UNIVERSITY OF Cincinnati

Final Project – T-Rex Jump, Survival of the Fittest

Presented by:
Aswin Balasubramaniam; Heath Palmer

Class section: 001
Due Date: December 9th, 2017

For instructor use

Students Last Name		
Report		
Conclusion		
Total		

Objective:

The objective of this project is to use the knowledge and experience gained from the Embedded Systems course and apply it to work on a final project to showcase creativity, and the expertise gained through this course.

Equipment (Hardware):

- Personal laptop/Desktop PC
- 8051 Development Board
- Keithley DC Power Supply

Equipment (Materials)

- Apex LCD (16x4) x1
- NO Pushbutton x2
- 4.53k Ω resistor x2
- Piezo buzzer x1
- Schmitt Trigger x2
- 22 µF Capacitor x2
- SN74LS245 Buffer x1
- Potentiometer x1

Equipment (Software):

- MCU8051 IDE
- Atmel FLIP 3.4.7

Reference:

- Hantronix LCD Datasheet
- 8051 Online resources
- Ricky's World – LCD Custom Character Creation

Project Description:

The final project uses the 8051-microcontroller and the Apex LCD to create a gaming experience, called the T-Rex Jump, Survival of the Fittest. The project uses two pushbuttons and a piezo buzzer to enhance the gaming experience.
The game is set in the Ice Age Period where our character, T-Rex, jumps above obstacles surrounding him, to survive his way out this apocalyptic period.
The two pushbuttons provided, allow the user to control the T-Rex and reset the game. Pushbutton 1, allows the user to help T-Rex jump above obstacles, and keep it safe. Pushbutton 2 allows the user to reset the game, when the user's negligence kills the T-Rex.
The piezobuzzer, augments the gaming experience by creating sound effects when the T-Rex jumps, or meets his demise.

Procedure:

In this final lab, the main objective is to use the knowledge and experience gained from the Embedded Systems course and apply it to work on a final project to showcase creativity, and the expertise gained through this course. To achieve everything in the Project Description above, we needed to understand the LCD initialization, Custom Character creation, circuit to de-bounce the push-buttons, and JB/JNB logic. The project was incremented into development phases that ended up coming together. The first phase included the adequate jumping in response to the push button, along with the Title and Game Over screen displaying at the proper time. The next phase included ground terrain placement, custom character creation, implementation of those characters, and collision detection. Finally, the code was implemented into one program and successfully tested. The circuit schematic in Figure 1 shows the circuit schematic for the Final Project.

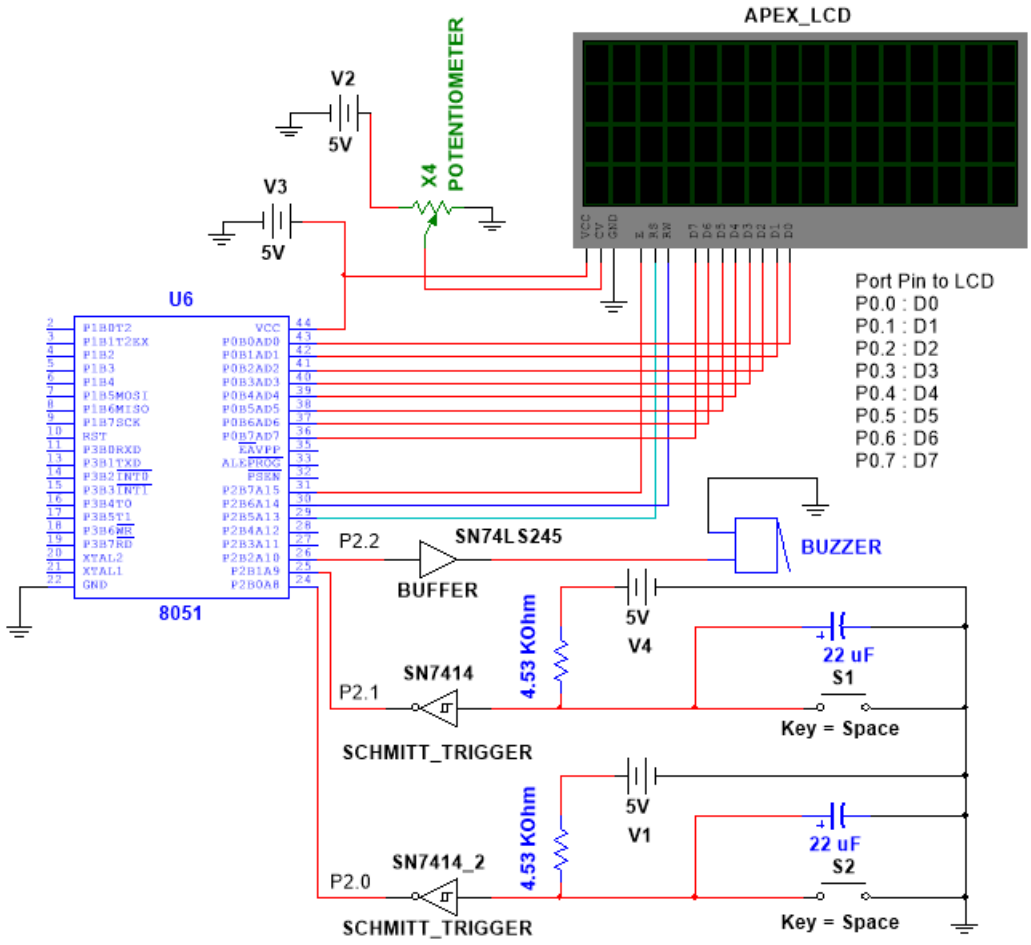

Figure 1: T-Rex Game Circuit Schematic

The LCD circuit in Figure 1 is very similar to setups from past labs, it is arranged in a similar fashion. The 8051 Port 0.0-0.7 is connected to the D0-D7 accordingly. A change includes the addition of the Buffer and Buzzer to Port 2.2. This buzzer will sound when the jump button has been pressed. Another change includes the addition of the de-bouncing pushbutton circuits connected to Ports 2.0 and 2.1. Port 2.0 is the button used to Jump in the game, while Port 2.1 is used to Reset the game. The capacitor filters out any quick changes in the signal as this helps prevent error in the signal sent to the 8051 board. The Schmitt Trigger Inverter IC inverts the signal sent to it and generates a smooth digital signal.

The first phase required understanding of the LCD initialization and display logic from recent labs. The initial cursor position changes from the Title Screen to the in-game screen and finally to the Game over screen. The logic to move the cursor while jumping did not require initialization of the LCD, but required the cursor location to be changed. Every time the Jump button was pressed, the character would move up a row on the LCD. When released, the character would move back down a row on the LCD. These parameters were met as part of the Pseudo Code.

