CHROME DINO GAME USING 8051 MICROCONTROLLER

5TH SEMESTER PROJECT
2017-21 BATCH

INTRODUCTION

In the fast growing field of software engineering and game development we as an end user don't necessarily require any basic knowledge on coding or basic working of the game. This project deals with the basic complexities in building a very simple 2D game such as "Chrome's Dino Game". The main complexity arises when a fully functional game has to be run on a basic microcontroller with its restricted and limited resources. This game has an LCD interface where the user can enjoy playing the game while his/her scores are displayed on a 7 segment display. The user controls the dino using a push button provided on board. Since LCD prints each and every character involved in the game using a single cursor that has to perform its task so quickly that the user doesn't even notice that only a single cursor is doing such a humongous task. Now that the core limitations of resources said now this becomes even more impossible when the microcontroller has to keep track of real time scores of the player and display their scores in real time in a 7 segment display while preserving the smooth motion of the game.

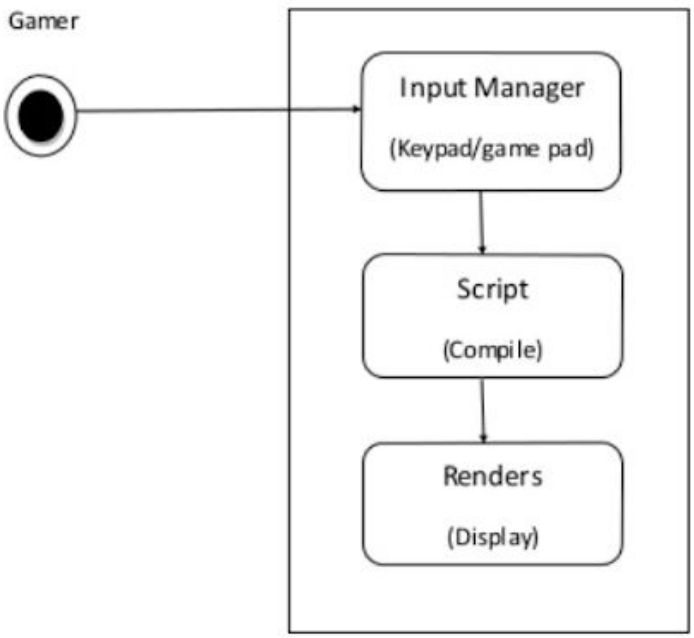
MATERIALS

1. 8051 Microcontroller trainer kit
2. Jumper cables
3. Push button
4. 7 Segment Display

CONNECTIONS

1. LCD CONNECTIONS
    a. rs - P3^2
    b. rw - P3^3
    c. en - P3^4
    d. P1 are Data cables (8-bits)
2. SWITCH CONNECTIONS - p3^7
3. 7 SEGMENT DISPLAY
    a. P2 are Data cables (8-bits)
    b. P0 7 segment display enable pins

GENERAL SCENARIO



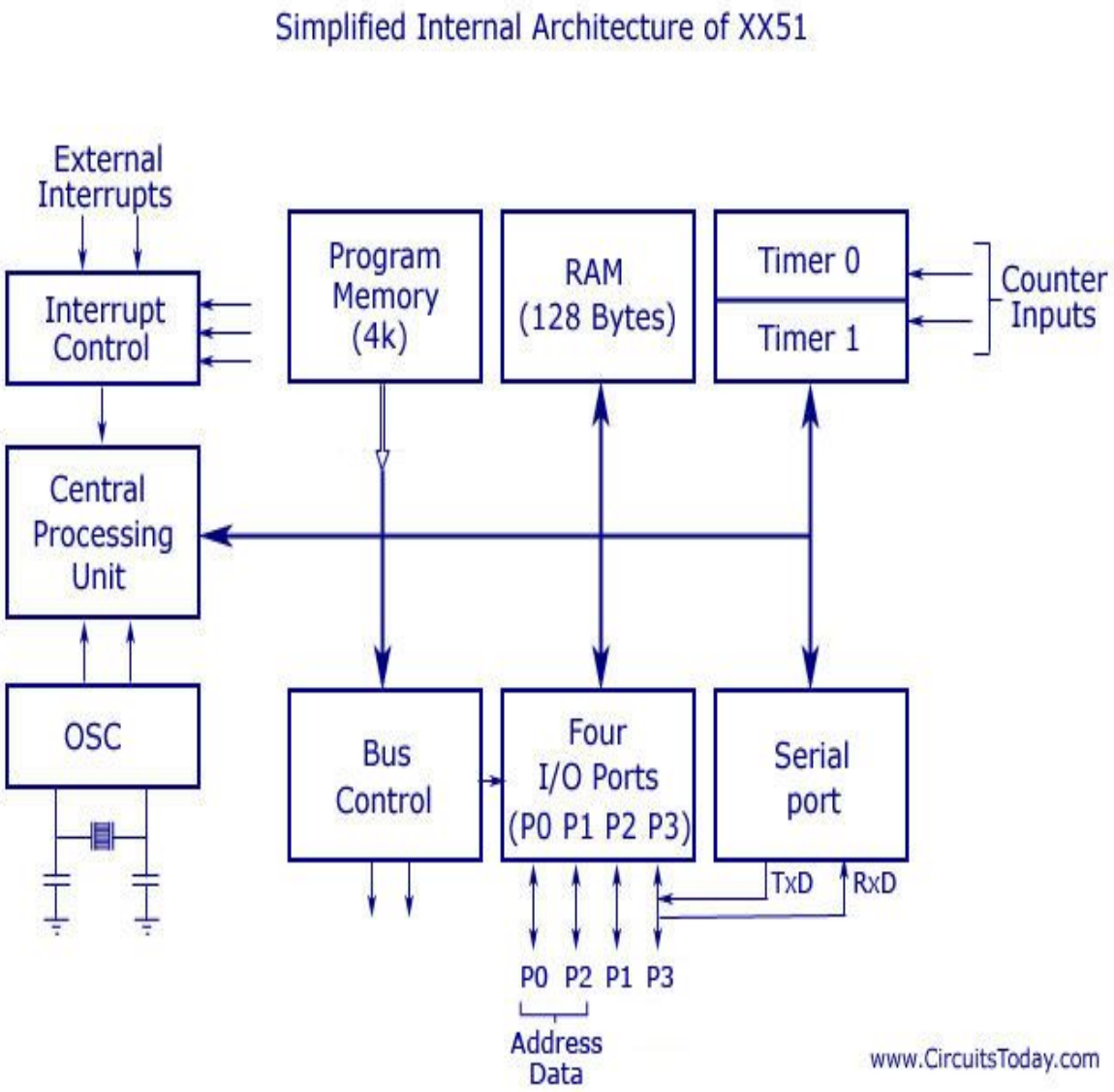
8051 MICROCONTROLLER SPECIFICATIONS AND LIMITATIONS:

