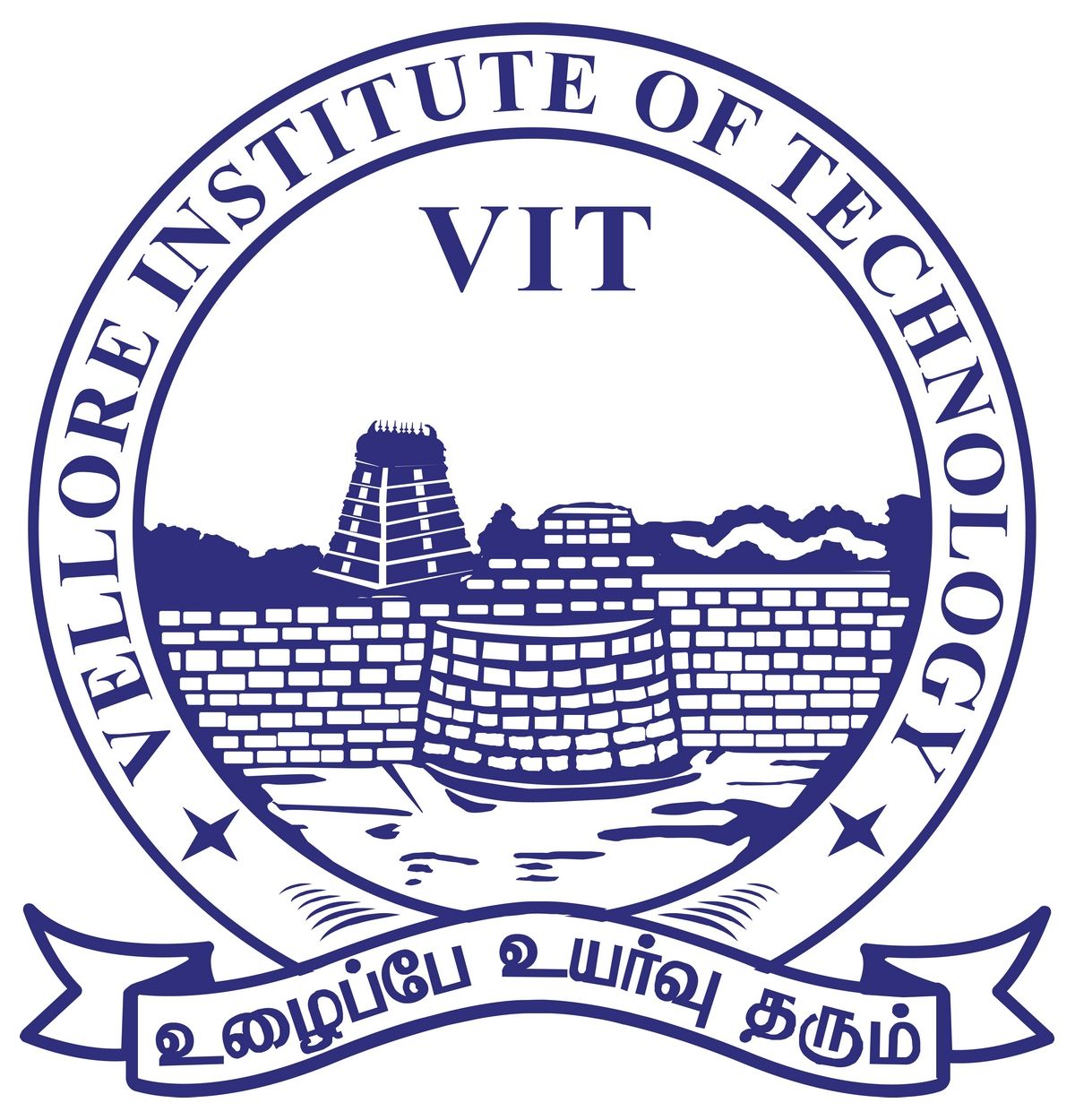
**ECE3003 - MICROCONTROLLERS & ITS APPLICATIONS**



PROJECT REPORT

**THE PONG GAME**

SUBMITTED BY:

T. TEJAS - 18BEC0396 (L33+34)

K. AKSHAR - 18BEC0863 (L33+34)

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| S. No. | TOPIC | PAGE |
| 1. | Abstract | 3 |
| 2. | Introduction | 3 |
| 3. | Overview of hardware | 4 |
| 4. | Project Specifications | 4 |
| 5. | Flow Chart of the Code | 5 |
| 6. | Code | 6 |
| 7. | Step-by-Step Report | 17 |
| 8. | Running the Game (DOSBOX Commands) | 22 |
| 9. | Results | 24 |
| 10. | Conclusion | 24 |
| 11. | References | 25 |

**ABSTRACT**

The aim of our project is to use 8086 assembly language, its interrupt services and an x86 DOSBOX emulator to create a legacy Pong game. It is a two-dimensional sports game that simulates table tennis. The player controls an in-game paddle by moving it vertically across the left or right side of the screen. It can also be played by two players who can control a paddle each using the keyboard. Players use the paddles to hit a ball back and forth. The goal is for each player to return the ball to the opponent successfully for as long as possible. The one who misses the ball first, loses.

We use three 8086 interrupt services out of the 256 available: INT 10H (for creating a new screen which provides video services, INT 16H (for allowing keyboard inputs to control the paddles on the screen), INT 21H (for using system time information to update the screen frame at regular intervals).

This project is completely based on software except for the keyboard inputs which the players will use to control the paddles.

**INTRODUCTION**

Video Games have become an essential part of any 21st Century kid, whether they be on mobile, PC or a console. Video Games have seen huge growth in the last 3 decades and are now truly unbelievable in terms of the overall experience they provide. But, the game industry had humble beginnings with the creation of games like Pong and Mario, written purely in assembly language. So, for our project, we have decided to create our own version of Pong written purely in Assembly Language as a tribute to the game developers of the days of yore.

This report contains details of the steps we went through to make this game and also highlights the issues we faced at each step. This game can be run on any system with the DOSBOX emulator and requires practically zero computing power due to its lightweight and super-fast execution times due to it being written in assembly language.

**OVERVIEW OF THE HARDWARE COMPONENTS**

We have used no hardware components except the keyboard which sends players’ inputs to the game. We have used an 8086 emulator which is a software that enables one computer system (called the host) to behave like another computer system (called the guest).

The host is the operating system in our laptops i.e. Windows 10, whereas the guest is the operating system being run on the host system i.e. DOS (which emulates the 8086 hardware functions).

GAME

SCREEN

KEYBOARD

**PROJECT SPECIFICATIONS**

**Software Specifications**

* Emulator: DOSBOX (A program to emulate an IBM PC running a DOS Operating System. Here, in this case, it is used to emulate an 8086 microprocessor.)
* Assembler: MASM (Microsoft Macro Assembler)
* Text Editor: Notepad++ (or any other adequate text editor)

**Hardware Specifications**

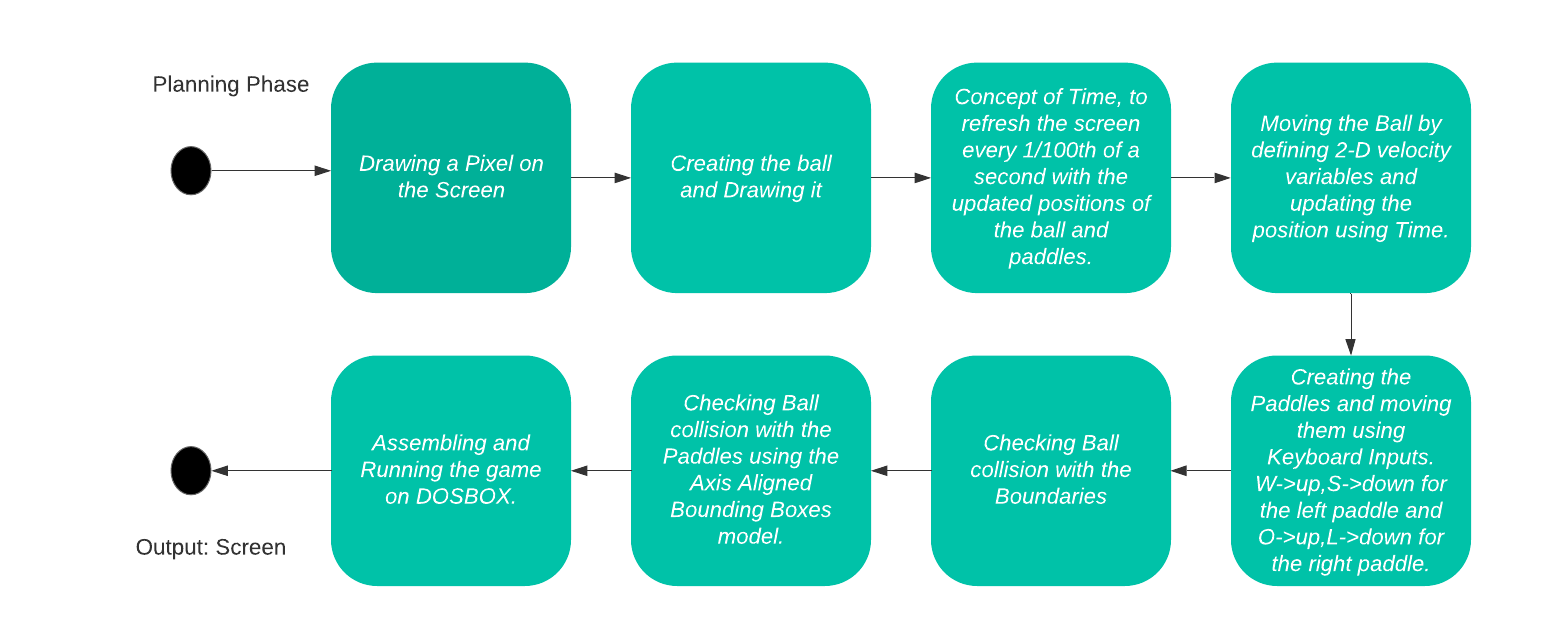
* Keyboard for key input.