The second phase required understanding of object collision and character creation on the LCD. To detect a collision, the program checks when the cactus reaches DDRAM address 11 on the LCD and then check if the Port 2.0 was pull HIGH. If it was HIGH, then that means the character jumped and there is no collision. If the Port 2.0 was LOW when the obstacle was in position 11 on the LCD, then that would result in a Game Over. When there is a Game Over, the program jumps to a routine to display “GAME OVER” centered on the second line of the LCD. While joining the two programs, close review of the variables and logic was required.

Pseudo Code:

1. Display the name of the project on the screen
2. Pushbutton 1 starts the game.
3. Display black boxes on the bottom 1 row.
4. Define the ASCII value for T-Rex (x2) potentially (x4) (to duck)
5. Define the ASCII values for obstacles
6. Display stationary T-Rex, and few obstacles.
7. Write program to display T-Rex, obstacles, and enable sound every time Pushbutton 1 is pushed.
8. Pushbutton 2 resets the game when the user loses the game.
9. The game’s display relies on timers.
10. The ASCII value for T-Rex is going to be saved in variable TREX1; LCD Location: 40H
11. The ASCII value for cactus is going to be saved in variable CACTUS1; LCD Location 48H
12. The ASCII value for left leg T-Rex is going to be saved in variable TREX2; LCD Location: 50H (Not used in this project)
13. The ASCII value for right leg T-Rex is going to be saved in variable TREX3; LCD Location: 58H (Not used in this project)
14. The ASCII value for big cactus is going to be saved in variable CACTUS2; LCD Location: 60H (Not used in this project)

15. The ASCII value for big cactus is going to be saved in variable CACTUS3; LCD Location: 68H (Not used in this project)
16. The ASCII value for big cactus is going to be saved in variable CACTUS4; LCD Location: 70H (Not used in this project)

Backend Code:

For collision:

1. Jump to Subroutine DETECTOBS
2. Check if obstacle in position 1 (fixed position of T-Rex), and if pushbutton is pressed
3. If pushbutton pressed:
    a. Check if the obstacle is in position 1. If obstacle in position 1 then game over.
    b. Check if the obstacle is in position 1. If obstacle is not in position 1 then game continues.
4. If the pushbutton is not pressed:
    a. Check if the obstacle is in position 1. If the obstacle is in position 1 then game over.
    b. Check if the obstacle is in position 1. If the obstacle is not in position 1 then game continues.
5. If the game is over, GAMEOVER flag is set high. If the GAMEOVER flag is set high then, pushbutton 2 resets the game.

For Jump:

1. When P2.0 is pulled High, jump to the JUMP tag
    a. Move a Space (‘ ‘) character into the original LCD position 11
    b. Move the character to the new LCD position 41 for the button press duration

For the Character:

1. The Dinosaur character has been coded; however, for functionality they are currently on hold.
    a. Three Dinosaur characters: Two while running and one while jumping
2. The cactus obstacles have been coded; however, for functionality they are currently on hold.
    a. There are four Cactus obstacles randomly generated

Only two of the six custom characters are used in this project. Due to some issues that arose during the troubleshooting phase, all the six characters could not be implemented in the project. The Characters below in Figure 2 are in line with their respective Hex and Decimal values.

Bitmap	Decimal	Hex
	2	0x2
	7	0x7
	7	0x7
	6	0x6
	7	0x7
	22	0x16
	30	0x1e
	10	0xa

Bitmap	Decimal	Hex
	2	0x2
	7	0x7
	7	0x7
	6	0x6
	7	0x7
	22	0x16
	30	0x1e
	2	0x2

Bitmap	Decimal	Hex
	2	0x2
	7	0x7
	7	0x7
	6	0x6
	7	0x7
	22	0x16
	30	0x1e
	8	0x8

Bitmap	Decimal	Hex
	0	0x0
	0	0x0
	4	0x4
	21	0x15
	21	0x15
	29	0x1d
	7	0x7
	4	0x4

Bitmap	Decimal	Hex
	0	0x0
	0	0x0
	4	0x4
	21	0x15
	21	0x15
	23	0x17
	28	0x1c
	4	0x4

Bitmap	Decimal	Hex
	0	0x0
	20	0x14
	20	0x14
	20	0x14
	29	0x1d
	5	0x5
	7	0x7
	4	0x4

Bitmap	Decimal	Hex
	0	0x0
	5	0x5
	5	0x5
	5	0x5
	23	0x17
	20	0x14
	28	0x1c
	4	0x4

Figure 2: T-Rex Game LCD Custom Characters

As the 8051 Board is powered on, the T-Rex Jump Game is initiated with the Intro screen – "TREX GAME" seen in Figure 3 (Left Photo) below. The game then loads in the Ground terrain, T-Rex avatar and the Cactus obstacle. The T-Rex then runs toward the Cactus and attempts to jump over the obstacle when the Pushbutton connected to P2.0 is pressed. This operation can be viewed in Figure 3 (Right Photo) below.

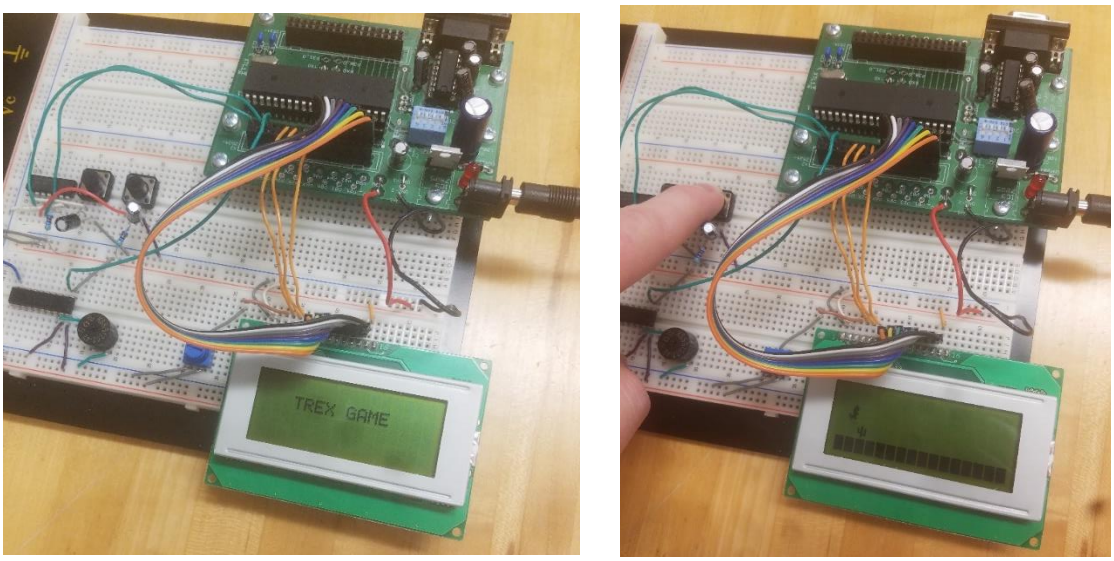

Figure 3: T-Rex Game Intro Screen and Jump Pushbutton Activated Screens

If the player does not time their jumping well enough, the T-Rex will run into the Cactus. This will result in a Game Over screen seen in Figure 4. The program is constantly comparing object placement with the signal of Port 2.0. Since P2.0 was LW while the Cactus was in LCD position 11, this results in a collision.