The main difference can be stated as on-chip memory i.e. a Microcontroller has both Program Memory (ROM) and Data Memory (RAM) on the same chip (IC) whereas a Microprocessor has to be externally interface with the memory modules.

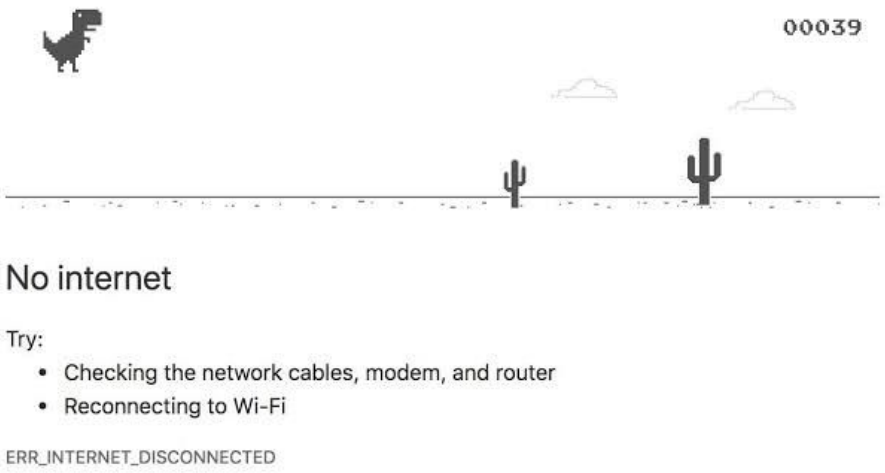
As seen in the internal architecture our point of interest will be the RAM which is only 128 bytes. It is a notable point to mention the fact that an Apple III that was released in the market in mid 1980 had 128 KB RAM which is 1000 times that of what we have got available with our 8051 board. Our next area of interest moves towards our inter program memory that can store only 4KB and an external program memory of 60KB can be connected if required. With all these specifications and limitations let's move on to discuss about the original complexities involved in running our Dino Run game on the board.

1. Limited memory.
2. Limited RAM availability.

3. 16X2 LCD screen that can print only one character per unit time.
4. 7 Segment display that requires continuous data transmission to display the 4 digit number perfectly since the 8 bits data for every single digit is transferred by only one set of 8 data lines.



