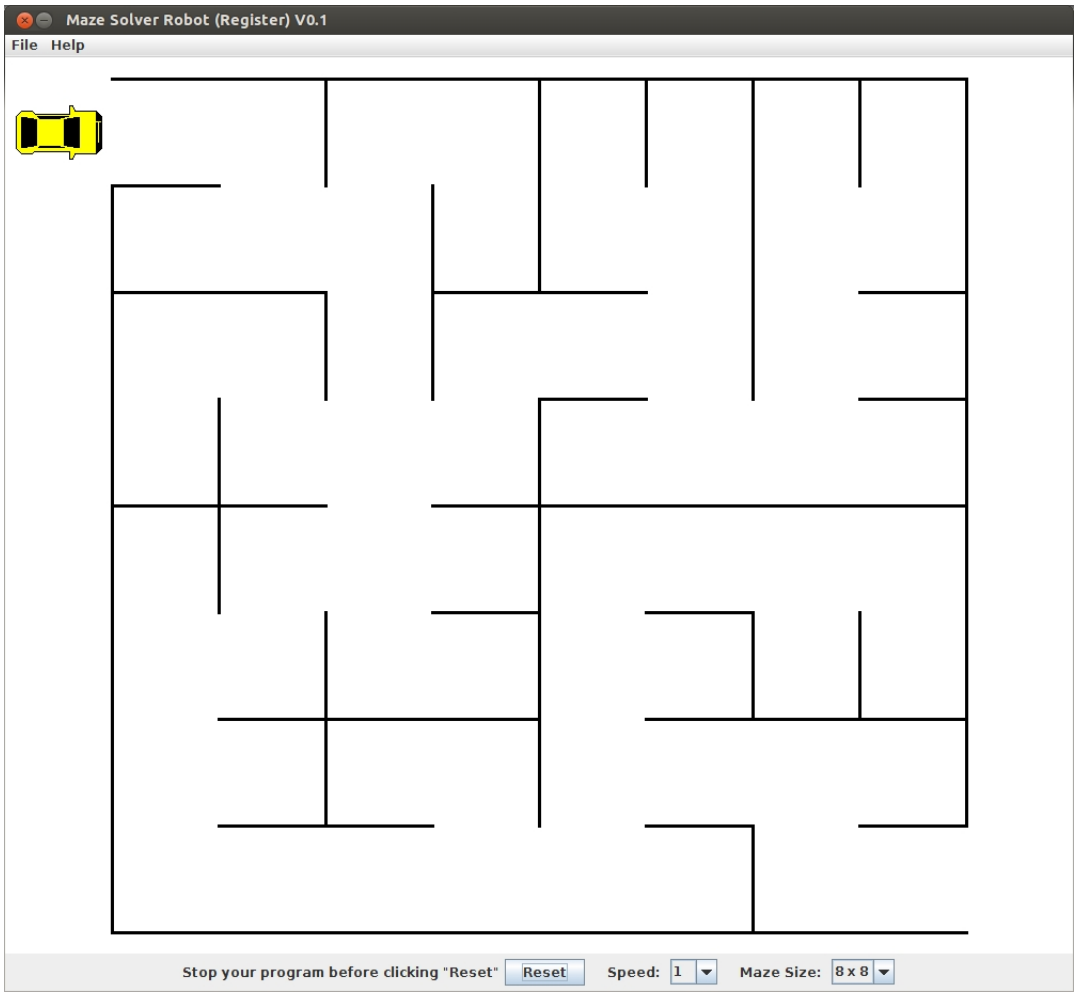
Project 3 - Maze Solver Robot

CS 0447 — Computer Organization & Assembly Language

Due Sunday Sunday 22, 2015 at 11:59pm

The purpose of this project is for you to practice writing a little bit more complex function and recursive function in assembly to control a car to navigate through a maze. The hardware for this project is a car robot with a maze as shown below:



The Maze Solver Robot (Mars Tool) shown above can be found in `mazeSolver.zip` located in the CourseWeb under this project. Extract all files to your `[..]/mars4_5/mars/tools` directory. If you extract all files to the right directory, when you run the MARS program, you should see "Maze Solver Robot (Register) V0.1" under the "Tools" menu.

