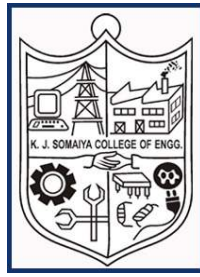




K.J. Somaiya College of Engineering, Mumbai-77.



# 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.

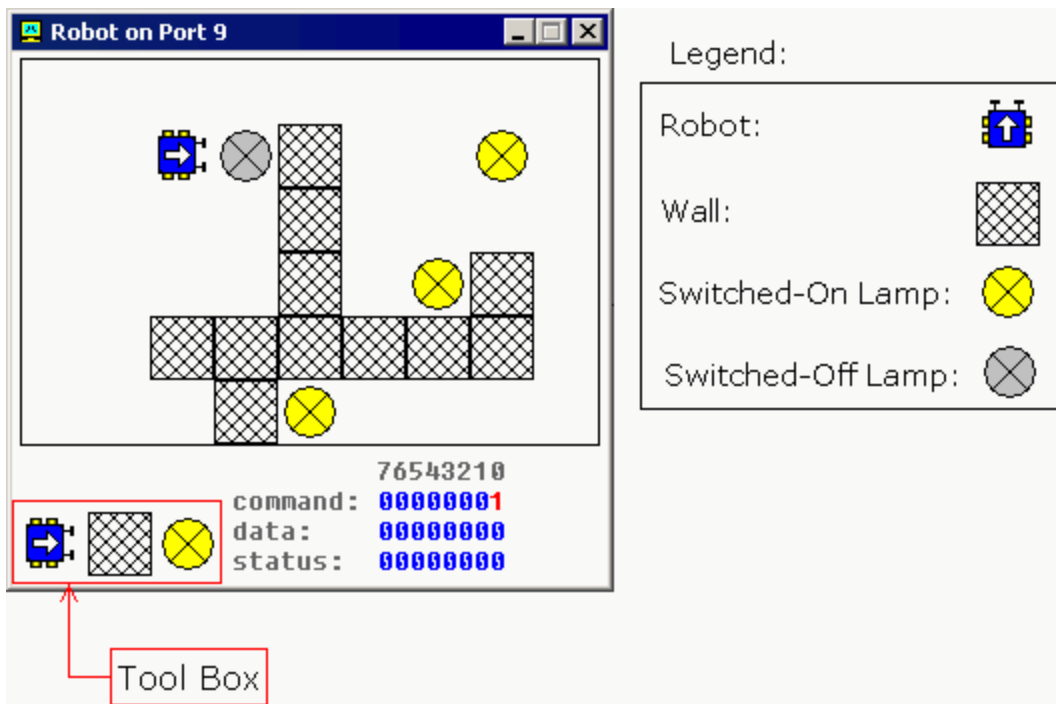
---

**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 <b>data register</b> and <b>bit #0</b> of the status <b>register</b> is set to <b>1</b> .
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
------------	-------------

<b>bit #0</b>	<b>zero</b> when there is no new data in <b>data register</b> , <b>one</b> when there is new data in <b>data register</b> .
<b>bit #1</b>	<b>zero</b> when robot is ready for the next command, <b>one</b> when robot is busy doing some task.
<b>bit #2</b>	<b>zero</b> when there is no error on last command execution, <b>one</b> 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/anshmehta379@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
```

---

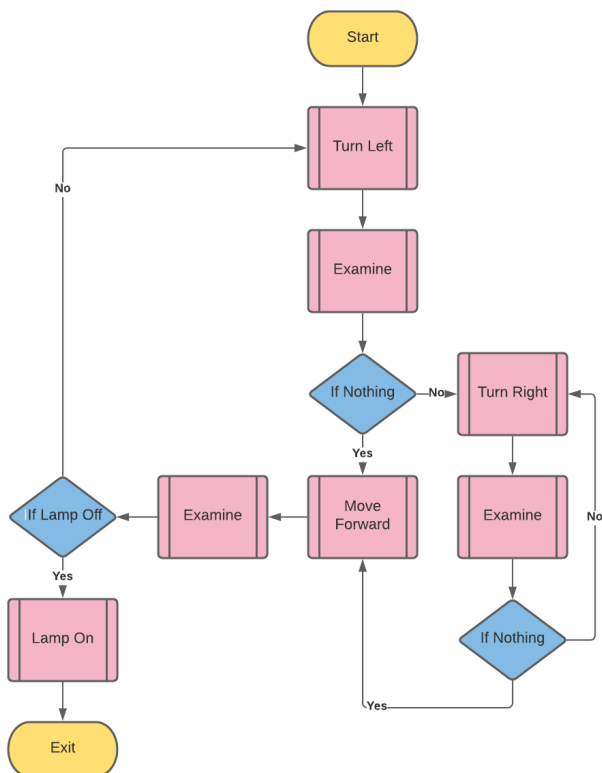
**Algorithm:**

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:**



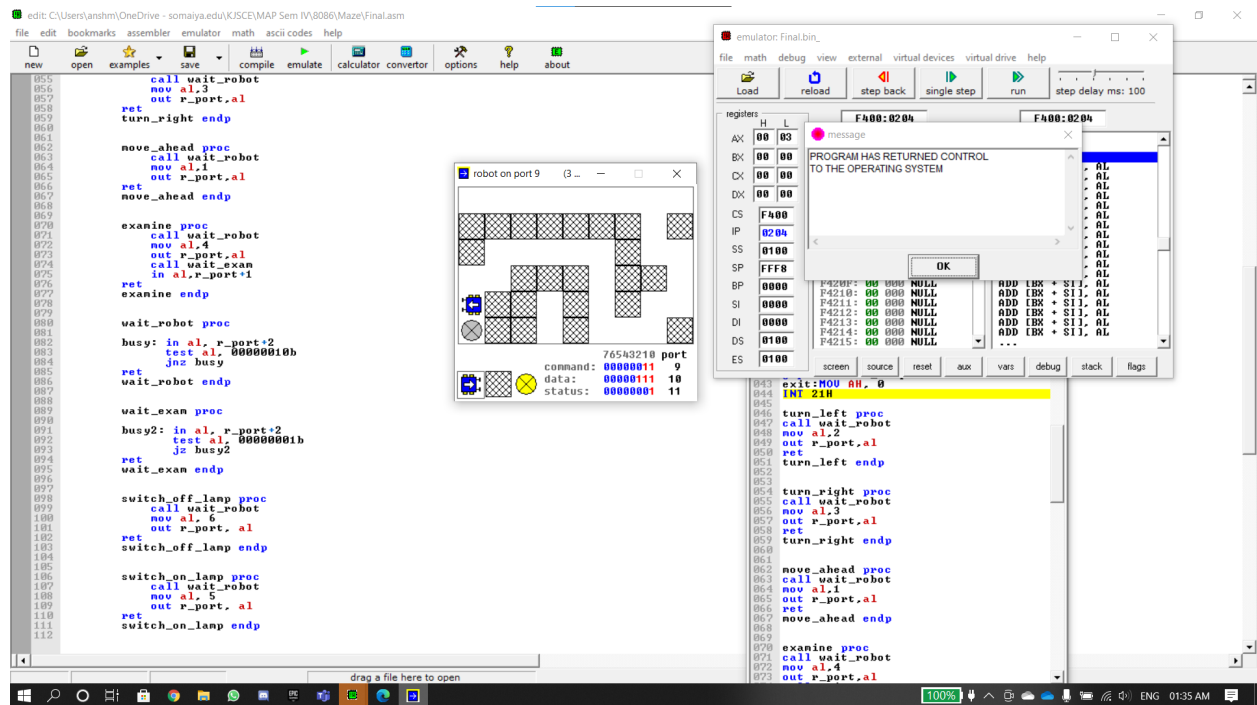


## Code Screenshots:

```
edit: C:\Users\anshm\OneDrive - somaiya.edu\KJSC\MAP Sem IV\8086\Maze\Final.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
0001 ;Code written by Ansh Mehta and Naisargi Doshi for Microprocessors and Peripherals Mini Project at K.J. Somaiya College of Engineering, ETRK SY B 2023
0002
0003 ;The following code can solve any maze provided:
0004 ;The final lamp is touching a wall which can be traced upto the starting position of the robot since the algorithm follows the LMS wall and navigates acc
0005 ;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 prob
0006 ; users are requested to run the code at a step delay of a minimum of 100ms
0007
0008 ;Any suggestions regarding better implementation can be emailed to ansh.n@somaiya.edu/anshmehta379@gmail.com
0009 r_port equ 9
0010
0011 infinite_loop:
0012     call turn_left
0013     call examine
0014     cmp al,0 ;check value to see if nothing is present
0015     je move_forward
0016     cmp al,255 ;check value to see if wall is present
0017     je alternate1
0018     cmp al,7
0019     je lamp_off
0020
0021 alternate1: call turn_right
0022     call examine
0023     cmp al,0 ;check value to see if nothing is present
0024     je move_forward
0025     cmp al,255 ;check value to see if wall is present
0026     je alternate1
0027     cmp al,7
0028     je lamp_off
0029
0030 lamp_on:call switch_on_lamp
0031     call turn_right
0032     call turn_right
0033     jmp exit
0034
0035 lamp_off:call switch_off_lamp
0036     call turn_right
0037     call turn_right
0038     jmp exit
0039
0040 move_forward: call move_ahead
0041
0042 jmp infinite_loop
0043 exit:MOV AH, 0
0044     INT 21H
0045
0046 turn_left proc
0047     call wait_robot
0048     mov al,2
0049     out r_port,al
0050     ret
0051     turn_left endp
0052
0053
0054 turn_right proc
0055     call wait_robot
0056     mov al,3
0057     out r_port,al
0058     ret
0059     turn_right endp
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
```

```
edit: C:\Users\anshm\OneDrive - somaiya.edu\KJSC\MAP Sem IV\8086\Maze\Final.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
0055     call wait_robot
0056     mov al,3
0057     out r_port,al
0058     ret
0059     turn_right endp
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
```

## 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.