Rock Paper Scissors in 8086 using 8255A PPI

Shubhankar Pandit: 18-13-104

June 23, 2020

Synopsis

The primary objective of this project was to implement a game of *Rock Paper Scissors* in **8086 Microprocessor** using buttons as user input and LEDs as output. **8255A PPI** was also interfaced with **8086** with the help of **74HC373** Latch to increase the number of disposable *I/O* ports.

Contents

1 Introduction

1.1 Motivation

Rock Paper Scissors is a broadly revered game and its sheer simplicity is the reason of its widespread popularity among the masses. **8086** and its instruction set provides the perfect platform to implement this game electronically.

1.2 About the 8086 Microprocessor

The **8086** is a 16-bit Microprocessor developed by *Intel* in early 1976. It is a successor of the *8085* Microprocessor with enhanced specifications and a more modified, robust instruction set. Some of the prominent features of **8086** Microprocessor are as follows

- It is capable of executing about 0.33 *MIPS* (millions instructions per second).
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It uses two stages of pipelining, known as *Fetch Stage* and *Execute Stage*, which greatly improves performance.
- It has a 20-bit address bus, in contrast to *8085*'s 16-bit address bus.
- It can access upto $1MB$ of memory.

1.3 Description

Rock Paper Scissors is a game usually played between two players, in which each player makes a symbol of one of the three possible shapes with their outstretched hand. It has nine possible permutations, wherein a player could win, lose or draw the game. According to the rules, *Rock* can beat *Scissors* but loses to *Paper*, and *Scissors* beats *Paper*. If both players make the same symbols, the game ends in a draw. An attempt has been made to implement this game using **8086**.

