CLASSIC PONG GAME

Following are the functions used in Project Code:

Data Movement Instructions

mov — Move

> The mov instruction copies the data item referred to by its first operand (i.e. register contents, memory contents, or a constant value) into the location referred to by its second operand (i.e. a register or memory). While register-to-register moves are possible, direct memory-to-memory moves are not. In cases where memory transfers are desired, the source memory contents must first be loaded into a register, they can be stored at the destination memory address.

push — Push on stack

> The push instruction places its operand onto the top of the hardware-supported stack in memory

pop — Pop from stack

> The pop instruction removes the data element from the top of the hardware-supported stack into the specified operand

lea — Load effective address

> The lea instruction places the *address* specified by its first operand into the register specified by its second operand.

Arithmetic and Logic Instructions

add — Integer addition

> The add instruction adds together its two operands, storing the result in its second operand.

sub — Integer subtraction

> The sub instruction stores in the value of its second operand the result of subtracting the value of its first operand from the value of its second operand.

inc, dec — Increment, Decrement

> The inc instruction increments the contents of its operand by one. The dec instruction decrements the contents of its operand by one.

neg — Negate

> Performs the two's complement negation of the operand contents.

Control Flow Instructions

jmp — Jump

> Transfers program control flow to the instruction at the memory location indicated by the operand.

cmp — Compare

> Compare the values of the two specified operands, setting the condition codes in the machine status word appropriately. This instruction is equivalent to the sub-instruction, except the result of the subtraction is discarded instead of replacing the first operand.

call, ret — Subroutine call and return

> These instructions implement a subroutine call and return. The call instruction first pushes the current code location onto the hardware-supported stack in memory (see the push instruction for details). It then performs an unconditional jump to the code location indicated by the label operand. Unlike the simple jump instructions, the call instruction saves the location to return to when the subroutine completes.
>
> The ret instruction implements a subroutine return mechanism. This instruction first pops a code location of the hardware-supported in-memory stack (see the pop instruction for details). It then performs an unconditional jump to the retrieved code location.

INT 10H

INT 10h, INT 10H, or INT 16 is shorthand for BIOS interrupts call 10_{hex}, the 17th interrupt vector in an x86-based computer system. The BIOS typically sets up a real mode interrupt handler at this vector that provides video services. Such services include setting the video mode, character and string output, and graphics primitives (reading and writing pixels in graphics mode).

Function	Function code	Parameters	Return
Set video mode	AH=00h	AL = video mode	AL = video mode flag / CRT controller mode byte
Set cursor position	AH=02h	BH = Page Number, DH = Row, DL = Column	
Set background/border color	AH=0Bh, BH = 00h	BL = Background/Border color (border only in text modes)	

DOS INT 21h - DOS Function Codes

AH = 2Ch - GET SYSTEM TIME

> Return: CH = hour CL = minute DH = second DL = 1/100 seconds
>
> on most systems, the resolution of the system clock is about 5/100sec, so returned times generally do not increment by 1 on some systems, DL may always return 00h

