**Assignment 4 Report**

**1.Problem：**

Suppose the skiing field is a rectangular area which is described by a two-dimensional array. Each element of the array denotes the height of the point. Michael can start from any point, and he can slide to one of the four adjacent points if its height is lower than current height. Michael cannot leave ski field when skiing.

e.g.

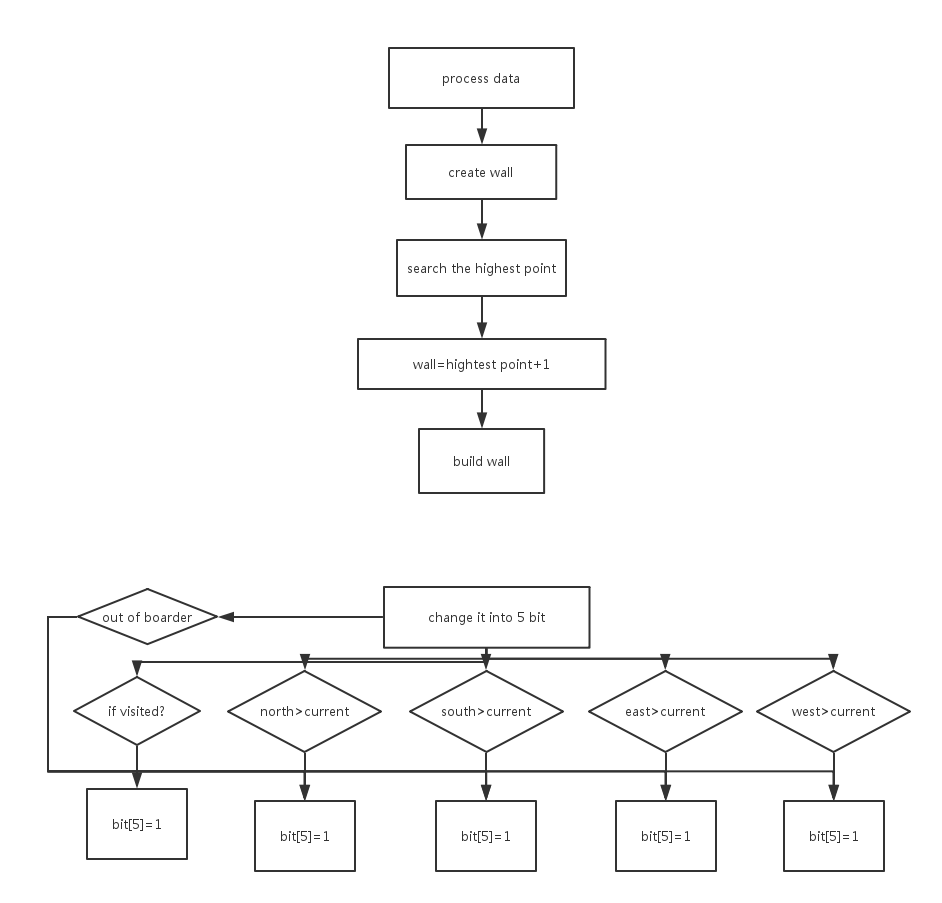
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 16 | **17** | **18** | **19** | **6** |
| 15 | **24** | **25** | **20** | **7** |
| 14 | **23** | **22** | **21** | **8** |
| 13 | **12** | **11** | **10** | **9** |