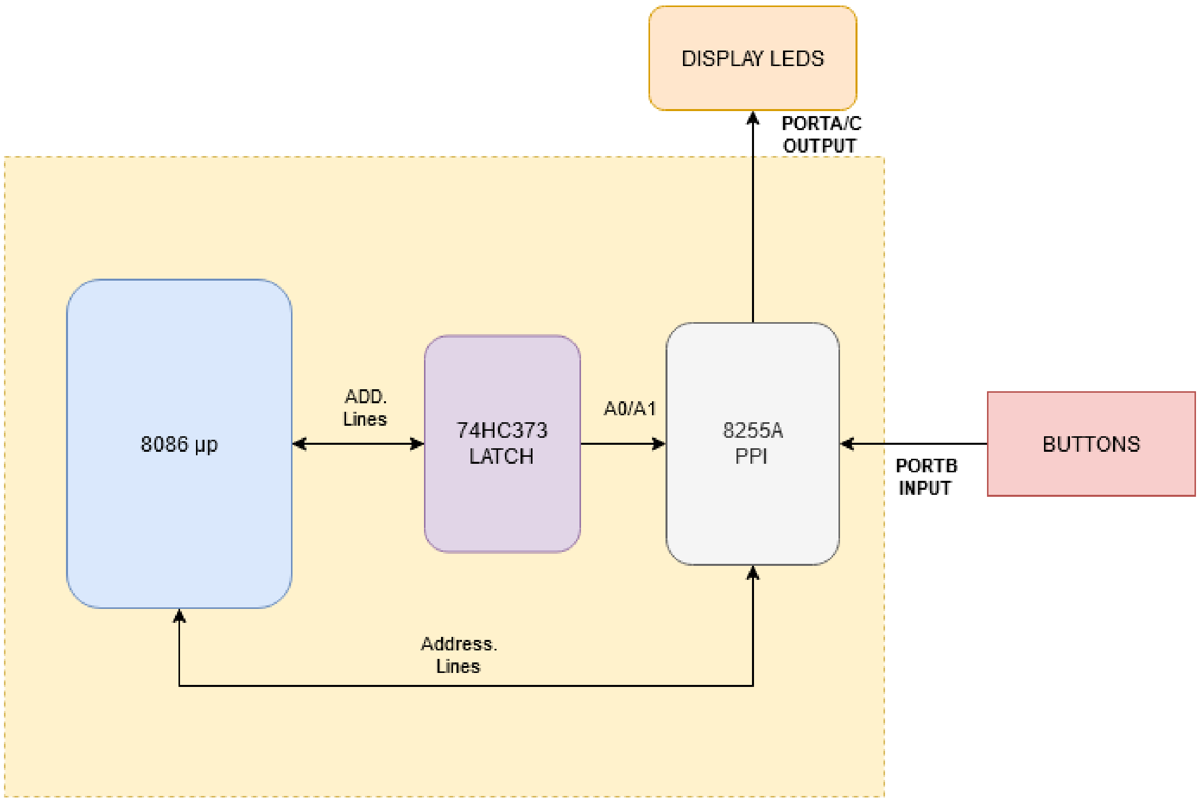

Figure 1: Block Diagram Representation of the Project

The input is taken from the buttons, which reset themselves after being pressed once. The result is displayed through *LEDs*.
Both the buttons and the *LEDs* are connected to the **8255A Programmable Peripheral Interface**, which acts as a link between **8086** and the Input/Output devices. The **74HC373** Latch is used to demultiplex the address/data bus.

1.4 Flow Chart of Operation

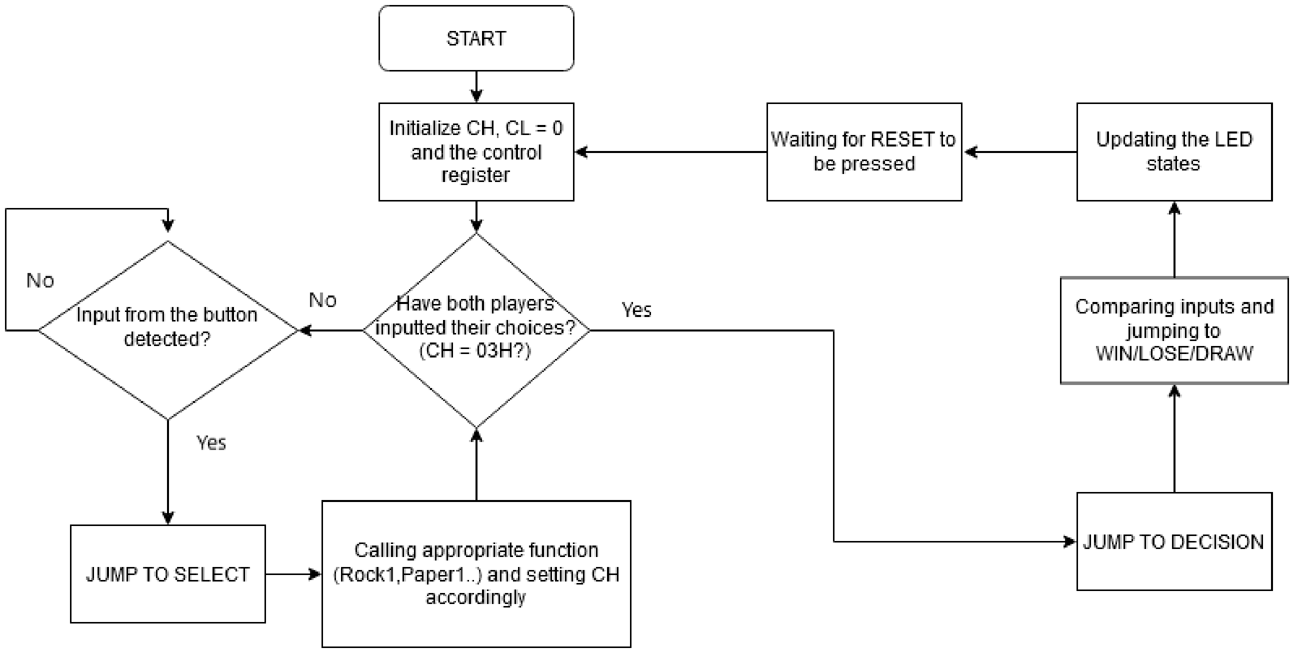
START
Initialize CH, CL = 0 and the control register
Waiting for RESET to be pressed
Updating the LED states
No
Input from the button detected?
No
Have both players inputted their choices? (CH = 03H?)
Yes
Comparing inputs and jumping to WIN/LOSE/DRAW
Yes
JUMP TO SELECT
Calling appropriate function (Rock1,Paper1..) and setting CH accordingly
JUMP TO DECISION

Figure 2: Flow Chart

The game starts with two counters, `CH` and `CL` initialized as 0. These counters are used to keep track of the user input. After an input is detected, the `SELECT` subroutine is called, which stores the user input in the `CL` register.

