Lab4

Name:苗昊田 ID: 3190104547

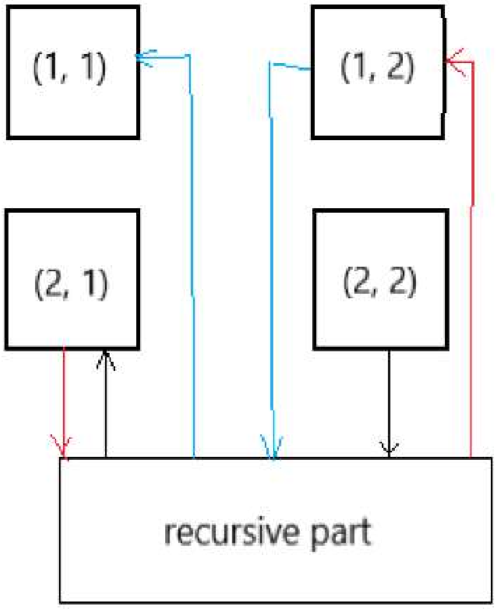
Introduction

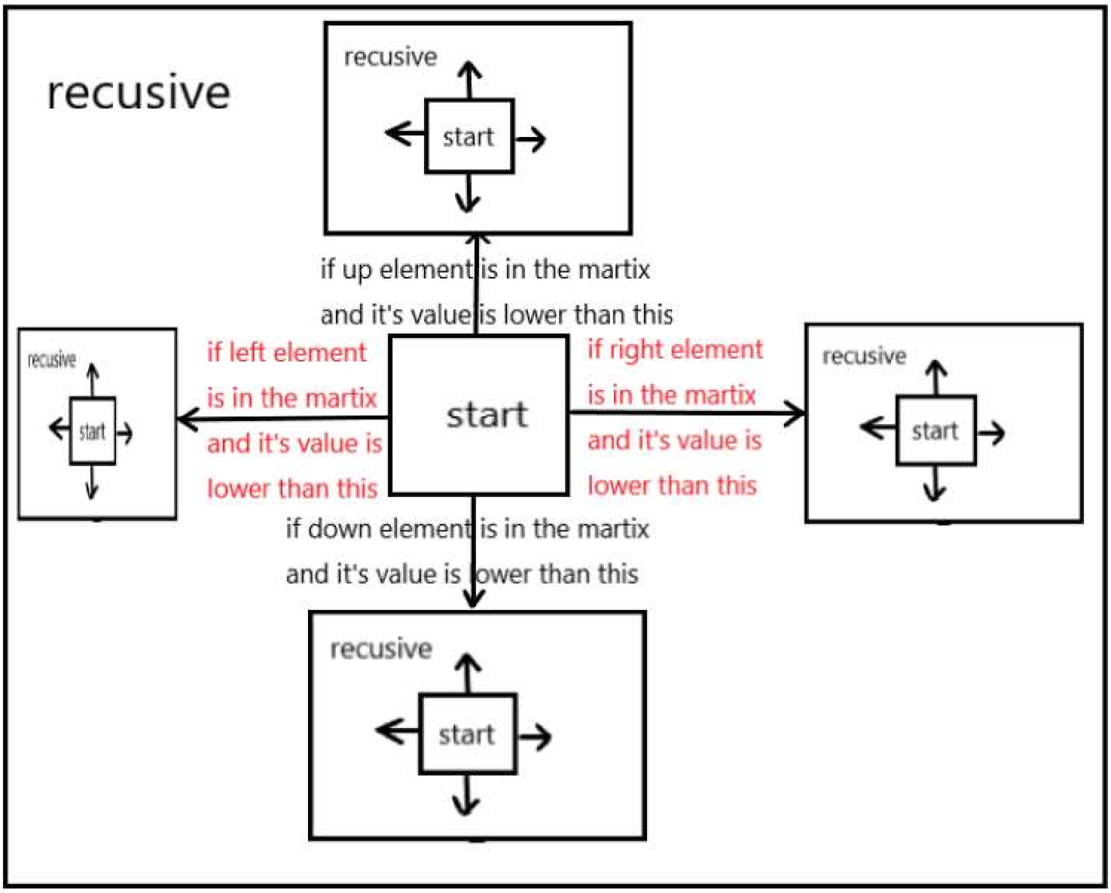
This program is to find the longest path in a matrix of map using recursive ways. And the report will represent the algorithm and some testing results.

