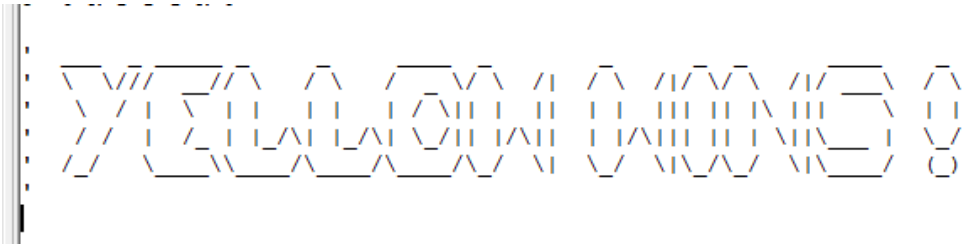
Ascii Art Usage

For the output of the win condition I decided to use ascii art to add something extra to the program. In this case I used the following images:

For red:



For yellow:



In both cases I used a string of the appropriate list of hex codes to form this output. Then loading said strings to the R0 register I then used puts to display then once a win condition had been meet.