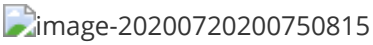
LAB 4 Report

1.INTRODUCTION

- Target: find the longest way which can slide (that means each choosed value is bigger than next one)
- learn how to use LC-3 assembly language and recursive ways to solve this problem
    - learn how to use array to read and storage many datas.
- Algorithm：the simple DFS and don't need to put the visited flag.

2.Algorithm

- initlize all the number you will use later, storage them in the correct address.
- for each position you storage the address in R1 and now value in R4, then use recursive DFS to find out all the possible situation.
    - if value is bigger than the MAX, then the MAX is value.
    - if Right way is exist and the Right value is bigger, then push some data in STACK and recirsive DFS
    - if left way is exist and the left value is bigger, then push some data in STACK and recirsive DFS
    - if Top way is exist and the Top value is bigger, then push some data in STACK and recirsive DFS
    - if Bottom way is exist and the Bottom value is bigger, then push some data in STACK and recirsive DFS
    - Then pop the data and choose where to Jump, whether continue DFS or Break.
- storage the value of Max into R2 and HALT



3.TESTING RESULT

the test input is under

```
.ORIG   x3200
.FILL   #4
.FILL   #7
.FILL   #7
.FILL   #6
.FILL   #5
.FILL   #4
```

```
.FILL   #3
.FILL   #2
.FILL   #1
.FILL   #1
.FILL   #5
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #4
.FILL   #3
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #1
.FILL   #5
.FILL   #6
.FILL   #7
.FILL   #8
.FILL   #1
.FILL   #1
.END

expect: 7

.ORIG   x3200
.FILL   #3
.FILL   #3
.FILL   #9
.FILL   #1
.FILL   #2
.FILL   #5
.FILL   #6
.FILL   #7
.FILL   #8
.FILL   #4
.FILL   #3
.END
expect:4

.ORIG   x3200
.FILL   #4
.FILL   #4
.FILL   #1
.FILL   #2
.FILL   #2
.FILL   #1
.FILL   #1
.FILL   #4
.FILL   #4
.FILL   #1
.FILL   #1
.FILL   #3
.FILL   #3
.FILL   #1
```

```
.FILL   #1
.FILL   #2
.FILL   #2
.FILL   #1
.END
expect:4
```

input	expect	output	status
First	7	7	pass
Second	4	4	pass
Third	4	4	pass

Th also some simple samples.

4.DISCUSSION AND EXPERIENCE

- in this program I learn some using skill about JMP and BRnzp, if we want set PC in some register, JMP is the best solution to change PC, BRnzp is useless in this situation
- this program we don't need to check the overflow of the stack, but normally it is necessary.
- we can calculate the used number before the loop and LD them when we used it
- the array of address is much difficult to use but it's save some register.