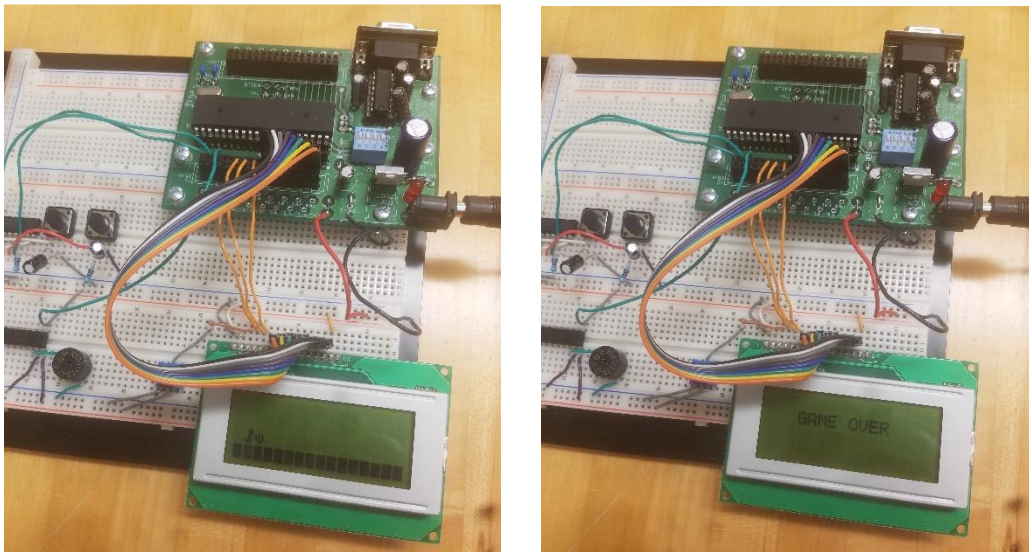

Figure 4: T-Rex Game Collision and Game Over Screens

Source Code:

```
; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ;
; ***          Embedded Systems Final Project                 *** ;
; ***      Authors: Aswin Balasubramaniam, Heath Palmer       *** ;
; ***         T-Rex Jump, Survival of The Fittest             *** ;
; ***                        12/7/2017                        *** ;
; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ;


.ORG 0000H


;; =====  VARIABLE LOCATION INITIALIZATION ===== ;;
.      equ COUNTDOWN, 30H  ;VARIABLE THAT STORES THE VALUE OF COUNT FOR 1 SEC
.equ VALUE, 31H            ;VARIABLE THAT STORES VALUE TO REFERNECE TABLE
.      equ LENGTH1, 32H    ;VARAIBLE THAT STORES LENGTH OF INTRO TEXT
.      equ LENGTH2, 33H    ;VARIABLE THAT STORES LENGTH OF END TEXT
.equ CURSORINTRO, 34H      ;VARIABLE THAT STORES THE STARTING ADDRESS FOR INTRO
.      equ CURSOREND, 35H  ;VARIABLE THAT STORES THE STARTING ADDRESS FOR OVER
.      equ TREXUP, 36H     ;VARIABLE THAT STORES THE TREX CHARACTER VALUE
.      equ VALUETER, 37H   ;VARIABLE THAT STORES THE VALUE OF LENGTH OF GROUND
.      equ CURSORTER, 38H  ;VARIABLE THAT STORES THE STARTING ADDRESS FOR GROUND
.      equ VALUEOBS, 39H   ;VARIABLE THAT STORES THE LENGTH THE OBSTACLE TRAVELS
.      equ CURSOROBS, 3AH  ;VARIABLE THAT STORES THE STARTING ADDRESS OF OBSTACLE
.equ CURSOROBS_1, 3BH      ;VARIABLE THAT STORES THE NEXT ADDRESS OF OBSTACLE
.equ COUNTDOWNOBS, 3CH     ;VARIABLE THAT STORES THE VALUE OF FPS FOR OBSTACLE (X50MS)
.      equ GAMEDONE, 3DH   ;VARIABLE THAT STORES 1 TO INDICATE GAME IS OVER
.      equ BUZZER, 3EH     ;VARIABLE THAT STORES HOW LONG THE BUZZER BEEPS


STARTOVER:
;; =====  VARIABLE VALUE INITIALIZATION ===== ;;
MOV COUNTDOWN, #14H         ;DECIMAL VALUE OF 20 TO HELP TO COUNT TO 1 SECOND
MOV VALUE, #00H             ;VARIABLE THAT HELPS CALL THE VALUES FROM THE INTRO AND
GAME OVER TABLE
MOV LENGTH1, #09H           ;VALUE OF 9, LENGTH OF INTRO TEXT
MOV LENGTH2, #09H           ;VALUE OF 9, LENGTH OF GAME OVER TEXT
```

```
MOV CURSORINTRO, #0C3H      ;CURSOR POSITION (80 + DDRAM ADDRESS)
MOV CURSOREND, #0C3H        ;CURSOR POSITION (80 + DDRAM ADDRESS)
MOV TREXUP, #00H            ;VALUE OF 0
MOV VALUETER, #10H          ;VALUE OF 17 IN DECIMAL (16X4 LCD)
MOV CURSORTER, #0D0H        ;CURSOR POSITION (80 + DDRAM ADDRESS)
MOV VALUEOBS, #0FH          ;VALUE OF 16 IN DECIMAL (16X4 LCD)
MOV CURSOROBS, #9FH         ;CURSOR POSITION (80 + DDRAM ADDRESS)
MOV CURSOROBS_1, #9FH       ;CURSOR POSITION (80 + DDRAM ADDRESS)
MOV COUNTDOWNOBS, #0FH      ;VALUE OF 16 IN DECIMAL (16X4 LCD)
MOV GAMEDONE, #00H          ;VALUE OF 0
MOV BUZZER, #0AH            ;VALUE OF 11
MOV A, #00H
LCALL TREX1                 ;CALLS THE TREX1 SUBROUTINE TO WRITE THE CUSTOM CHARACTER
TO LCD
LCALL CACTUS1               ;CALLS THE CACTUS1 SUBROUTINE TO WRITE THE CUSTOM CHARACTER
TO LCD


;; ===== LCD INITIALIZATION ===== ;;
LCDINIT:                    ;(PINS: P2.5-RS(DATA/INSTRUCTION SELECT), P2.6-
R/W(READ/WRITE), P2.7-E(ENABLE))
MOV TMOD, #01H              ;SETS TIMER 1 MODE
CLR P2.5                    ;SETS RS FOR INSTRUCTION MODE
CLR P2.6                    ;SETS R/W FOR WRITE MODE
LCALL TIMERSUB40M           ;40 MS TIMER DELAY TO SETUP THE LCD
MOV P0, #38H                ;DATA WRITTEN TO LCD FOR SETUP
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV P0, #0CH                ;SWITCHES ON LCD
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV P0, #01H                ;CLEARS THE LCD
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND


;; ===== DISPLAY INTRO AND GAME'S GROUND ===== ;;
```

```
GAMEINTRO:
SETB P2.0                   ;PUSH BUTTON 1 - PLAYER JUMP UP BUTTON
SETB P2.1                   ;CLEAR SCREEN - RESET GAME
LCALL INTRO                 ;DISPLAY TREX GAME
LCALL TIMERSUB40M           ;CEREATES A 40 MS TIME DELAY
LCALL CLEARSCR              ;CALLS SUBROUTINE TO CLEAR THE LCD SCREEN
LCALL GROUND                ;CALLS SUBROUTINE TO DISPLAY GROUND

;; ===== GAME AND OBSTACLE LOGIC ===== ;;
MAINGAME:
OBS:
LCALL GAME                  ;CALLS SUBROUTINE TO DISPLAY TREX
LCALL OBSTACLE              ;CALLS SUBROUTINE TO DISPLAY OBSTACLE
MOV A, GAMEDONE             ;MOVES VALUE OF GAMEDONE VARIABLE TO ACCUMULATOR
CJNE A, #01H, CONTINUE1     ;CHECKS IF THE GAME IS OVER OR NOT
SJMP STARTOVER              ;JUMPS TO TAG STARTOVER IF GAME IS OVER
CONTINUE1:
SJMP OBS                    ;JUMPS TO TAG OBS IF GAME IS NOT OVER
SJMP ENDALL                 ;WILL NEVER HAPPEN


;; ===== SHOULD NEVER JUMP HERE ===== ;;
ENDALL:
SJMP ENDALL


;; ================================ ;;
;; ===== SUBROUTINES UTILIZED ===== ;;
;; ================================ ;;


;; ===== SUBROUTINE THAT DISPLAYS GROUND ===== ;;
;; ===== START ===== ;;
```

```
GROUND:
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV P0, #0D0H              ;MOVES THE CURSOR TO THE FOURTH ROW FIRST POSITON
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40M          ;CREATES A 40 MS DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #20H               ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LOOPGND:
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, CURSORTER           ;MOVES CURRENT VALUE AT CURSORTER TO ACCUMULATOR
LCALL SETCURSOR            ;SUBROUTINE THAT MOVES THE CURSOR TO THE LOCATION ON LCD
MOV A, CURSORTER
ADD A, #01H                ;UPDATES THE VALUE IN VARIABLE CURSORTER
MOV CURSORTER, A
LCALL TIMERSUB40M          ;CREATES A 40 MS DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #0FFH              ;WRITES A DARK BLOCK TO THE POSITION ON THE LCD SCREEM