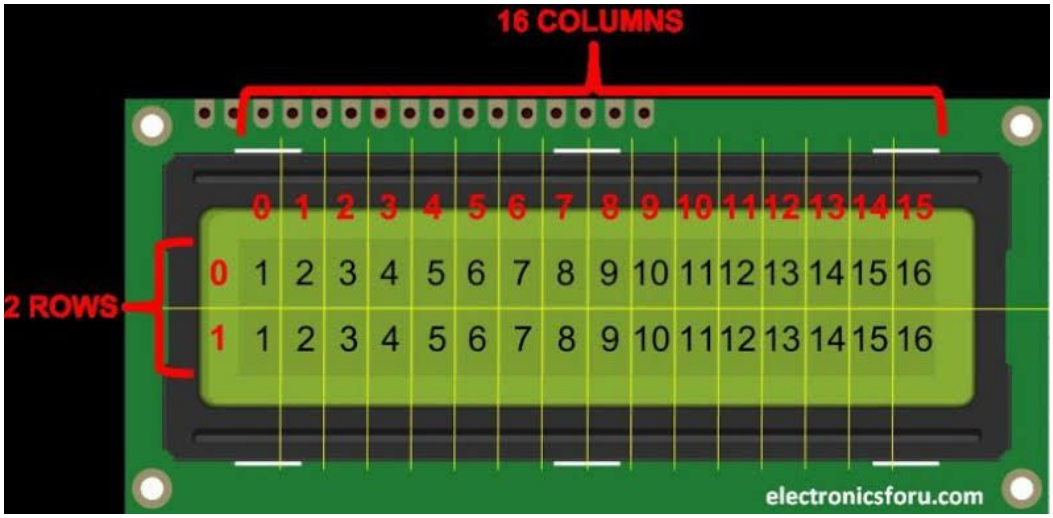
DEVELOPMENT PROCESS



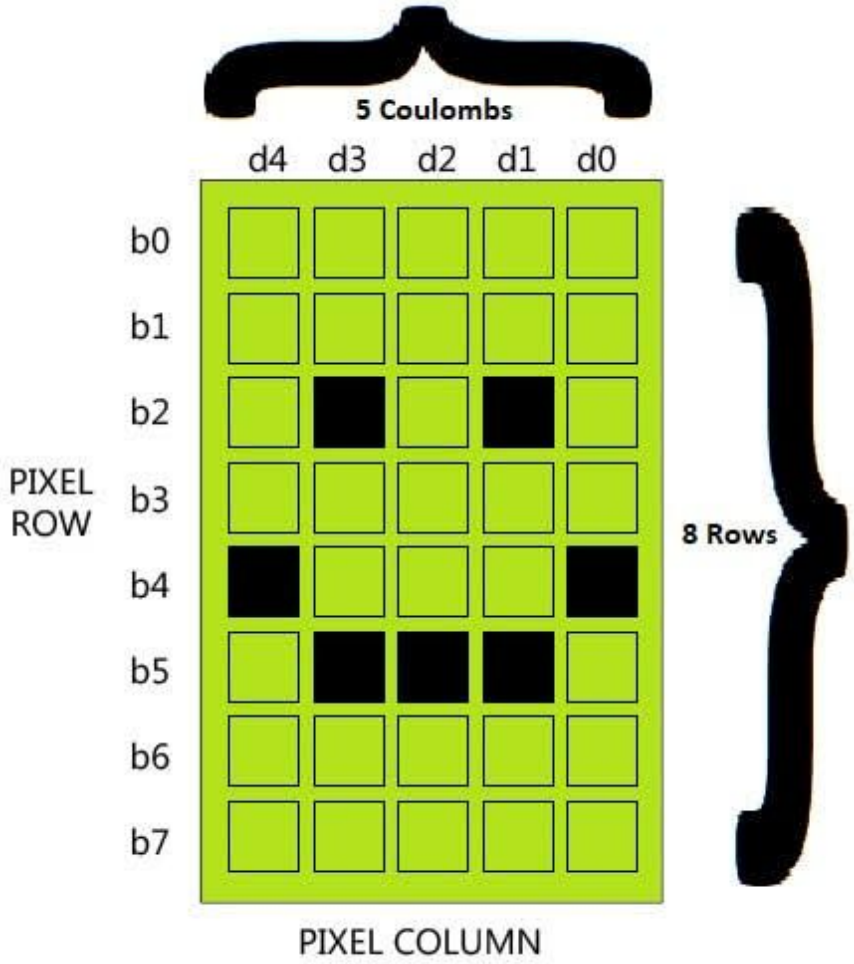
The above picture is the original Chrome's Dino Run game originally designed to start when a user presses the spacebar button whenever he/she doesn't have an internet connection. So the main motivation of this project is to replicate the same game on a 8051 Microcontroller with our own twist in it.

1. BUILDING OUR GAME CHARACTERS

Since we are going to use a 16x2 LCD for our display purposes, let us now have a quick review on how things work in an LCD screen display and how can we achieve our objective of building our on custom characters that is to be displayed on screen.



