K.J. Somaiya College of Engineering, Mumbai-77.

MAP Mini-Project

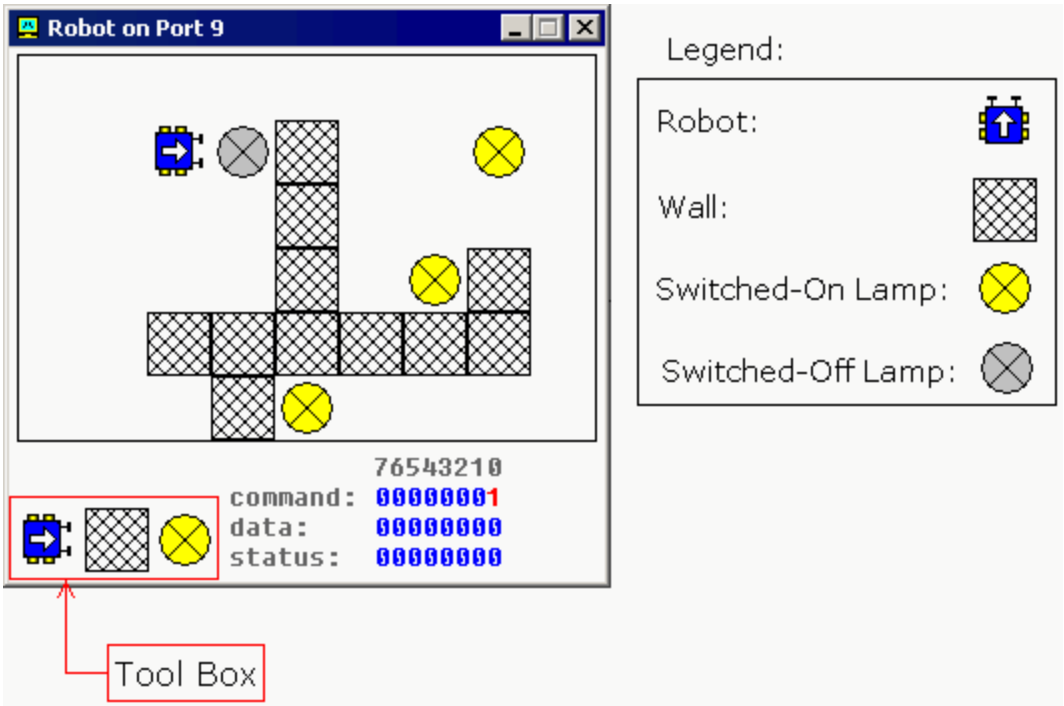
ROBOT MAZE SOLVER

Submitted by:
Roll no: 1912041 Name: Ansh Mehta
Roll no: 1912049 Name: Naisargi Doshi

Aim: To control the robot provided by emu8086 as a virtual device and solve any given maze.

Software Used: emu8086

Theory:
Emu8086 provides various i/o devices via the virtual devices option. This project uses the robot device. The robot is controlled by sending data to i/o port 9.



The first byte (port 9) is a **command register**. set values to this port to make the robot do something.

decimal value	binary value	action
0	00000000	do nothing.

1	00000001	move forward.
2	00000010	turn left.
3	00000011	turn right.
4	00000100	examine. examines an object in front using a sensor. when robot completes the task, result is set to **data register** and **bit #0** of the status **register** is set to **1**.
5	00000101	switch on a lamp.
6	00000110	switch off a lamp.