Output: 25

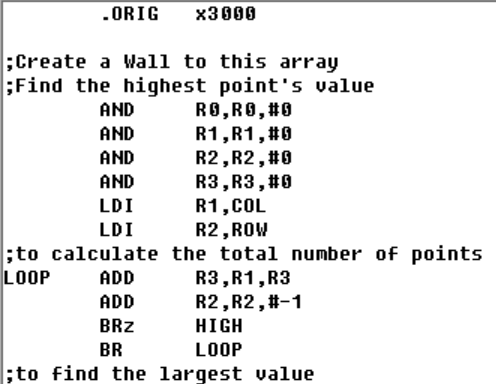
**2.Main Idea:**

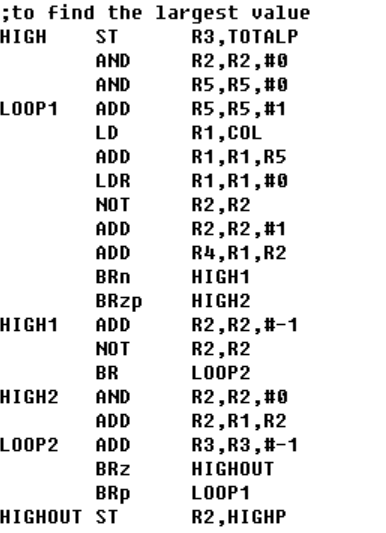
**（1）Data Processing**

**1．Build Wall**

****

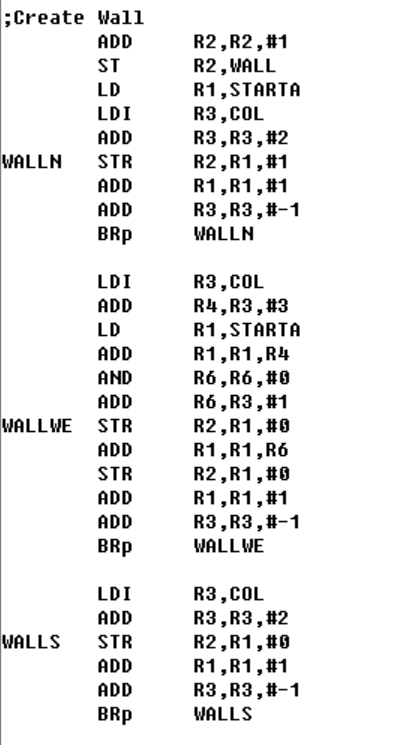
**a. In order to build a wall whose every point is higher than the inner input value. We need to find the highest value first.**





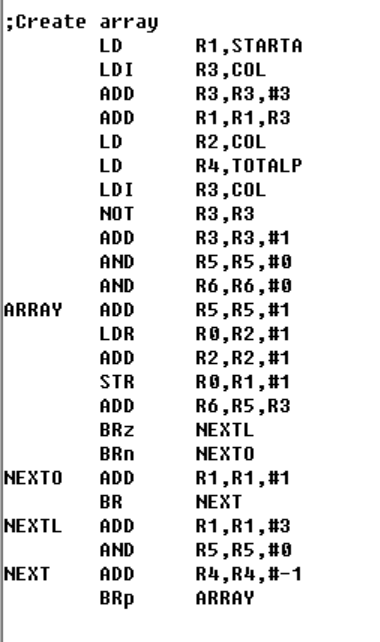
**b. Than Create a wall first in another block of memory and in th memory space it should be a 1-dimenional array.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **26** | **26** | **26** | **26** | **26** | **26** | **26** |
| **26** |  |  |  |  |  | **26** |
| **26** |  |  |  |  |  | **26** |
| **26** |  |  |  |  |  | **26** |
| **26** |  |  |  |  |  | **26** |
| **26** |  |  |  |  |  | **26** |
| **26** | **26** | **26** | **26** | **26** | **26** | **26** |

