EE306 - Microprocessors

Term Project
Sliding Text

June 8, 2020

Ali Bahadır Bulut
#041602017

Alp Gokcek
#041701014

Erdal Sidal Dogan
#041702023

1 Project Description

2-3 cümlelik abstract/description (0-1): açıklayıcı, net olmalı. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer at lectus est. Etiam pellentesque, dui eget tempor suscipit, turpis dui pharetra sapien, vel molestie enim felis ac est. Aliquam erat volutpat. Duis eget dictum lacus, nec luctus magna. Sed et cursus erat. Vivamus lobortis sollicitudin fringilla. In tincidunt mi arcu, quis suscipit metus hendrerit vitae. Sed ut libero sit amet orci pretium fermentum sit amet eu augue. Vivamus lobortis ante ut nunc porta posuere. Quisque vel leo a dui ultrices imperdiet.

Our design supports the characters provided in Figure 1.

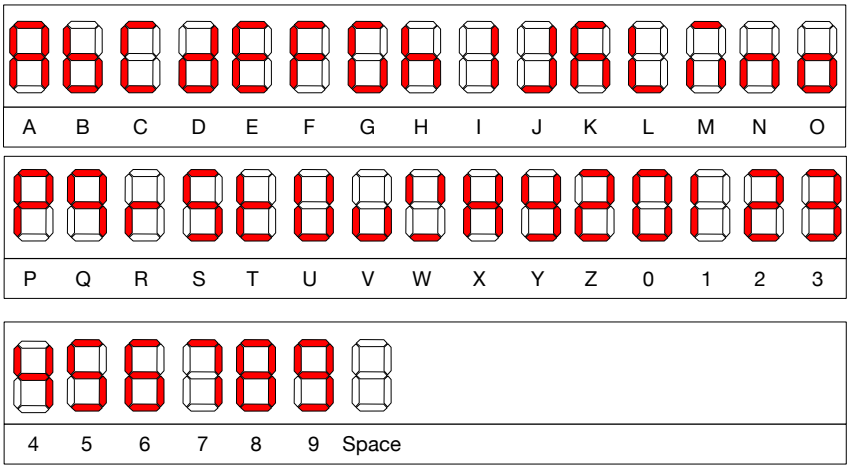

Figure 1: Sample output on 7-segment displays for input "EE306 Fun"

2 Sample Output

Figure 2: Sample output on 7-segment displays for input "EE306 Fun"

Figure 3: Sample output on 7-segment displays for input "1234567890"

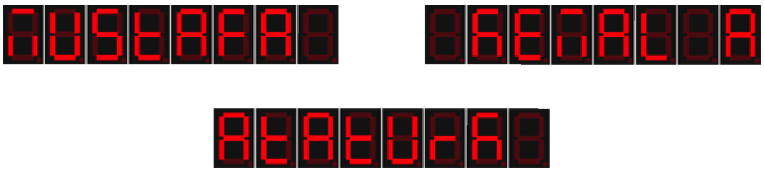

Figure 4: Sample output on 7-segment displays for input "Mustafa Kemal Ataturk"