Algorithm

First we take every position as start point each time, to find the longest path of every start point. And we always store the longest one.

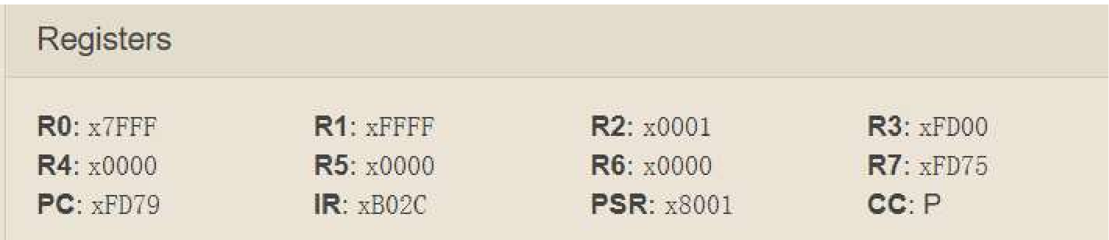
Second is the recursive algorithm for every start. We use stacks to store the position we past through. And we check the “left”, “right”, “up” and “down” of this position, if the value of that position is lower than position now, we will go to that position and do second step again. If all four directions are not allowed to pass we will go back to last position through the position stack. Last we check the next start point until all positions have been taken as the start point. The longest path is showed on R2.-------------------------------------





Testing Result

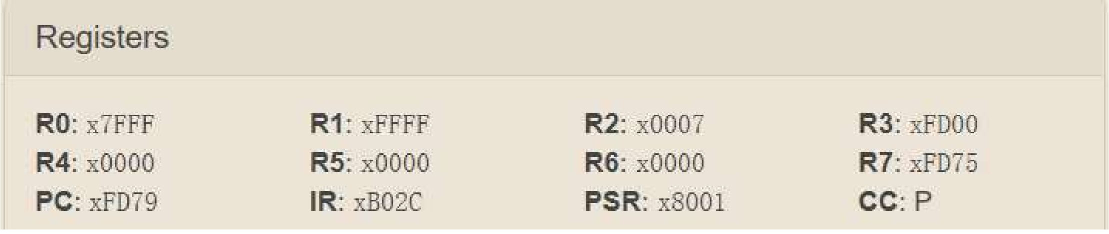
If the matrix is 1 1 then the result is 1, so the R2 is x0001
1 1



Registers

R0: x7FFF **R1**: xFFFF **R2**: x0001 **R3**: xFD00
R4: x0000 **R5**: x0000 **R6**: x0000 **R7**: xFD75
PC: xFD79 **IR**: xB02C **PSR**: x8001 **CC**: P

If the matrix is 1 7 6 5 then the result should be 7, so the R2 is x0007
2 9 8 3
3 4 5 2



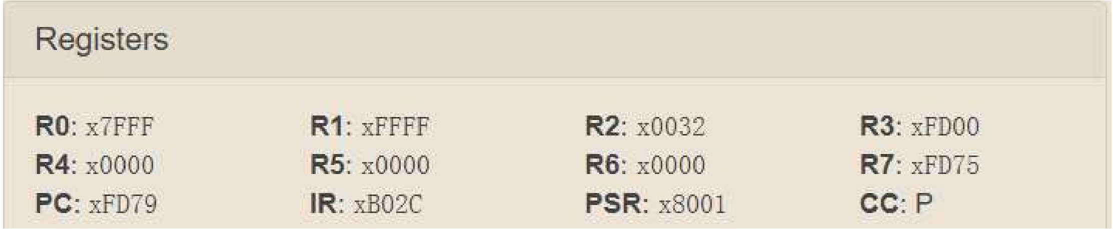
Registers

R0: x7FFF **R1**: xFFFF **R2**: x0007 **R3**: xFD00
R4: x0000 **R5**: x0000 **R6**: x0000 **R7**: xFD75
PC: xFD79 **IR**: xB02C **PSR**: x8001 **CC**: P