As



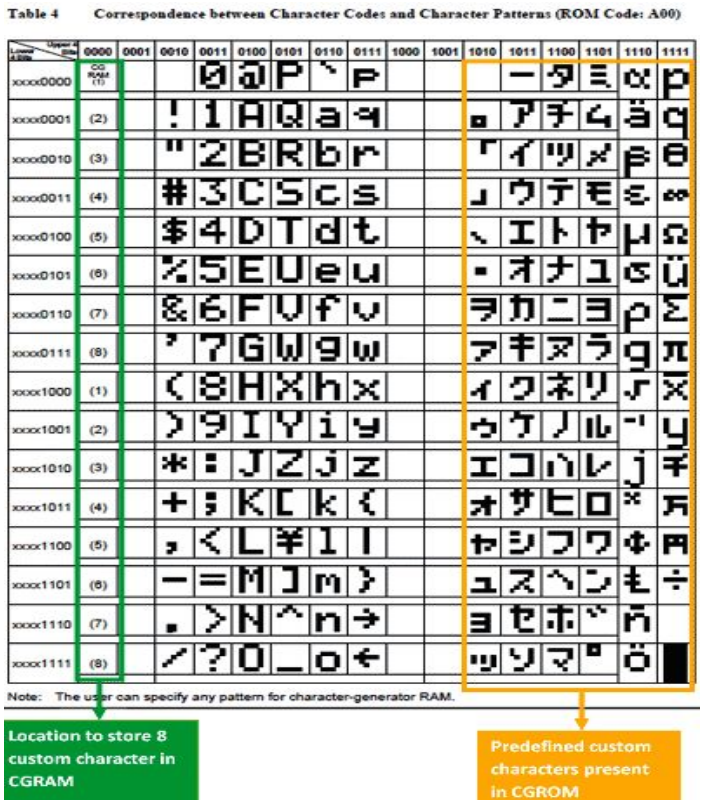
As we can see here an LCD screen displays a character by lighting up specific pixels in a pattern in order to print our desired character on screen.
In order to display a custom character, we have to somehow tell the IC that how the

custom character will look like. To do that we should know about the **Three types of Memories present inside the HD44780 LCD controller IC**:

Character Generator ROM (CGROM): It is the read only memory which, as said earlier, contains all the patterns of the characters pre-defined inside it. This ROM will vary from each type of Interface IC, and some might have some pre-defined custom character with them.

Display Data RAM (DDRAM): This is a random access memory. Each time we display a character its pattern will be fetched from the CGROM and transferred to the DDRAM and then will be placed on the screen. To put it simple, DDRAM will have the patterns of all characters that are currently being displayed on the LCD Screen. This way for each cycle the IC need not fetch data from CGROM, and helps in getting a short update frequency

Character generator RAM (CGRAM): This is also a Random access memory, so we can write and read data from it. As the name implies this memory will be the one which can used to generate the custom character. We have to form a pattern for the character and write it in the CGRAM, this pattern can be read and displayed on the Screen when required.



As, the datasheet implies, the HD44780 IC has provided as 8 Locations to store our custom patterns in CGRAM, also on the right we can see that there are some pre-defined characters which can also be displayed on our LCD Screen. Let us see how we can do it.

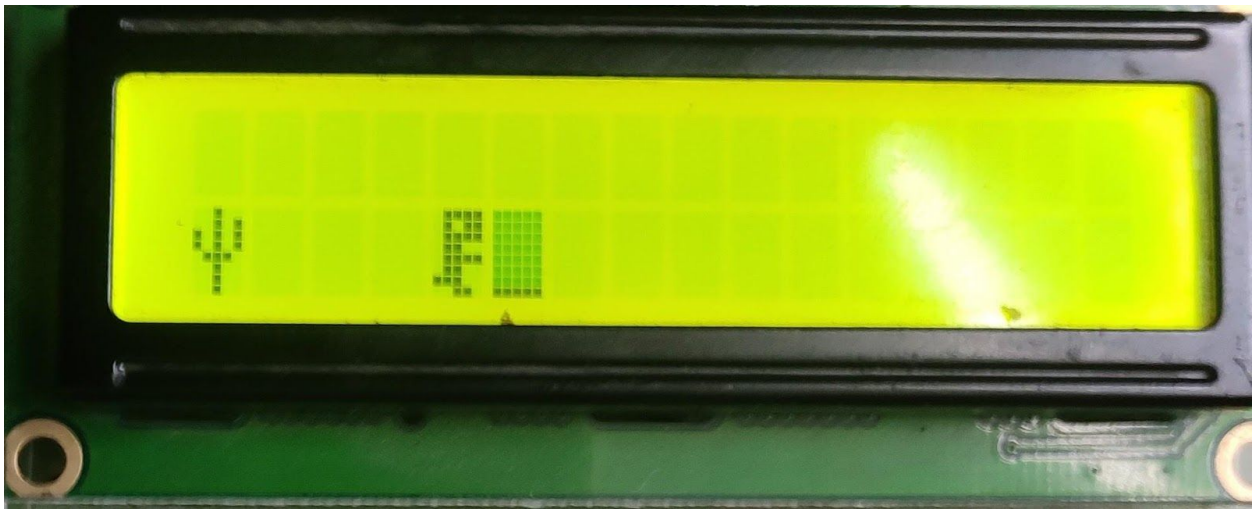
To display a custom character we must first generate a pattern for it and then save it to the CGRAM.

