PROJECT REPORT

MAZE GAME USING 8086

SUBMITTED BY:

00101012018 – AAMIYA GARG

01301012018 – ARCHITA VARSHNEY

01701012018 – SAKSHI JAIN

02801012018 – APOORVA GUPTA

02901012018 – VANSHIKA UNIYAL

B. TECH CSE-1, 5TH SEMESTER

SUBMITTED TO:

MS. NAJME ZEHRA

DEPARTMENT OF CSE

IGDTUW

INDEX

PROJECT GOAL

To implement and develop a maze game using 8086 instruction set and emu8086 assembly language.

ABOUT THE PROJECT

We have developed this game using 8086 microprocessors, it is a 16-bit microprocessor with a 20-bit address line and 16-bit data bus. It has a memory capacity of 1MB. It is a general-purpose register-based processor that supports pipelining and memory segmentation.
It is a 20 x 20 maze game wherein our motive is to reach the destination, while traveling at valid positions in the four directions (right, left, up, and down). The current position of the pointer is marked by a smiley and the path traversed is marked by diamonds. We also calculate the total number of moves taken to reach the destination.

WORKING OF THE PROJECT

- We load the instruction set of 8086.
- The program is loaded at location 100h in the RAM using **ORG** instruction.
- We initialize some set of variables using **dw** instruction.
- We initialized the msg variable with the initial message of the starting instructions that needs to be followed while playing the game.
- For drawing a 20 x 20 maze we take 20 rows with strings of 1 and 0 of length 20 where 0 and 1 represent valid and invalid positions in the maze respectively.
- Libraries included: SCAN_NUM, PRINT_STRING, PRINT_NUM, PTHIS, PRINT_NUM_UNS, CLAEAR_SCREEN
- To print the welcome message, we use the combination of **int 21h** and function code; **ah = 09h**
- Then we call clear screen function to erase the messages and start printing the maze
- To print each row, we store the effective address of the row in **si** and call the print function
- In the print function, we declare cx register as counter having value 20 and then use the compare instruction to compare the content of si and if si = 0, print a space using **putc 32** else print a wall using **putc 219** and decrement the content of cx register and move to next value in that row
- To set the starting position of the game we use **int 10h** and function code; **ah = 01h** to set text-mode cursor shape where **ch = 2bh** (scan row start) and **cl = 0bh** (scan row end)
- To select an active display page, we use **int 10h** and function code; **ah = 05h**. We select page 0.
- To set cursor position we use **int 10h** and function code; **ah = 02h**
- To make the smiley at the cursor position we use **int 10h** and function code; **ah = 09h** where **al = 01h** symbolizing the smiley, **bl = 0eh** symbolizing the yellow of the smiley, and **cx = 01h** symbolizing the number of times to print the character. Here the yellow-coloured smiley gets printed once on the cursor position.
- Here we call the subroutine **check_for_key** in which we first check the state of the keyboard buffer using **int 16h** and function code; **ah = 01h** and if we a key is pressed, we read key press using **int 16h** and function code; **ah = 00h** else jump to no_key. In case the value of al = 1bh, we jump to stop game
- After reading the key press, we store it in **curr_dir** and increment the **keypress** variable
- In the subroutine **no_key**, it keeps on waiting as well as checking whether the key is entered by calling check_for_key and adds 4 to the wait_time
- In **stop_game** we use **int 10h** and function code; **ah = 01h** to set text-mode cursor shape where **ch = 0bh** (scan row start) and **cl = 0bh** (scan row end)
- Now to decide the direction in which the smiley will move we first compare b with 19 and if we reach the end of the maze, we jump to stop
- Else we compare **curr_dir** with **left, right, up and down** and jump to **move_left, move_right, move_up, move_down** respectively
- If curr_dir matches none we jump to stop_move

Moving the player

To move the player in different directions we have 4 functions: move_left, move_right, move_up, move_down:

move_left -

- To move left we first decrement **b** (current column number).
- Then we compare **a** (current row number) with numbers from 0 to 19.
- According to the compare instructions, we jump to the respective maze move function which moves the player from the current position to the next position
- For example, if we are at row number 3, then we jump to lmaze4
- In the **lmazeX** (say, lmaze4) functions we first load the row address to **si** and add **b** to it to specify the new position of the player. Then we check if the new specified position in the maze is valid or not by comparing it with 0.
- If invalid, we jump to beepl, which produces a beep sound using **int 21h** and function code; **ah = 02h** with **dl=07h** and redos the values of row and column changed(here increment **b**).
- If valid, we call **mvlt** through which the move is made.

Move_right -

- To move left we first increment **b** (current column number).
- Rest all the steps are the same except that we jump to **rmazeX** according to the current row.
- **rmazeX** works in the same way as **lmazeX**.

