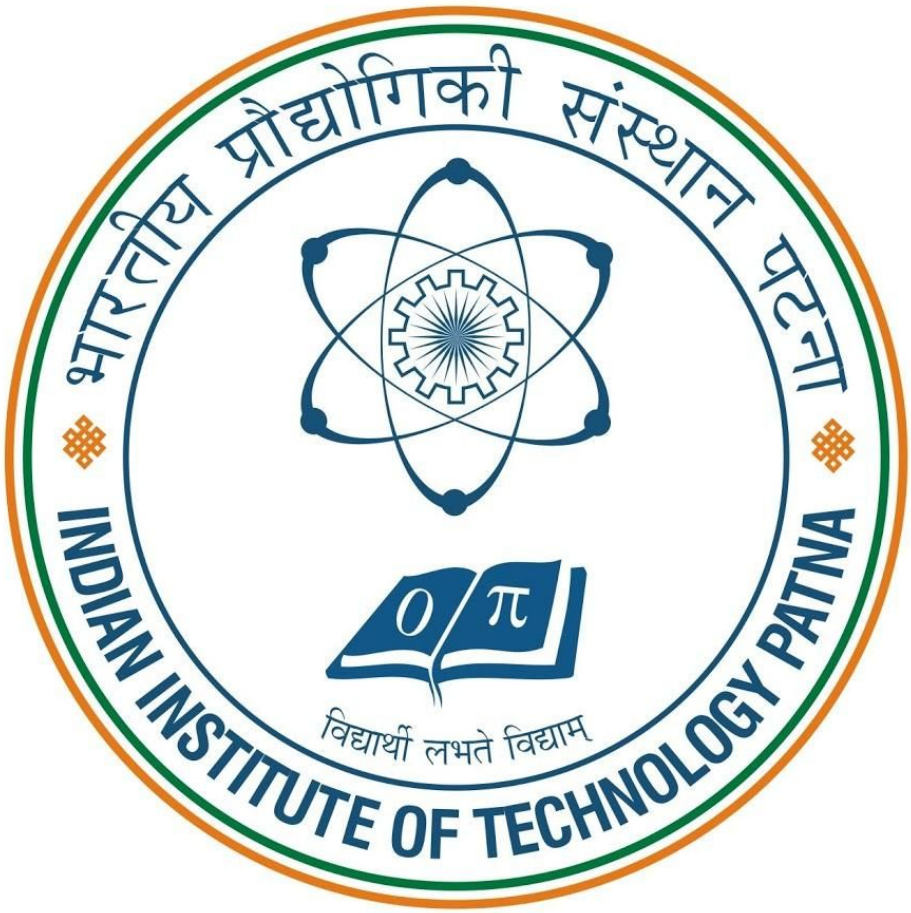
Computer Architecture

PONG IN 8086 ASSEMBLY

PROJECT REPORT



RAJ SHEKHAR

Introduction

Pong is one of the earliest arcade video games. It is a table tennis sports game featuring simple two-dimensional graphics. The player controls an in-game paddle by moving it vertically across the screen.

This project aims to implement it using 8086 assembly language. It can be run on DOSBOX compiled with nasm.

Game Sections

There are basically two sections of the game-

I. Initialization & Drawing
II. Looping

In the process of initialization and drawing, we have to,

1. Hide Cursor
2. Clear Screen
3. Draw Player
4. Draw CPU
5. Draw Ball

--- Here, we will draw the player paddle on the left center side of screen, ball on the center and CPU paddle on the right center of the screen

In the loop, we have to,

1. Check for input from keyboard
2. Move CPU paddle with ball
3. Check if ball is colliding with either paddle
4. Move Ball according to its direction
5. TIme Delay

--- This part will be in loop , if ball does not touch the paddle when it is on left or right side, the game will end. The loop will call drawing function whenever it needs it.

We also have game-end subroutine which exits from the game.