The custom character is constructed by making an array of binary values representing each pixel rows in a single cell of the 16x2 LCD screen display. The piece of code that does the job can be seen in the following picture.

```
60
61              unsigned char dino[8] =  {
62  0x0F,
63  0x0B,
64  0x0F,
65  0x0C,
66  0x0F,
67  0x0C,
68  0x1C,
69  0x06
70 };
```

```
93
94     P1 = (0x40+16);
95     cmdwrite();
96     for(i=0;i<8;i++)
97       {
98         P1 = dino[i];
99         datawrite();
100      }
101        delay(50);       //making dino and storing it in 0x02 CGRAM
```

THE RESULTING CUSTOM CHARACTERS - CACTUS AND THE DINO



2. CODING THE BASIC MOTIONS INVOLVED IN THE GAME

The basic motions involved in the game are

1. Cactus moving towards the Dino.
2. Dino's jumping motion.

Now let's discuss about making the cactus to move towards the Dino. This objective can be achieved by two ways:

1. Manually print the character one by one in each necessary positions while erasing the previous ones.
2. Using the left/right shift commands of LCD.

Here we would be using option two i.e., Using the left shift LCD command because manually printing the character one by one would overload the microcontroller and suck up the available processing resources.

END RESULT:

https://photos.app.goo.gl/Po2C77aEK6wTE1MQA

The Dino is made to make the jumping illusion by erasing the current position of the character and printing the same character in the upper row of the 16x2 LCD screen. This transition should be so neat that user doesn't even notice the real mechanism happening here.

END RESULT

https://photos.app.goo.gl/dBAMvbJfjp5pvSnA6

THE REAL CHALLENGE

The real challenge awaits when we want both of our characters need to work with a mutual coordination. The tricky part here is the movement of the cactus character is dependent on the left shift command of the LCD and when we left shift a character from the starting position of the screen i.e., from either 0x80 or 0xc0, the character appears on the end of the screen. This becomes a problem when we already have the Dino in the starting position of the screen. If we make the position of the Dino constant i.e., 0xc0 and 0xc1 (two positions required in order to make the Dino look like its running), the left shift command would reprint the Dino character on the end of the screen which is undesirable. Not only does it print the Dino character at the end of the screen but also moves the Dino character towards the left again in the next iteration again making it look like cactus is getting chased by Dino and not the other way around.

This problem is solved by printing the Dino character one step ahead after each left shift command. But this solution gives rise to another problem i.e., the already printed character that was displayed earlier will still be displayed on the end of the screen giving rise to multiple Dinos on completion of one full cycle. Using clear screen LCD command might seem like a good solution but it worsens the situation because it can't be used to clear any specific characters and rather clears all the characters that the screen was displaying before rendering the left shift loop useless. Thus a white space custom character was made to tackle the situation by printing the white space character on top of every previously printed Dinos but leaving only the currently printed Dino.

```
156 //****************process of moving the cactus while keeping the dino c
157   for(i=0;i<14;i++)
158   {
159              cnt=++cnt;
160     P1=mov[i+1];
161     cmdwrite();
162     scores();
163 P1=0x02;
164 datawrite();
165     scores();
166
167 P1=0xCF;
168 cmdwrite();
169     scores();
170 //delay(100);
171
172 P1=0x01;    //display cactii
173     datawrite();
174
175 P1=0x18;
176 cmdwrite();
177     scores();
178
```

RESULT

https://photos.app.goo.gl/ERKogrC5L9zPSLRA9

The next tricky part comes where the Dino has to jump while the cactus is approaching towards it. This is also achieved with the same solution but with few changes.

1. The pre-existing Dino character in row 1 is erased.
2. The cursor jumps to row 2 and prints the Dino custom character.
3. Erases the character in row 2 and again prints the character in row 1 again.

This full process has to take place in a smooth transition with the press of a button and the whole screen shifting left the whole time.

END RESULT

https://photos.app.goo.gl/aP7sQr6isxaQTcby7

On the last seen video a few of the glitches can be seen since it was taken during the development process and the final version had much more improvements with the Dino's movements and responses.

The next problem that was solved was showing the scores on a 4 digit 7 segment display. The problem with the 7 segment display is that if it is not fed with data continuously, the 4 digit 7 segment display displays the previously transmitted data on all four digits until it is again fed any set of data. **For example let us say it displays 0012 at one point of time and now it is not being fed with any data. This makes the 7 segment to display 2222 in the mean time which is totally undesirable.**

This problem was also solved by placing the 7 segment score counting function in specific targets of interest thus using very few left over microcontroller's resources.

Finally the end logic is made in such a way that whenever the Dino crashes onto the cactus the game shall stop displaying **"GAME OVER…"** displaying the player's total score.

SCORE COUNTER

Adding a multiplexed 4 digit 7 segment display as a counter to give the score of the player. It gets initialized to zero at start and will get paused when the game is over.