The second byte (port 10) is a **data register**. this register is set after robot completes the **examine** command:

decimal value	binary value	meaning
255	11111111	wall
0	00000000	nothing
7	00000111	switched-on lamp
8	00001000	switched-off lamp

The third byte (port 11) is a **status register**. read values from this port to determine the state of the robot. each bit has a specific property:

bit number	description

bit #0	**zero** when there is no new data in **data register**, **one** when there is new data in **data register**.
bit #1	**zero** when robot is ready for the next command, **one** when robot is busy doing some task.
bit #2	**zero** when there is no error on last command execution, **one** when there is an error on command execution (when robot cannot complete the task: move, turn, examine, switch on/off lamp).

example:

```
MOV AL, 1  ; move forward.
OUT 9, AL  ;

MOV AL, 3  ; turn right.
OUT 9, AL  ;

MOV AL, 1  ; move forward.
OUT 9, AL  ;

MOV AL, 2  ; turn left.
OUT 9, AL  ;

MOV AL, 1  ; move forward.
OUT 9, AL  ;
```

keep in mind that the robot is a mechanical creature and it takes some time for it to complete a task. you should always check bit#1 of the status **register** before sending data to port 9, otherwise the robot will reject your command and "**busy!**" will be shown. see **robot.asm** in c:\emu8086\examples.

Creating Custom Robo-World Map

It is possible to change the default map for the robot using the tool box.

if you click the robot button and place the robot over the existing robot it will turn 90 degrees counterclockwise. To manually move the robot just place it anywhere else on the map.

If you click the lamp button and click switched-on lamp the lamp will be switched-off, if lamp is already switched-off it will be deleted. clicking over empty space will create a new switched-on lamp.

Placing wall over existing wall deletes the wall.

Current version is limited to a single robot only. if you forget to place a robot on the map it will be placed in some random coordinates.

When robot device is closed the map is automatically saved inside this file:
..**emu8086\devices\robot_map.data**
It is possible to have several maps by renaming and copying this file before starting the robot device.

The right-click over the map brings up a popup menu that allows to switch-on or switch-off all the lamps at once.
The above theory was mentioned in the **documentation of emu8086 under I/O ports and Hardware Interrupts.**

Objective:
The robot provided in emu8086 virtual devices should be able to navigate any given maze given that all walls of the maze are touching at least one of the outside walls.

Procedure:

1) The documentation provided for the virtual device was understood properly.
2) Basic assembly code was written on emu8086 in accordance with the documentation for testing purposes.
3) A random maze was made for testing the algorithm used.
4) After multiple iterations, the most reliable algorithm was finalised to be used. The algorithm used was the left-hand side algorithm for maze solving. Further information about this algorithm and its implementation can be found further down.
5) Multiple mazes were used to check for errors in the algorithm.

Code:

```
;Code written by Ansh Mehta and Naisargi Doshi for Microprocessors and Peripherals Mini Project at K.J. Somaiya College of Engineering, ETRX SY B 2023

;The following code can solve any maze provided:
; The final lamp is touching a wall which can be traced upto the starting position of the robot since the algorithm follows the LHS wall and navigates accordingly
;NOTE: The code cannot work at high speeds for unknown reasons as of 29 April, 2021. Further investigation may reveal the cause and solution to this problem, until then,
;      users are requested to run the code at a step delay of a minimum of 100ms

;Any suggestions regarding better implementation can be emailed to ansh.m@somaiya.edu/anshmmehta379@gmail.com
r_port equ 9
```

```
infinite_loop:
   call turn_left
   call examine
   cmp al,0       ;check value to see if nothing is present
   je move_forward
   cmp al,255     ;check value to see if wall is present
   je alternate1
   cmp al,7
   je lamp_off

alternate1: call turn_right
   call examine
   cmp al,0       ;check value to see if nothing is present
   je move_forward
   cmp al,255     ;check value to see if wall is present
   je alternate1
   cmp al,7
   je lamp_off

lamp_on:call switch_on_lamp
   call turn_right
   call turn_right
   jmp exit

lamp_off:call switch_off_lamp
   call turn_right
   call turn_right
   jmp exit

move_forward: call move_ahead

jmp infinite_loop
exit:MOV AH, 0
   INT 21H

turn_left proc
   call wait_robot
   mov al,2
   out r_port,al
ret
turn_left endp


turn_right proc
```

```
    call wait_robot
    mov al,3
    out r_port,al
ret
turn_right endp


move_ahead proc
    call wait_robot
    mov al,1
    out r_port,al
ret
move_ahead endp


examine proc
    call wait_robot
    mov al,4
    out r_port,al
    call wait_exam
    in al,r_port+1
ret
examine endp


wait_robot proc

busy: in al, r_port+2
      test al, 00000010b
      jnz busy
ret
wait_robot endp


wait_exam proc

busy2: in al, r_port+2
       test al, 00000001b
       jz busy2
ret
wait_exam endp


switch_off_lamp proc
```

