


Programming Lab 7D

Mazes & Gyroscopes

Topics: Shift, bitwise and bitfield instructions; address calculation with pointers.

 Click to download
Lab7D-Main.c

Prerequisite Reading: Chapters 1-7

Revised: January 22, 2021

Background¹: This lab uses a randomized version of a depth-first search algorithm to create a maze. Frequently implemented with a stack, this approach is one of the simplest ways to generate a maze using a computer. Consider the space for a maze being a large grid of cells (like a large chess board), each cell starting with four walls. Starting from a random cell, the computer then selects a random neighboring cell that has not yet been visited. The computer removes the wall between the two cells and marks the new cell as visited, and adds it to the stack to facilitate backtracking. The computer continues this process, with a cell that has no unvisited neighbors being considered a dead-end. When at a dead-end it backtracks through the path until it reaches a cell with an unvisited neighbor, continuing the path generation by visiting this new, unvisited cell (creating a new junction). This process continues until every cell has been visited, causing the computer to backtrack all the way back to the beginning cell.

Assignment: As described above this algorithm involves deep recursion which may cause stack overflow issues on some computer architectures. This lab uses an algorithm in which the recursion has been changed into a loop by storing backtracking information in the maze itself. This also provides a quick way to display a solution, by starting at any given point and backtracking to the beginning. The program keeps track of the status of cell walls in an array of 4-bit "nibbles" (two per byte) – one nibble per cell.

The main program will compile and run without writing any assembly. However, your task is to create equivalent replacements in assembly language for the following two functions found in the C main program to store and retrieve individual nibbles within the array. The original C versions have been defined as “weak” so that the linker will automatically replace them in the executable image by those you create in assembly; you do not need to remove the C versions. This allows you to create and test your assembly language functions one at a time. Your solutions should execute as fast as possible, so using loops is not acceptable.

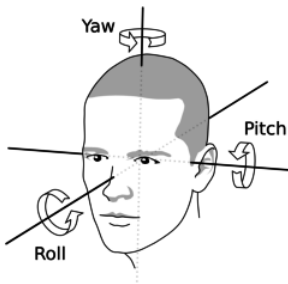
```
void PutNibble(void *nibbles, uint32_t which, uint32_t value) ;  
uint32_t GetNibble(void *nibbles, uint32_t which) ;
```

Parameter Description

nibbles Starting address of the array of nibbles.

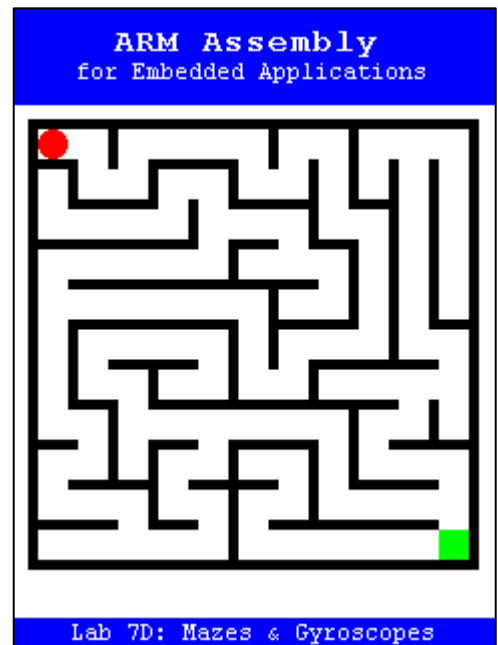
which A nibble position within the array of nibbles.

value A four-bit binary value (0000 to 1111)



The board incorporates a gyroscope that senses angular velocity in each of three directions. The program integrates the angular velocities to determine angular positions (roll, pitch and yaw). Roll is like tilting your head left or right; pitch is like tilting your head forward and backward; yaw is like turning your head left and right. The program only uses the roll and pitch sensors. Test your code using the C main program. When the program starts (or

when you press the black pushbutton) it first performs a gyro calibration after which a randomly generated maze should be displayed, similar to the one shown on the right. The object is to move the red ball to the green square by tilting the board towards the direction you would like the ball to move. Pressing the blue push button starts over with a different maze.



¹ https://en.wikipedia.org/wiki/Maze_generation_algorithm#Depth-first_search