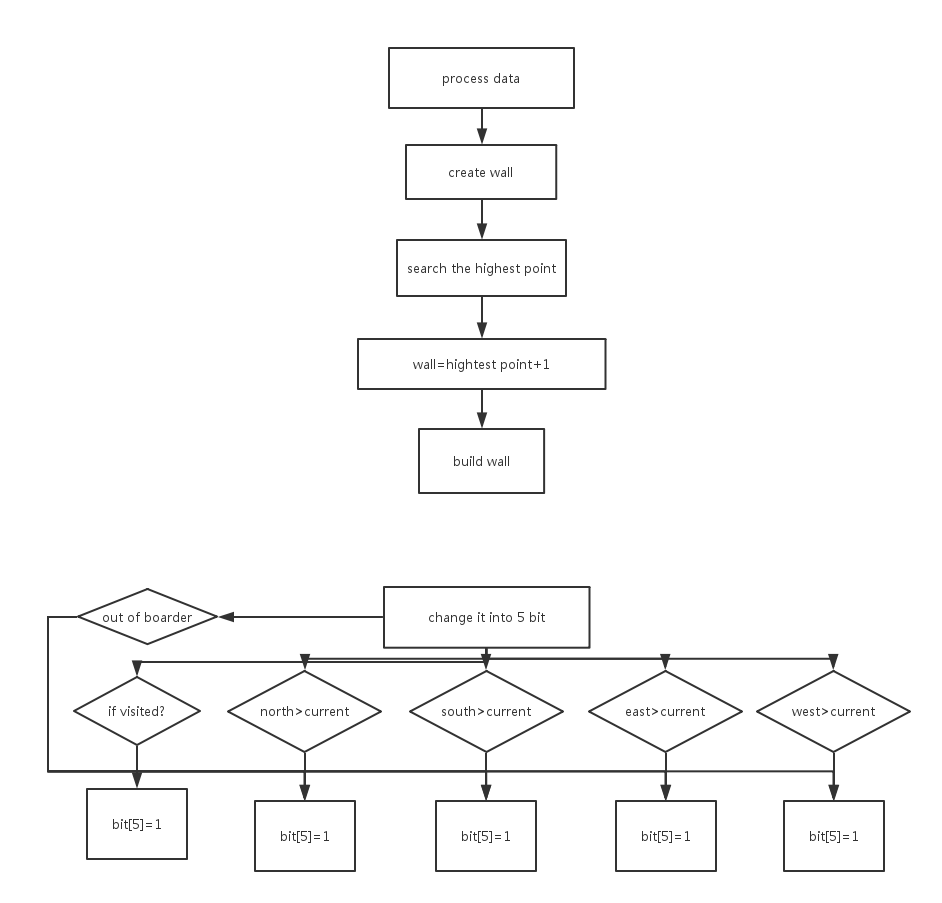


**c. In the end, store original input value into new memory space and in th memory space it should be a 1-dimenional array.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **26** | **26** | **26** | **26** | **26** | **26** | **26** |
| **26** | **1** | **2** | **3** | **4** | **5** | **26** |
| **26** | **16** | **17** | **18** | **19** | **6** | **26** |
| **26** | **15** | **24** | **25** | **20** | **7** | **26** |
| **26** | **14** | **23** | **22** | **21** | **8** | **26** |
| **26** | **13** | **12** | **11** | **10** | **9** | **26** |
| **26** | **26** | **26** | **26** | **26** | **26** | **26** |

