DEPARTMENT OF COMPUTER SCIENCE
COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 341: Computer Organization
HW 7: Recursion (Again)

When writing code, comments are important! Note the rubric specifies that about half your lines should be commented - this is not atypical for assembly.

We **strongly recommend** you write the pseudocode first (it is a required element in the submitted README), and if needed, rewrite it using only the constructs if-goto and goto, to make the leap from abstraction to RISC-V smaller. Have your pseudocode handy if you come to office hours for assistance with this.

An Ouroboros is "an ancient symbol of a serpent or dragon eating its own tail" (Just in case you didn't know that). It symbolizes cycles that begin again as soon as they end, or something that is constantly recreating itself.[1] It may not be the exact definition of recursion, but the concept is similar. That cannot be good for the serpent though, so in this assignment you will find all the paths through a rectangular grid that start at the top left corner and end at the bottom right corner, that way the serpent doesn't double back on itself accidentally.

In this program you will be implementing a recursive searching procedure that prints out **the values** in all the paths from `(0,0)` in a grid to `(xMax-1,yMax-1)` in a grid, given that you can only move down, to the right, and diagonally to the bottom right **in that order**. For consistency purposes, the grid will have (0,0) in the top left and x and y will increase as you move to the right and down respectively. An example of a 3x3 grid is shown below. Note that the values in the cells are not guaranteed to be sequential, unique, or organized in any manner.

y\x	0	1	2
0	3	5	8
1	4	2	10
2	2	1	2

The grid will be represented as a 2D array in memory, with dimensions `xMax` and `yMax`.

[1] https://en.wikipedia.org/wiki/Ouroboros

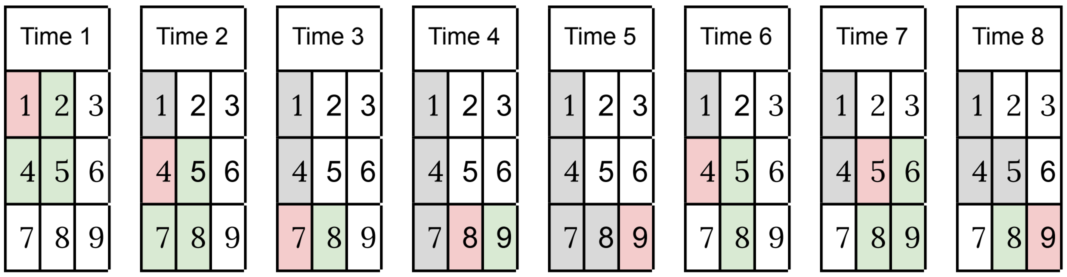
In Memory	xMax and yMax	The Interpretation as a Grid
3 | 5 | 8 | 4 | 2 | 10 | 2 | 1 | 2	xMax: 3; yMax: 3	3 5 8 / 4 2 10 / 2 1 2

3	5	8	4	2	10	2	1	2

xMax	3
yMax	3

3	5	8
4	2	10
2	1	2

The example below illustrates how the first two paths in a `3x3` grid will be started. Red cells represent the current cell, green cells represent potential moves, and gray cells are cells that have been chosen, representing the current path. The values in the cells represent the values of the grid at each cell. The values in the cells are what will be printed when each path is finalized.



Time 1	(0,0) is the current cell (1,0), (0,1), and (1,1) are possible moves Moving down is chosen because it has the highest precedence 1 is the current path
Time 2	(0,1) is the current cell (0,2), (1,1), and (1,2) are possible moves Moving down is chosen because it has the highest precedence 1, 4 is the current path
Time 3	(0,2) is the current cell (1,2) is the only possible move Moving right is chosen because it is the only possible move 1, 4, 7 is the current path
Time 4	(1,2) is the current cell (2,2) is the only possible move Moving right is chosen because it is the only possible move

	1, 4, 7, 8 is the current path
Time 5	(2,2) is the current cell There are no possible moves The end of this path has been reached The path will need to be printed
Time 6	The recursive case for cells (1,2) and (0,2) both fall through having explored every possible move (0,1) is now current cell (1,1) and (1,2) are the possible moves Moving right is chosen because it has precedence over moving down 1, 4, 5 is now the current path
…	…

As hinted at in the table above, **this is a recursive problem**. Recursion will actually *simplify* the problem significantly, reducing the amount of bookkeeping you will need to do.