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
DJNZ VALUETER, LOOPGND     ;LOOPS TILL IT DECREMENTS THE VALUE IN VALUETER VARIABLE TO
0
MOV CURSORTER, #0D0H       ;RESETS CURSORTER VALUE
LCALL SETCURSOR            ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR


RET
;; ===== END ===== ;;



;; ===== SUBROUTINE THAT DISPLAYS OBSTACLE ===== ;;
;; ===== START ===== ;;
OBSTACLE:
CLR P2.5                   ;SETS RS FOR INSTRUCTION MODE
CLR P2.6                   ;SETS R/W FOR WRITE MODE
LCALL TIMERSUB40M          ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
```

```
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV P0, #9FH               ;MOVES THE CURSOR TO THE 3RD LINE LAST POSITION
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40M          ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #20H               ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, CURSOROBS           ;MOVES CURRENT VALUE OF CURSOROBS TO ACCUMULATOR
LCALL SETCURSOR            ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
MOV A, CURSOROBS_1         ;MOVES CURRENT VALUE OF CURSOROBS_1 TO ACCUMULATOR
SUBB A, #01H               ;UPDATES THE VALUE IN CURSOROBS_1 VARIABLE
MOV CURSOROBS_1, A
LCALL TIMERSUB40M          ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #01H               ;WRITES THE CACTUS CUSTOM CHARACTER TO CURSOR LOCATION
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL OBSDELAY             ;CALLS SUBROUTINE TO DELAY VIEWING THE NEXT OBSTACLE
LCALL GAME                 ;CALLS THE SUBROUTINE TO INTERACT WITH THE CHARACTER
LOOPOBS:                   ;LOOP TO DISPLAY OBSTACLES
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, CURSOROBS_1         ;MOVES CURRENT VALUE OF CURSOROBS_1 TO ACCUMULATOR
LCALL SETCURSOR            ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
MOV A, CURSOROBS_1         ;MOVES CURRENT VALUE OF CURSOROBS_1 TO ACCUMULATOR
SUBB A, #01H               ;UPDATES THE VALUE IN CURSOROBS_1 VARIABLE
MOV CURSOROBS_1, A
LCALL TIMERSUB40M          ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #01H               ;WRITES THE CACTUS CUSTOM CHARACTER TO CURSOR LOCATION
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, CURSOROBS           ;MOVES CURRENT VALUE OF CURSOROBS TO ACCUMULATOR
LCALL SETCURSOR            ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
```

```
MOV A, CURSOROBS          ;MOVES CURRENT VALUE OF CURSOROBS TO ACCUMULATOR
SUBB A, #01H              ;UPDATES THE VALUE IN CURSOROBS_1 VARIABLE
MOV CURSOROBS, A
LCALL TIMERSUB40M         ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                 ;SETS RS FOR DATA MODE
MOV P0, #20H              ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE           ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL OBSDELAY            ;CALLS SUBROUTINE TO DELAY VIEWING THE NEXT OBSTACLE
LCALL GAME                ;CALLS THE SUBROUTINE TO INTERACT WITH THE CHARACTER
MOV A, CURSOROBS_1        ;MOVES CURRENT VALUE OF CURSOROBS TO ACCUMULATOR
INC A                     ;UPDATES THE VALUE IN CURSOROBS_1 VARIABLE
CJNE A, #92H, CHECKED     ;CHECKS IF THE CHARACER HAS COLLIDED
SJMP COLLISION            ;JUMPS TO SUUBROUTINE COLLISION IF COLLIDED
CHECKED:
DJNZ VALUEOBS, LOOPOBS    ;LOOPS TILL IT DECREMENTS THE VALUE IN VALUETER VARIABLE TO
0
LCALL TIMERSUB40U         ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, CURSOROBS          ;MOVES CURRENT VALUE OF CURSOROBS TO ACCUMULATOR
LCALL SETCURSOR           ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
LCALL TIMERSUB40M         ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                 ;SETS RS FOR DATA MODE
MOV P0, #20H              ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE           ;CYCLES ENABLE TO INITIATE THE COMMAND
MOV VALUEOBS, #0FH        ;RESETS THE VALUE IN VALUEOBS
MOV CURSOROBS, #9FH       ;RESETS THE VALUE IN CURSOROBS
MOV CURSOROBS_1, #9FH     ;RESETS THE VALUE IN CURSOROBS_2
SJMP RETURNOBS            ;JUMPS TO RETURN TAG TO RETURN FROM SUBROUTINE

COLLISION:
JB P2.0, CHECKED          ;IF PUSHBUTTON IS PUSHED THEN GAME NOT OVER
LCALL ONESECOND           ;IF NOT PUSHED DELAY FOR 3 SECONDS
LCALL ONESECOND
LCALL CLEARSCR            ;SUBROUTINE THAT CLEARS THE LCD SCREEN
LCALL OVER                ;CALLS SUBROUTINE TO DISPLAY GAMEOVER TEXT
```

```
LCALL PAUSEOVER            ;PAUSES THE GAME IN THE SUBROUTINE TILL RESET PUSHBUTTON IS
PUSHED


RETURNOBS:

RET

;; ===== END ===== ;;



;; ===== SUBROUTINE THAT CREATES CACTUS ===== ;;

;; ===== START ===== ;;

CACTUS1:

LCALL LCD_LOCATIONCMD

MOV P0,#48H

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV P0, #00H

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#00H                  ;LOADS ROW 1 DATA

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#04H                   ;LOADS ROW 1 DATA

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#15H                   ;LOADS ROW 3 DATA

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#15H                   ;LOADS ROW 4 DATA

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#1DH                   ;LOADS ROW 5 DATA

LCALL LCD_COMMAND

LCALL LCD_SENDDATA

MOV   P0,#07H                   ;LOADS ROW 6 DATA
```

```
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#04H                    ;LOADS ROW 7 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA

RET                         ;RETURNS FROM SUBROUTINE
;; ===== END ===== ;;



;; ===== SUBROUTINE THAT CREATES TREX ===== ;;
;; ===== START ===== ;;
TREX1:
LCALL LCD_LOCATIONCMD
MOV P0,#40H
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV P0, #02H