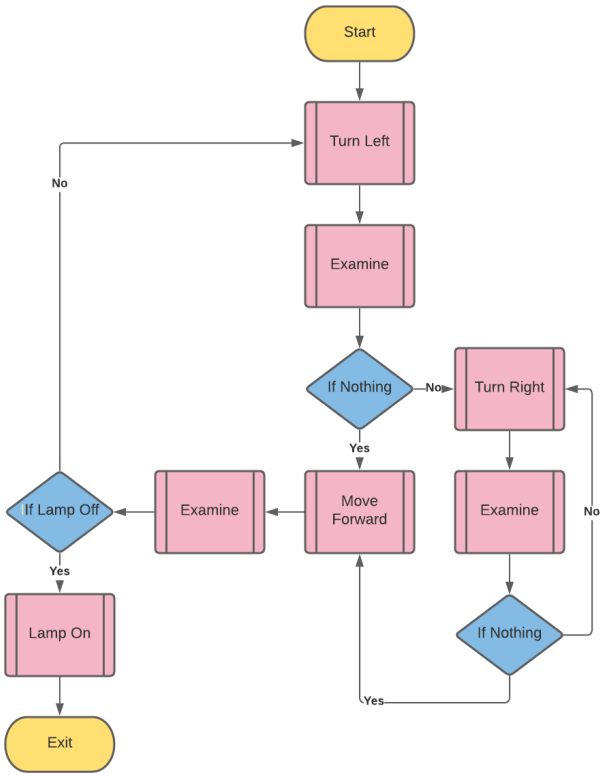
```
    call wait_robot
    mov al, 6
    out r_port, al
ret
switch_off_lamp endp


switch_on_lamp proc
    call wait_robot
    mov al, 5
    out r_port, al
ret
switch_on_lamp endp
```

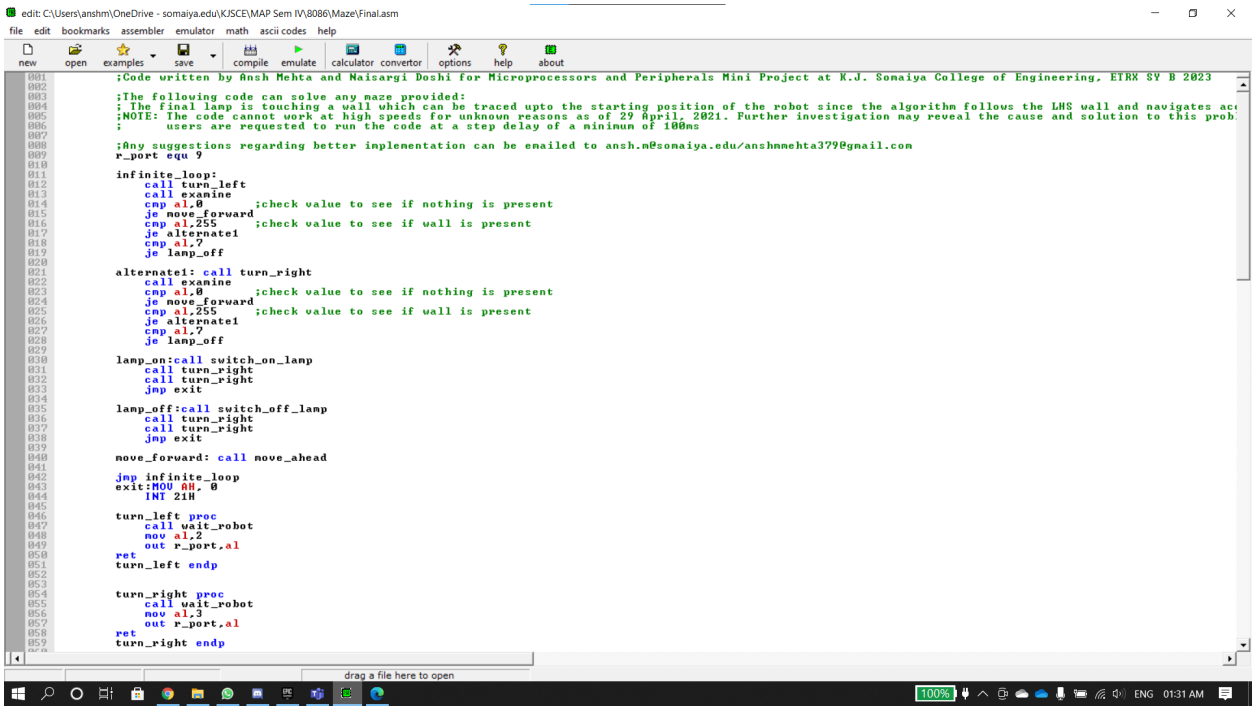
Algorithm:

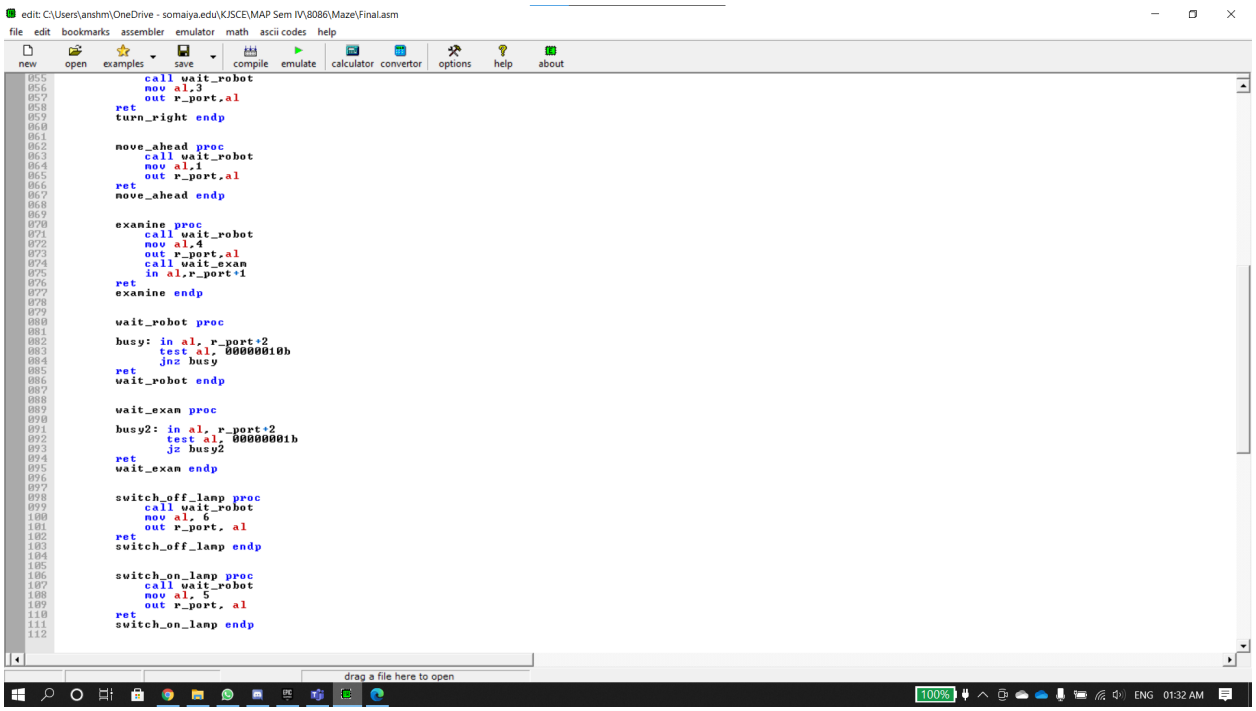
The algorithm used was LHS following for maze solving.
According to this, the maze solving entity should follow the left wall. Limitations of this algorithm are that it works only when the walls of the maze touch the outer box of the maze.

Flowchart:

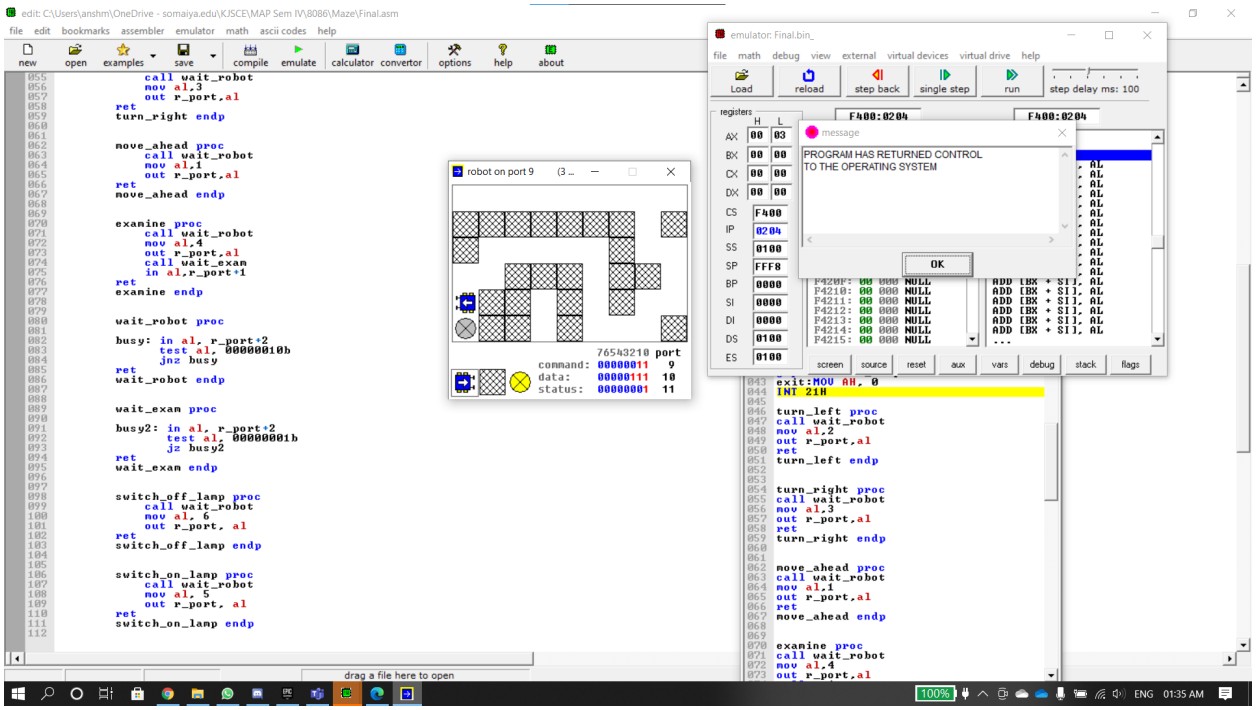


Code Screenshots:





Output:



A video showing the robot solving the maze:

Limitations:

1) The current algorithm implemented cannot solve all mazes provided to it since it requires all the walls to be touching the outer box of the arena of the robot.
2) The current code needs to be run with a step delay of 100ms due to the latency in the response of the robot.
3) The current code is not optimised for time.

Conclusion:

We conclude from this project that a robot can be controlled through a maze using an 8086 microprocessor.

We can successfully control the robot in any given random maze using the microprocessor.