CODE IN KEIL UVISION

```
#include<reg51.h>
//#define display_port P1      //Data pins connected to port 2 on microcontroller
sbit rs = P3^2;  //RS pin connected to pin 2 of port 3
sbit rw = P3^3;  // RW pin connected to pin 3 of port 3
sbit en =  P3^4;  //E pin connected to pin 4 of port 3
sbit cen = P3^7;          //switch connection for dino movement
code unsigned char a[5]={0x02,0x38,0x0c,0x01,0x06};
code unsigned char
mov[16]={0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf}
;
code unsigned char
moc[16]={0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x
8f};
code unsigned char intro[]="WELCOME TO DINO";
code unsigned char intro2[]=" RUN....!!";
code unsigned char intro3[]="ENJOY...!!";
code unsigned char end[]="GAME OVER...";
int score[]={0xfc,0x60,0xda,0xf2,0x66,0xb6,0xbe,0xe0,0xfe,0xf6};
#define SegOne   0x01
#define SegTwo   0x02
#define SegThree 0x04
#define SegFour  0x08

void scores(void);
int dinojump(int);
unsigned int i,j,k,l,count;
int cnt, num, temp,temp1,temp2,temp3,h,zeo;
void delay(unsigned int time)
 {
unsigned int i,j;
for(i=0;i< time;i++)
for(j=0;j<5;j++);
 }
void cmdwrite()
 {
   rs=0;
   rw=0;
   en=1;
   delay(1);
```

```
    en=0;
  }
void datawrite()
  {
   rs=1;
   rw=0;
   en=1;
  delay(1);
  en=0;
  }
void main()
{
              unsigned char cactii[8]={
  0x04,
  0x05,
  0x15,
  0x15,
  0x0E,
  0x04,
  0x04,
  0x04
};
                                        unsigned char dino[8] =  {
  0x0F,
  0x0B,
  0x0F,
  0x0C,
  0x0F,
  0x0C,
  0x1C,
  0x06
};
                                              unsigned char blank[8]={
          0x00,
  0x00,
  0x00,
  0x00,
  0x00,
  0x00,
  0x00,
  0x00
};
    int i;
```

```
                P1 = (0x40+8);
                cmdwrite();
   for(i=0;i<8;i++)
                        {
                                P1 = cactii[i];
      datawrite();
                        }
                                delay(50);                      //making cactii and storing in 0x01
CGRAM
                P1 = (0x40+16);
                cmdwrite();
   for(i=0;i<8;i++)
                        {
                                P1 = dino[i];
      datawrite();
                        }
                                delay(50);                      //making dino and storing in 0x02
CGRAM

                        P1 = (0x40+24);
                cmdwrite();
   for(i=0;i<8;i++)
                        {
                                P1 = blank[i];
      datawrite();
                        }
                                delay(50);
for(i=0;i<=5;i++)
{
P1=a[i];
cmdwrite();
delay(100);
}
//***************INTRO***********************//
P1=0x80;
cmdwrite();
i=0;
for(i=0;i<15;i++)
{
        P1=intro[i];
        datawrite();
        delay(1000);
}
```

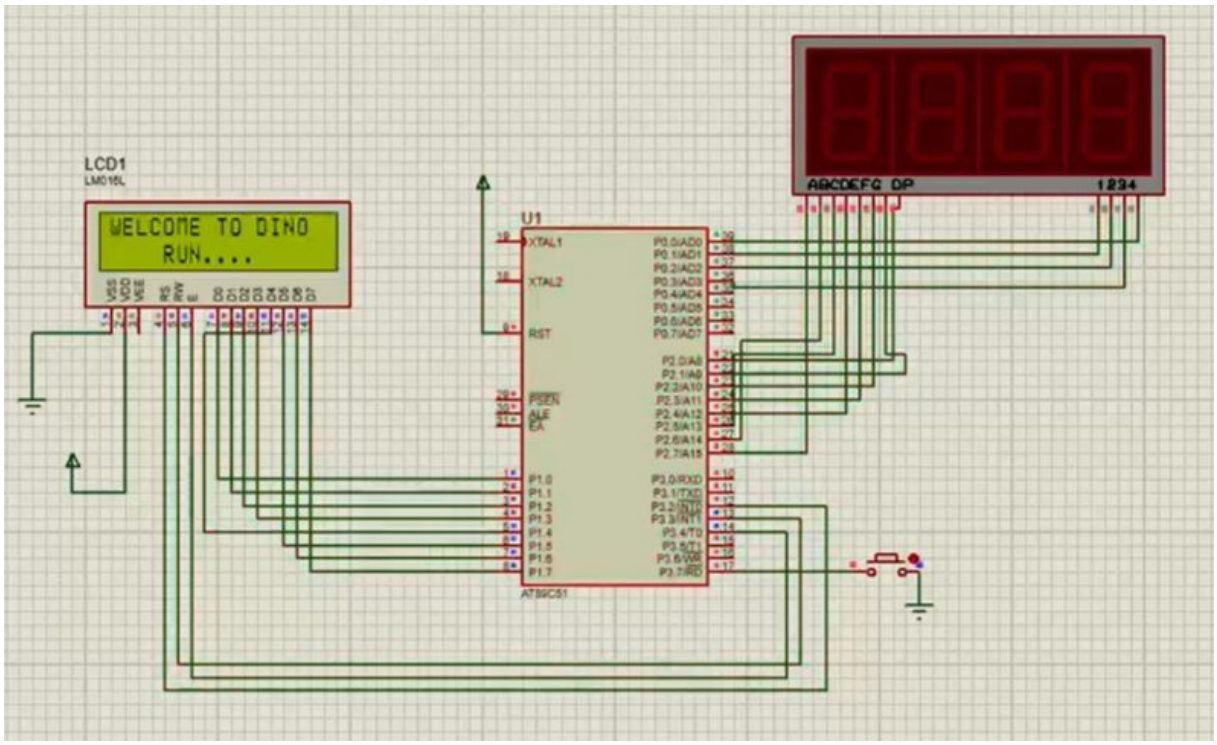
```
P1=0xc3;
cmdwrite();
i=0;
for(i=0;i<10;i++)
{
        P1=intro2[i];
        datawrite();
        delay(1000);
}
P1=0X01;
cmdwrite();
delay(100);
P1=0x82;
cmdwrite();
i=0;
for(i=0;i<10;i++)
{
        P1=intro3[i];
        datawrite();
        delay(1000);
}
        delay(1000);
//*****************************************************//
cnt= 0;
while(1)
{
cen = 1;
zeo=0;
//***************process of moving the cactus while keeping the dino constant by erasing the
dinos created behind every original dino*******************************//
        for(i=0;i<14;i++)
        {
                                                cnt=++cnt;
                P1=mov[i+1];
                cmdwrite();
                scores();
P1=0x02;
datawrite();
                scores();

P1=0xCF;
cmdwrite();
                scores();
```