Data Section

We will use several variables and manipulate the function of the game using the data stored in it.

```
curX db 0x00
curY db 0x00
playerLoc db 0x0a
cpuLoc db 0x0a
ballLoc db 0x0b,0x27
ballDirection db 0x01
gameOver db 'Game over!$'
```

curX, curY is used to give location to set cursor and draw the necessary character at that location.

playerLoc will store the head location of the player paddle.

cpuLoc will store the head location of CPU paddle.

ballLoc will store Y-coordinate and X-coordinate of the ball.

ballDirection will store the current direction of the ball.

gameOver stores the ‘Game over!’ message to print it when game is over.

Drawing

Hiding Cursor

Interrupt int 10h is used to set text mode cursor shape. When bit 5 of CH is set to 0, the cursor is visible. when bit 5 is 1, the cursor is not visible.

```
mov cx,0x2000
mov ah,0x01
int 0x10
```

We store 20h in CH register to hide the blinking text cursor.

Clear Screen

To clear the screen, we use interrupt int 10h / ah = 06h. It scrolls up the current window with the attributes given the DX and CX register. Row and Column of window's upper left corner is stored in CH, CL register and row, column of window's lower right corner is stored in DH, DL register. Also AL register stores the number of lines to scroll up (00h to clear). Below is the snippet to clear the screen.

```
mov ah,0x06
mov al,0x00
mov bh,0x07
mov cx,0x0000
mov dl,0x79
mov dh,0x24
int 0x10
```

Drawing Player

To draw the player, we can use interrupt int 21h/ ah = 02h. This interrupt writes character to standard output. We have to move our cursor to the playerLoc and draw a '|' character and do it twice moving down two rows. This will make a paddle or player on the left center of screen.

```
mov cl,0x00 ;as player is always on left side, x-axis is zero
mov ch,[playerLoc] ;gets the player location
mov [curX],cl
mov [curY],ch
call .setCur ;subroutine to move cursor at that location
mov ah,0x02
mov dl,0xb3
int 0x21
```

Int 21h/ ah = 02h will print out whatever it is in the DL register.

0xb3 = 179 represents '|' character in extended ASCII table.

We move down be adding 1 to CH twice to make paddle of length 3.

Drawing CPU

To draw the CPU, we can use the same interrupt and character as player, as we have to make paddle for the CPU.

The head position of the CPU is stored in cpuLoc and since it is always on the right side of the screen, X-coordinates will always be 0x4f = 79.

Drawing Ball

To draw the ball, we use the same interrupt as drawing player and CPU. Now, since ball can move around the screen, neither of its coordinates are fixed.

```
    mov cl,[ballLoc+1]   ; X coordinate
     mov ch,[ballLoc]    ; Y-coordinate
     mov [curX],cl
     mov [curY],ch
     call .setCur
     mov ah,0x02
     mov dl,0x2a
     int 0x21
```

Here, dl = 0x2a.

In ASCII, 0x2a = 42 is '*' character.

Looping

Process Input - Check for keystrokes

To check if there is any keystroke stored in the buffer, interrupt INT 16h / AH = 01h is used. If any keystroke is available, ZF = 0, else ZF = 1. Also it will return the stored ASCII character in AL register and BIOS scan code in AH register.

Note - arrow keys are BIOS scan code

```
     mov ah,0x01
     int 0x16
     jz .processInputEnd ;this subroutine returns null
```

Process Input Get - get the keystrokes from the buffer if available

If after the above interrupt, ZF = 0, we will get the keystroke from the keyboard buffer using interrupt INT 16h / AH = 00h. This interrupt will store BIOS scan code in AH and ASCII in AL register.

```
    mov ah,0x00
    int 0x16
    cmp al,0x1b ; if ESC is hit
    je .endProgram
    cmp ah,0x48  ; up arrow
    je .playerMoveUp ; subroutine to move player paddle up
    cmp ah,0x50  ; down arrow
    je .playerMoveDown  ; subroutine to move player paddle down
```

0x1b = 33 is ESC character in ASCII table. We will end the program immediately when escape is hit.

0x48 is up arrow key in BIOS scan code and 0X50 is down arrow in BIOS scan code.

Clear Player paddle

To clear the player, we simply move the cursor down through the left side of the screen and print ' ' (space character) using loop.

Clear CPU paddle

Similarly to clear Player method, we move our cursor from top to bottom on the right side of our screen and print out ' ' character.