Once it is detected that both users have inputted their choices, the flow of control is transferred to the `DECISION` subroutine, which determines the final result (win, lose or draw). These results are then accordingly displayed on the LEDs. After the user presses the `RESET` button, the game restarts and all the counters are reset to their initial values.

2 Details of Hardware Implementation

2.1 Circuit Description

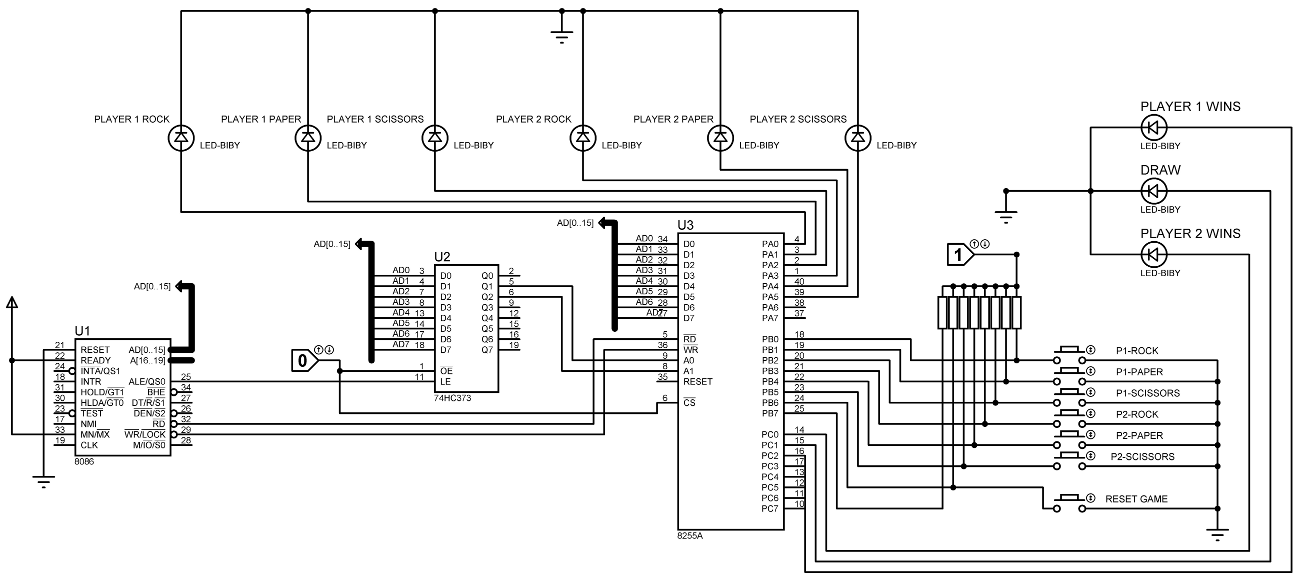

Figure 3: Circuit Layout

This project is based on the **8086** microprocessor. For interacting with the user, there are 7 *Push Buttons* and 9 *LEDs*. Six of the *Push Buttons* are for the users to make their choice *(Rock, Paper or Scissors)*, three for each of the user. There's a separate `RESET` button for resetting the game and starting it afresh. Of the nine *LEDs*, six are for displaying the move chosen by each player. The rest of three are for displaying the result of the match (`PLAYER 1 WINS, PLAYER 2 WINS, DRAW`).
The **8086** microprocessor is interfaced with the **8255A** PPI through the **74HC373** Latch. The address lines of 8086 from $A[0]$ to $A[15]$ are linked with both **8255A** and **74HC373**. The $\overline{OE}$ *(Output Enable)* pin of **74HC373** is set to `LOGIC 0`, and *LE (Latch Enable)* is connected to the *ALE (Address Latch Enable)* pin of **8086**. The Q_1 and Q_2 pins of **74HC373** are connected to the A_0 and A_1 pins of **8255A** respectively.

