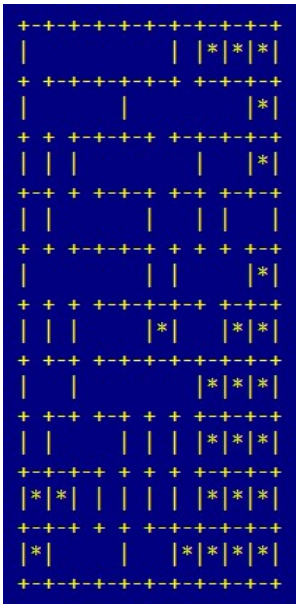
ECE220: Computer Systems & Programming **Fall 2020 ZJUI**
Lab 4 **Tuesday 13 October**

Searching a Maze



In today's lab, your group must write a depth-first search using a stack to find all reachable locations in the maze shown to the right. The goal of this exercise is to help you formulate code that uses DFS to solve a problem, as is necessary for MP3.

Begin by updating your Git repository to obtain the **`lab4`** subdirectory. The directory contains a copy of this document and a starting point for your solution in the **`lab4.asm`** file.

Most of the code has already been written for you, including a subroutine that clears the array of reachable data, pseudo-code in the form of comments for the maze searching subroutine, and a subroutine that prints the maze using the reachability data.

The subroutine that you must complete is called **`SEARCH_MAZE`**. A version marked with the registers that I used in my version and pseudo-code for the DFS appears in the file given to you. Including comments and blank lines, completing the subroutine requires fewer than 50 lines of code.

The Task

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

Starting from location (0,0) in the **`MAZE`**, you must use a depth-first search to identify all reachable locations. The maze consists of 100 (10×10) memory locations, each with a 4-bit vector (the upper 12 bits are always 0). The bits represent the ability to go left (1), right (2), up (4), and down (8) from the given room in the maze. In memory, the maze rooms are ordered from right to left and from top to bottom. An example using a 3×5 maze is shown to the right. Note that moving left or right changes the offset of the room by 1, but moving up or down changes the offset by 5 (in the example). In the maze given to you, there are 10 columns, so moving up or down changes the offset by 10.

The **`REACH`** array uses the same organization as the **`MAZE`** data, and **`R1`** has already been prepared with **`(REACH - MAZE)`** in the code given to you, so you need merely add **`R1`** to the address of a "room" in the maze to find the address of its reachability marking. When the **`SEARCH_MAZE`** subroutine starts, all 100 locations in the **`REACH`** array have been set to 0. When **`SEARCH_MAZE`** finishes, any room reachable from (0,0) should have its reach entry set to 1. Unreachable rooms should remain marked with 0 in the **`REACH`** array.

In the picture at the top of this page, unreachable locations are marked with asterisks ('*'). In the printout from **`lab4.asm`**, reachable locations are marked as asterisks ('*'), while unreachable locations appear as periods ('.'). A copy of the correct output is given to you in the **`goldout`** file. Use the script file (**`lc3sim -s script > out`**) and diff your result (**`diff out goldout`**). If your code is correct, the register values may differ, but the printed maze should not.