Player paddle move UP

When we get up arrow from the ‘process input get’, we decrease the head of player by one(as y-coordinate increases downwards on the screen) and store it in playerLoc, call the clear player function and then call the drawPlayer function, thus moving the player up.

Also, if the player head is already on the top row, nothing is done.

Player paddle move DOWN

Similarly to the UP subroutine, when we get the down arrow keystroke, head of the player paddle is increased by 1 and store in playerLoc and player paddle is deleted and the draw player subroutine is called.

Move CPU paddle

In this game, CPU can never be defeated as it always follows the ball.

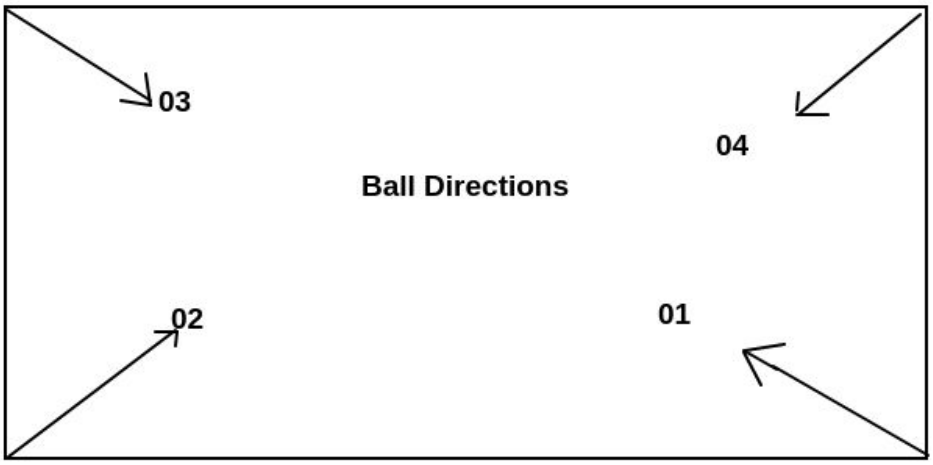
To move the CPU paddle we get the Y coordinate of the ball and compare it with middle of CPU paddle by adding one to CPU head. If is it less than the ball, CPU is moved down and if it is more, CPU is moved up.

CPU paddle move UP

Similarly to moving player paddle UP, we decrease the cpuLoc by one, clear the CPU and then draw it again.

CPU paddle move DOWN

Similarly to moving player paddle down, we increase the cpuLoc by one, clear the CPU and draw it at new location.



Ball Direction

We have four different ball direction to check the collision with different surface and act accordingly.

Paddle Collision Check

We check if there is any collision with either paddle or not. If ball is on extreme left on x-axis, player paddle check is called and if it is extreme right on x-axis, cpu paddle check is called.

There it is check whether the ball is touching paddle or wall. If ball is touching wall, game is ended. Else direction of ball is changed according to the previous directions.

Moving Ball

In this subroutine, the previous position of ball is deleted and it is moved according to its direction and location.

Time Delay - Sleep

Interrupt INT 1Ah / AH = 0h is used to give time delay in the process.

```
    mov ah,0  ; function no. for read
    int 1ah   ; get the time of day count
    add dx,1  ; add one half second delay to low word
    mov bx,dx ; store end of delay value in bx
.sleepLoop:
    int 1ah
    cmp dx,bx
    jne .sleepLoop ;loop for delay
```

Game Over

When game over is called, it will clear the screen print game over message and call endProgram

```
    call .clearScreen
    mov ah,0x09
    mov dx,gameOver
    int 0x21
    jmp .endProgram
```

End Program

Interrupt INT 21h / AH = 4Ch is used to end the program and return control to OS

```
    call .cursorShow
    mov ax,0x4c00
    int 0x21
```

Acknowledgement

1. Ralf Brown's Interrupt List:
   http://www.ctyme.com/rbrown.htm
2. http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_and_DOS_interrupts.pdf
3. StackOverflow
   https://stackoverflow.com/

Instructions-

Use the nasm.exe file to compile the asm file in DOSBOX

Type,

- nasm pong.asm -f bin -o pong.com

It will create a .com file

To run it, simply enter

- pong.com