Introduction to the Maze Solver Robot (Mars Tool)

This Maze Solver Robot (Mars Tool) uses registers `$t8` and `$t9`. Thus, do not use these registers in your program other than communicating with the robot car.

The control panel of this Mars tool consist of the following:

- **Reset**: This will reset the car back to the original top left corner. Note that before clicking the 'Reset' button, make sure that you stop your program first.
- **Speed**: This control the speed of the animation where 1 is the slowest and 10 is the fastest. The speed can be changed while the program is running.
- **Maze Size**: This set the size of the maze, regenerate a new maze according to the new size, and set the car back to the top left corner. Note that before chaning the size of the maze make sure that you stop your program first.

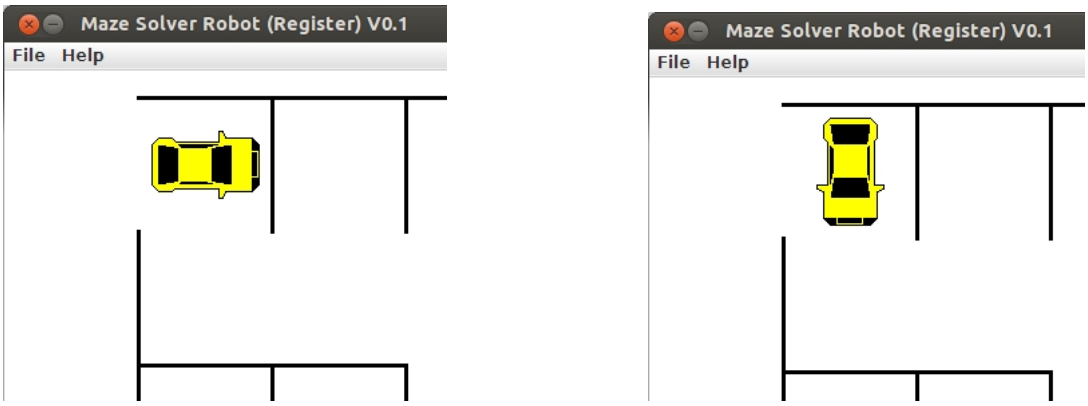
Imagine that the yellow car robot equips with a GPS, a compass, and four wall sensors (front, left, right, and back). When the car robot stops moving, it will report its status by writing a value to the register `$t9`. A value in the register `$t9` consists of a various values representing the status of the car robot as follows

- **Bit 31 to Bit 24 (8 bits)** represents the current row position. This value is in 8-bit two's complement format.
- **Bit 23 to Bit 16 (8 bits)** represents the current column position. This value is in 8-bit two's complement format.
- **Bit 11 (1 bit)**: If the value of this bit is 1, it means the car robot is facing north (up).
- **Bit 10 (1 bit)**: If the value of this bit is 1, it means the car robot is facing east (right).
- **Bit 9 (1 bit)**: If the value of this bit is 1, it means the car robot is facing south (down).
- **Bit 8 (1 bit)**: If the value of this bit is 1, it means the car robot is facing west (left).
- **Bit 3 (1 bit)**: If the value of this bit is 1, it means there is a wall in front of the car robot.
- **Bit 2 (1 bit)**: If the value of this bit is 1, it means there is a wall on the left side of the car robot.
- **Bit 1 (1 bit)**: If the value of this bit is 1, it means there is a wall on the right side of the car robot.
- **Bit 0 (1 bit)**: If the value of this bit is 1, it means there is a wall behind the car robot.

Note that the maze is organized the same way as a table. For example, Consider a maze size 8 x 8. The top left corner (inside the maze) is row 0, column 0. The top right corner (inside the maze) is row 0, column 7. The bottom left corner (inside the maze) is row 7, column 0. The bottom right corner (inside the maze) is row 7, column 7. When the car robot is a the top left corner (outside the maze) the location of the car robot is row 0, column -1. When the car robot is a the bottom right corner (outside the maze) the location of the car robot is row 7, column 8.

Status Example

Consider a section of a maze shown below.