**Technical Details**

1. We will use three Interrupts in the x86 instruction set (10H, 16H & 21H), and AH and AL 8-bit registers to call various services through these interrupts.
2. INT 10H provides video services. The values of AH and AL together are used for calling specific functions like setting the video mode, choosing color output etc. We use it for graphics rendering.
3. INT 16H provides I/O operation services. It helps in updating output based on keyboard inputs.
4. INT 21H provides system time information. We use it as the basis for screen refresh at 100fps. More the fps, smoother the graphical transitions.

**Process**

1. Creating a blank screen.
2. Drawing a pixel on the blank screen using Interrupt 10H.
3. Drawing a ball by accumulating multiple pixels.
4. Moving the ball by assigning a preset 2-D velocity.
5. Refreshing the Screen at 100FPS with the updated position of the ball.
6. Creating two Paddles and enabling them to move in the Y axis based on the Keyboard input.
7. Collision Detection between the ball and the paddle by using the Axis Aligned Bounding Boxes Collision Model.

**FLOW CHART OF THE CODE**

**CODE**

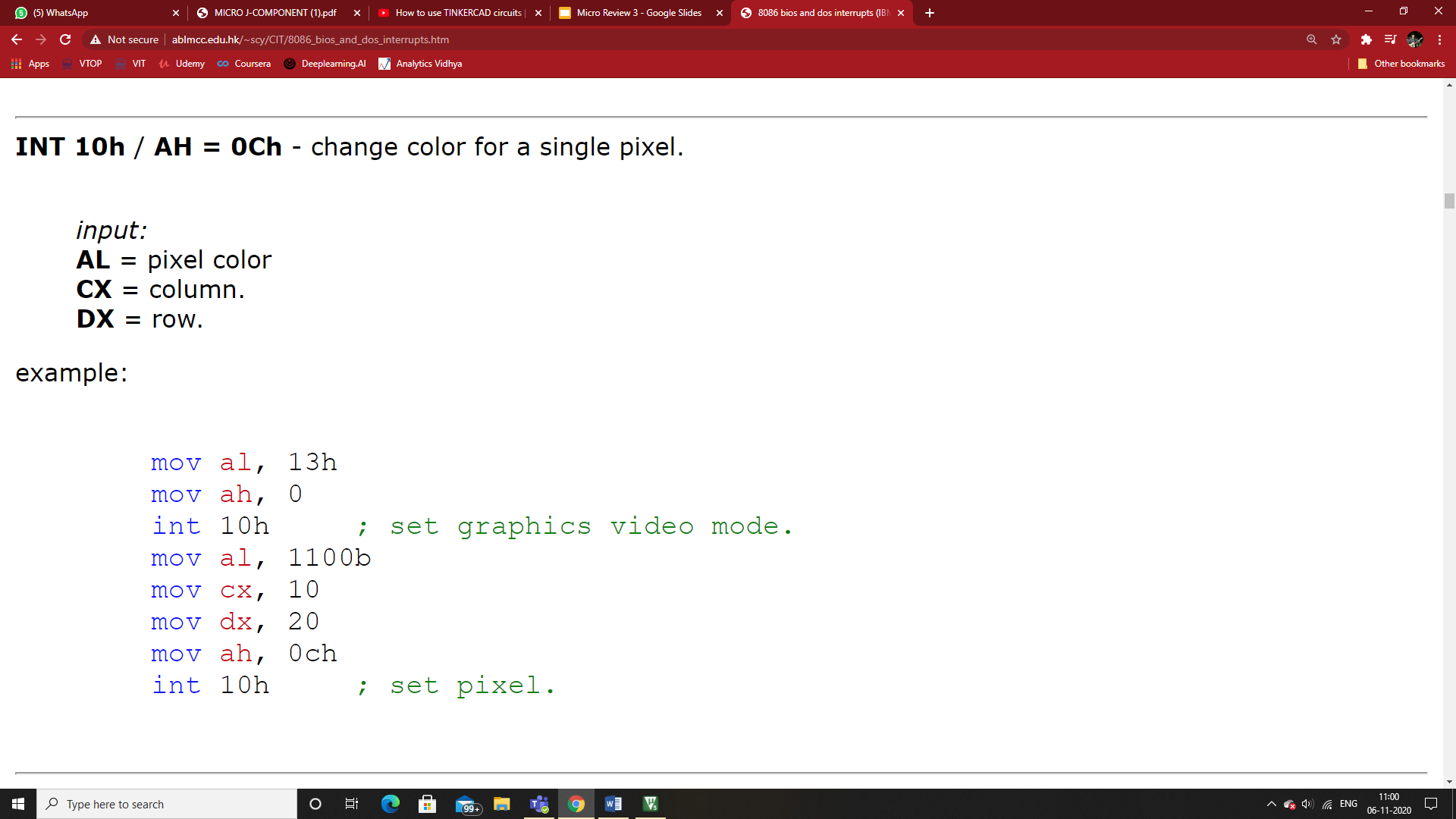
This code has been written by us entirely with reference from Wikipedia pages, documentation and domain specific resources about various interrupts and how to use them for each task as after that, the implementation is very straightforward.

The assembly directives used in our code:

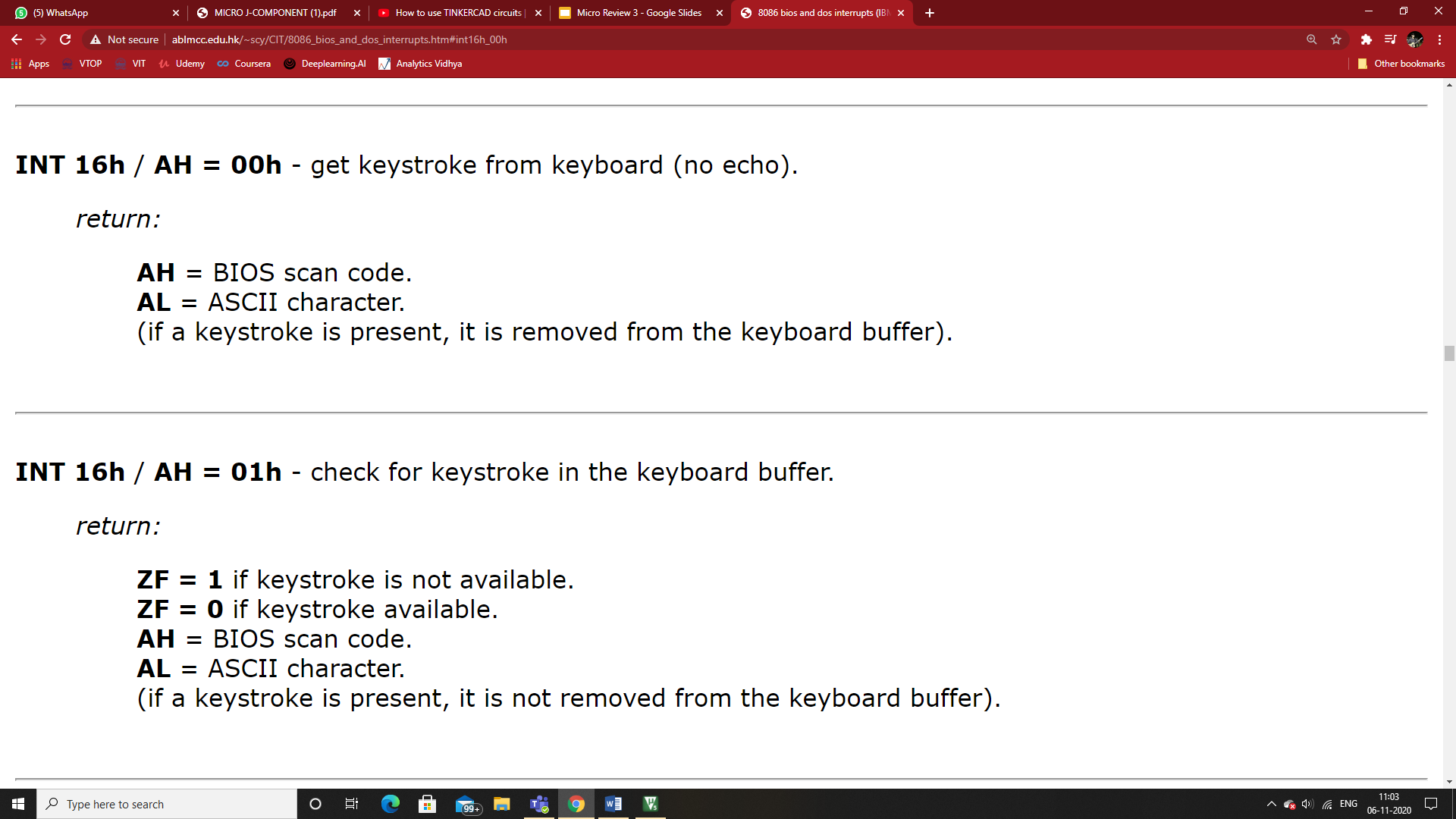
* DUP: to initialize several locations at once (duplicate)
* DB/DW: *Define Byte* (declares 8-bit variable) and *Define Word* (declares 16-bit variable)
* SEGMENT: indicates the start of a logical segment.
* PROC: identifies the start of a procedure.
* PARA: used to align segment on paragraph boundary