Due to the complexity of this assignment, C++ pseudo code will be provided to you. If you are confused, take the time to understand the C++ code. When implemented properly, you will only need a stack to keep track of the current path.

```
// functions provided by grader script
void addValueToStack(int numElements, int value);
void printStack(int numElements);

void printPaths(int* grid, int xMax, int yMax, int curX, int curY,
int stackLen) {
    // base case
    if (xMax <= 0 || yMax <= 0) { return; }

    // include current cell in route
    addValueToStack(stackLen, grid[curX + curY * xMax]);
    stackLen++;

    // if the last cell is reached
    if (curY == yMax - 1 && curX == xMax - 1) {
        printStack(stackLen);
        return;
    }
    // move down if (curY + 1 < yMax) {
        printPaths(grid, xMax, yMax, curX, curY + 1, stackLen);
    }
    // move right
    if (curX + 1 < xMax) {
        printPaths(grid, xMax, yMax, curX + 1, curY, stackLen);
    }
    // move diagonally
    if (curY + 1 < yMax && curX + 1 < xMax) {
        printPaths(grid, xMax, yMax, curX + 1, curY + 1, stackLen);
    }
}
```

This program will be written in a similar manner to the previous one, with the `printPaths` procedure declared `.globl`. This assignment's grader script (provided in Canvas) will be used when grading to call your procedure. Note that *there are a few additional procedures provided to you in this grader script*. There are two more global functions that perform operations on the stack declared in the data section. Not only may you find these useful when writing your procedure, but it will also give you practice with calling procedures.

Note how the grid size is variable, and not guaranteed to be square. Do not assume only a `3x3` grid will always be entered. The smallest grid size your code should handle is `1x1` and the largest grid will contain no more than `200` cells. `xMax` and `yMax` will never be 0.

The `printPaths` procedure must match the following signature:

1. Name: printPaths
2. Arguments:
   - a2: address of the grid
   - a3: xMax
   - a4: yMax
   - a5: current x index
   - a6: current y index
   - a7: current stack length
3. Return Value:
   - None

An example run through of the program is shown below.

```
Enter a number to add to the array: 1
Enter a number to add to the array: 2
Enter a number to add to the array: 3
Enter a number to add to the array: 4
Enter a number to add to the array: 5
Enter a number to add to the array: 6
Enter a number to add to the array: 7
Enter a number to add to the array: 8
Enter a number to add to the array: 9
Enter a number to add to the array: 0
X size: 3
Y size: 3
1 4 7 8 9
1 4 5 8 9
1 4 5 6 9
1 4 5 9
1 4 8 9
1 2 5 8 9
1 2 5 6 9
1 2 5 9
1 2 3 6 9
1 2 6 9
1 5 8 9
1 5 6 9
1 5 9

-- program is finished running (0) --
```

When printing the path, the cell values **must be separated by whitespace characters** (for grading purposes). The procedure provided to you in the grader script does this for you.

As always, assume a perfect user and no malformed input.

What to turn in:

1. A zipped file with the file name lastnameFirstname.zip (.zip is the file extension) containing:
    a. Your README, with the file name lastnameFirstname_README.txt (.txt is the file extension, this should be a plain text file).
    b. Your code, with the file name lastnameFirstname_ASSEMBLY.s **Do not submit the file you downloaded to test your code.** (.s is the file extension)

In your README, include the following elements:

1. Include YOUR NAME and the names of all people who helped/collaborated as per the syllabus and CS@Mines collaboration policy. [This is an individual assignment]
2. Any additional pseudocode (in C/Java/Python/etc) for your implementation [yes it should also be recursive].

3. Describe the challenges you encountered and how you surmounted them.
4. What did you like about the assignment?
5. How long did you spend working on this assignment?
6. A description of any features you added for extra credit (if any)

Extra Credit Options:

1. None. This assignment is challenging enough.

Section	**Points**
Procedures Implemented Properly: Caller: $jal\, ra,\ LABEL$ Callee: $jalr\ zero,\ ra,\ 0$ binarySearch implemented recursively This is a big part of this grade: DOUBLE CHECK IT IS DONE CORRECTLY	10 points
Public Test Cases Pass	2 points each, 4 points total
Private Test Cases Pass	2 points each, 20 points total
Adequate comment coverage (50%)	5 pts
No pseudo instructions other than la to load addresses (hint: la should not be needed for this assignment)	6 pts, -1 for each up to -6 pts
README has required elements and effort is demonstrated	5 pts
Total	50 pts