**Kuki |Microprocessor Systems Project Report**

**Introduction**

The main purpose of this project is to make a game, simulation, or any application with using a virtual LCD screen on ARM Cortex M0 microprocessor in Assembly language. While doing this project we used Microprocessor concepts such as: ARM Instructions, ARM registers, memory structure of ARM, Subroutines, Stacks, Interrupts… that we learned during the course. In this project we did a “Flappy Bird” game on assembly. Flappy Bird is a single player game. The flapping bird rises with each command (UP Button) and tries to break through the barriers. The bird that touches the barriers or the ground dies and the game is over.

**Team Info**

Mevlüt Yıldırım (040190253) => Game map & character drawing
Çağrı Tarakçıoğlu (040190246) => Button interrupt, end game condition
Mertcan Çelik (040190415) => Preparing the project report

Following tasks were done together in collaboration: Determining the game logic & the code structure

**Implementation**

In our project we have 3 main parts:

1- **Main function**

This is where all the code work together. Such as map drawing, character drawing, end game condition, bird moving…

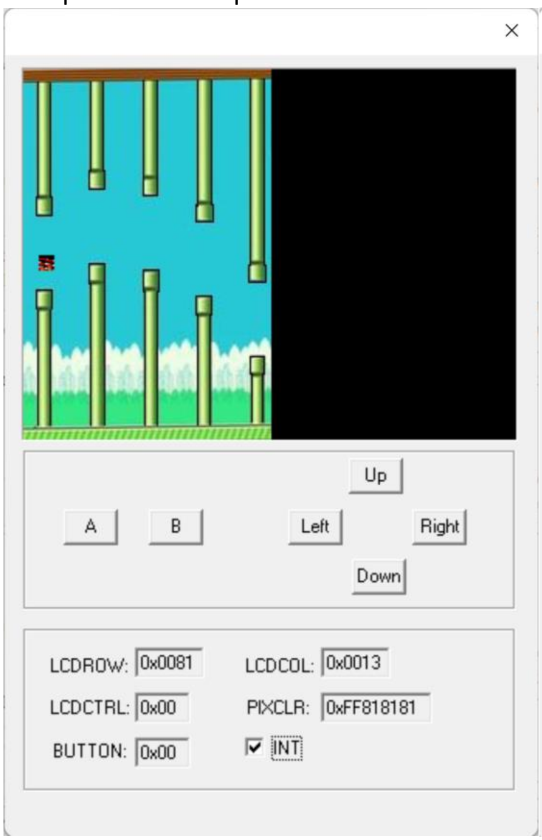
-We initialize our values (like bird’s borders, LCD base address) to registers that we are using everywhere in the code.

```
LDR     R0, =0x40010000     ;LCD base address
MOVS    R7, #120            ;TOP border of bird
MOVS    r5, #130            ;BOTTOM border of bird
MOVS    r4, #10             ;LEFT border of bird
MOVS    r6, #20             ;RIGHT border of bird
```

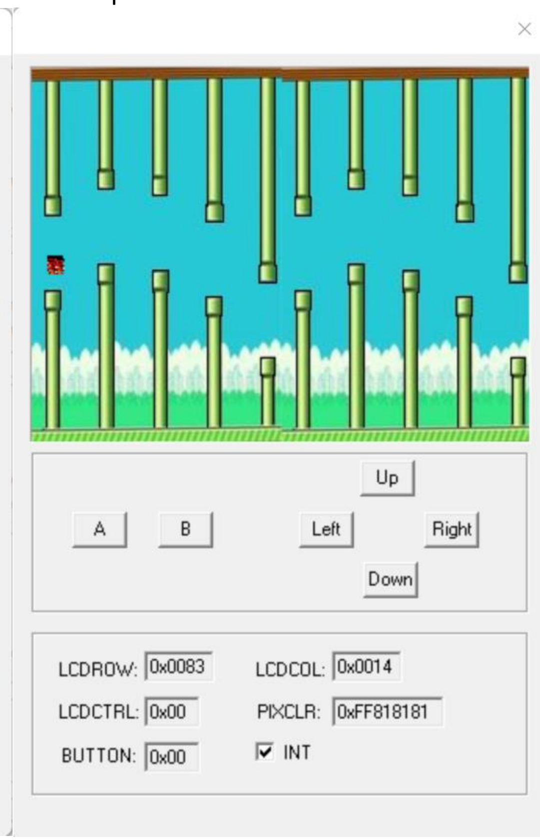
- Our map is an image of the real flappy bird game. The map image size is 160x240px which is half of the LCD. So, we draw it side by side 2 times.

- In the original game the map moves to the left constantly. But we did not to that. Our map does not move.

First part of the map




Full map




- While reading the image.c file which our map we should convert the RGBA order to ARGB.

```
ldr     r1, [r2]            ; Load the image addres to the R1 register
rev     r1, r1              ; Reverse the R1 register thus we get RGBA->ABGR
movs    r3, #8              ; For getting the ARGB order we should rotate our image 8bit
rors    r1, r1, r3          ; Rorate the ABGR-> ARGB thus we get the correct order of the color
STR     r1, [R0, #0x8]      ; Store the color to the color address
adds    r2, r2, #4          ; Move next pixel of the map
```

- In this part we are converting it to correct order for the LCD.