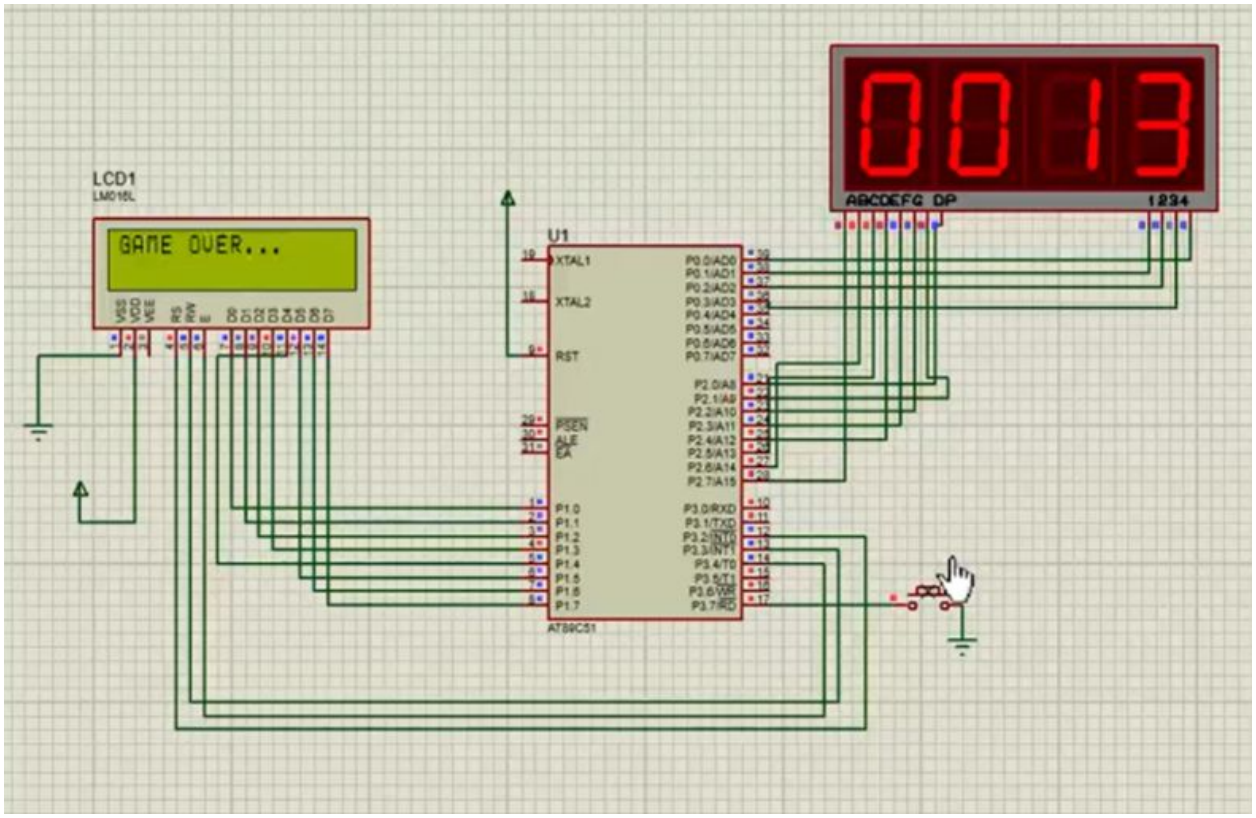
```
//delay(100);

P1=0x01;                //display cactii
                datawrite();
P1=0x18;
cmdwrite();
                scores();

delay(1000);
                                        if(cen==0)
{
i=dinojump(i+1);

}
        for(l=0;l<i+1;l++)
{
P1=mov[i+1];
                cmdwrite();
                P1=0x03;
                datawrite();
}
if(i==13 && cen==1)             //game over
{
                P1=0x01;
        cmdwrite();
                P1=0x82;
        cmdwrite();
                for(i=0;i<12;i++)
        {
                P1=end[i];
                        scores();
                datawrite();
                        scores();
        }
                while(1)
{
P1=0x08;
        cmdwrite();
        delay(100);
        P1=0x0c;
        cmdwrite();
        delay(50);
        scores();
```