The links for these resources have been given in the ‘References’ section at the end of this report. Below are the codes of the interrupt services and the formulae we referred to online:

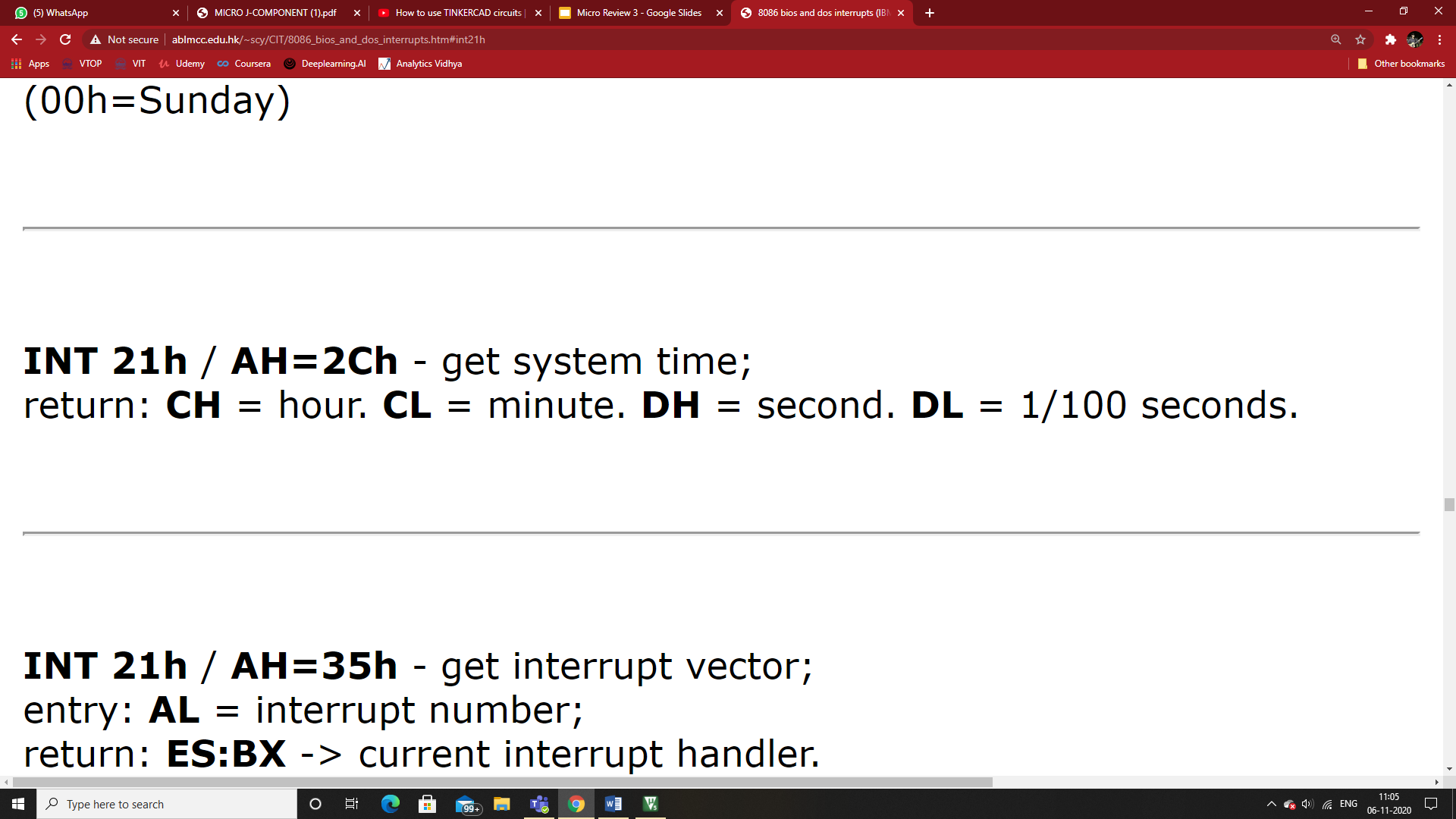
For *INT 10H*



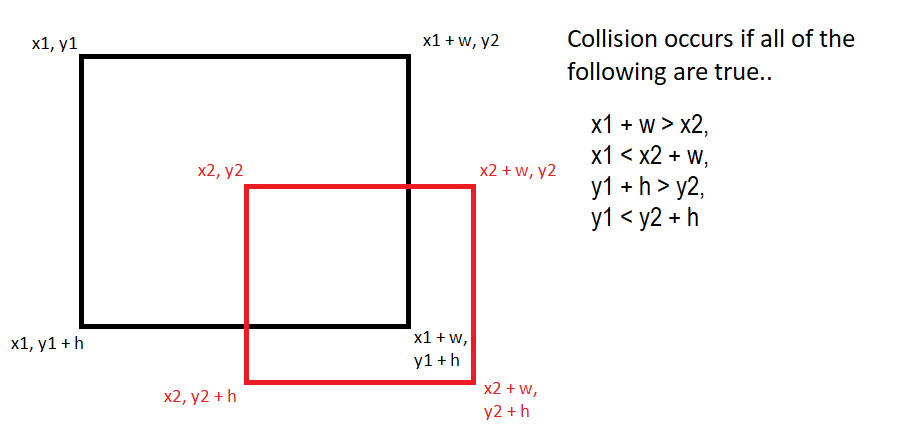
*For INT 16H*



*For INT 21H*



*Collision Detection Formula*



FINAL CODE

|  |
| --- |
| STACK SEGMENT PARA STACK                            ;start by creating a stack segment      DB 64 DUP (' ')                                 ;fill the stack with 64 empty spaces.  STACK ENDS  DATA SEGMENT PARA 'DATA'                            ;segment where we will save our variables      TIME\_AUX DB 0      WINDOW\_WIDTH DW 140h                            ;the width of the window (320 pixels)      WINDOW\_HEIGHT DW 0C8h                           ;the height of the window (200 pixels)      WINDOW\_BOUNDS DW 6                              ;variable used to check collisions early      BALL\_ORIGINAL\_X DW 0A0h                         ;x position of the ball at the beginning of the game.      BALL\_ORIGINAL\_Y DW 64h      BALL\_X DW 0A0h                                  ;declaring the ball variables here (x and y posn's of the ball)      BALL\_Y DW 64h       ;DWcan store 16 bits of info. We use it because we are using 16 bits registers.                                        ;DB can only store 8 bits of data.      BALL\_SIZE DW 04h      ;size of the ball, 4 pixels on x(width) and on y(height). 16 pixels in total.      BALL\_VELOCITY\_X DW 05h      BALL\_VELOCITY\_Y DW 02h        PADDLE\_LEFT\_X DW 0Ah      PADDLE\_LEFT\_Y DW 0Ah        PADDLE\_RIGHT\_X DW 130h      PADDLE\_RIGHT\_Y DW 0Ah        PADDLE\_WIDTH DW 05h      PADDLE\_HEIGHT DW 1Fh      PADDLE\_VELOCITY DW 05h  DATA ENDS  CODE SEGMENT PARA 'CODE'        MAIN PROC FAR                                   ;creating our main procedure(main function jaise)        ASSUME CS:CODE,DS:DATA,SS:STACK                 ;to clearly define the segments of our program.      ;code segment,data segment,stack segment.      ;assembler assumes that these are called code,data and stack.      ;useful for telling the assembler where to get the variables from(where they have been defined).      ;rather than using random garbage values.        PUSH DS                                         ;push to the stack the DS segment.      SUB AX,AX                                       ;clean the AX Register by subtracting it with itself      PUSH AX                                         ;push ax to the stack      MOV AX,DATA                                     ;save on the AX Register the contents of the DATA segment      MOV DS,AX                                       ;save on the data segment,the contents of AX.      POP AX                                          ;release the top item from the stack to the register      POP AX        CALL CLEAR\_SCREEN        CHECK\_TIME:          MOV AH,2Ch                                  ;set the configuration to get the system time. 2Ch is the function code on AH to get the system time.          INT 21h                                     ;we will use the interruption 21h. CH = hour CL = minute DH = second DL = 1/100 seconds          CMP DL,TIME\_AUX                             ;check if the current time is equal to the previous one          JE CHECK\_TIME                               ;JE stands for "Jump if condition is met". If DL and TIME\_AUX are the same,check again. If not the same,then draw ball,move etc.            MOV TIME\_AUX,DL                             ;Update the value of time\_aux with the new value of dl. basically,update the time.            CALL CLEAR\_SCREEN          CALL MOVE\_BALL          CALL DRAW\_BALL            CALL MOVE\_PADDLES          CALL DRAW\_PADDLES            JMP CHECK\_TIME          RET        MAIN ENDP      MOVE\_BALL PROC NEAR            MOV AX,BALL\_VELOCITY\_X          ADD BALL\_X,AX                               ;move the ball horizontally            MOV AX,WINDOW\_BOUNDS          CMP BALL\_X,AX          JL RESET\_POSITION                           ;BALL\_X < 0 + WINDOW\_BOUNDS (Y -> collided)            MOV AX,WINDOW\_WIDTH          SUB AX,BALL\_SIZE          SUB AX,WINDOW\_BOUNDS          CMP BALL\_X,AX                               ;BALL\_X > WINDOW\_WIDTH - BALL\_SIZE  - WINDOW\_BOUNDS (Y -> collided)          JG RESET\_POSITION            MOV AX,BALL\_VELOCITY\_Y          ADD BALL\_Y,AX                               ;move the ball vertically            MOV AX,WINDOW\_BOUNDS          CMP BALL\_Y,AX                               ;BALL\_Y < 0 + WINDOW\_BOUNDS (Y -> collided)          JL NEG\_VELOCITY\_Y            MOV AX,WINDOW\_HEIGHT          SUB AX,BALL\_SIZE          SUB AX,WINDOW\_BOUNDS          CMP BALL\_Y,AX          JG NEG\_VELOCITY\_Y                           ;BALL\_Y > WINDOW\_HEIGHT - BALL\_SIZE - WINDOW\_BOUNDS (Y -> collided)            ;here we check if the ball is colliding with the right paddle          ;maxx1>minx2 && minx1<maxx2 && maxy1>miny2 && miny1<maxy2. Condition for collision.          ;therefore BALL\_X+BALL\_SIZE>PADDLE\_RIGHT\_X && BALL\_X<PADDLE\_RIGHT\_X+PADDLE\_WIDTH && BALL\_Y+BALL\_SIZE>PADDLE\_RIGHT\_Y  && BALL\_Y<PADDLE\_RIGHT\_Y+PADDLE\_HEIGHT            MOV AX,BALL\_X          ADD AX,BALL\_SIZE          CMP AX,PADDLE\_RIGHT\_X          JNG CHECK\_COLLISION\_WITH\_LEFT\_PADDLE        ;jump if not greater,check if collision with left paddle          MOV AX,PADDLE\_RIGHT\_X          ADD AX,PADDLE\_WIDTH          CMP BALL\_X,AX          JNL CHECK\_COLLISION\_WITH\_LEFT\_PADDLE            MOV AX,BALL\_Y          ADD AX,BALL\_SIZE          CMP AX,PADDLE\_RIGHT\_Y          JNG CHECK\_COLLISION\_WITH\_LEFT\_PADDLE            MOV AX,PADDLE\_RIGHT\_Y          ADD AX,PADDLE\_HEIGHT          CMP BALL\_Y,AX          JNL CHECK\_COLLISION\_WITH\_LEFT\_PADDLE          ;if it reaches this point,means the ball collided with the right paddle.          NEG BALL\_VELOCITY\_X          RET            RESET\_POSITION:              CALL RESET\_BALL\_POSITION              RET            NEG\_VELOCITY\_Y:              NEG BALL\_VELOCITY\_Y                     ;BALL\_VELOCITY\_Y = - BALL\_VELOCITY\_Y              RET          ;here we check if the ball is colliding with the left paddle          ;maxx1>minx2 && minx1<maxx2 && maxy1>miny2 && miny1<maxy2. Condition for collision.          ;therefore BALL\_X+BALL\_SIZE>PADDLE\_LEFT\_X && BALL\_X<PADDLE\_LEFT\_X+PADDLE\_WIDTH && BALL\_Y+BALL\_SIZE>PADDLE\_LEFT\_Y  && BALL\_Y<PADDLE\_LEFT\_Y+PADDLE\_HEIGHT          CHECK\_COLLISION\_WITH\_LEFT\_PADDLE:              MOV AX,BALL\_X              ADD AX,BALL\_SIZE              CMP AX,PADDLE\_LEFT\_X              JNG EXIT\_BALL\_MOVEMENT                  ;jump if not greater,check if collision with left paddle              MOV AX,PADDLE\_LEFT\_X              ADD AX,PADDLE\_WIDTH              CMP BALL\_X,AX              JNL EXIT\_BALL\_MOVEMENT                MOV AX,BALL\_Y              ADD AX,BALL\_SIZE              CMP AX,PADDLE\_LEFT\_Y              JNG EXIT\_BALL\_MOVEMENT                MOV AX,PADDLE\_LEFT\_Y              ADD AX,PADDLE\_HEIGHT              CMP BALL\_Y,AX              JNL EXIT\_BALL\_MOVEMENT              ;if it reaches this point,means the ball collided with the left paddle.              NEG BALL\_VELOCITY\_X              RET                EXIT\_BALL\_MOVEMENT:                  RET              RET        MOVE\_BALL ENDP        MOVE\_PADDLES PROC NEAR          MOV AH,01h                                   ;to check keystroke status          INT 16h          JZ CHECK\_RIGHT\_PADDLE\_MOVEMENT               ;JUMP if ZERO FLAG(of Ah=01) is set. Meaning,if ZF=1,this means a key is being pressed.          ;left paddle movement          ;check if any key is being pressed(if not then exit procedure)          ;if being pressed,then check which key is being pressed(AL will store the ASCII character of the key being pressed)          MOV AH,00h                                   ;to read which key has been pressed          INT 16h          ;if it is 'w' or 'W', move up          CMP AL,77h                                   ;for 'w'. ASCII value of 'w' in hexadecimal          JE MOVE\_LEFT\_PADDLE\_UP          CMP AL,57h                                   ;for 'W'.          JE MOVE\_LEFT\_PADDLE\_UP            ;if it is 's' or 'S', move down          CMP AL,73h                                   ;for 's'. ASCII value of 's' in hexadecimal          JE MOVE\_LEFT\_PADDLE\_DOWN          CMP AL,53h                                   ;for 'S'.          JE MOVE\_LEFT\_PADDLE\_DOWN            JMP CHECK\_RIGHT\_PADDLE\_MOVEMENT              ;if it reaches here,it means that they key pressed was neither w nor s            MOVE\_LEFT\_PADDLE\_UP:              MOV AX,PADDLE\_VELOCITY              SUB PADDLE\_LEFT\_Y,AX              MOV AX,WINDOW\_BOUNDS              CMP PADDLE\_LEFT\_Y,AX              JL FIX\_PADDLE\_LEFT\_TOP\_POSITION          ;jumps if paddle left y is lesser than ax. 'y' position is 0 on top,AND IT INCREASES as we go down.              JMP CHECK\_RIGHT\_PADDLE\_MOVEMENT            FIX\_PADDLE\_LEFT\_TOP\_POSITION:              MOV PADDLE\_LEFT\_Y,AX              JMP CHECK\_RIGHT\_PADDLE\_MOVEMENT            MOVE\_LEFT\_PADDLE\_DOWN:              MOV AX,PADDLE\_VELOCITY              ADD PADDLE\_LEFT\_Y,AX              MOV AX,WINDOW\_HEIGHT              SUB AX,WINDOW\_BOUNDS              SUB AX,PADDLE\_HEIGHT              CMP PADDLE\_LEFT\_Y,AX              JG FIX\_PADDLE\_LEFT\_BOTTOM\_POSITION       ;jumps if paddle left y is greater than ax              JMP CHECK\_RIGHT\_PADDLE\_MOVEMENT            FIX\_PADDLE\_LEFT\_BOTTOM\_POSITION:              MOV PADDLE\_LEFT\_Y,AX              JMP CHECK\_RIGHT\_PADDLE\_MOVEMENT          CHECK\_RIGHT\_PADDLE\_MOVEMENT:              ;if it is 'o' or 'O', move up              CMP AL,6Fh                              ;for 'o'. ASCII value of 'o' in hexadecimal              JE MOVE\_RIGHT\_PADDLE\_UP              CMP AL,4Fh                              ;for 'O'.              JE MOVE\_RIGHT\_PADDLE\_UP                ;if it is 'l' or 'L', move down              CMP AL,6Ch                              ;for 'l'. ASCII value of 'l' in hexadecimal              JE MOVE\_RIGHT\_PADDLE\_DOWN              CMP AL,4Ch                              ;for 'L'.              JE MOVE\_RIGHT\_PADDLE\_DOWN              JMP EXIT\_PADDLE\_MOVEMENT            MOVE\_RIGHT\_PADDLE\_UP:              MOV AX,PADDLE\_VELOCITY              SUB PADDLE\_RIGHT\_Y,AX              MOV AX,WINDOW\_BOUNDS              CMP PADDLE\_RIGHT\_Y,AX              JL FIX\_PADDLE\_RIGHT\_TOP\_POSITION        ;jumps if paddle left y is lesser than ax. 'y' position is 0 on top,AND IT INCREASES as we go down.              JMP EXIT\_PADDLE\_MOVEMENT            FIX\_PADDLE\_RIGHT\_TOP\_POSITION:              MOV PADDLE\_RIGHT\_Y,AX              JMP EXIT\_PADDLE\_MOVEMENT            MOVE\_RIGHT\_PADDLE\_DOWN:              MOV AX,PADDLE\_VELOCITY              ADD PADDLE\_RIGHT\_Y,AX              MOV AX,WINDOW\_HEIGHT              SUB AX,WINDOW\_BOUNDS              SUB AX,PADDLE\_HEIGHT              CMP PADDLE\_RIGHT\_Y,AX              JG FIX\_PADDLE\_RIGHT\_BOTTOM\_POSITION     ;jumps if paddle left y is greater than ax              JMP EXIT\_PADDLE\_MOVEMENT            FIX\_PADDLE\_RIGHT\_BOTTOM\_POSITION:              MOV PADDLE\_RIGHT\_Y,AX              JMP EXIT\_PADDLE\_MOVEMENT          JMP EXIT\_PADDLE\_MOVEMENT          EXIT\_PADDLE\_MOVEMENT:              RET      MOVE\_PADDLES ENDP        RESET\_BALL\_POSITION PROC NEAR          MOV AX,BALL\_ORIGINAL\_X          MOV BALL\_X,AX            MOV AX,BALL\_ORIGINAL\_Y          MOV BALL\_Y,AX          RET      RESET\_BALL\_POSITION ENDP        DRAW\_BALL PROC NEAR                             ;to define a noew procedure that belongs to the same code segment. the main procedure can call this drawball procedure            MOV CX,BALL\_X                               ;set the initial column (X position) of the pixel to 10          MOV DX,BALL\_Y                               ;set initial y (row position) position to 10.          ;we draw one pixel in one line till we reach the width of 4.          ;do the same for each line till we reach 4 again.          ;we print 16 pixels(4\*4).          ;this is done using the below loop          DRAW\_BALL\_HORIZONTAL:              MOV AH,0Ch                              ; set the configuration to writing a pixel              MOV AL,0Fh                              ; chooose white colour for the pixel              MOV BH,00h                              ; set the page number to zero              INT 10h                                 ;execute the configuration                INC CX                                  ;CX=CX+1 incrementing the column by 1.                                                      ;CX - BALL\_X > BALL\_SIZE (We check this condition. If this occurs,means we have reached the width of the ball,so we reset and go to the next line. If not,then we go to next column)              MOV AX,CX                               ;Using an auxiliary register AX so that we dont lose the current values.              SUB AX,BALL\_X                           ;This difference increases with each iteration. This Diff is stored in AX.              CMP AX,BALL\_SIZE                        ;comparing the two values              JNG DRAW\_BALL\_HORIZONTAL                ;JUMP,if NOT GREATER. if AX is not greater than ballsize,then loop this again.              ;this basically draws the columns.                ;if it reaches the below lines,means that AX became greater than ball\_size.              ;now we have to go to the next line and do this again.              MOV CX,BALL\_X                           ;resetting the column value to initial x position of the ball. CX register goes back to the initial column              INC DX                                  ;move to the next line              ;DX - BALL\_Y > BALL\_SIZE(to check if we have reached the final line).              ;if true,then we exit this procedure. otherwise,we move to next line.              MOV AX,DX              SUB AX,BALL\_Y              CMP AX,BALL\_SIZE              JNG DRAW\_BALL\_HORIZONTAL            RET                                         ;to exit this procedure      DRAW\_BALL ENDP                                  ;denotes the end of this procedure        DRAW\_PADDLES PROC NEAR          MOV CX,PADDLE\_LEFT\_X          MOV DX,PADDLE\_LEFT\_Y            DRAW\_PADDLE\_LEFT\_HORIZONTAL:              MOV AH,0Ch                              ;set the configuration to writing a pixel              MOV AL,0Fh                              ;chooose white colour for the pixel              MOV BH,00h                              ;set the page number to zero              INT 10h                                 ;execute the configuration                INC CX              MOV AX,CX              SUB AX,PADDLE\_LEFT\_X              CMP AX,PADDLE\_WIDTH              JNG DRAW\_PADDLE\_LEFT\_HORIZONTAL              MOV CX,PADDLE\_LEFT\_X              INC DX              MOV AX,DX              SUB AX,PADDLE\_LEFT\_Y              CMP AX,PADDLE\_HEIGHT              JNG DRAW\_PADDLE\_LEFT\_HORIZONTAL            MOV CX,PADDLE\_RIGHT\_X                       ;set the initial column (X)          MOV DX,PADDLE\_RIGHT\_Y                       ;set the initial line (Y)            DRAW\_PADDLE\_RIGHT\_HORIZONTAL:              MOV AH,0Ch                              ;set the configuration to writing a pixel              MOV AL,0Fh                              ;choose white as color              MOV BH,00h                              ;set the page number              INT 10h                                 ;execute the configuration                INC CX                                  ;CX = CX + 1              MOV AX,CX                               ;CX - PADDLE\_RIGHT\_X > PADDLE\_WIDTH (Y -> We go to the next line,N -> We continue to the next column              SUB AX,PADDLE\_RIGHT\_X              CMP AX,PADDLE\_WIDTH              JNG DRAW\_PADDLE\_RIGHT\_HORIZONTAL                MOV CX,PADDLE\_RIGHT\_X                   ;the CX register goes back to the initial column              INC DX                                  ;we advance one line                MOV AX,DX                               ;DX - PADDLE\_RIGHT\_Y > PADDLE\_HEIGHT (Y -> we exit this procedure,N -> we continue to the next line              SUB AX,PADDLE\_RIGHT\_Y              CMP AX,PADDLE\_HEIGHT              JNG DRAW\_PADDLE\_RIGHT\_HORIZONTAL          RET      DRAW\_PADDLES ENDP        CLEAR\_SCREEN PROC NEAR                          ;clear the screen by resetting the video mode              MOV AH,00h                              ;set the configuration to video mode              MOV AL,13h                              ;choose the video mode. set it to 13h(is a part of provides 256 color graphics)              INT 10h                                 ; execute the configuration                MOV AH,0Bh                              ;set the configuration              MOV BH,00h                              ;to the background color              MOV BL,00h                              ;choose black as background. 00h is the colour code for black              INT 10h                                 ;execute the configuration              RET      CLEAR\_SCREEN ENDP    CODE ENDS  END |

**STEP-BY-STEP REPORT**

* Creating a blank screen

CLEAR\_SCREEN PROC NEAR

    MOV AH,00h

    MOV AL,13h

    INT 10h

    MOV AH,0Bh

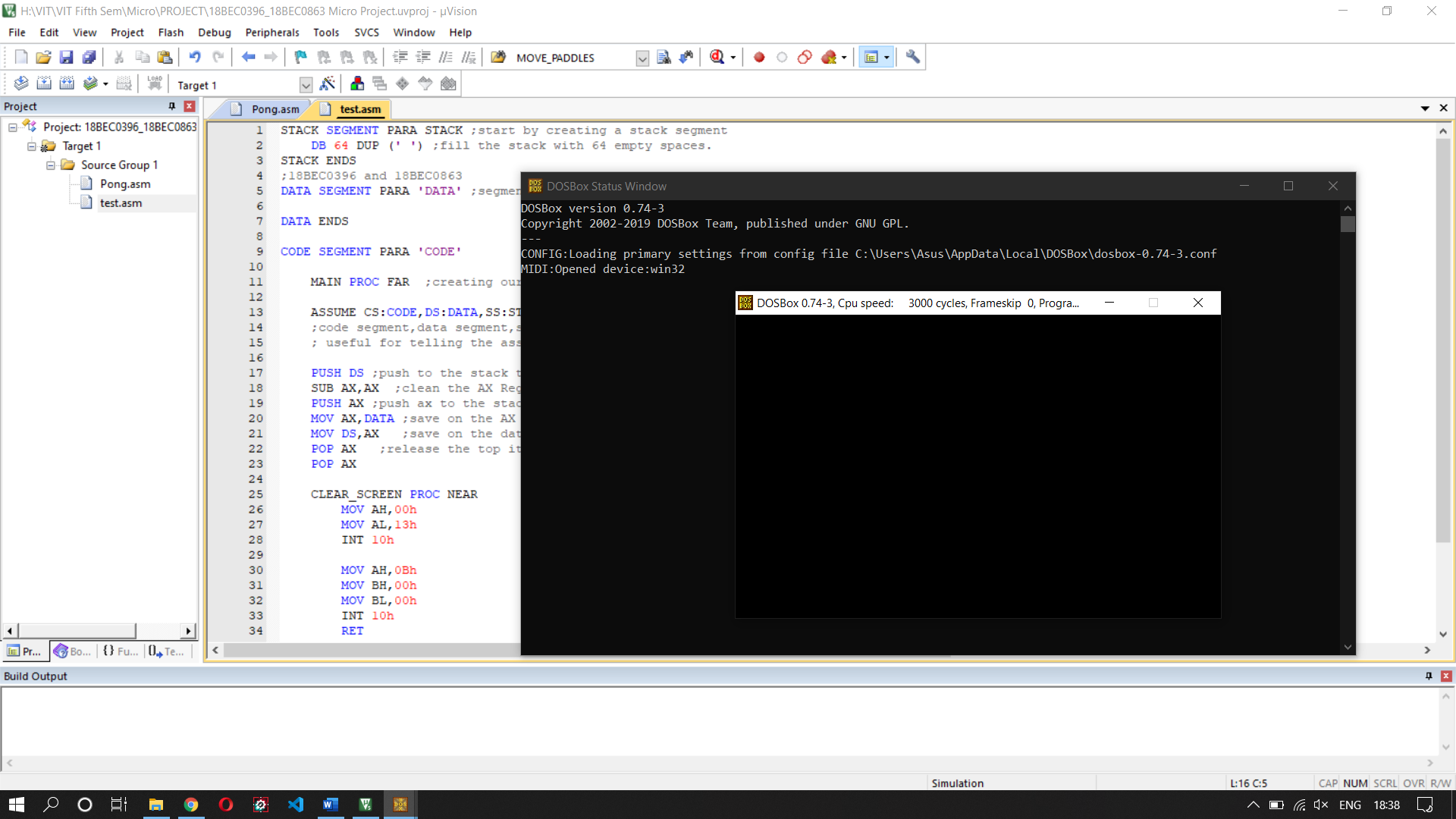
    MOV BH,00h

    MOV BL,00h

    INT 10h

    RET

CLEAR\_SCREEN ENDP



* Drawing the ball

MOV CX,BALL\_X

MOV DX,BALL\_Y

DRAW\_BALL\_HORIZONTAL:

    MOV AH,0Ch

    MOV AL,0Fh

    MOV BH,00h

    INT 10h

    INC CX

    MOV AX,CX

    SUB AX,BALL\_X

    CMP AX,BALL\_SIZE

    JNG DRAW\_BALL\_HORIZONTAL

    MOV CX,BALL\_X

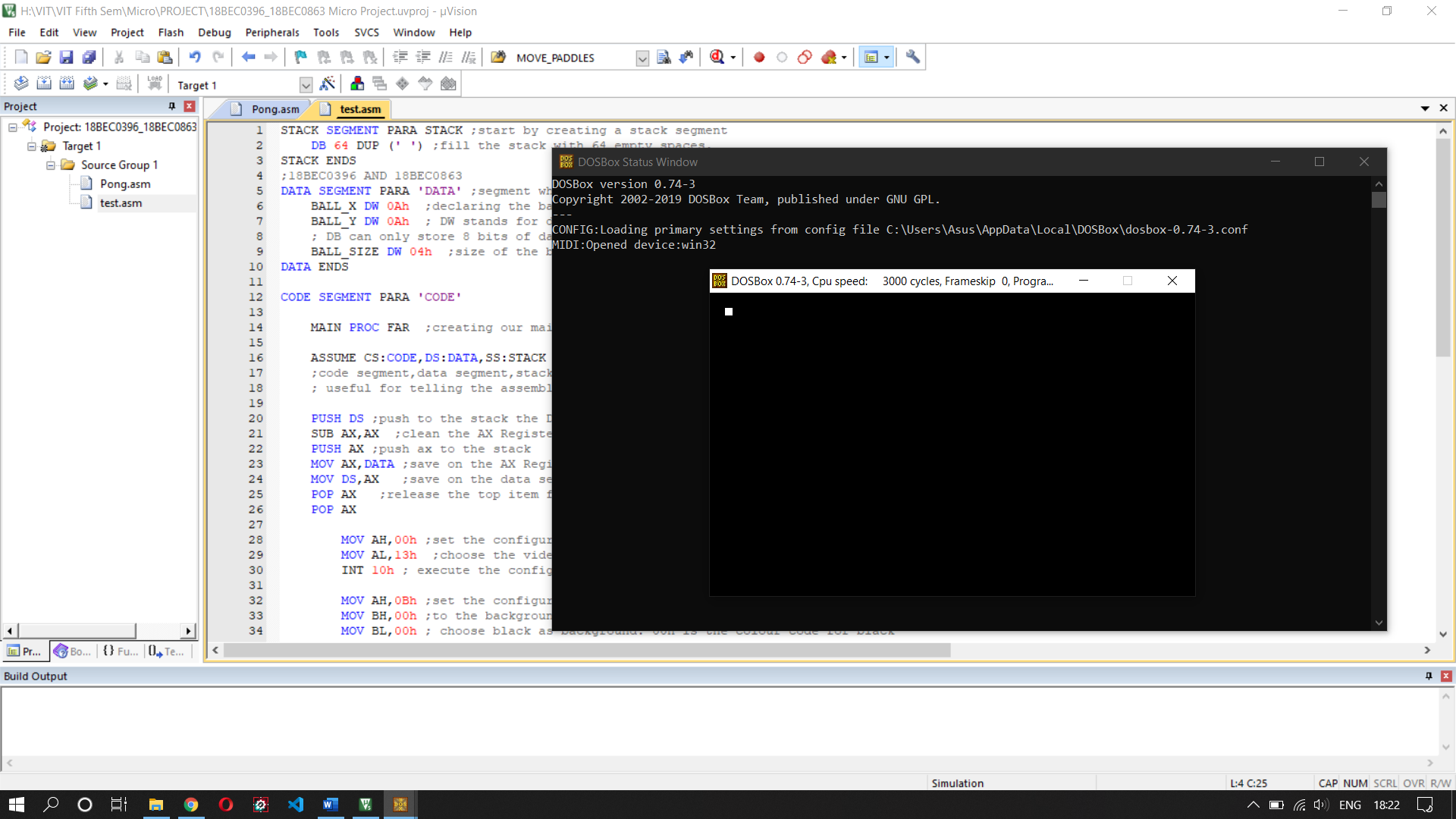
    INC DX

    MOV AX,DX

    SUB AX,BALL\_Y

    CMP AX,BALL\_SIZE

    JNG DRAW\_BALL\_HORIZONTAL



* Time Monitoring

CHECK\_TIME:

    MOV AH,2Ch

    INT 21h

    CMP DL,TIME\_AUX

    JE CHECK\_TIME

    MOV TIME\_AUX,DL

    CALL CLEAR\_SCREEN

    CALL MOVE\_BALL

    CALL DRAW\_BALL

    JMP CHECK\_TIME

* Moving the ball

MOVE\_BALL PROC NEAR

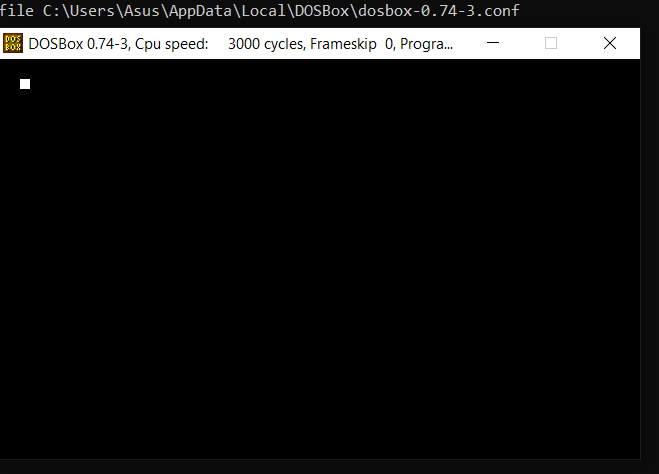
    MOV AX,BALL\_VELOCITY\_X

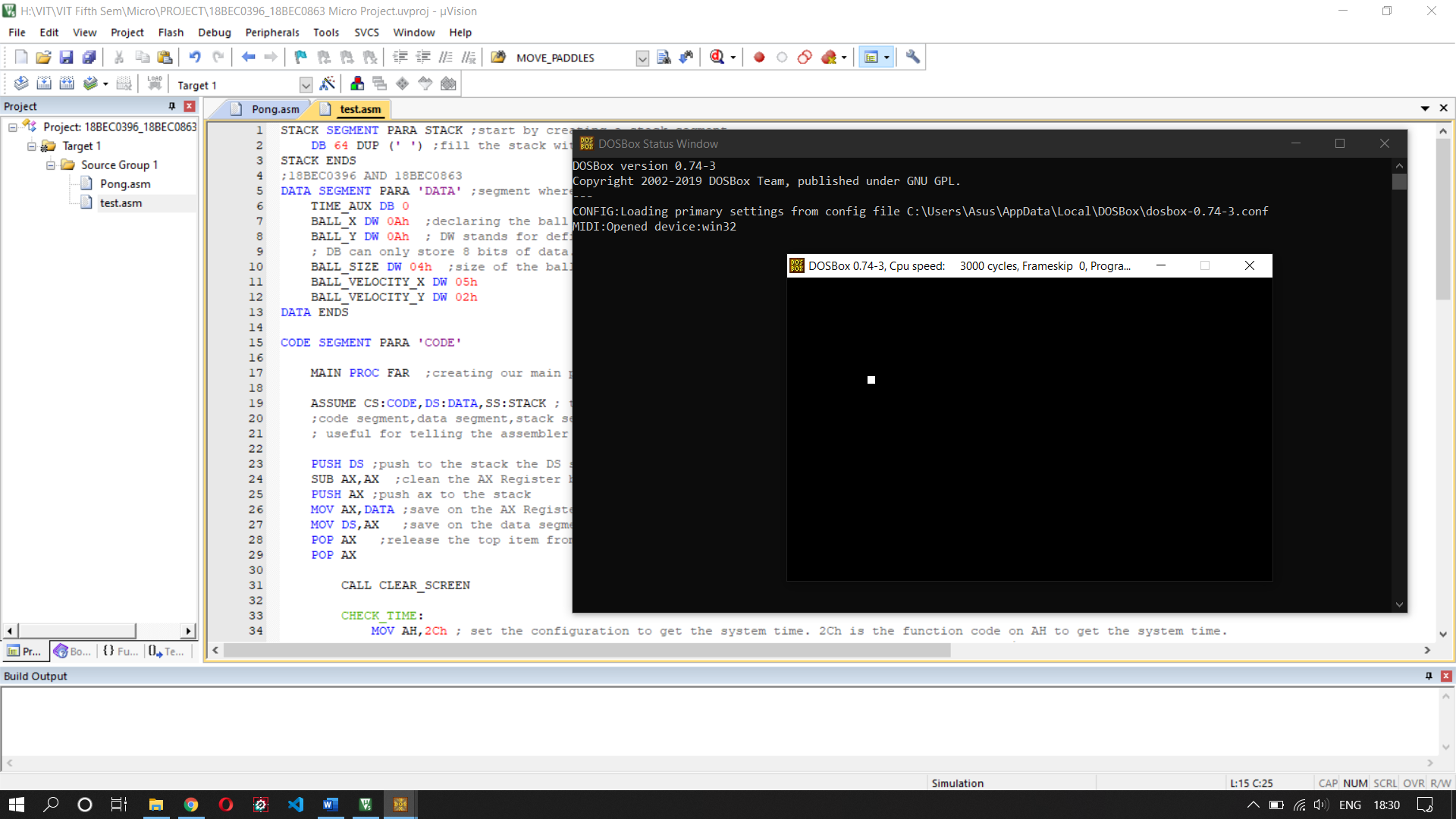
    ADD BALL\_X,AX

    MOV AX,BALL\_VELOCITY\_Y

    ADD BALL\_Y,AX

MOVE\_BALL ENDP





* Creating Paddles

MOV AX,BALL\_X

ADD AX,BALL\_SIZE

CMP AX,PADDLE\_LEFT\_X

JNG EXIT\_BALL\_MOVEMENT

MOV AX,PADDLE\_LEFT\_X

ADD AX,PADDLE\_WIDTH

CMP BALL\_X,AX

JNL EXIT\_BALL\_MOVEMENT

MOV AX,BALL\_Y

ADD AX,BALL\_SIZE

CMP AX,PADDLE\_LEFT\_Y

JNG EXIT\_BALL\_MOVEMENT

MOV AX,PADDLE\_LEFT\_Y

ADD AX,PADDLE\_HEIGHT

CMP BALL\_Y,AX

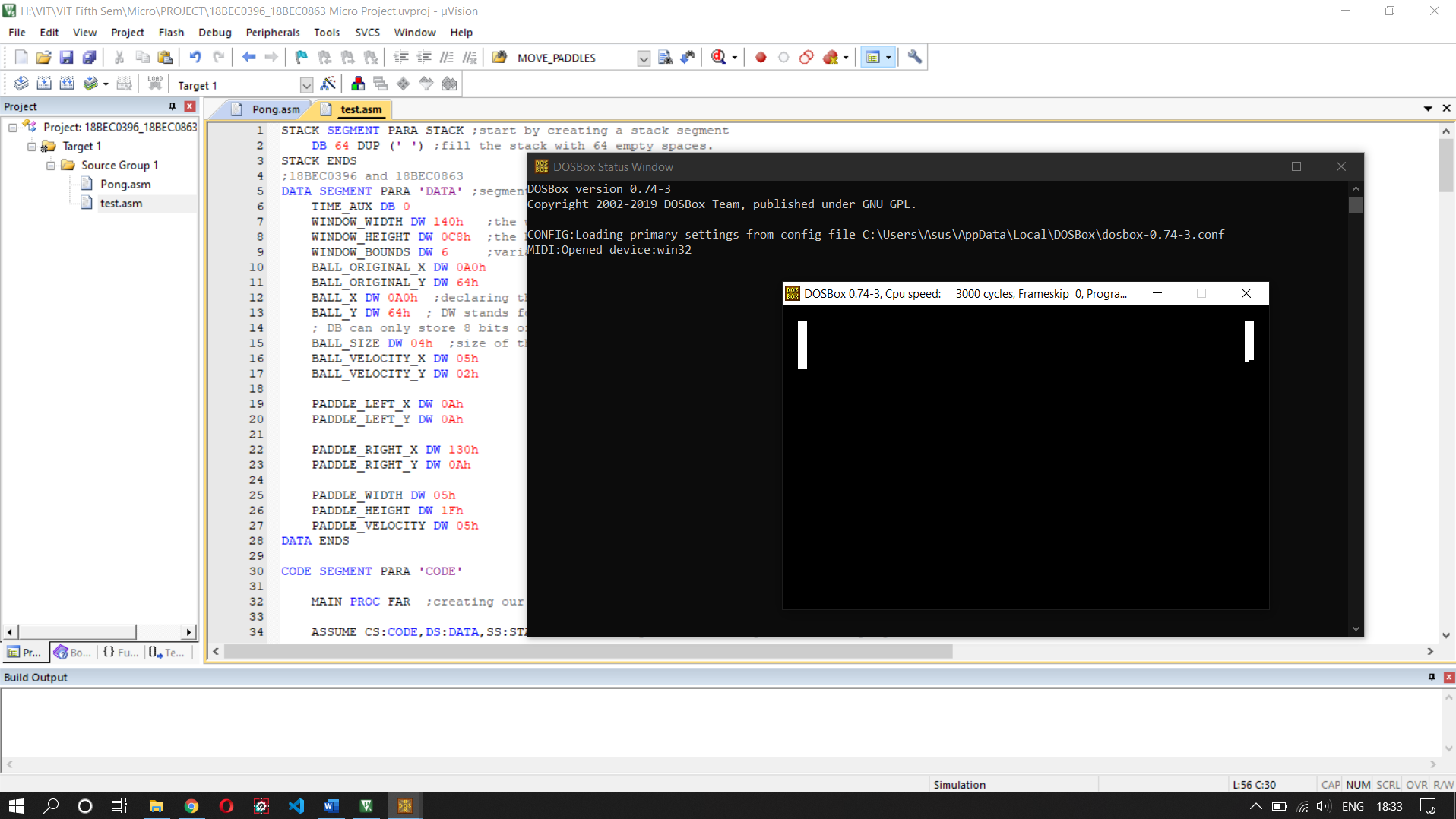
JNL EXIT\_BALL\_MOVEMENT

NEG BALL\_VELOCITY\_X

RET

EXIT\_BALL\_MOVEMENT:

    RET



* Collision

MOV AX,BALL\_X

ADD AX,BALL\_SIZE

CMP AX,PADDLE\_LEFT\_X

JNG EXIT\_BALL\_MOVEMENT

MOV AX,PADDLE\_LEFT\_X

ADD AX,PADDLE\_WIDTH

CMP BALL\_X,AX

JNL EXIT\_BALL\_MOVEMENT

MOV AX,BALL\_Y

ADD AX,BALL\_SIZE

CMP AX,PADDLE\_LEFT\_Y

JNG EXIT\_BALL\_MOVEMENT

MOV AX,PADDLE\_LEFT\_Y

ADD AX,PADDLE\_HEIGHT

CMP BALL\_Y,AX

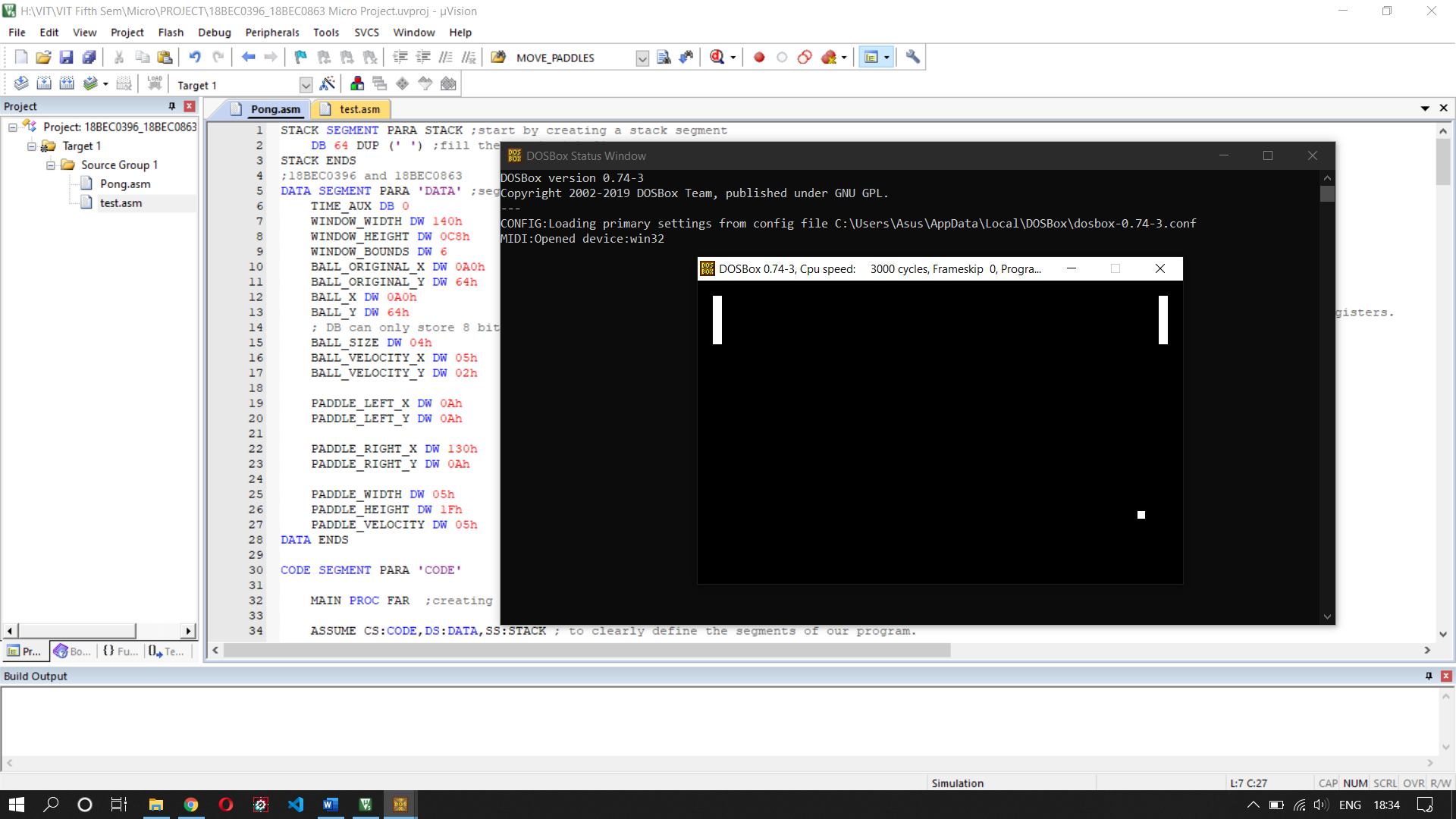
JNL EXIT\_BALL\_MOVEMENT

NEG BALL\_VELOCITY\_X

RET

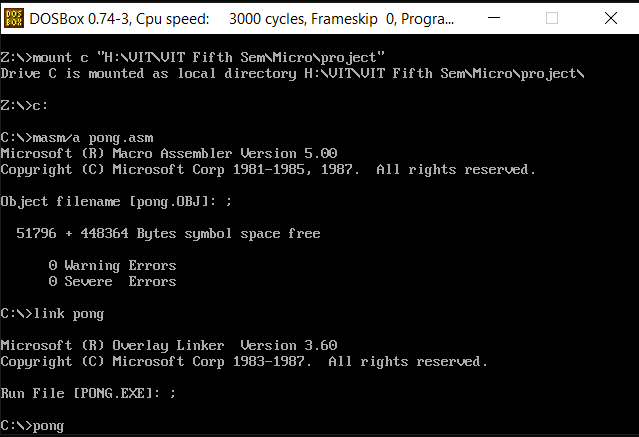
EXIT\_BALL\_MOVEMENT:

    RET



If ball collides with any of the paddles or the top/bottom edges of the frame, the game continues.

**RUNNING THE GAME (DOSBOX COMMANDS)**

****

What you see here is the DOSBOX terminal, which is basically used for emulating an 80806 processor so that legacy programs can be executed on it. This is a snapshot of the commands used to play the game. The commands are:

1. *masm /a pong.asm*

MASM stands for Microsoft Macro Assembler. It is an x86 assembler, and we use it for converting our assembly code file (pong.asm) into machine language. This process creates an obj file.

1. *link pong*

This command is used to link the pong.obj file (object file) by using the linker. The process of linking resolves all the references to other .obj files and is therefore responsible for linking all the keywords/pre-defined functions we use in our code with their true meaning. The process of linking generates an executable (.exe) file.

1. *pong*

This command is just to run the application’s executable file that was generated by the linker. Running this command starts the game.

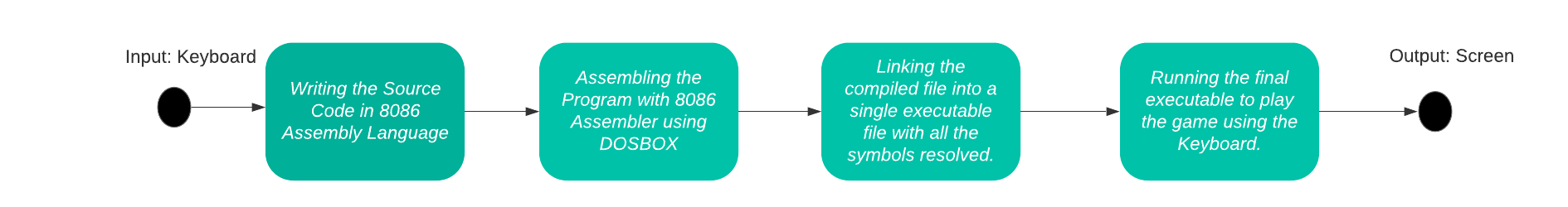


Fig: Flowchart of the above process

**RESULTS**

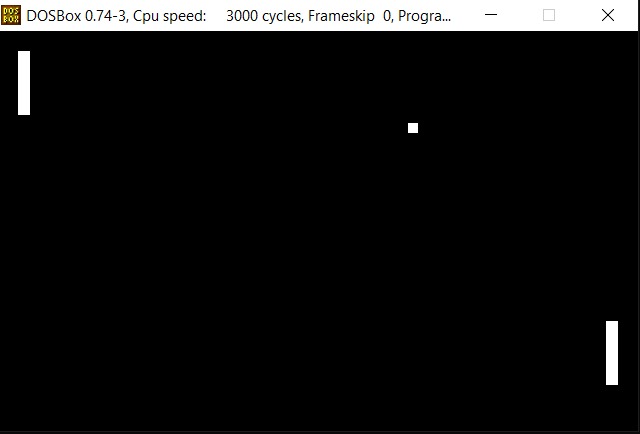
****

Fig: Final game running smoothly in our x86 emulator DOSBOX.

**CONCLUSION**

We have successfully recreated the Pong game using only Assembly Language, and in the process, have gained useful knowledge about 8086 architecture and its programming.

**REFERENCES**

1. <https://en.wikipedia.org/wiki/INT_10H>
2. <https://stanislavs.org/helppc/int_10-0.html>
3. <https://www.stanislavs.org/helppc/int_16.html>
4. <http://spike.scu.edu.au/~barry/interrupts.html>
5. <https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/>
6. <https://www.tutorialspoint.com/microprocessor/microprocessor_8086_overview.htm>
7. <http://ftp.it.murdoch.edu.au/units/ICT106/Practicals/ict106_pract_week9.pdf>
8. <https://www.elprocus.com/8086-assembly-language-programs-explanation/>