The `PORT A` of **8255A** from A_0 to A_5 is connected to the six *LEDs* that display player moves. Similarly, the C_0 to C_2 pins of `PORT C` are attached to the three result *LEDs*. `PORT B` is used as an input port, and hence is connected to the seven *Push Buttons*.
The *Push Buttons* are kept at active low, meaning an input is registered when a pin detects a low signal.

2.2 Bill of Materials

Part	Specification	Quantity
8086 µP		1
74HC373 Latch		1
8255A PPI		1
LED	2.2V,10mA	9
Push Button		7

3 Details of Software Implementation

3.1 Code Description

Throughout the code, `CH` and `CL` registers have been used as counters to keep track of user input and related information.
`CH` has been used to check whether or not inputs from both the players have been recieved. If both inputs have been given, the flow of control is then transferred to another segment of the code which then decides the final result of the game.
`CL` is used to store the input from both the users. This information is then later utilized to decide the outcome of the game.
To make sure the previous information isn't overwritten, only *OR* operations are done on both the registers.
`DX` and `AL` registers are solely used for *I/O* operations.

3.2 Code Layout

The code makes use of the following functions:

- `START`
  Initializes all the values of the registers used in the program. Also sets the *Control Word* of **8255A** as 100000010B (`PORT A` and `PORT C` as output and `PORT B` as input).

- `GAME`
  Executes as an infinite loop. Gets the current value of `PORT B` from the `IN` instruction and checks constantly for an input. If an input is detected, jumps to `SELECT` function. Also compares the value of `CH` with 00000011B, which if turns out to be true, jumps to `DECISION` function.

- `SELECT`
  Compares the inputted value in `AL` with all possible choices. Jumps to the respective function (`ROCK1, PAPER1, SCISSORS1, ROCK2, PAPER2, SCISSORS2, RESET`).

- `ROCK1, PAPER1, SCISSORS1`
  These functions set the value of `CH` and `CL` registers. For the case of Player 1, the *LSB* of `CH` should be set. Therefore, `CH` is `OR`ed with 00000001B. The value of `CL` register depends upon whether Player 1 chose *Rock, Paper or Scissors.* For each of these choices, the value of `CH` should be

  - Rock - 00000001B
  - Paper - 00000010B
  - Scissors - 00000100B

  The respective values are `OR`ed with `CL` to set that particular bit.

- `ROCK2, PAPER2, SCISSORS2`
  The Player 2 functions do the same task as the Player 1 functions, but set different bits. The second last bit of `CH` register is set if Player 2 has inputted their choice. For `CL` register, the following values are used
  - Rock - 00001000B
  - Paper - 00010000B
  - Scissors - 00010000B
- `RESET`
  The `RESET` function effectively resets the game and jumps to `START`.
- `DECISION`
  It first calls the `LEDOUT` function to turn on the *LEDs* corresponding to the user's inputs. Then, it compares the value of the `CL` register to every possible case, and hence accordingly calls the `WIN, LOSE` and `DRAW` functions.
- `LEDOUT`
  Turns on the *LEDs* by using `OUT` operation on `PORT A`.
- `DRAW, WIN, LOSE`
  These functions are called to display the result to the players. Hence, they use the `OUT` operation on `PORT C` and turn on their corresponding *LED*. They then call the `PAUSE` function.
- `PAUSE`
  It is an infinite loop. It pauses the operations after the game has ended to display the results to the user. It waits until the `RESET` button is pressed and then transfers control to `START` function to restart the game.