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#07H                    ;LOADS ROW 1 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#07H                    ;LOADS ROW 2 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#06H                    ;LOADS ROW 3 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#07H                    ;LOADS ROW 4 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#16H                    ;LOADS ROW 5 DATA
```

```
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#1EH                    ;LOADS ROW 6 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA
MOV   P0,#0AH                    ;LOADS ROW 7 DATA
LCALL LCD_COMMAND
LCALL LCD_SENDDATA

RET                         ;RETURNS FROM SUBROUTINE
;; ===== END =====


;; ===== SUBROUTINES THAT AIDS IN CREATING CUSTOM CHARACTER ===== ;;
;; ===== START ===== ;;
LCD_LOCATIONCMD:
CLR P2.5                    ;SETS RS FOR INSTRUCTION MODE
CLR P2.6                    ;SETS R/W FOR WRITE MODE
LCALL TIMERSUB40M           ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION

LCD_COMMAND:
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
RET

LCD_SENDDATA:
SETB P2.5                   ;SETS RS FOR DATA MODE
CLR P2.6                    ;SETS R/W FOR WRITE MODE
LCALL TIMERSUB40M           ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
RET
;; ===== END ===== ;;
```

```
;; ===== SUBROUTINE AND TAG THAT DISPLAYS CHARACTER ===== ;;
;; ===== START ===== ;;
GAME:
CHARACTER:
SETB P2.0                   ;PUSH BUTTON 1 - PLAYER JUMP UP BUTTON
SETB P2.1                   ;CLEAR SCREEN - RESET GAME
JB P2.1, RESETGAME          ;RESETS GAME IF PB2 IS PRESSED
JB P2.0, JUMP               ;WHEN HIGH THE CHARACTER IS JUMPING UP
LCALL STAND                 ;CALLS THE STAND SUBROUTINE
RETURN:
RET

JUMP:
LCALL JUMPUP                ;CALLS THE JUMPUP SUBROUTINE
LCALL BEEP                  ;CALLS THE BEEP SUBROUTINE
SJMP RETURN

RESETGAME:
CLR P2.5                    ;SETS RS FOR INSTRUCTION MODE
CLR P2.6                    ;SETS R/W FOR WRITE MODE
MOV P0, #01H                ;CLEAR LCD SCREEN
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL SETCURSOR1            ;CALLS THE SUBROUTINE TO SET THE CURSOR
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SJMP RETURN
;; ===== END ===== ;;



;; ===== SUBROUTINE THAT DISPLAYS THE CHARACTER STANDING ===== ;;
;; ===== START ===== ;
STAND:
LCALL SETCURSOR2            ;SUBROUTINE THAT POINTS TO THE LOCATION WHERE THE CHARACTER
IS STANDING
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
```

```
CLR P2.6                    ;SETS R/W FOR WRITE MODE
MOV P0, #10H
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR1            ;CALLS THE SUBROUTINE TO SET THE CURSOR
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
CLR P2.6                    ;SETS R/W FOR WRITE MODE
SETB P2.5                   ;SETS LCD TO DATA MODE
MOV P0, #00H                ;MOVES DATA TO LCD DATA BUS
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR1            ;CALLS THE SUBROUTINE TO SET THE CURSOR
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
CLR P2.6                    ;SETS R/W FOR WRITE MODE
MOV DPTR, #STANDING         ;POINTS TO STANDING TABLE
MOV A, TREXUP               ;MOVES TREXUP VALUE TO ACCUMULATOR
ADD A, #01H                 ;ADDS 1 TO ACCUMMULATOR AND DECIMAL ADJUSTS THE VALUE
MOV TREXUP, A               ;MOVES ACCUMULATOR VALUE TO COUNTUP VARIABLE
MOVC A, @A+DPTR             ;MOVES VALUE IN TABLE TO ACCUMULATOR
SETB P2.5                   ;SETS LCD TO DATA MODE
MOV P0, #00H                ;MOVES CUSTOM CHARACTER OF TREX TO LCD
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M            ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
MOV A, 00H                  ;RESETS THE VALUE IN THE ACCUMULATOR

RET
;; ===== END =====;;



;; ===== SUBROUTINE THAT DISPLAYS THE CHARACTER JUMPING ===== ;;
;; ===== START ===== ;;
JUMPUP:
LCALL SETCURSOR1            ;SUBROUTINE THAT POINTS TO THE LOCATION WHERE THE CHARACTER
IS JUMPING
```

```
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
CLR P2.6                  ;SETS R/W FOR WRITE MODE
MOV P0, #10H
LCALL TIMERSUBE           ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR2          ;CALLS THE SUBROUTINE TO SET THE CURSOR
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
CLR P2.6                  ;SETS R/W FOR WRITE MODE
SETB P2.5                 ;SETS LCD TO DATA MODE
MOV P0, #00H              ;MOVES CUSTOM CHARACTER OF TREX TO LCD
LCALL TIMERSUBE           ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR2          ;CALLS THE SUBROUTINE TO SET THE CURSOR
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
CLR P2.6                  ;SETS R/W FOR WRITE MODE
MOV DPTR, #UP             ;POINTS TO UP TABLE
MOV A, TREXUP
ADD A, #01H               ;ADDS 1 TO ACCUMMULATOR AND DECIMAL ADJUSTS THE VALUE
MOV TREXUP, A             ;MOVES ACCUMULATOR VALUE TO COUNTUP VARIABLE
MOVC A, @A+DPTR           ;MOVES VALUE IN TABLE TO ACCUMULATOR
SETB P2.5                 ;SETS LCD TO DATA MODE
MOV P0,#00H               ;MOVES CUSTOM CHARACTER OF TREX TO LCD
LCALL TIMERSUBE           ;CYCLES ENABLE TO INITIATE THE COMMAND
LCALL TIMERSUB5M          ;CREATES A 5 MILLISECOND DELAY BEFORE NEXT INSTRUCTION

RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT POINTS THE CURSOR TO STAND POSITION OF CHARACTER ===== ;;
;; ===== START ===== ;;
SETCURSOR1:
CLR P2.5                  ;SETS RS FOR INSTRUCTION MODE
```

```
CLR P2.6                    ;SETS R/W FOR WRITE MODE
NOP
MOV P0, #91H                ;MOVES THE CURSOR TO DDRAM ADDRESS 11 (80+11)