If the matrix is 1 2 3 … 10 then the result should be 50, so the R2 is x0032

```
If the matrix is   1    2    3    …    10
                  20   19   18    …    11
                  …
                  41   42   43    …    50
```

then the result should be 50, so the R2 is x0032 though it takes about 32s



Registers

```
R0: x7FFF        R1: xFFFF        R2: x0032        R3: xFD00
R4: x0000        R5: x0000        R6: x0000        R7: xFD75
PC: xFD79        IR: xB02C        PSR: x8001       CC: P
```

Discussion and Experience

Though this experiment, I gain a deep knowledge and understanding about recursive which I didn't make clear in C. Also I learn much about stack and function. The problem I met is that the initialization of registers which causes a covert problem. And also about the DFS. All of those were solved by debugging step by step. Because I don't make the recursive clear so I decide to spend a lot of time on debugging which takes me about 5 hours at the beginning of the working on lab4.

APPENDIX:SOURCECODE

R0: row now   R1: column now   R2: the path we go through   R3: value now in matrix.
R5: position now in matrix   R6: pointer of stacks   R7: PC

```
.ORIG x3000
        AND R0, R0, #0
        ST R0, DIST
        LD R0, ROW
        LDR R0, R0, #0
        LD R1, COL
        LDR R1, R1, #0
        ADD R0, R0, #1
        ADD R1, R1, #1
        ST R0, SAVEROW
        ST R1, SAVECOL
        LD R6, SAVER7
        ST R6, R7STACK
LOOP1   AND R6, R6, #0          ;loop in row
        LD R0, SAVEROW
        LD R1, COL
        LDR R1, R1, #0
        ADD R1, R1, #1
        ST R1, SAVECOL
        ADD R0, R0, #-1
        ST R0, SAVEROW
        BRz OVERALL
LOOP2   AND R6, R6, #0              ;loop in column
        LD R0, SAVEROW
        LD R1, SAVECOL
        ADD R1, R1, #-1
        ST R1, SAVECOL
        BRz LOOP1
        AND R2, R2, #0
        JSR DFS
        BR LOOP2
OVERALL LD R2, DIST                ;every element is done
        ADD R2, R2, #1
        HALT
DFS     ADD R6, R6, #1              ;the recursive function
        ST R6, R01STACK
        LD R6, R7STACK
        ADD R6, R6, #-1
        STR R7, R6, #0
        ST R6, R7STACK                ;store R7,
        ADD R0, R0, #0                   ;-----
        BRnz OVERIT
        LD R3, ROW
        LDR R3, R3, #0
        NOT R3, R3
        ADD R3, R3, #1
        ADD R3, R3, R0
        BRp OVERIT
        ADD R1, R1, #0
        BRnz OVERIT
        LD R3, COL
        LDR R3, R3, #0
        NOT R3, R3
        ADD R3, R3, #1
        ADD R3, R3, R1
        BRp OVERIT                       ;-----     return when the element is not in matrix
        ADD R2, R2, #1                ;-----
        AND R5, R5, #0
        AND R4, R4, #0
        AND R3, R3, #0
        ADD R4, R1, #0
        ADD R3, R0, #0
        ST R4, SAVER4
```

```
MUL     ADD R3, R3, #-1
        BRnz NEXT
        LD R4, COL
        LDR R4, R4, #0
        ADD R5, R5, R4
        BR MUL
NEXT    LD R4, COL