AH = 09h - WRITE STRING TO STANDARD OUTPUT

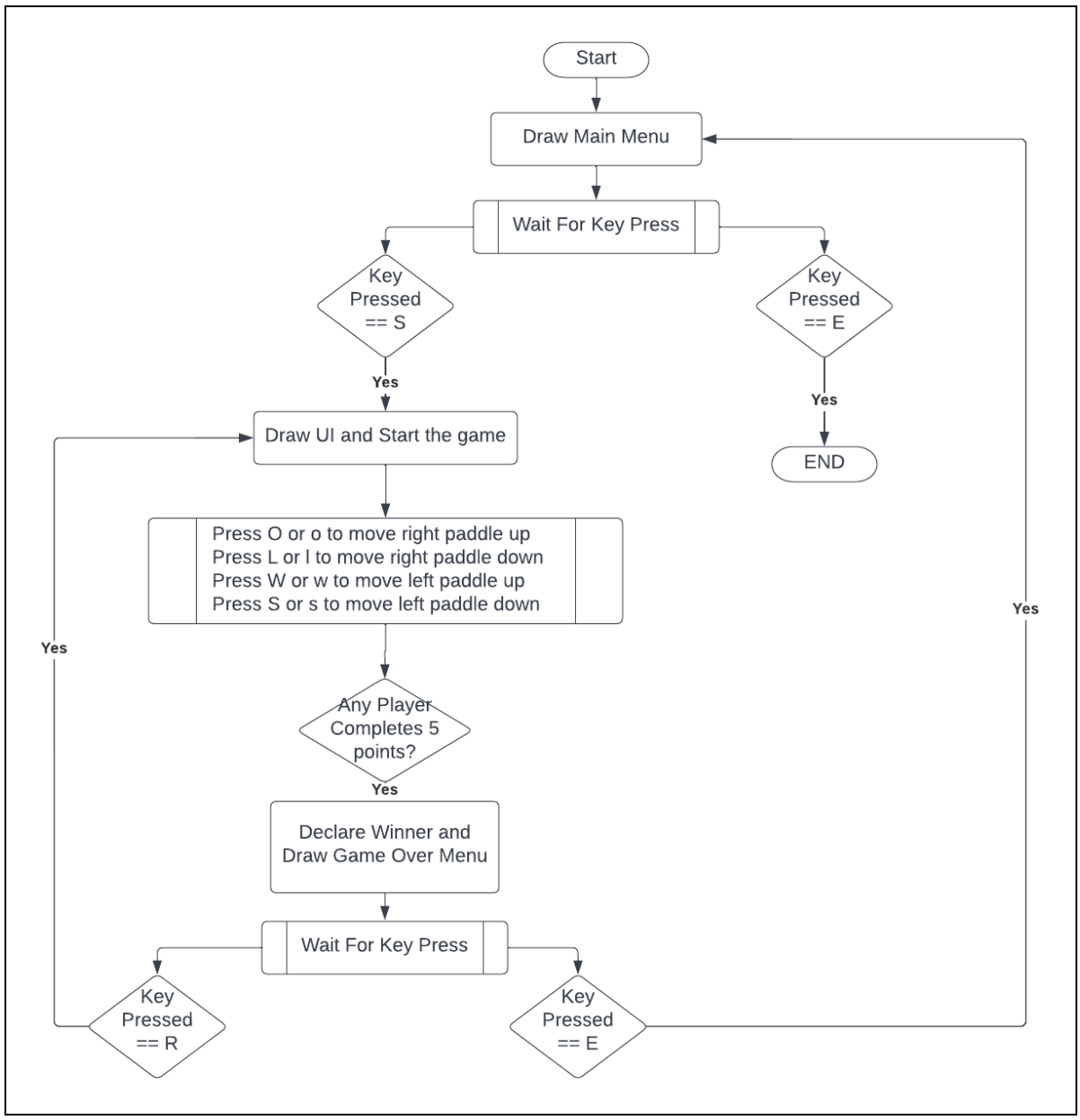
INT 16 - Keyboard BIOS Services

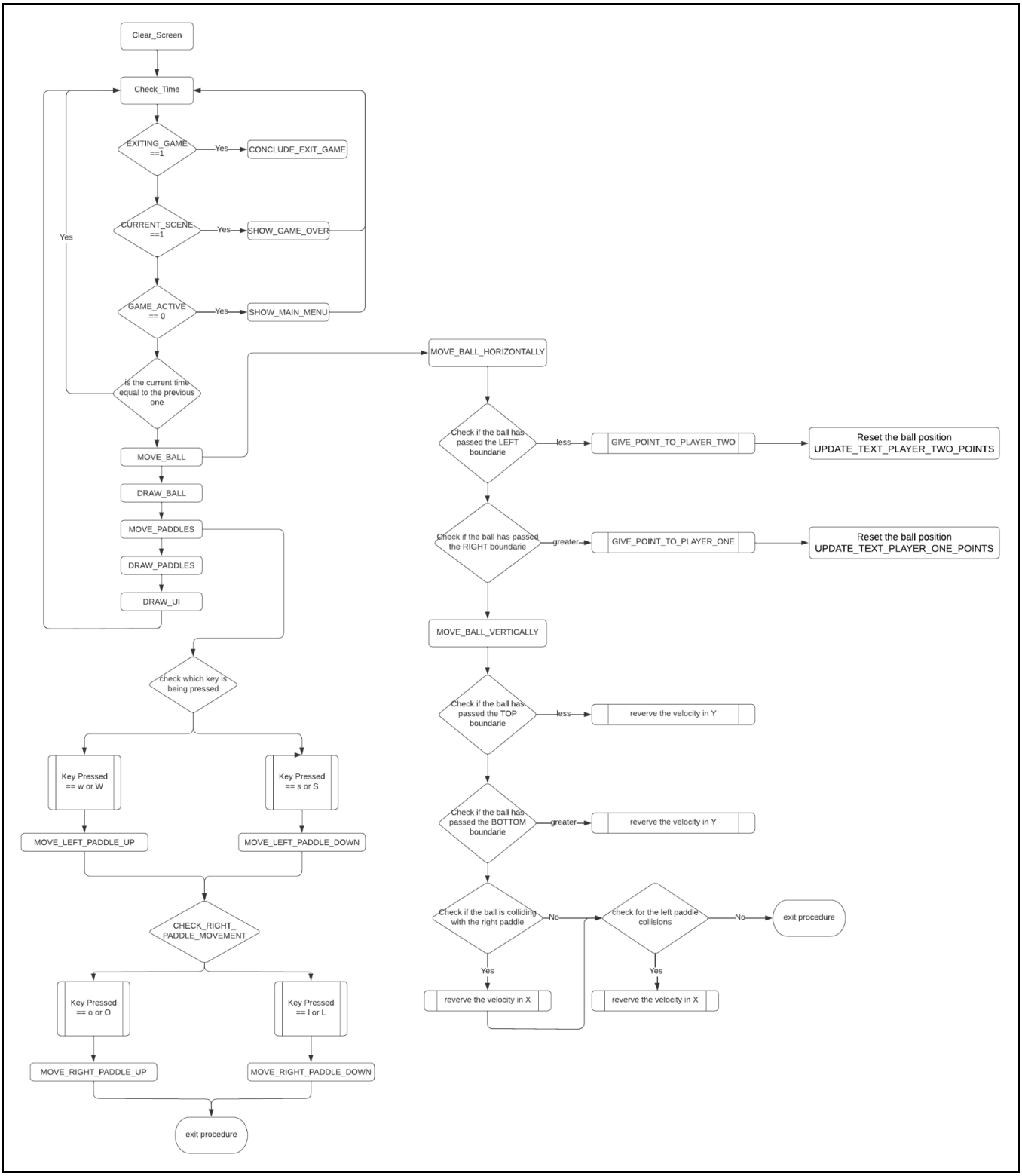
INT 16,0 Wait for keystroke and read

AH = 00