Move_up -

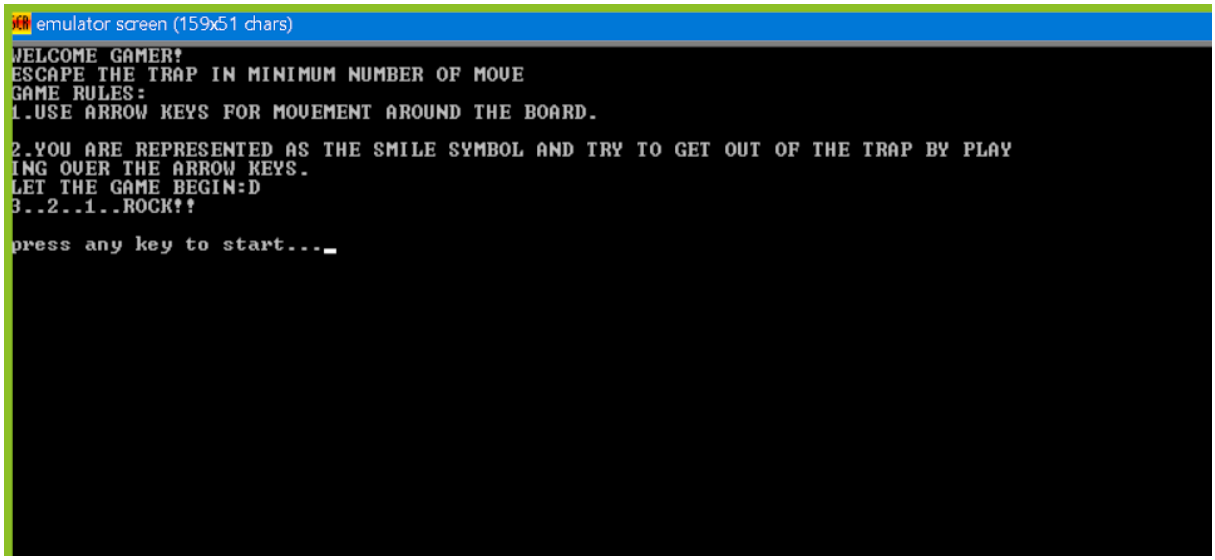
- To move left we first decrement **a** (current row number).
- Rest all the steps are the same except that we jump to **umazeX** according to the current row.
- **umazeX** works in the same way as **lmazeX**.

Move_down -

- To move left we first increment **a** (current row number).
- Rest all the steps are the same except that we jump to **dmazeX** according to the current row.
- **dmazeX** work in the way as **lmazeX**.

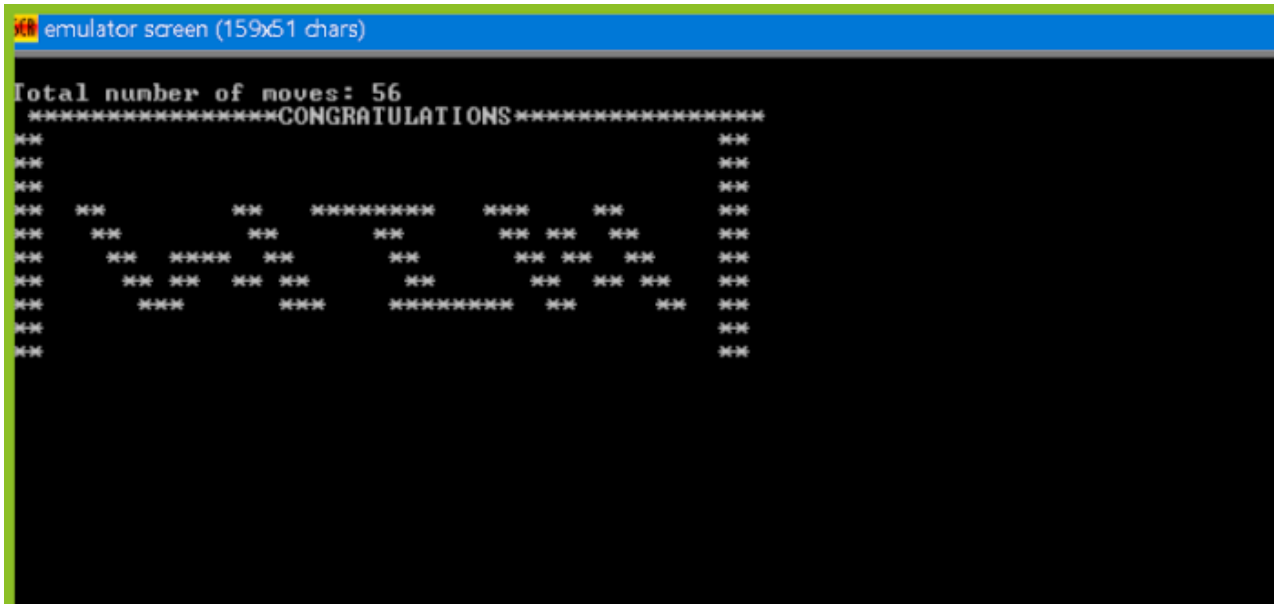
- After reaching the destination, the screen is cleared and we print the total number of moves.
- Then the winning message is displayed on the screen.

OUTPUT


emulator screen (159x51 chars)
WELCOME GAMER!
ESCAPE THE TRAP IN MINIMUM NUMBER OF MOVE
GAME RULES:
1.USE ARROW KEYS FOR MOVEMENT AROUND THE BOARD.
2.YOU ARE REPRESENTED AS THE SMILE SYMBOL AND TRY TO GET OUT OF THE TRAP BY PLAY
ING OVER THE ARROW KEYS.
LET THE GAME BEGIN:D
3..2..1..ROCK!!
press any key to start...


emulator screen (159x51 chars)


emulator screen (159x51 chars)


emulator screen (159x51 chars)
Total number of moves: 56
CONGRATULATIONS