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
SETB P2.5                   ;SETS R/W FOR READ MODE
SETB P2.6                   ;SETS RS FOR DATA MODE
NOP

RET
;; ===== END =====;;


;; ===== SUBROUTINE THAT POINTS THE CURSOR TO JUMP POSITION OF CHARACTER ===== ;;
;; ===== START ===== ;;
SETCURSOR2:
CLR P2.5                    ;SETS RS FOR INSTRUCTION MODE
CLR P2.6                    ;SETS R/W FOR WRITE MODE
NOP
MOV P0, #0C1H               ;MOVES THE CURSOR TO DDRAM ADDRESS 41 (80+C1)
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
SETB P2.5                   ;SETS R/W FOR READ MODE
SETB P2.6                   ;SETS RS FOR DATA MODE
NOP

RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT CLEARS THE LCD SCREEN ===== ;;
;; ===== START ===== ;;
CLEARSCR:                   ;SUBROUTINE TO CLEAR THE LCD SCREEN
CLR P2.5                    ;SETS RS FOR INSTRUCTION MODE
MOV P0, #01H                ;CLEARS LCD SCREEN
```

```
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND

RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT ENABLES BEEP ===== ;;
;; ===== START ===== ;;
BEEP:
CLR P2.2                   ;SETS P2.2 LOW
MAIN:
LCALL DELAYBUZZER          ;CALLS THE DELAYBUZZER SUBROUTINE
SETB P2.2                  ;SETS P2.2 HIGH
LCALL DELAYBUZZER          ;CALLS THE DELAYBUZZER SUBROUTINE
CLR P2.2                   ;SETS P2.2 LOW
DJNZ BUZZER, MAIN          ;LOOPS TILL IT DECREMENTS THE VALUE IN BUZZER VARIABLE TO 0
MOV BUZZER, #0AH           ;RESETS THE VALUE IN BUZZER

RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT DISPLAYS THE GAME TITLE ===== ;;
;; ===== START ===== ;;
INTRO:
MOV A, CURSORINTRO         ;MOVES CURSORINTRO VALUE TO ACCUMULATOR
LCALL TIMERSUB40U          ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR            ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
LCALL TIMERSUB40M          ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                  ;SETS RS FOR DATA MODE
MOV P0, #20H               ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND
INTROLOOP:
```

```
MOV A, CURSORINTRO          ;MOVES CURSORINTRO VALUE TO ACCUMULATOR
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR             ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
MOV A, CURSORINTRO          ;MOVES CURSORINTRO VALUE TO ACCUMULATOR
ADD A, #01H                 ;UPDATES THE VALUE IN CURSORINTRO VARIABLE
MOV CURSORINTRO, A
MOV DPTR, #TITLE            ;POINTS TO TABLE TITLE TO DISPLAY TREX GAME
MOV A, VALUE
MOVC A, @A+DPTR             ;MOVES VALUE IN TABLE TO ACCUMULATOR
LCALL TIMERSUB40M           ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                   ;SETS LCD TO DATA MODE
MOV P0, A                   ;MOVES DATA TO LCD DATA BUS
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
MOV A, VALUE
INC A                       ;UPDATES VARIABLE IN VALUE
MOV VALUE, A
DJNZ LENGTH1, INTROLOOP     ;LOOPS TILL IT DECREMENTS THE VALUE IN LENGTH1 VARIABLE TO
0
LCALL ONESECOND             ;CALLS ONESECOND DELAY
MOV VALUE, #00H             ;RESETS THE VALUE IN VARIABLE VALUE
LCALL PAUSE                 ;CALLS PAUSE SUBROUTINE
MOV CURSORINTRO, #0C3H      ;RESETS THE VALUE IN CURSORINTRO VARIABLE


RET
;; ===== END ===== ;;



;; ===== SUBROUTINE THAT DISPLAYS GAME OVER ===== ;;
;; ===== START ===== ;;
OVER:
MOV A, CURSOREND            ;MOVES CURSOREND VALUE TO ACCUMULATOR
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR             ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
LCALL TIMERSUB40M           ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
```

```
SETB P2.5                   ;SETS LCD TO DATA MODE
MOV P0, #20H                ;CLEARS THE BLOCK AT THE CURSOR POSTION (20 IS A CLEAR BOX)
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
OVERLOOP:
MOV A, CURSOREND            ;MOVES CURSOREND VALUE TO ACCUMULATOR
LCALL TIMERSUB40U           ;CREATES A 40 MICROSECOND DELAY BEFORE NEXT INSTRUCTION
LCALL SETCURSOR             ;CALLS SUBROUTINE TO SET THE POSITION OF CURSOR
MOV A, CURSOREND            ;MOVES CURSOREND VALUE TO ACCUMULATOR
ADD A, #01H                 ;UPDATES THE VALUE IN CURSOREND VARIABLE
MOV CURSOREND, A
MOV DPTR, #GAMEOVER         ;POINTS TO TABLE TITLE TO DISPLAY GAME OVER
MOV A, VALUE
MOVC A, @A+DPTR             ;MOVES VALUE IN TABLE TO ACCUMULATOR
LCALL TIMERSUB40M           ;CREATES A 40 MILLISECOND DELAY BEFORE NEXT INSTRUCTION
SETB P2.5                   ;SETS LCD TO DATA MODE
MOV P0, A                   ;MOVES DATA TO LCD DATA BUS
LCALL TIMERSUBE             ;CYCLES ENABLE TO INITIATE THE COMMAND
MOV A, VALUE
INC A                       ;INCREMENTS VALUE'S VALUE TO DISLAY THE NEXT ENTRY IN TABLE
MOV VALUE, A
DJNZ LENGTH2, OVERLOOP      ;LOOPS TILL IT DECREMENTS THE VALUE IN LENGTH1 VARIABLE TO
0
LCALL ONESECOND             ;CALLS ONESECOND DELAY
MOV VALUE, #00H             ;RESETS THE VALUE IN VARIABLE VALUE
MOV CURSOREND, #0C3H        ;RESETS THE VALUE IN CURSOREND VARIABLE

RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT AIDS IN PAUSING THE PROGRAM ===== ;;
;; ===== START ===== ;;
PAUSE:
JNB P2.0, PAUSE                  ;PAUSES THE GAME HERE TILL THE PUSHBUTTON 1 IS PUSHED
```

```
RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT AIDS IN PAUSING THE PROGRAM ===== ;;
;; ===== START ===== ;;
PAUSEOVER:
JNB P2.1, PAUSEOVER        ;PAUSES THE GAME HERE TILL THE PUSHBUTTON 2 IS PUSHED
MOV GAMEDONE, #01H         ;WRITES 1 TO GAMEDOWN VARIABLE
RET
;; ===== END ===== ;;


;; ===== SUBROUTINE THAT POINTS CURSOR ON LCD ===== ;;
;; ===== START ===== ;;
SETCURSOR:                 ;SUBROUTINE THAT SETS THE CURSOR POSITION
CLR P2.5                   ;SET RS LOW
CLR P2.6                   ;SET RW LOW
MOV P0, A                  ;MOVES THE CURSOR TO THE BEGININNG OF LCD
LCALL TIMERSUBE            ;CYCLES ENABLE TO INITIATE THE COMMAND

RET
;; ===== END ===== ;;

;; ================================ ;;
;; ===== TIMER INITIALIZATIONS ===== ;;
;; ================================ ;;

;; ===== FRAME RATE DELAY FOR OBSTACLE ===== ;;
;; ===== START ===== ;;
OBSDELAY:                  ;TAG THAT HELPS WITH THE MOVEMENT OF OBSTACLE
LCALL TIMERSUB             ;CALLS TIMERSUB SUBROUTINE
```

```
CLR TF0                    ;CLEAR TIMER FLAG

SETB TR0                   ;RESTART THE TIMER AFTER PROGRAM PAUSE

DJNZ COUNTDOWNOBS,OBSDELAY ;DECREMENTS THE VALUE AT COUNTDOWN BY 1 AND JUMPS TO NEXT
LINE ONLY IF VLAUE IS 0

MOV COUNTDOWNOBS, #0FH     ;RESETS COUNTDOWNOBS VALUE BEFORE RETURNING


RET

;; ===== END ===== ;;



;; ===== ONE SECOND DELAY ===== ;;

;; ===== START ===== ;;

ONESECOND:                 ;TAG THAT HELPS WITH 1 SECOND DELAY

LCALL TIMERSUB             ;CALLS TIMERSUB SUBROUTINE

CLR TF0                    ;CLEAR TIMER FLAG

SETB TR0                   ;RESTART THE TIMER AFTER PROGRAM PAUSE

DJNZ COUNTDOWN,ONESECOND   ;DECREMENTS THE VALUE AT COUNTDOWN BY 1 AND JUMPS TO NEXT
LINE ONLY IF VLAUE IS 0

MOV COUNTDOWN, #14H        ;RESET THE COUNTDOWN VALUE BEFORE RETURNING


RET

;; ===== END ===== ;;



;; ===== 50MS DELAY VALUE ===== ;;

;; ===== START ===== ;;

TIMERSUB:                  ;TAG THAT HELPS WITH 1 SECOND DELAY

MOV TH0, #3CH              ;MOVE TIMER VALUE INTO THE HIGH BYTE

MOV TL0, #0AFH             ;MOVE TIMER VALUE INTO THE LOW BYTE

SETB TR0                   ;SET TR0 HIGH (STARTS TIMER)

LCALL TIMERDELAY           ;LCALL TIMERDELAY SUBROUTINE (HELPS CREATE THE DELAY)


RET

;; ===== END ===== ;;
```

```
;; ===== 40MS DELAY VALUE ===== ;;
;; ===== START ===== ;;
TIMERSUB40M:              ;TAG THAT HELPS WITH 40 MILISECOND DELAY
MOV TH0, #63H             ;MOVE COUNTER VALUE INTO THE HIGH BYTE
MOV TL0, #0BFH            ;MOVE COUNTER VALUE INTO THE LOW BYTE
SETB TR0                  ;SET TR0 HIGH (STARTS TIMER)
LCALL TIMERDELAY          ;LCALL TIMERDELAY SUBROUTINE (HELPS CREATE THE DELAY)
CLR TF0

RET
;; ===== END ===== ;;


;; ===== 5MS DELAY VALUE ===== ;;
;; ===== START ===== ;;
TIMERSUB5M:               ;TIMERSUB SUBROUTINE THAT INITIALIZES THE VALUE FOR TIMER
MOV TH0, #0ECH            ;MOVE COUNTER VALUE INTO THE HIGH BYTE
MOV TL0, #77H             ;MOVE COUNTER VALUE INTO THE LOW BYTE
SETB TR0                  ;SET TR0 HIGH (STARTS TIMER)
LCALL TIMERDELAY          ;LCALL TIMERDELAY SUBROUTINE (HELPS CREATE THE DELAY)
CLR TF0

RET
;; ===== END ===== ;;


;; ===== 40US DELAY VALUE ===== ;;
;; ===== START ===== ;;
TIMERSUB40U:              ;TAG THAT HELPS WITH 40 MICROSECOND DELAY
MOV TH0, #0FFH            ;MOVE TIMER VALUE INTO THE HIGH BYTE
MOV TL0, #0D7H            ;MOVE TIMER VALUE INTO THE LOW BYTE
SETB TR0                  ;SET TR0 HIGH (STARTS TIMER)
```

```
LCALL TIMERDELAY          ;LCALL TIMERDELAY SUBROUTINE (HELPS CREATE THE DELAY)
CLR TF0

RET
;; ===== END ===== ;;


;; ===== 2MS DELAY VALUE ===== ;;
;; ===== START ===== ;;
TIMERSUB2M:               ;TAG THAT HELPS WITH 2 MILISECOND DELAY
MOV TH0, #0F8H            ;MOVE TIMER VALUE INTO THE HIGH BYTE
MOV TL0, #2FH             ;MOVE TIMER VALUE INTO THE LOW BYTE
SETB TR0                  ;SET TR0 HIGH (STARTS TIMER)
LCALL TIMERDELAY          ;LCALL TIMERDELAY SUBROUTINE (HELPS CREATE THE DELAY)
CLR TF0

RET
;; ===== END ===== ;;


;; ===== ENABLE DELAY  ===== ;;
;; ===== START ===== ;;
TIMERSUBE:                ;ENABLE TIMER CLOCK CYCLE
SETB P2.7                 ;SETS ENABLE PORT HIGH
NOP                       ;WAITS FOR FEW MICROSECONDS
CLR P2.7                  ;SETS ENABLE PORT LOW

RET
;; ===== END ===== ;;

;; ===== BUZZER DELAY  ===== ;;
;; ===== START ===== ;;
DELAYBUZZER:
```

```
CLR TR0                     ;CLEARS TIMER RUN FLAG FOR TIMER 0
CLR TF0                     ;CLEARS TIMER DONE FLAG FOR TIMER 0
MOV TH0, #0FFH              ;LOADS HIGH BYTE VALUE INTO TIMER 0 REGISTER
MOV TL0, #98H               ;LOADS LOW BYTE VALUE INTO TIMER 0 REGISTER
SETB TR0                    ;STARTS TIMER 0
LOOPDELAY_ONE:
JNB TF0, LOOPDELAY_ONE      ;KEEP CHECKING TIMER FLAG TILL IT OVERFLOWS WITH 1
CLR TR0                     ;CLEARS TIMER RUN FLAG FOR TIMER 0
CLR TF0                     ;CLEARS TIMER DONE FLAG FOR TIMER 0

RET                         ;RETURNS FROM SUBROUTINE
;; ===== END ===== ;;



;; ===== PAUSE TAG FOR DELAYS ===== ;;
;; ===== START ===== ;;
TIMERDELAY:
JNB TF0, TIMERDELAY         ;IF TIMER OVERFLOW FLAG IS NOT HIGH THE TIMER KEEPS RUNNING
UNTIL TF0 IS HIGH

RET                         ;RETURN THE SUBROUTINE TO ITS CALL LOCATION
;; ===== END ===== ;;

;; ===== THIS SHOULD NOT HAPPEN ===== ;;
ERRCASE:                    ; THIS IS AN ERROR CASE THAT SHOULD NEVER HAPPEN
SJMP ERRCASE



;; ============================== ;;
;; ===== TABLE DEFINITIONS ===== ;;
;; ============================== ;;

;; ===== TITLE LETTERS ===== ;;
;; ===== START ===== ;;
```

```
TITLE:
.DB 'T'
.DB 'R'
.DB 'E'
.DB 'X'
.DB ' '
.DB 'G'
.DB 'A'
.DB 'M'
.DB 'E'
;; ===== END ===== ;;

;; ===== GAME OVER LETTERS ===== ;;
;; ===== START ===== ;;
GAMEOVER:
.DB 'G'
.DB 'A'
.DB 'M'
.DB 'E'
.DB ' '
.DB 'O'
.DB 'V'
.DB 'E'
.DB 'R'
;; ===== END ===== ;;

;; ===== JUMP CHARACTER ===== ;;
;; ===== START ===== ;;
UP:
.DB 00H                          ;TREX IN THE UPPER LCD LOCATION
;; ===== END ===== ;;

;; ===== STAND CHARACTER ===== ;;
```

```
;; ===== START ===== ;;
STANDING:
.DB 00H                          ;TREX IN THE LOWER LCD LOCATION
;; ===== END ===== ;;
.END
```