3.3 Assembly Code

```
DATA SEGMENT

PORTA EQU 00H
PORTB EQU 02H
PORTC EQU 04H
PORT_CON EQU 06H ;sets alias of the i/o addresses of each port
    as their name

DATA ENDS

CODE SEGMENT

ORG 0000H

START:

MOV DX, PORT_CON
MOV AL, 10000010B; port C (output), port A (output) in mode 0
    and PORT B (INPUT) in mode 0
OUT DX, AL
MOV CL, 00000000B
MOV CH, 00000000B ;initializing CH and CL as 0
MOV DX, PORTA
MOV AL, 00000000B
OUT DX, AL
MOV DX, PORTC
OUT DX, AL ;turning off all the LEDs

GAME:
CMP CH, 03H ;checks if both the users have given their inputs
JZ DECISION ;if so, jumps to decision function
MOV DX, PORTB
IN AL,DX
CMP AL, 11111111B ;takes input from portb and checks if the user
    has pressed any button
JNZ SELECT ;jumps to select if an input is detected
JMP GAME ;repeats the process indefinitely
```

```
SELECT:
CMP AL, 11111110B
JZ ROCK1
CMP AL, 11111101B
JZ PAPER1
CMP AL, 11111011B
JZ SCISSORS1
CMP AL, 11110111B
JZ ROCK2
CMP AL, 11101111B
JZ PAPER2
CMP AL, 11011111B
JZ SCISSORS2
CMP AL, 10111111B ;compares each choice with their respective
    buttons
JZ RESET
JMP GAME

ROCK1:
OR CH,01H ;sets the lsb of CH to indicate that player1 has
    inputted their choice
AND CL, 11111000B ;clears the previous choice if any
OR CL, 00000001B ;sets the bit according to the choice made
    (rock/paper/scissors)
JMP GAME

PAPER1:
OR CH,01H
AND CL, 11111000B
OR CL, 00000010B
JMP GAME

SCISSORS1:
OR CH,01H
AND CL, 11111000B
OR CL, 00000100B
JMP GAME

ROCK2:
```

```
OR CH,02H ;sets the 2nd last bit of CH to indicate player 2 has
    made their choice
AND CL, 11000111B ;clears the previous choices if any (the first
    two bits are don't care)
OR CL, 00001000B
JMP GAME

PAPER2:
OR CH,02H
AND CL, 11000111B
OR CL, 00010000B
JMP GAME

SCISSORS2:
OR CH,02H
AND CL, 11000111B
OR CL, 00100000B
JMP GAME

RESET: ;resets the game
JMP START

DECISION: ;compares every possible combination and accordingly
    jumps to the draw, lose and win functions
CALL LEDOUT
CMP CL, 00001001B
JZ DRAW
CMP CL, 00010001B
JZ LOSE
CMP CL, 00100001B
JZ WIN
CMP CL, 00001010B
JZ WIN
CMP CL, 00010010B
JZ DRAW
CMP CL, 00100010B
JZ LOSE
CMP CL, 00001100B
JZ LOSE
CMP CL, 00010100B
```

```
JZ WIN
CMP CL, 00100100B
JZ DRAW

LEDOUT:
MOV AL, CL
MOV DX, PORTA
OUT DX, AL ;turns on the LEDs corresponding to the choice the
    players made
RET


DRAW:
MOV AL, 00000010B
MOV DX, PORTC
OUT DX,AL ;turns on the draw LED
JMP PAUSE

WIN:
MOV AL, 00000100B
MOV DX, PORTC
OUT DX,AL ;turns on the player 1 win LED
JMP PAUSE

LOSE:
MOV AL, 00000001B
MOV DX, PORTC
OUT DX,AL ;turns on the player 2 win LED
JMP PAUSE

PAUSE: ;waits until reset button is set to maintain the state of
    the LEDs
MOV DX, PORTB
IN AL,DX
CMP AL, 10111111B
JZ START
JMP PAUSE
JMP START
CODE ENDS
END
```

3.4 The Simulation

The project was simulated with the help of *Proteus 8* Windows Application as a **8086** Firmware Project. The following are some exports of the project while being simulated.