        ADD R5, R5, R4
        LD R4, SAVER4
        ADD R5, R5, R4          ;-----      find the element at (R0, R1)
        LD R6, R01STACK       ;-----
        LD R4, SAVER5
        ADD R4, R4, R6
        STR R5, R4, #0
        LD R6, R01STACK
        LD R5, SAVER0
        ADD R6, R6, R5
        STR R0, R6, #0
        LD R6, R01STACK
        LD R5, SAVER1
        ADD R6, R6, R5
        STR R1, R6, #0         ;-----      store R0, R1, R5 into stack
RIGHT   LD R6, R01STACK       ;check right
        LD R5, SAVER5
        ADD R5, R5, R6
        LDR R5, R5, #0
        LD R0, SAVER0
        ADD R0, R0, R6
        LDR R0, R0, #0
        LD R1, SAVER1
        ADD R1, R1, R6
        LDR R1, R1, #0
        LDR R3, R5, #0
        LDR R4, R5, #1
        NOT R4, R4
        ADD R4, R4, #1
        ADD R4, R4, R3
        BRnz LEFT
        ADD R1, R1, #1
        LD R5, SAVER5
        JSR DFS
LEFT    LD R6, R01STACK       ;check left
        LD R5, SAVER5
        ADD R5, R5, R6
        LDR R5, R5, #0
        LD R0, SAVER0
        ADD R0, R0, R6
        LDR R0, R0, #0
        LD R1, SAVER1
        ADD R1, R1, R6
        LDR R1, R1, #0
        LDR R3, R5, #0
        LDR R4, R5, #-1
        NOT R4, R4
        ADD R4, R4, #1
        ADD R4, R4, R3
        BRnz UP
        ADD R1, R1, #-1
        JSR DFS
UP      LD R6, R01STACK         ;check up
        LD R5, SAVER5
        ADD R5, R5, R6
        LDR R5, R5, #0
        LD R0, SAVER0
        ADD R0, R0, R6
        LDR R0, R0, #0
        LD R1, SAVER1
        ADD R1, R1, R6
        LDR R1, R1, #0
        LDR R3, R5, #0
        LD R4, COL
        LDR R4, R4, #0
        NOT R4, R4
        ADD R4, R4, #1
        ADD R5, R5, R4
        LDR R4, R5, #0
        NOT R4, R4
        ADD R4, R4, #1
        ADD R4, R4, R3
        BRnz DOWN
        ADD R0, R0, #-1
        JSR DFS
DOWN    LD R6, R01STACK           ;check down
        LD R5, SAVER5
        ADD R5, R5, R6
        LDR R5, R5, #0
        LD R0, SAVER0
        ADD R0, R0, R6
        LDR R0, R0, #0
        LD R1, SAVER1
        ADD R1, R1, R6
        LDR R1, R1, #0
        LDR R3, R5, #0
        LD R4, COL
        LDR R4, R4, #0
        ADD R5, R5, R4
        LDR R4, R5, #0
        NOT R4, R4
        ADD R4, R4, #1
        ADD R4, R4, R3
        BRnz RETURN
        ADD R0, R0, #1
        JSR DFS
RETURN  ADD R2, R2, #-1           ;if the check is over, store R2
        LD R5, DIST
        LDR R3, R5, #0
        NOT R5, R5
        ADD R5, R5, #1
        ADD R5, R5, R2
        BRn OVERIT
        ST R2, DIST
OVERIT  LD R6, R01STACK           ;return
        ADD R6, R6, #-1
        ST R6, R01STACK
        LD R6, R7STACK
        LDR R7, R6, #0
        ADD R6, R6, #1
        ST R6, R7STACK
        LD R6, R01STACK
        RET
ROW .FILL x3200
COL .FILL x3201
SAVER7 .FILL x2FFF
SAVER0 .FILL x4000
SAVER1 .FILL x4100
SAVER5 .FILL x4200
SAVER4 .BLKW #1
SAVEROW .BLKW #1
SAVECOL .BLKW #1
R7STACK .BLKW #1
R01STACK .BLKW #1
DIST .BLKW #1
```

.END