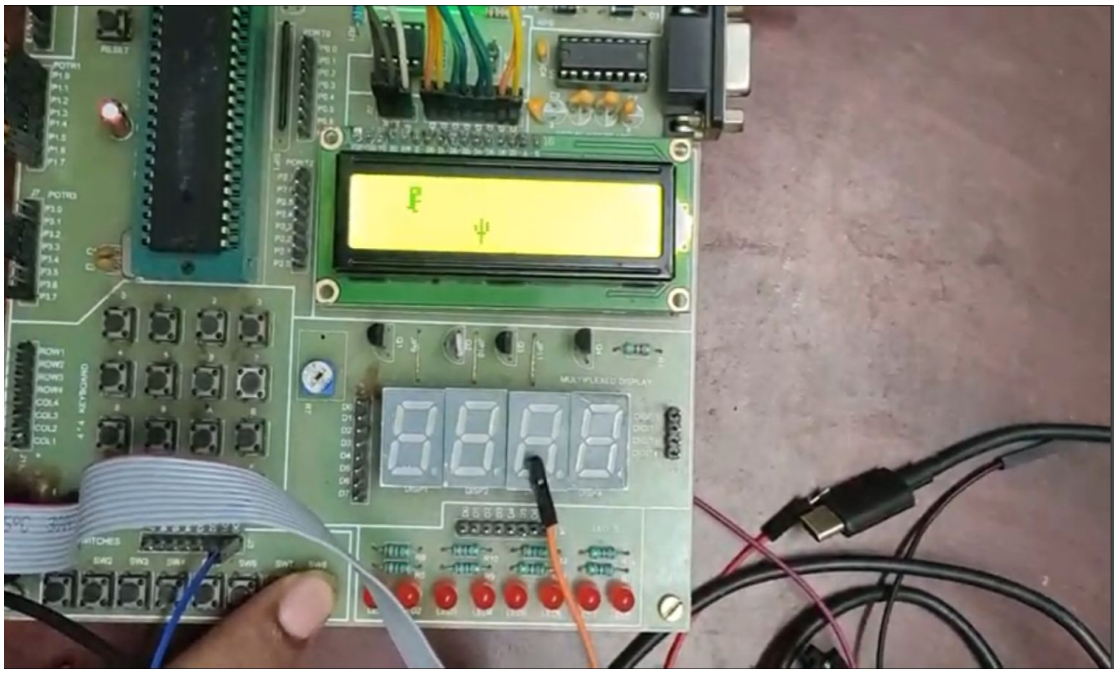
```
}
}
//*********************************************************************//

        }
        P1=0x01;
        cmdwrite();
        scores();
}
}

int dinojump(k)
{

        count=0;
                        P1=mov[k];
                cmdwrite();
                P1=0x03;
                datawrite();

        P1=moc[k+2];
        cmdwrite();
P1=0x02;
datawrite();
        delay(10000);

        P1=moc[k+2];
                cmdwrite();
                P1=0x03;
                datawrite();

        P1=mov[k+3];
        cmdwrite();
P1=0x02;
datawrite();
delay(10000);

        P1=mov[k+3];
        cmdwrite();
P1=0x03;
datawrite();
```

```
return(k);
}
void scores()
{
//*****************************************************//
        for (h = 0; h < 300; h++)
        {
                num = cnt;
           temp = num / 1000;
           num = num % 1000;
           P0 = SegOne;
           P2 = score[temp];
           //delay(10);

           temp1 = num / 100;
           num = num % 100;
           P0 = SegTwo;
           P2 = score[temp1];
           //delay(10);

           temp2 = num / 10;
           P0 = SegThree;
           P2 = score[temp2];
           //delay(10);

           temp3 = num % 10;
           P0 = SegFour;
           P2 = score[temp3];
           //delay(10);
                                                    }
        //*************************************************************//

                                                }

//**********FULLY WORKING DINO RUN GAME ON 8051***************//
//*********************CLEAN VERSION**********************//
```

FINAL GAME PROTOTYPE







https://photos.app.goo.gl/Ve8CM5rmkGjjEVde9 - PROTEUS SIMULATION

https://drive.google.com/open?id=1tpBkdMlkrsaau97OTtKyMms27SQ0TX7IznWfeYYbTC4 - original C file code

CONCLUSION

I would like to conclude with the ideas or knowledge we got about the basic registers,working of the 8051,LCD module and seven segment display. The dino game is successfully created and tested.All the constraints were eradicated.

Group Members:

M.Kowsyap Pranay(117EC0022)

D.Ajay(117EC0024)

M.Sreemukhi(117EC0034)

G.Sadhana(117EC0035)

Upendra(117EC0021)