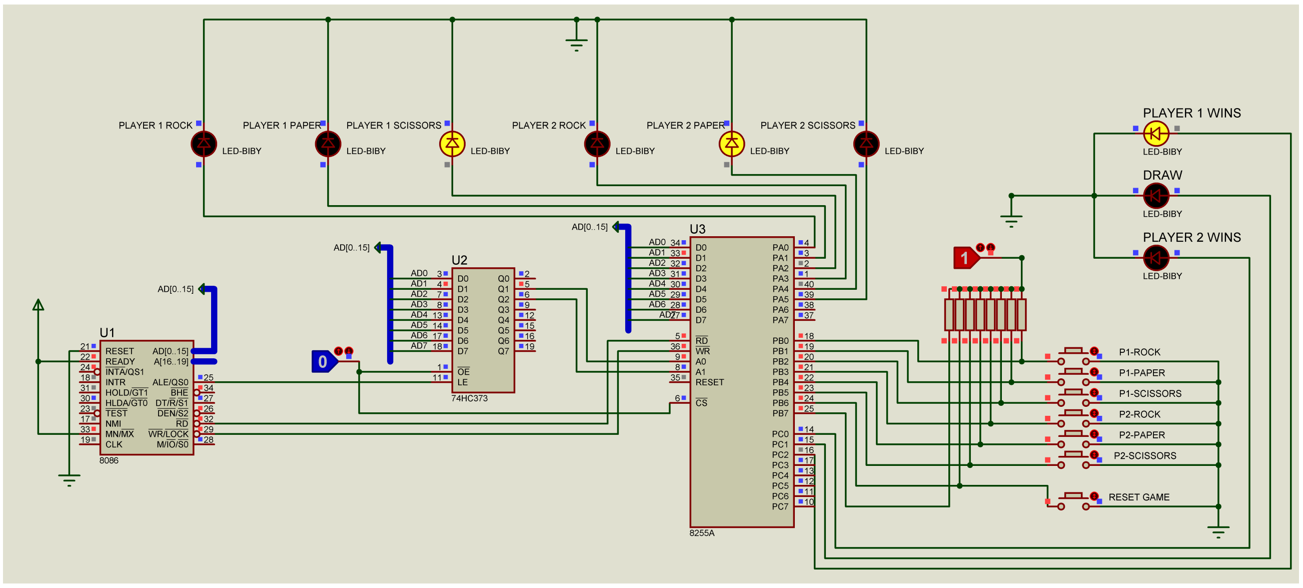

Figure 4: Player 1 Wins the Game

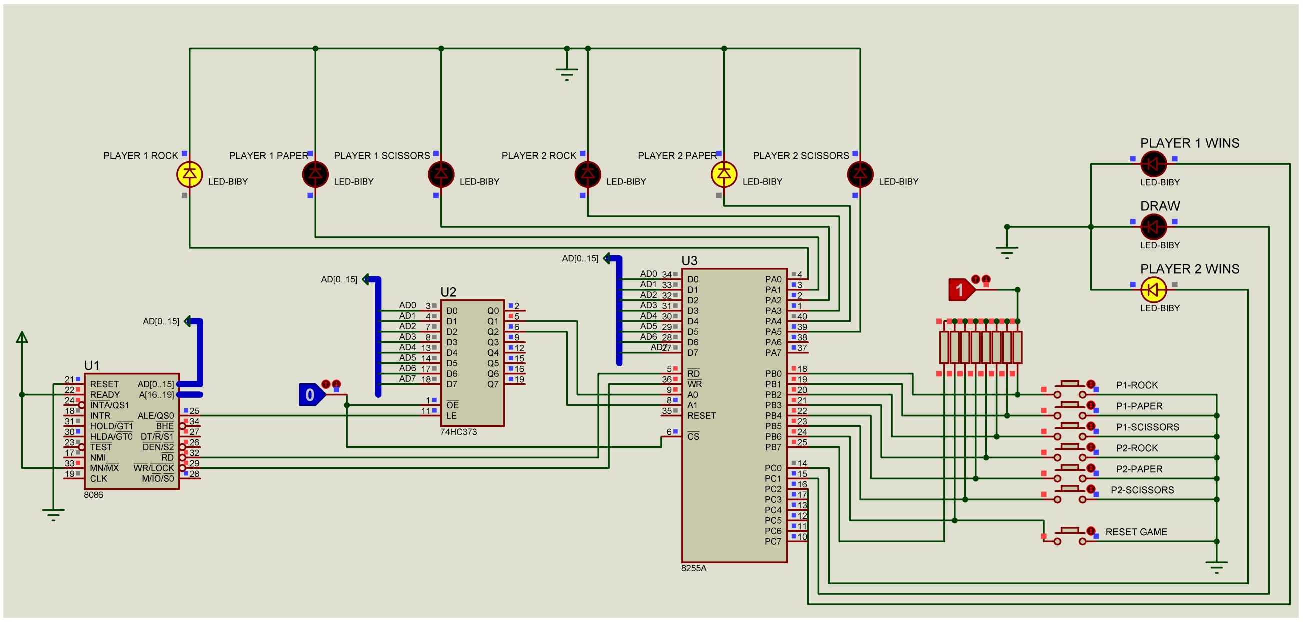

Figure 5: Player 2 Wins the Game

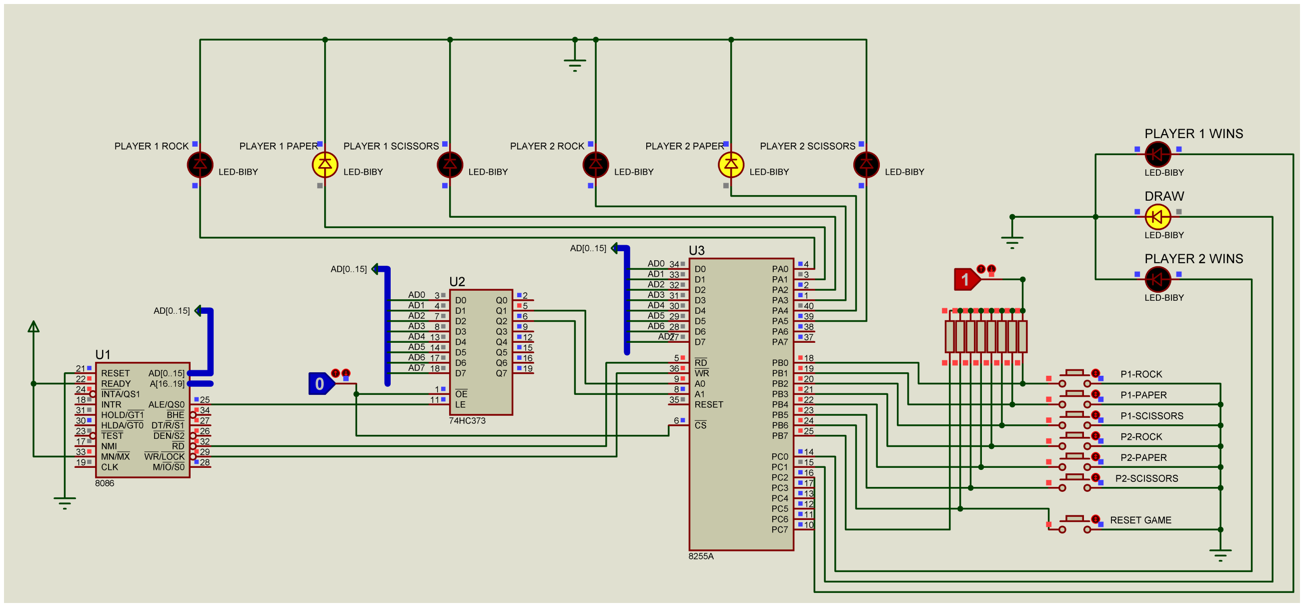

Figure 6: Player 1 and Player 2 Draw the Game

4 Conclusion

Hence, the *Rock, Paper, Scissors* Game was succesfully implemented and simulated in *Proteus 8*. Several skills were also picked up in the process of making this project, including, interfacing **8086** with **8255A** with the help of **7HC373** Latch, the *Control Word Format* of **8255A** and further sharpening the *Instruction Set* of **8086**.