3 Source Code

Listing 1: Source code of our sliding text project

```
.global _start
_start:
        LDR R9, BUTTON_BASE // push buttons address
    LDR R11, TIMER_BASE //timer adress
    LDR R12, =200000000 // 200M
    STR R12, [R11] //set load register of the timer
    MOV R2, #0b011 //control configuration
    STR R2, [R11, #8] //set config bits
        MOV R7, #0 //current input status flag

    /* AT THIS POINT WE CAN USE R1 & R2 REGISTERS AS WE WISH */
    /* DON'T FORGET R11 & R12 ARE RESERVED FOR TIMER OPERATION */

    LDR R0, DISPLAY_BASE // 7-segment adress
    LDR R1, =CHAR_SET // address of the character set
    LDR R2, =INPUT0 // input register
        MOV R3, #0 // string character iterator index
        MOV R5, #0 // length
        B MAIN

MAIN:
        BL SET_WORD // setting the word according to the switches
        BL CHECK_PAUSE // checking pause state from push button 0
        BL SET_SPEED // setting the speed of words according to push button 1 and 2
        BL PRINT_STRING // updating 7 segments
        B MAIN

CHECK_PAUSE:
        PUSH {R7} // save state of r7
        PAUSE:
                LDR R7, [R9] // load push buttons' state
                ANDS R7, R7, #0b001 // if push button 0 is pressed
                BNE PAUSE // branch until not pressed
                POP {R7} // restore state of r7
                BX LR // return to main loop

SET_SPEED:
        PUSH { R6 } // save state of registers
        LDR R6, [R9] // load buttons state
        CMP R6, #0b100 // check if button 2 is pressed
        BEQ INCREASE_SPEED // if yes branch to increase speed
        CMP R6, #0b010 // check if button 1 is pressed
        BEQ DECREASE_SPEED // if yes branch to decrease speed

        NORMALIZE_SPEED:
                LDR R12, =200000000 // load 200m to normalize speed
                STR R12, [R11] // update load register of the timer
                POP { R6 } //restore the register
                BX LR // return to main loop
```

```
        INCREASE_SPEED:
                LDR R12, =50000000 // load 50m to increase the speed
                STR R12, [R11] // update load register of the timer
                POP { R6 } // restore the register
                BX LR // return to main loop
        DECREASE_SPEED:
                LDR R12, =800000000 // load 800m to decrease the speed
                STR R12, [R11] // update load register of the timer
                POP { R6 } // restore the register
                BX LR // return to main loop

SET_WORD:
        PUSH {R0} // save register state
        set_word_loop:
                LDR R0, SWITCH_BASE // load base address of switches
                LDR R0, [R0] // load switches' status
                CMP R0, #0 // check if any of the switches are on
                BEQ set_word_loop // if not continue until one of them is pressed
                CMP R0, R7 // if not changed
                POP {R0} // restore r0
                BXEQ LR // return to main loop
                PUSH {R0} // save r0 again
                LDR R0, SWITCH_BASE // load base address of switches
                LDR R0, [R0] // load switches' status
                CMP R0, #1 // check if switch 0 is on
                LDREQ R2, =INPUT0 // update input
                MOVEQ R7, #1 // update flag
                CMP R0, #2 // check if switch 1 is on
                LDREQ R2, =INPUT1 // update input
                MOVEQ R7, #2 // update flag
                CMP R0, #4 // check if switch 2 is on
                LDREQ R2, =INPUT2 // update input
                MOVEQ R7, #4 // update flag
                CMP R0, #8 // check if switch 3 is on
                LDREQ R2, =INPUT3 // update input
                MOVEQ R7, #8 // update flag
                MOV R3, #0 // reset string character iterator index
                MOV R5, #0 // reset length
