Report for LAB-5

1 Algorithm (Pseudo Code)

- `func main()` load `N`, `M` from `x4000` and try to take each element in the 2D-array as the start point, and return the maximum result in `R2` at the end.

- `func MaxL(int i, int j)` try to find the longest path start `a[i][j]`. It would check each of its 4 neighbors (if exists) and return `Max(MaxL@neighbors) + 1` => that's where the recursion happens.

  Recursion ends when `pos(i,j)` is illegal, since we would check the validity of neighbor's position before call `func MaxL()` recursively.

```
int *base = 0x4002;

int main () {
   int res = 0;
   for(int i = 0 ; i < N ; i++) {
      for (int j = 0 ; j < M ; j++) {
         Max(res, MaxL(i, j));
      }
   }
   return res;
}

int MaxL(int i, int j) {
   int res = 0;
   int cur = *(base + i*M + j);
   // check 4 neighbors
   if(Less(i-1, j, cur)) Max(&res, MaxL(i-1, j));
   if(Less(i+1, j, cur)) Max(&res, MaxL(i+1, j));
   if(Less(i, j-1, cur)) Max(&res, MaxL(i, j-1));
   if(Less(i, j+1, cur)) Max(&res, MaxL(i, j+1));

   return res+1;
}
```

```
int Less(int i, int j, int cur) {
   // is (i,j) valid ?
   if(i == -1)  return 0;
   if(i == N)   return 0;
   if(j == -1)  return 0;
   if(j == M)   return 0;
   // (a[i][j] < cur)?1:0
   int a = *(base + i*M + j);
   return (a-cur < 0) ? 1 : 0;
}

void Max(int* res, int i) {
   *res = (*res > i) ? *res : i;
}
```

The interfaces in assembly are as follows:

Function	Args	Return Value
MAIN		R2: Longest Distance
MAX	R2(res), R5	R2: Max(R2, R5)
MULM	R0(i)	R2: R0 * M
LESS	R0(i), R1(j), R4(cur)	R2: 0 – Illegal / a[i][j] >= cur 1 – Legal & a[i][j] < cur
MAXL	R0(i), R1(j)	R2: Longest Distance at a[i][j]

2 Code (Essential Parts with Comments)

This part mainly shows how `func MaxL(int i, int j)` works:

- Input & Output:
  Input `i` in `R0`, `j` in `R1`; Return maxLength in `R2`
- How to visit array element `a[i][j]`?
  If we cast the Matrix `a[N][M]` to a 1D-array, then we can visit `a[N][M]` by accessing `Memory[base + i*M + j]`
- What the index of array element `a[i][j]`'s neighbors?
  In matrix, the 4 neighbors are `UP:(i-1, j)`, `DOWN:(i+1,j)`,`LEFT: (i,j-1)`, `RIGHT:(i,j+1)`, we can also cast their position to 1D-array.
- When should the recursion ends?
  The recursion stops when position is out of range, there are 4 cases: `i == -1 / N, j == -1, M`, this would be checked by `func Less(i, j, cur)`

And since it would be called recursively, we should save & restore those Callee-Saved registers in the stack.

```
MAXL ; store R0, R1, R2, R4, R7 in stack
      ADD R6, R6, #-1;
      STR R0, R6, #0;
      ; …
      AND R2, R2, #0; res=0
      ; load a[i][j]
      JSR MULM;       R5 = i*m
      LD R4, BASE;    R4 = BASE
      ADD R4, R4, R5; R4 = BASE + i*m
      ADD R4, R4, R1; R4 = BASE + i*m + j
      LDR R4, R4, #0; R4 = a[i][j]
      ; check a[i-1][j]
      ADD R0, R0, #-1; R0 = i-1
      JSR LESS;       R5 = less(i-1, j, cur)
      ADD R5, R5, #0;  invalid, next case
      BRz CASE2
      JSR MAXL;       R5 = maxL (i-1, j)
      JSR MAX;        R2 = max(R2, R5)
      ; check a[i+1][j]
CASE2 ADD R0, R0, #2;  R0 = i+1
      JSR LESS
      ADD R5, R5, #0
      BRz CASE3
      JSR MAXL
      JSR MAX
      ; check a[i][j-1]
CASE3 ADD R0, R0, #-1; R0 = i
      ADD R1, R1, #-1; R1 = j-1
      JSR LESS
      ADD R5, R5, #0
      BRz CASE4
      JSR MAXL
      JSR MAX
      ; check a[i][j+1]
CASE4 ADD R1, R1, #2; R1 = j+1
      JSR LESS
      ADD R5, R5, #0
      BRz RESTML
      JSR MAXL
      JSR MAX
RESTML ADD R5, R2, #1; return res+1
      ; restore from stack
      LDR R7, R6, #0;
      ADD R6, R6, #1;
      ; …
      RET
```

3 Q & A

1. **Explain your recursion function.**

   It has been explained in Section 2.

2. **If `M * N = 50`, how large might your User Stack be?**

   The worst case is that `max(len(path)) == M * N`, then there would be `50 * func MAXL()`'s active records in the User Stack. Since each active records would push 5 register's value (`R0, R1, R2, R4, R7`) into the stack, the max size of the User Stack would be `5 * 50`.

   In fact it would be `5 * 50 + 1`, because `func LESS()` would push `R7` into the stack and there could only be ONE active record of `func LESS()` at the same time.