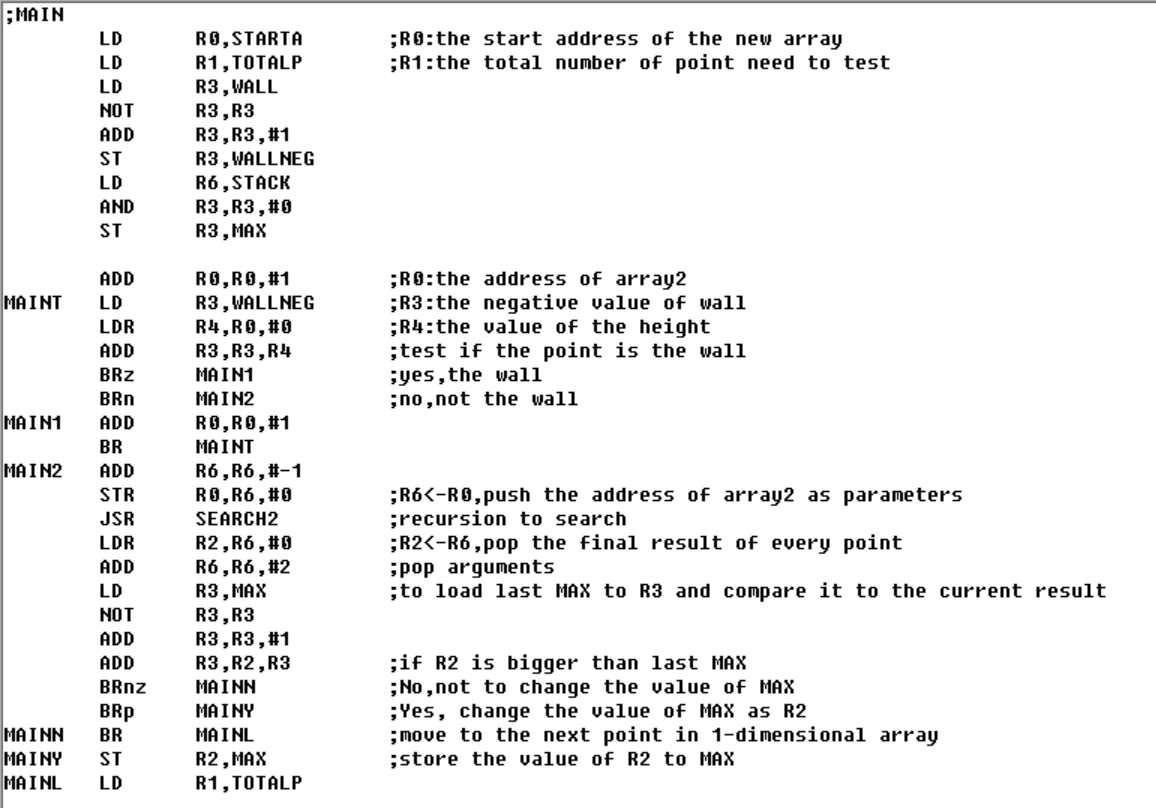


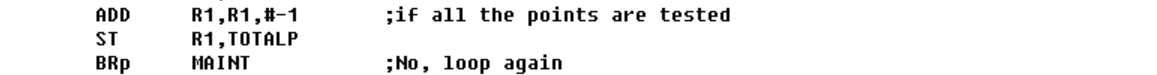
**（2）Maze**

****

**(2) Main Function**

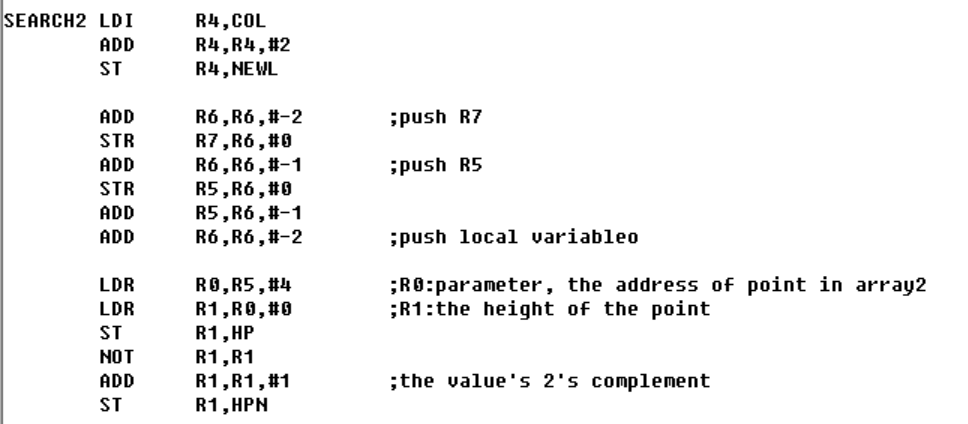
**Use recursion to test every point’s available path length. First test if it’s a wall point, if so, move on; otherwise use JSP instruction to do the search part.**