                POP {R0} // restore state of register
                BX LR


PRINT_STRING:
        PUSH {LR} // save state of link register
        PRINT_STR:
                LDRB R4, [R2, R3] // load ascii code of the character
                CMP R4, #0x00 // reached to the end of the string
                MOVEQ R5, R3 // length
                MOVEQ R3, #0 // start from the beginning of the string
                BEQ PRINT_STR
                CMP R4, #32 // 32 is ascii code of the space char
                SUBEQ R4, R4, #32 // 0th location of array contains space char
                PUSH {LR}
                BLNE FIND_LOC
```

```
                /* ascii(A) = 65 and goes sequentially, when we subtract
                64 from the ascii code, we get the index of the char
                representation in the array */
                LDRB R4, [R1, R4] // retrieve the character representation
                STRB R4, [R0] // write on to the 7-segment
                BL UPDATE_LOCATION
                POP {LR} // restore link register
                BX LR // return to main

//this subroutine finds the location of the input char on CHAR_SET
FIND_LOC:
        CMP R4, #65 // check if it is upper case "A"
        SUBLT R4, R4, #47 // for numbers
        BXLT LR // return to PRINT_STR
        CMP R4, #97 // check if it is lower case "a"
        SUBLT R4, R4, #54 // for upper case letters
        BXLT LR // return to PRINT_STR
        SUB R4, R4, #86 // for lower case letters
        BX LR // return to PRINT_STR

UPDATE_LOCATION:
        SUB R0, R0, #1 /* R0 is the leftmost 7-segment.
        at each iteration, print the character to the left of the
        previous char */
        LDR R10, =0xFF20002F // non-entry zone for display addresses.
        CMP R0, R10 // check if in non-entry zone
        LDREQ R0, =0xFF200023 // if yes reset
        LDR R10, =0xFF20001F // non-entry zone for display addresses.
        CMP R0, R10 // out of the available 7-segment adress. (too right)
        ADDNE R3, R3, #1 /* increase the string char index to read the next char */
        BXNE LR // return to PRINT_STR
        LDR R0, =0xFF200033 // address of the leftmost of the 7-segments
        SUB R3, R3, #6
        CMP R3, #0
        ADDLT R3, R3, R5
        PUSH {LR} // save link register
        BL DELAY // do delay
        POP {LR} // restore link register
        BX LR // return to PRINT_STR

DELAY:
    LDR R8, [R11, #0xC] // load timer state
    CMP R8, #1 // check the interrupt
    BNE DELAY // branch until interrupt
    STR R8, [R11, #0xC] // reset timer state
    BX LR // return to UPDATE_LOCATION

END: B END

DISPLAY_BASE: .word 0xff200033 //display base address
TIMER_BASE: .word 0xfffec600 //timer base address
BUTTON_BASE: .word 0xff200050 //push buttons base address
SWITCH_BASE: .word 0xff200040 //switches base address
//.asciz appends a zero at the end of the string as an finish indicator
```

```
INPUT0: .asciz "abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVXYZ "
INPUT1: .asciz "1234567890 "
INPUT2: .asciz "Mustafa Kemal Ataturk "
INPUT3: .asciz "MeF RoCkS EE306 MiCrOpRoCesSorS "
CHAR_SET: .byte 0x00, 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77,
     0x7C, 0x39, 0x5E, 0x79, 0x71, 0x7D, 0x74, 0x06, 0x0E, 0x75, 0x38, 0x15, 0x54,
    0x5C, 0x73, 0x67, 0x50, 0x6D, 0x78, 0x3E, 0x1C, 0x2A, 0x76, 0x6E, 0x5B
// 1 0 = 0x3F 11 A = 0x77 21 K = 0x75 31 U = 0x3E
// 2 1 = 0x06 12 B = 0x7C 22 L = 0x38 32 W = 0x1C
// 3 2 = 0x5B 13 C = 0x39 23 M = 0x15 33 V = 0x2A
// 4 3 = 0x4F 14 D = 0x5E 24 N = 0x54 34 X = 0x76
// 5 4 = 0x66 15 E = 0x79 25 O = 0x5C 35 Y = 0x6E
// 6 5 = 0x6D 16 F = 0x71 26 P = 0x73 36 Z = 0x5B
// 7 6 = 0x7D 17 G = 0x7D 27 Q = 0x67
// 8 7 = 0x07 18 H = 0x74 28 R = 0x50
// 9 8 = 0x7F 19 I = 0x06 29 S = 0x6D
// 10 9 = 0x6F 20 J = 0x0E 30 T = 0x78

.end
```