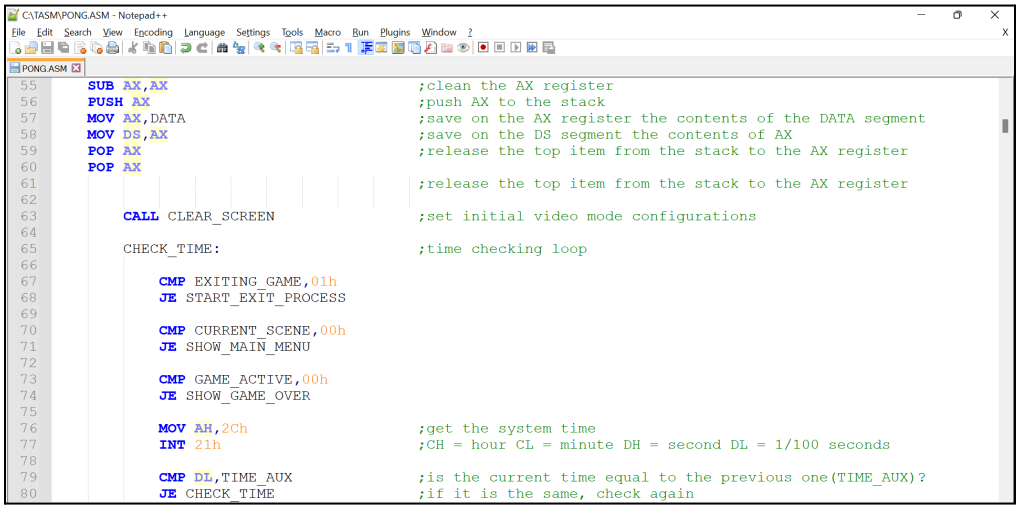
> on return:
> AH = keyboard scan code
> AL = ASCII character or zero if special function key
>
> - halts program until key with a scancode is pressed

Algorithm and Flowchart of the Game:

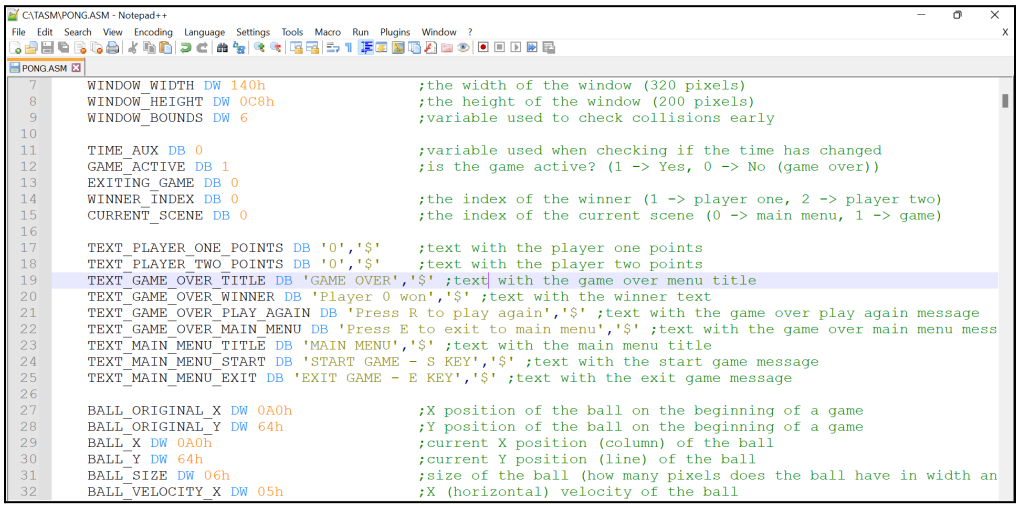



Clear_Screen
Check_Time
EXITING_GAME ==1
Yes
CONCLUDE_EXIT_GAME
CURRENT_SCENE ==1
Yes
SHOW_GAME_OVER
Yes
GAME_ACTIVE == 0
Yes
SHOW_MAIN_MENU
is the current time equal to the previous one
Yes
MOVE_BALL
DRAW_BALL
MOVE_PADDLES
DRAW_PADDLES
DRAW_UI
check which key is being pressed
Key Pressed == w or W
Key Pressed == s or S
MOVE_LEFT_PADDLE_UP
MOVE_LEFT_PADDLE_DOWN
CHECK_RIGHT_ PADDLE_MOVEMENT
Key Pressed == o or O
Key Pressed == l or L
MOVE_RIGHT_PADDLE_UP
MOVE_RIGHT_PADDLE_DOWN
exit procedure
MOVE_BALL_HORIZONTALLY
Check if the ball has passed the LEFT boundarie
less
GIVE_POINT_TO_PLAYER_TWO
Reset the ball position
UPDATE_TEXT_PLAYER_TWO_POINTS
Check if the ball has passed the RIGHT boundarie
greater
GIVE_POINT_TO_PLAYER_ONE
Reset the ball position
UPDATE_TEXT_PLAYER_ONE_POINTS
MOVE_BALL_VERTICALLY
Check if the ball has passed the TOP boundarie
less
reverve the velocity in Y
Check if the ball has passed the BOTTOM boundarie
greater
reverve the velocity in Y
Check if the ball is colliding with the right paddle
No
check for the left paddle collisions
No
exit procedure
Yes
reverve the velocity in X
Yes
reverve the velocity in X