APPENDIX Code

- program

```
.ORIG   x3000
LD  R0, BEGIN;
LDR R1, R0 ,#0;
LDR R2, R0, #1;
ST  R1, N;  store the row of Matrix
ST  R2, M;   store the column of Matrix
NOT R4, R2;
ADD R4, R4, #1;
ST  R4 ,Negative_M;
NOT R4, R1;
ADD R4, R4, #1;
ST  R4, Negative_N;
AND R3, R3, #0;
ST R3 MAX;  reset the MAX number
LOOP1 ADD   R3, R3,R1;
ADD R2, R2, #-1;
BRp LOOP1;  R3 is the N*M
ADD R0, R0, R3;
ADD R0, R0, #1;
ST  R0, LAST; store the last data's address
NOT R0, R0;
```

```
ADD R0, R0, #1;
ST  R0, Negative_LAST; store the negative last address
LD  R6, STACK; now R6 is the ptr of stack

AND R0, R0, #0; clear the value of output address
LEA R1,ANSWER;
LOOP2 STR   R0, R1, #0;
ADD R1,R1,  #1;
ADD R3, R3, #-1;
BRp LOOP2;
;init all the register
AND R3, R3, #0;
AND R1, R1, #0;
AND R2, R2, #0;
AND R0, R0, #0;
AND R4, R4, #0;
AND R5, R5, #0;


LD  R1, FIRST;
Each_Begin LD   R4, LAST;
NOT R4, R4;
ADD R4, R4, #1;
ADD R4, R1, R4;
BRp BREAK; every address is checked
AND R4, R4, #0;
ADD R4, R4, #1;
AND R3, R3, #0;
AND R5, R5, #0;
AND R2, R2, #0;
JSR DFS; run dfs and storage the output
ADD R1, R1 ,#1;
BRnzp   Each_Begin;

;DFS likes its means , use C language program and translate it to LC-3
DFS  LD R2, MAX;   if R4 > MAX then MAX=R4
NOT R2, R2;
ADD R2,R2,  #1;
ADD R2, R2, R4; check which is bigger;
BRnz  #1;
ST  R4, MAX;

;check whether can go RIGHT
RIGHT LD    R2,FIRST;
NOT R2,R2;
ADD R2, R2, #2;
ADD R2, R2, R1;
LD  R3, Negative_M;
RIGHT_LOOP ADD  R2, R2,R3; whether can  divide exactly
BRz LEFT;  check left
BRp RIGHT_LOOP;
;now we check whether can go to the right;
LDR R3, R1,#0;
NOT R3, R3;
ADD R3, R3, #1; -Value of R1
ADD R2, R1, #1;
LDR R2, R2, #0;
ADD R2, R2, R3; check which is higher
```

```
BRnz LEFT;
ADD R2, R1, #1;
LEA R5, LEFT;
;put now station into stack
ADD R6, R6, #-1;
STR R1, R6, #0;
ADD R6, R6, #-1;
STR R4, R6, #0;
ADD R6, R6, #-1;
STR R5, R6, #0;

ADD R1, R2, #0;
ADD R4, R4, #1;
BRnzp   DFS;

;now check whether can go left
LEFT LD R2,FIRST;
NOT R2,R2;
ADD R2, R2, #1; it's a simple way to do that
ADD R2, R2, R1;
LD  R3, Negative_M;
LEFT_LOOP BRz TOP;
ADD R2, R2, R3;
BRzp LEFT_LOOP;

;now we check whether can go to the right;
LDR R3, R1,#0;
NOT R3, R3;
ADD R3, R3, #1; -Value of R1
ADD R2, R1, #-1;
LDR R2, R2, #0;
ADD R2, R2, R3; check which is higher
BRnz TOP;
ADD R2, R1, #-1;
LEA R5, TOP;
;put now station into stack
ADD R6, R6, #-1;
STR R1, R6, #0;
ADD R6, R6, #-1;
STR R4, R6, #0;
ADD R6, R6, #-1;
STR R5, R6, #0;

ADD R1, R2, #0;
ADD R4, R4, #1;
BRnzp   DFS;


;check whether can go top position
TOP LD  R2,FIRST;
LD  R3, M;
ADD R2, R2, R3;
NOT R2, R2;
ADD R2, R2, #1;
ADD R2, R2, R1;
BRn BOTTOM;
;now we check whether can go to the right;
LDR R3, R1,#0;
```

```
NOT R3, R3;
ADD R3, R3, #1; -Value of R1
LD  R2, Negative_M;
ADD R2, R1, R2;
LDR R2, R2, #0;
ADD R2, R2, R3; check which is higher
BRnz BOTTOM;
LD  R2, Negative_M;
ADD R2, R1, R2; R2 is the  now address
LEA R5, BOTTOM;
;put now station into stack
ADD R6, R6, #-1;
STR R1, R6, #0;
ADD R6, R6, #-1;
STR R4, R6, #0;
ADD R6, R6, #-1;
STR R5, R6, #0;

ADD R1, R2, #0;
ADD R4, R4, #1;
BRnzp   DFS;

;check whether can go down
BOTTOM LD   R2,Negative_LAST;
LD  R3, M;
ADD R3,R3,  R1;
ADD R2, R2, R3;
BRp RETURN;
;now we check whether can go to the right;
LDR R3, R1,#0;
NOT R3, R3;
ADD R3, R3, #1; -Value of R1
LD  R2, M;
ADD R2, R1, R2;
LDR R2, R2, #0; R2 now is address
ADD R2, R2, R3; check which is higher
BRnz RETURN;
LD  R2, M;
ADD R2, R1, R2; R2 is the  now address
LEA R5, RETURN;
;put now station into stack
ADD R6, R6, #-1;
STR R1, R6, #0;
ADD R6, R6, #-1;
STR R4, R6, #0;
ADD R6, R6, #-1;
STR R5, R6, #0;

ADD R1, R2, #0;
ADD R4, R4, #1;
BRnzp   DFS;

;now it's time to RETURN
RETURN LD   R2, Negative_STACK;
ADD R2, R2, R6;
BRnp #1; if R6 is x3000 then return ;
RET;
LDR R5, R6, #0;
```

```
ADD R6, R6, #1;
LDR R4, R6, #0;
ADD R6, R6, #1;
LDR R1, R6, #0;
ADD R6, R6, #1;
JMP R5;
;now DFS is over and output the needed number

BREAK  LD   R2, MAX;
 HALT;

N .FILL #1; the number of Row
M .FILL #1; the number of Column
Negative_N .FILL    #1;
Negative_M .FILL    #1;
MAX .FILL   #-1;
ANSWER .BLKW    #50;
STACK .FILL x3000;  the beginner of STACK
Negative_STACK .FILL    x-3000;
BEGIN .FILL x3200;
FIRST .FILL x3202; the first ptr of DATA
LAST .FILL  #1; the last ptr of DATA
Negative_LAST .FILL #1;
.END
```