**2- Draw char function**

- This function draws our bird. It takes the border variables of the bird which we initialized in the main part and draws the bird.c image between these borders.
- Our bird.c image's size is 10x10 px.
- After we drew the bird, we are moving our bird 2px down and 1px right by changing its border values.

**3- Game condition**

- We determine the win-lose condition by checking the birds borders with the pillars' position. We could not determine a dynamic logic here. At least we wanted to implement the end game screen. So, we are hard-coded the end-game logic part. We are checking the bird's right border with every pillar left border. If both are same, we are ending the game with a RED screen which means lose, if not we check is the bird reached to the end of the map. If it reaches, we are ending the game with a GREEN screen which means win.

For an example for the hard-coded part. The R6 is the right border & R5 is the bottom border of the bird. The numbers that we are comparing it with are the left border of the pillars.

```
check_wall      CMP     R6, #43
                BEQ     check_col_1
col_1_ok        CMP     R6, #78
                BEQ     check_col_2
col_2_ok        CMP     R6, #112
                BEQ     check_col_3
col_3_ok        CMP     R6, #147
                BEQ     check_col_4
col_4_ok        CMP     R6, #203
                BEQ     check_col_5
col_5_ok        CMP     R6, #238
```

```
check_col_1     CMP     R5, #125
                BHS     gameover
                CMP     R5, #77
                BLS     gameover
                b       col_1_ok
check_col_2     CMP     R5, #129
                BHS     gameover
                CMP     R5, #81
                BLS     gameover
                b       col_2_ok
check_col_3     CMP     R5, #146
                BHS     gameover
```

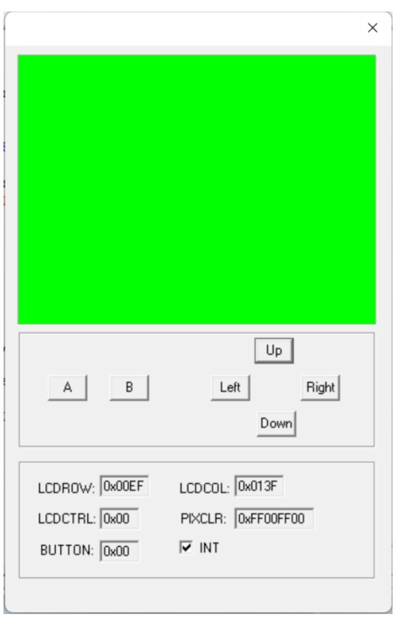
Above the code shows that how we did the game over check.

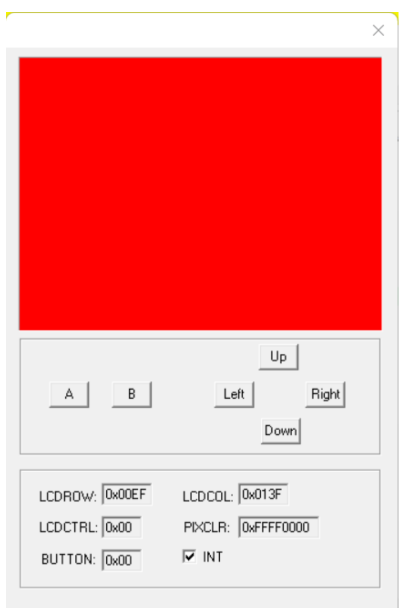
**4- End game screen**

At the end of the game, we are calling our end_game function which paint the lcd according to win/lose.

```
win             LDR R4,=0xFF00FF00
                BL end_game
                B finish_code

gameover        LDR R4,=0xFFFF0000
                BL end_game
                B finish_code

end_game
                LDR R0, =0x40010000

                MOVS R6, #0
                LDR R7, =320
                MOVS R5, #0
end_draw_rows   CMP R5, #240
                BHS end_row_end
                MOVS R6, #0
end_draw_cols   CMP R6, R7
                BHS end_col_end
                STR R5, [R0]
                STR R6, [R0, #0x4]
                STR R4, [R0, #0x8]
                ADDS R6, R6, #1
                B end_draw_cols
end_col_end     ADDS R5, R5, #1
                B end_draw_rows
end_row_end     MOVS R6, #1
                STR R6, [R0, #0xC]
                BX LR
```

Here in the "win" label we are changing the color to green and in the "gameover" label to red.
Then we are calling the function and printing the end game screen.




These are our end game screens.

**Discussion**

-While developing the flappy bird game, our initial challenge was the button interrupt. It took a large part of the project to make a properly working button system.

-The second challenge that we faced is: trying to store data (Borders of the bird, images' base addresses, lcd base address and so on.) with a limited number of registers. We used the stack memory structure for this.

-The last challenge that we faced is trying to paint the background image twice side by side.

In fact, after solving the first problem, we did the rest in a very short time. If we had started the project earlier, we could have made a better-quality endgame mechanic and sliding map.

We think that this project has added a lot of knowledge and experience to all of us about microprocessors. Especially about assembly language and ARM microprocessor.