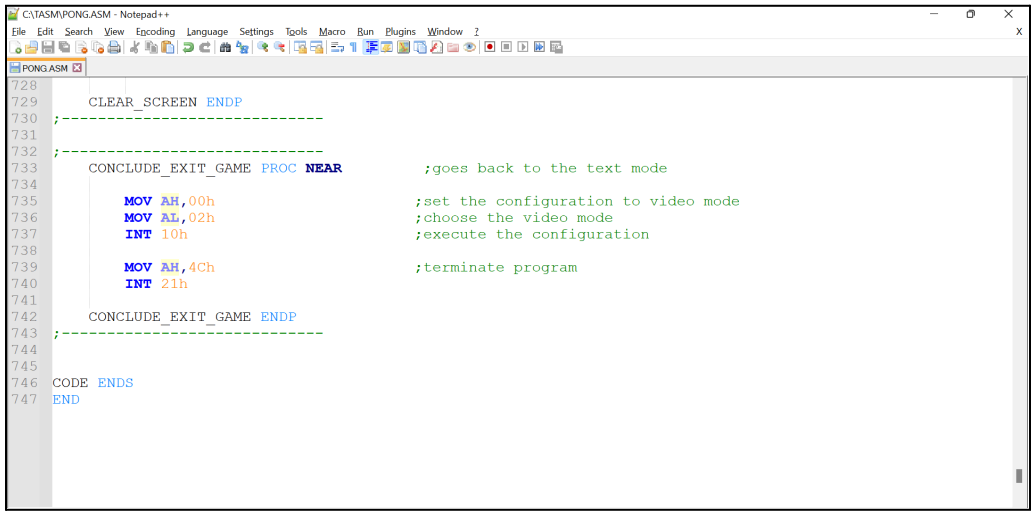
Screenshot of code:



```
55      SUB AX,AX                       ;clean the AX register
56      PUSH AX                         ;push AX to the stack
57      MOV AX,DATA                     ;save on the AX register the contents of the DATA segment
58      MOV DS,AX                       ;save on the DS segment the contents of AX
59      POP AX                          ;release the top item from the stack to the AX register
60      POP AX
61                                      ;release the top item from the stack to the AX register
62
63          CALL CLEAR_SCREEN           ;set initial video mode configurations
64
65          CHECK_TIME:                 ;time checking loop
66
67              CMP EXITING_GAME,01h
68              JE START_EXIT_PROCESS
69
70              CMP CURRENT_SCENE,00h
71              JE SHOW_MAIN_MENU
72
73              CMP GAME_ACTIVE,00h
74              JE SHOW_GAME_OVER
75
76              MOV AH,2Ch              ;get the system time
77              INT 21h                 ;CH = hour CL = minute DH = second DL = 1/100 seconds
78
79              CMP DL,TIME_AUX         ;is the current time equal to the previous one(TIME_AUX)?
80              JE CHECK_TIME           ;if it is the same, check again
```



```
7       WINDOW_WIDTH DW 140h            ;the width of the window (320 pixels)
8       WINDOW_HEIGHT DW 0C8h           ;the height of the window (200 pixels)
9       WINDOW_BOUNDS DW 6              ;variable used to check collisions early
10
11      TIME_AUX DB 0                   ;variable used when checking if the time has changed
12      GAME_ACTIVE DB 1                ;is the game active? (1 -> Yes, 0 -> No (game over))
13      EXITING_GAME DB 0
14      WINNER_INDEX DB 0               ;the index of the winner (1 -> player one, 2 -> player two)
15      CURRENT_SCENE DB 0              ;the index of the current scene (0 -> main menu, 1 -> game)
16
17      TEXT_PLAYER_ONE_POINTS DB '0','$'   ;text with the player one points
18      TEXT_PLAYER_TWO_POINTS DB '0','$'   ;text with the player two points
19      TEXT_GAME_OVER_TITLE DB 'GAME OVER','$' ;text with the game over menu title
20      TEXT_GAME_OVER_WINNER DB 'Player 0 won','$' ;text with the winner text
21      TEXT_GAME_OVER_PLAY_AGAIN DB 'Press R to play again','$' ;text with the game over play again message
22      TEXT_GAME_OVER_MAIN_MENU DB 'Press E to exit to main menu','$' ;text with the game over main menu mess
23      TEXT_MAIN_MENU_TITLE DB 'MAIN MENU','$' ;text with the main menu title
24      TEXT_MAIN_MENU_START DB 'START GAME - S KEY','$' ;text with the start game message
25      TEXT_MAIN_MENU_EXIT DB 'EXIT GAME - E KEY','$' ;text with the exit game message
26
27      BALL_ORIGINAL_X DW 0A0h         ;X position of the ball on the beginning of a game
28      BALL_ORIGINAL_Y DW 64h          ;Y position of the ball on the beginning of a game
29      BALL_X DW 0A0h                  ;current X position (column) of the ball
30      BALL_Y DW 64h                   ;current Y position (line) of the ball
31      BALL_SIZE DW 06h                ;size of the ball (how many pixels does the ball have in width an
32      BALL_VELOCITY_X DW 05h          ;X (horizontal) velocity of the ball
```



```
728
729     CLEAR_SCREEN ENDP
730 ;-----------------------------
731
732 ;-----------------------------
733     CONCLUDE_EXIT_GAME PROC NEAR        ;goes back to the text mode
734
735         MOV AH,00h                      ;set the configuration to video mode
736         MOV AL,02h                      ;choose the video mode
737         INT 10h                         ;execute the configuration
738
739         MOV AH,4Ch                      ;terminate program
740         INT 21h
741
742     CONCLUDE_EXIT_GAME ENDP
743 ;-----------------------------
744
745
746 CODE ENDS
747 END
```

Screenshot of Output: