

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

image-20200720200750815

3.TESTING RESULT

the test input is under

```
1  .ORIG    x3200
2  .FILL    #4
3  .FILL    #7
4  .FILL    #7
5  .FILL    #6
6  .FILL    #5
7  .FILL    #4
```

```
8 .FILL #3
9 .FILL #2
10 .FILL #1
11 .FILL #1
12 .FILL #5
13 .FILL #1
14 .FILL #1
15 .FILL #1
16 .FILL #1
17 .FILL #1
18 .FILL #1
19 .FILL #4
20 .FILL #3
21 .FILL #1
22 .FILL #1
23 .FILL #1
24 .FILL #1
25 .FILL #1
26 .FILL #5
27 .FILL #6
28 .FILL #7
29 .FILL #8
30 .FILL #1
31 .FILL #1
32 .END
33
34 expect: 7
35
36 .ORIG x3200
37 .FILL #3
38 .FILL #3
39 .FILL #9
40 .FILL #1
41 .FILL #2
42 .FILL #5
43 .FILL #6
44 .FILL #7
45 .FILL #8
46 .FILL #4
47 .FILL #3
48 .END
49 expect:4
50
51 .ORIG x3200
52 .FILL #4
53 .FILL #4
54 .FILL #1
55 .FILL #2
56 .FILL #2
57 .FILL #1
58 .FILL #1
59 .FILL #4
60 .FILL #4
61 .FILL #1
62 .FILL #1
63 .FILL #3
64 .FILL #3
65 .FILL #1
```

```

66 .FILL #1
67 .FILL #2
68 .FILL #2
69 .FILL #1
70 .END
71 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

```

1  .ORIG    x3000
2  LD  R0, BEGIN;
3  LDR R1, R0 ,#0;
4  LDR R2, R0, #1;
5  ST  R1, N;  store the row of Matrix
6  ST  R2, M;  store the column of Matrix
7  NOT R4, R2;
8  ADD R4, R4, #1;
9  ST  R4 ,Negative_M;
10 NOT R4, R1;
11 ADD R4, R4, #1;
12 ST  R4, Negative_N;
13 AND R3, R3, #0;
14 ST  R3 MAX;  reset the MAX number
15 LOOP1 ADD  R3, R3,R1;
16 ADD R2, R2, #-1;
17 BRp LOOP1;  R3 is the N*M
18 ADD R0, R0, R3;
19 ADD R0, R0, #1;
20 ST  R0, LAST; store the last data's address
21 NOT R0, R0;

```

```

22 ADD R0, R0, #1;
23 ST R0, Negative_LAST; store the negative last address
24 LD R6, STACK; now R6 is the ptr of stack
25
26 AND R0, R0, #0; clear the value of output address
27 LEA R1, ANSWER;
28 LOOP2 STR R0, R1, #0;
29 ADD R1, R1, #1;
30 ADD R3, R3, #-1;
31 BRp LOOP2;
32 ;init all the register
33 AND R3, R3, #0;
34 AND R1, R1, #0;
35 AND R2, R2, #0;
36 AND R0, R0, #0;
37 AND R4, R4, #0;
38 AND R5, R5, #0;
39
40
41 LD R1, FIRST;
42 Each_Begin LD R4, LAST;
43 NOT R4, R4;
44 ADD R4, R4, #1;
45 ADD R4, R1, R4;
46 BRp BREAK; every address is checked
47 AND R4, R4, #0;
48 ADD R4, R4, #1;
49 AND R3, R3, #0;
50 AND R5, R5, #0;
51 AND R2, R2, #0;
52 JSR DFS; run dfs and storage the output
53 ADD R1, R1, #1;
54 BRnzp Each_Begin;
55
56 ;DFS likes its means , use C language program and translate it to LC-3
57 DFS LD R2, MAX; if R4 > MAX then MAX=R4
58 NOT R2, R2;
59 ADD R2, R2, #1;
60 ADD R2, R2, R4; check which is bigger;
61 BRnz #1;
62 ST R4, MAX;
63
64 ;check whether can go RIGHT
65 RIGHT LD R2, FIRST;
66 NOT R2, R2;
67 ADD R2, R2, #2;
68 ADD R2, R2, R1;
69 LD R3, Negative_M;
70 RIGHT_LOOP ADD R2, R2, R3; whether can divide exactly
71 BRZ LEFT; check left
72 BRp RIGHT_LOOP;
73 ;now we check whether can go to the right;
74 LDR R3, R1, #0;
75 NOT R3, R3;
76 ADD R3, R3, #1; -Value of R1
77 ADD R2, R1, #1;
78 LDR R2, R2, #0;
79 ADD R2, R2, R3; check which is higher

```

```

80  BRnz LEFT;
81  ADD R2, R1, #1;
82  LEA R5, LEFT;
83  ;put now station into stack
84  ADD R6, R6, #-1;
85  STR R1, R6, #0;
86  ADD R6, R6, #-1;
87  STR R4, R6, #0;
88  ADD R6, R6, #-1;
89  STR R5, R6, #0;
90
91  ADD R1, R2, #0;
92  ADD R4, R4, #1;
93  BRnzp DFS;
94
95  ;now check whether can go left
96  LEFT LD R2, FIRST;
97  NOT R2, R2;
98  ADD R2, R2, #1; it's a simple way to do that
99  ADD R2, R2, R1;
100 LD R3, Negative_M;
101 LEFT_LOOP BRz TOP;
102 ADD R2, R2, R3;
103 BRzp LEFT_LOOP;
104
105 ;now we check whether can go to the right;
106 LDR R3, R1, #0;
107 NOT R3, R3;
108 ADD R3, R3, #1; -value of R1
109 ADD R2, R1, #-1;
110 LDR R2, R2, #0;
111 ADD R2, R2, R3; check which is higher
112 BRnz TOP;
113 ADD R2, R1, #-1;
114 LEA R5, TOP;
115 ;put now station into stack
116 ADD R6, R6, #-1;
117 STR R1, R6, #0;
118 ADD R6, R6, #-1;
119 STR R4, R6, #0;
120 ADD R6, R6, #-1;
121 STR R5, R6, #0;
122
123 ADD R1, R2, #0;
124 ADD R4, R4, #1;
125 BRnzp DFS;
126
127
128 ;check whether can go top position
129 TOP LD R2, FIRST;
130 LD R3, M;
131 ADD R2, R2, R3;
132 NOT R2, R2;
133 ADD R2, R2, #1;
134 ADD R2, R2, R1;
135 BRn BOTTOM;
136 ;now we check whether can go to the right;
137 LDR R3, R1, #0;

```

```

138 NOT R3, R3;
139 ADD R3, R3, #1; -Value of R1
140 LD R2, Negative_M;
141 ADD R2, R1, R2;
142 LDR R2, R2, #0;
143 ADD R2, R2, R3; check which is higher
144 BRnz BOTTOM;
145 LD R2, Negative_M;
146 ADD R2, R1, R2; R2 is the now address
147 LEA R5, BOTTOM;
148 ;put now station into stack
149 ADD R6, R6, #-1;
150 STR R1, R6, #0;
151 ADD R6, R6, #-1;
152 STR R4, R6, #0;
153 ADD R6, R6, #-1;
154 STR R5, R6, #0;
155
156 ADD R1, R2, #0;
157 ADD R4, R4, #1;
158 BRnzp DFS;
159
160 ;check whether can go down
161 BOTTOM LD R2, Negative_LAST;
162 LD R3, M;
163 ADD R3, R3, R1;
164 ADD R2, R2, R3;
165 BRp RETURN;
166 ;now we check whether can go to the right;
167 LDR R3, R1, #0;
168 NOT R3, R3;
169 ADD R3, R3, #1; -Value of R1
170 LD R2, M;
171 ADD R2, R1, R2;
172 LDR R2, R2, #0; R2 now is address
173 ADD R2, R2, R3; check which is higher
174 BRnz RETURN;
175 LD R2, M;
176 ADD R2, R1, R2; R2 is the now address
177 LEA R5, RETURN;
178 ;put now station into stack
179 ADD R6, R6, #-1;
180 STR R1, R6, #0;
181 ADD R6, R6, #-1;
182 STR R4, R6, #0;
183 ADD R6, R6, #-1;
184 STR R5, R6, #0;
185
186 ADD R1, R2, #0;
187 ADD R4, R4, #1;
188 BRnzp DFS;
189
190 ;now it's time to RETURN
191 RETURN LD R2, Negative_STACK;
192 ADD R2, R2, R6;
193 BRnp #1; if R6 is x3000 then return ;
194 RET;
195 LDR R5, R6, #0;

```

```

196 ADD R6, R6, #1;
197 LDR R4, R6, #0;
198 ADD R6, R6, #1;
199 LDR R1, R6, #0;
200 ADD R6, R6, #1;
201 JMP R5;
202 ;now DFS is over and output the needed number
203
204 BREAK LD R2, MAX;
205 HALT;
206
207 N .FILL #1; the number of Row
208 M .FILL #1; the number of Column
209 Negative_N .FILL #1;
210 Negative_M .FILL #1;
211 MAX .FILL #-1;
212 ANSWER .BLKW #50;
213 STACK .FILL x3000; the beginner of STACK
214 Negative_STACK .FILL x-3000;
215 BEGIN .FILL x3200;
216 FIRST .FILL x3202; the first ptr of DATA
217 LAST .FILL #1; the last ptr of DATA
218 Negative_LAST .FILL #1;
219 .END

```