**(3) recursion**

**a. build stack**



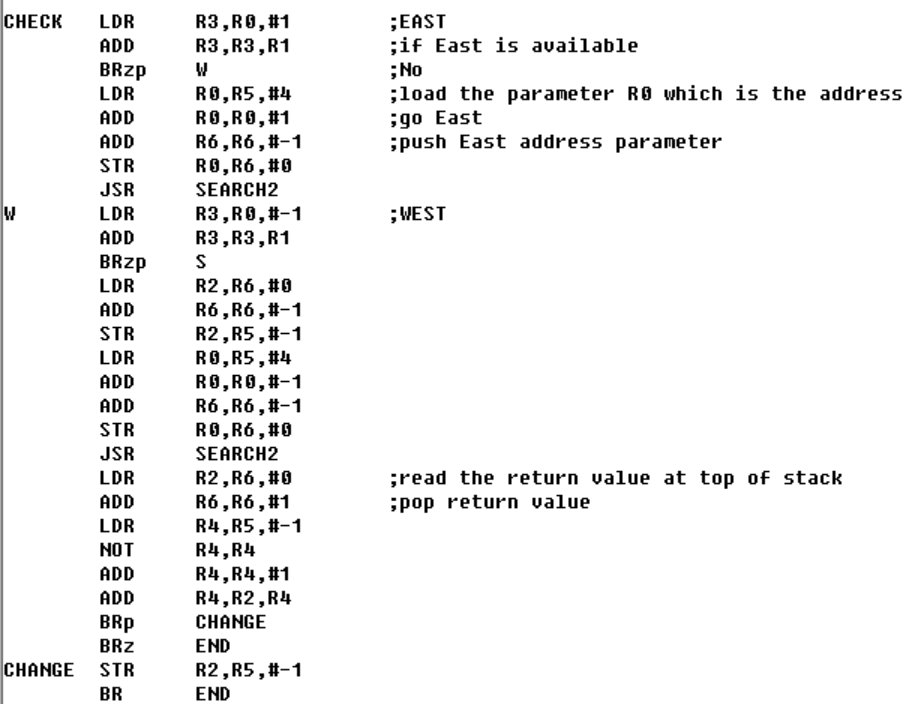
**b. check four direction**

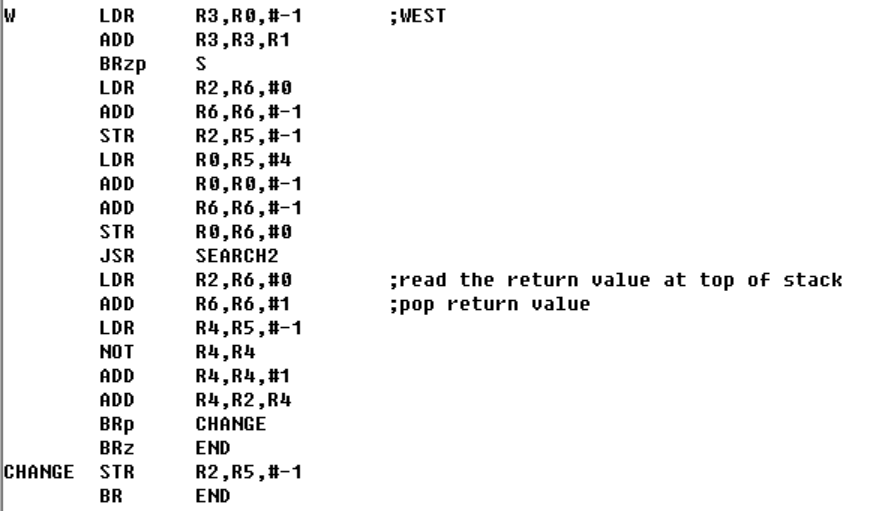
**East: if it’s out of order (as the last column)**

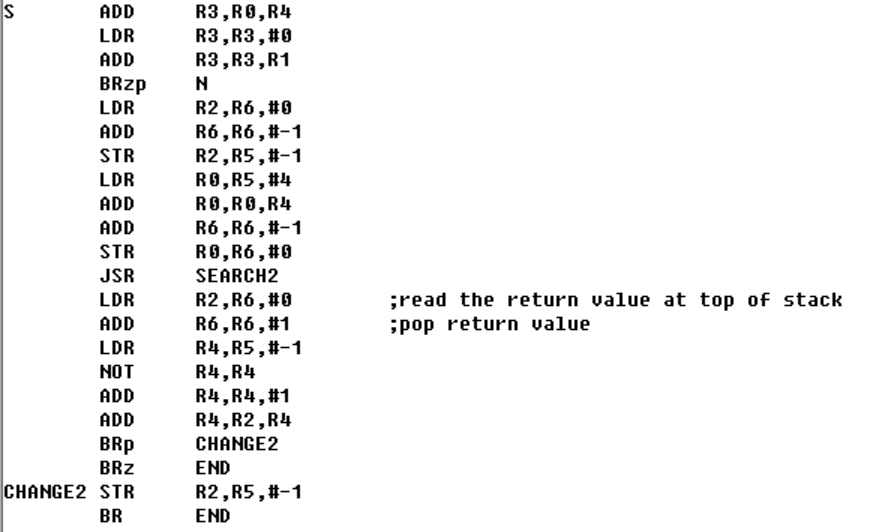
**If it’s available to go east (the east of current point’s value is smaller than that of current point)**

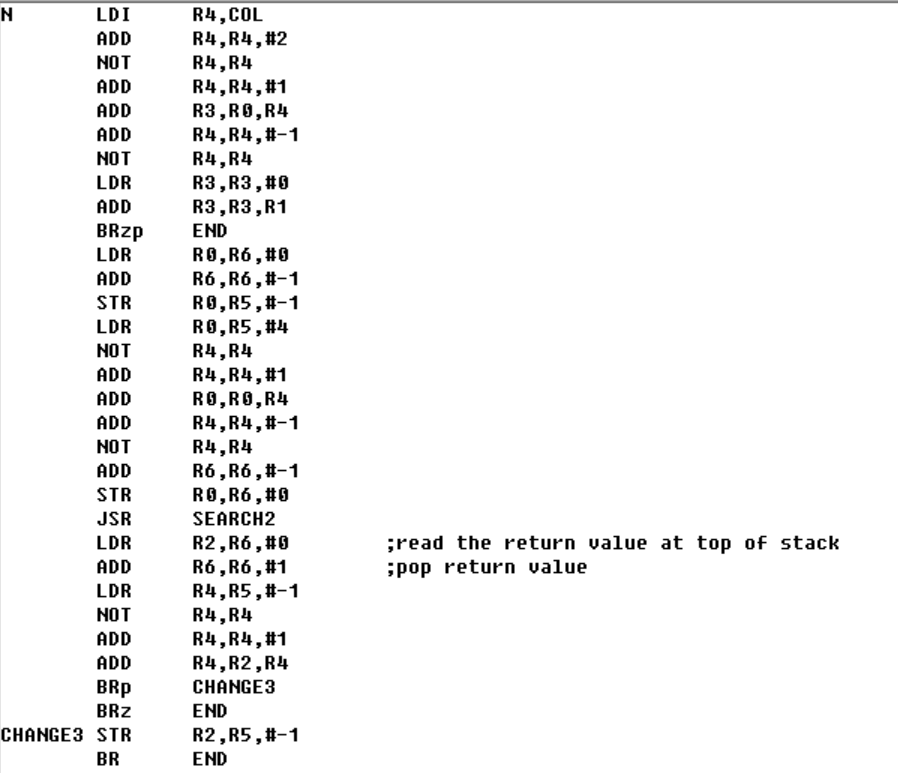
**If so, go to the east and recurs; otherwise check the west.**

**And do loop until the four directions have already been visited.**

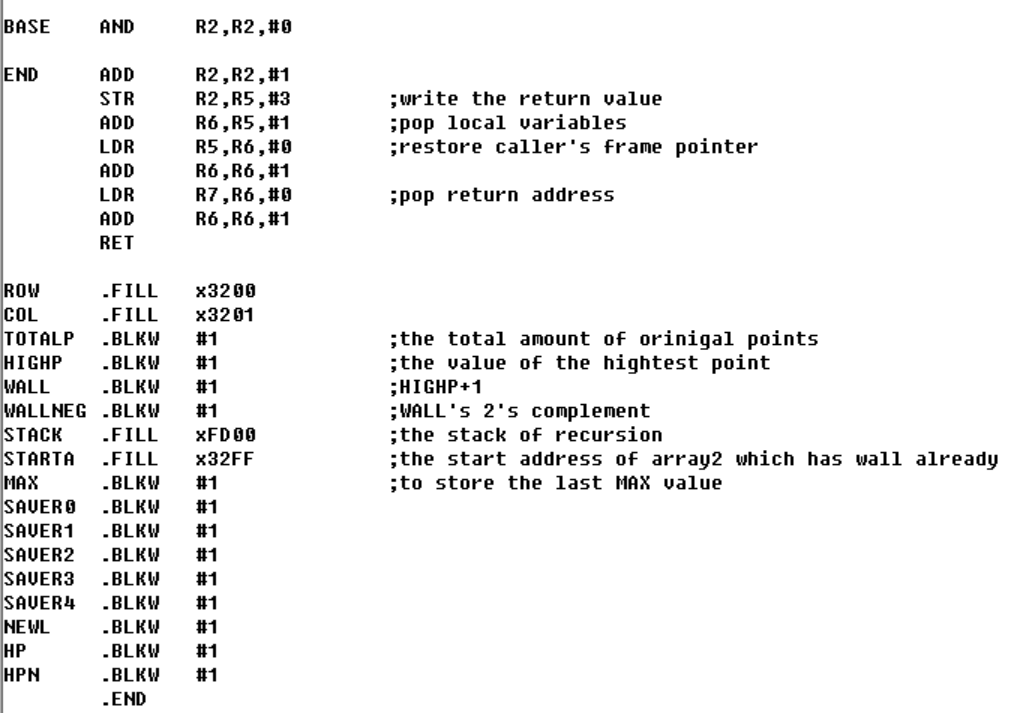








**(4) Exit**



**4.Result：**

Store it into R2.