Aswin Balasubramaniam

Analysis and Conclusion:

The main of this project was to use the knowledge gained from this course and apply it to build a system that uses the 8051-microcontroller and the assembly language. The 8051-microcontroller along with the Apex LCD, pushbuttons and a piezo buzzer, was used to build an interactive gaming platform. The T-Rex game is a simple pushbutton obstacle avoidance game, where the user controls the character of the game with a simple pushbutton. The user should be able to jump over obstacles and avoid the character colliding with the obstacle. The first pushbutton in the system helps the user to jump and start the game. The second pushbutton allows the user to reset the game and start over. Assembly language was used to the code the program for this game. Custom characters were created to make the game more interactive. The custom characters that were created are listed above in the report.

The program logic flow can be followed with the help of the pseudocode provided in the report. The assembly program is also well commented in case a third person has to refer on how the program works.

The project required extensive troubleshooting with displaying the characters at a particular DDRAM address, and displaying multiple custom characters at the same time. The team could not deliver all that was promised as given in the initial project description, due to the time spent on troubleshooting. The project does not include displaying random obstacles during varying time periods, and the various sound effects as promised in the initial project description. The code to write a custom character is given in case the user wants to change the obstacle used in the game.

Since the project did not involve recording any data, there were no visible errors produced to affect the quality of the project.

Heath Palmer

Analysis and Conclusion:

The final lab results were positive after extensive troubleshooting. The lab intended the use of knowledge and experience gained from past labs in Embedded Systems and apply it on the final project. We implemented logic ranging from JB/JNB to LCD initialization and set cursor movement. The logic for displaying "T-REX GAME" and "GAME OVER" did not require much troubleshooting as it was like Lab 9. One minor error of an incorrect variable VALUE was corrected. After pulling P2.0 HIGH, the program would jump to a routine JUMP to set the new cursor location on the LCD and display the T-Rex there.

The GROUND Subroutine was called after the intro screen has been displayed. This caused some initial problems when a lookup table was used to display the blocks. The solution was to remove the lookup table and move 0FFH to the LCD while moving the cursor across the bottom row of the LCD.

The T-Rex creation process in Ricky's World was followed to create the T-Rex and Cactus characters. One issue was the characters would display upside-down on the LCD. While implementing the other functionality of the program, this feature was removed for a short time. To address this, the character creation was reviewed and the display logic in OBSTACLE was edited. The obstacle would be written and then cleared, instead of writing a space to the previous position.

The lab results proved positive as the program ran the T-Rex Game according to the Pseudo Code and Project Description. The hardware was assembled per the circuit schematic seen in Figure 1 and did not require major troubleshooting. As the two programs were integrated, the lab was successful in function prior to the deadline.