Suppose originally the robot car is outside the maze (row 0 and column -1). If you send the command to the robot to move forward (set `$t8` to 1), after the robot stops moving (shown above on left), the register `$t9` will be set to `0x0000040c`. In binary `0x0000040c` is as follows:

00000000 00000000 0000 0100 0000 1100

which tells you the following information:

1. **Bit 31 to Bit 24** are 0s. Thus, this robot is located at row 0.
2. **Bit 23 to Bit 16** are 0s. Thus, this robot is located at column 0.
3. **Bit 10** is 1. Thus, this robot is facing east.
4. **Bits 3 and 4** are 1s. Thus, it has a wall in front of the robot and on the left of the robot.

Now, if we send a new command to turn right (set `$t8` to 3), after the robot stops moving (shown above on right), the register `$t9` will be set to `0x00000205`. In binary `0x00000205` is as follows:

00000000 00000000 0000 0010 0000 0101

which tells you the following information:

1. **Bit 31 to Bit 24** are 0s. Thus, this robot is located at row 0.
2. **Bit 23 to Bit 16** are 0s. Thus, this robot is located at column 0.
3. **Bit 9** is 1. Thus, this robot is facing south.
4. **Bits 2 and 0** are 1s. Thus, it has a wall on the left of the robot and in the back of the robot.

Controlling the Robot

To control the movement of the car robot, simply set the value of the register `$t8` as follows:

- **1**: move forward one block
- **2**: turn left 90 degrees

- **3**: turn right 90 degrees
- **4**: do not move, just update the status

Note that the yellow car robot cannot move backward or sideways. For example, if the car robot is facing north and you want to make it to go to the cell on its right side, you have to send the command to the car robot to turn right first. Then send another command to move forward on block.

Note that when the car robot is moving, it will ignore the command in the register `$t8`. When it stops moving and finishes update its status (the register `$t9`), it will change the value of the register `$t8` back to 0. Thus, it is the programmer responsibility to wait until the value of the register `$t8` is changed to 0 before sending the next command.

What to Do?

For this project, write a MIPS assembly program named `mazeSolver.asm`. Your program must control the yellow car robot to perform the following tasks in order (assume that the maze size is 8 by 8):

1. Navigate from the top-left corner (row 0 and column -1) to bottom-right corner (row 7 and column 8). For this task, your robot must use left-hand rule algorithm. During this task, your program should also record the direct path which will be used in the next task. After the robot reach the destination, make a 180 degree turn. **Note** the left hand rule algorithm will be explained in class.
2. Navigate from the bottom-right corner (row 7 and column 8) to top-left corner (row 0 and column -1). For this task, your robot must take the shortest path (recorded from the first task). After the robot reach the destination, make a 180 degree turn.
3. Navigate from the top-left corner (row 0 and column -1) to bottom-right corner (row 7 and column 8). For this task, your robot must use backtracking with recursion. Your car robot **MUST** try north, west, south, and then east (**in that order**). **Note** the backtracking with recursion will be explained in class.

Requirements

1. Since registers `$t8` and `$t9` are used for sending command and receiving button input, do not use these registers for any other purpose.
2. Your program must consist of at least three functions as follows:
   - `_leftHandRule`: This function, when called, it will control the car robot from the top-left corner to the bottom-right corner using left hand rule algorithm.
   - `_traceBack`: This function, when called, it will control the car robot from the bottom-right corner to the top-left corner using the direct path.
   - `_backtracking`: This function, when called, it will control the car robot from the top-left corner to the bottom-right corner using backtracking and recursion. This function **MUST** be a recursive function.
3. You are allowed to have more functions.

4. Every functions must follow the calling convention discussed in class.

Note about the size of the maze

For simplicity, the size of the maze should be hard coded to your code. For example, if you set your program to solve 8 by 8 maze, it should be able to solve only 8 by 8 maze. If you change the size of the maze in "Maze Solve Robot" (Mars tools), you also need to change the size of the maze in your program as well. This is important since you program must know the size of the maze such that it will know where is the starting point (which row and which column) and where is the destination (again, which row and which column).

Submission

Your project is due on Sunday March 22, 2015 at 11:59pm. Late submissions will not be accepted. You should submit the file `mazeSolver.asm` via CourseWeb.