4 Flowcharts

4.1 MAIN

After the initialization, the program starts executing the *Main* subroutine. It consists of chaining other subroutines sequentially such as *SET_WORD*, *CHECK_PAUSE*, *SET_SPEED*, *PRINT_STRING*.

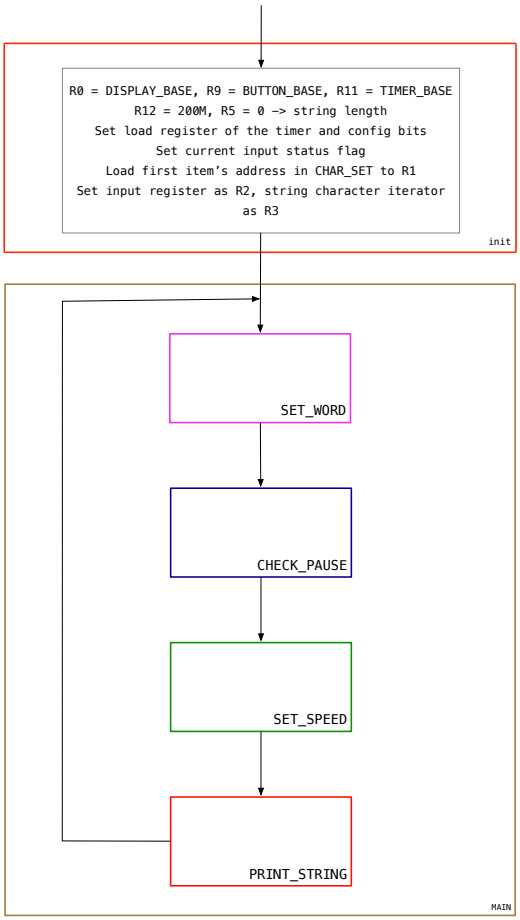

Figure 5: Flowchart of Main Loop

4.2 SET_WORD

The program supports multiple words. Selection is being made by the 'Switch' input. The subroutine *SET_WORD* is responsible for choosing the correct word according to the input.

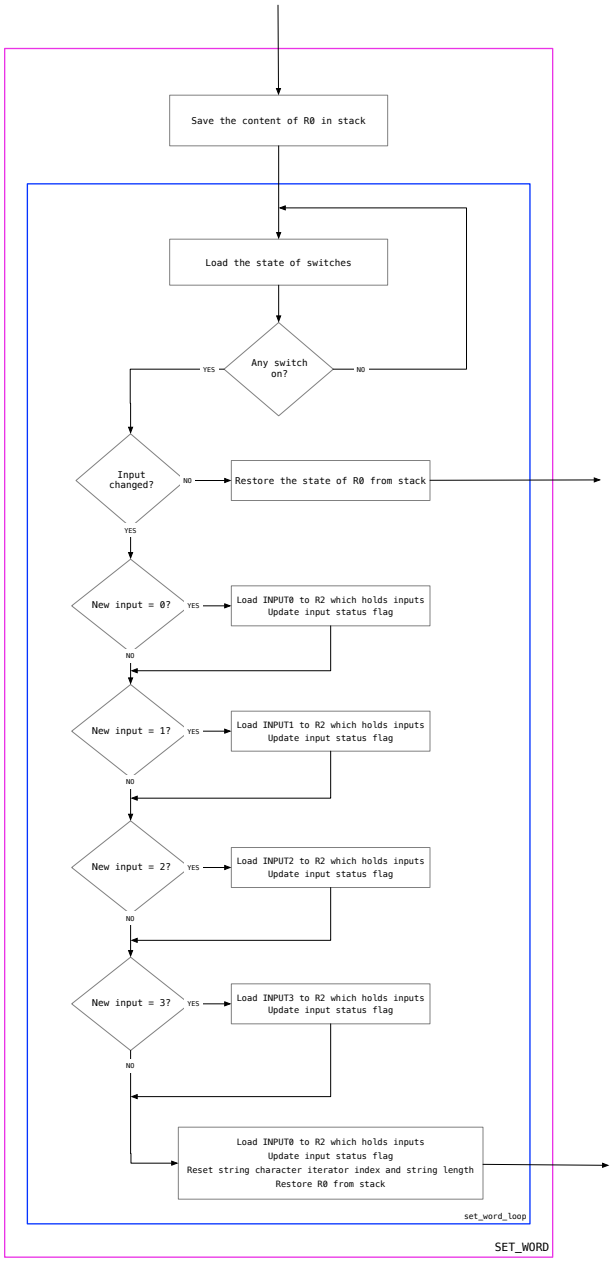

Figure 6: Flowchart of Set Word Subroutine

4.3 SET_SPEED & CHECK_PAUSE

As the names suggests, *SET_SPEED* and *CHECK_PAUSE* subroutines are pausing the slider and sets its speed. Both of these functions are triggered by certain combinations of push buttons. They check the latest push button configuration pause or increase/decrease the speed accordingly. There are 3 different predefined levels of speed;

- Normal: 1 sec
- Fast: .25 sec
- Slow: 4 sec

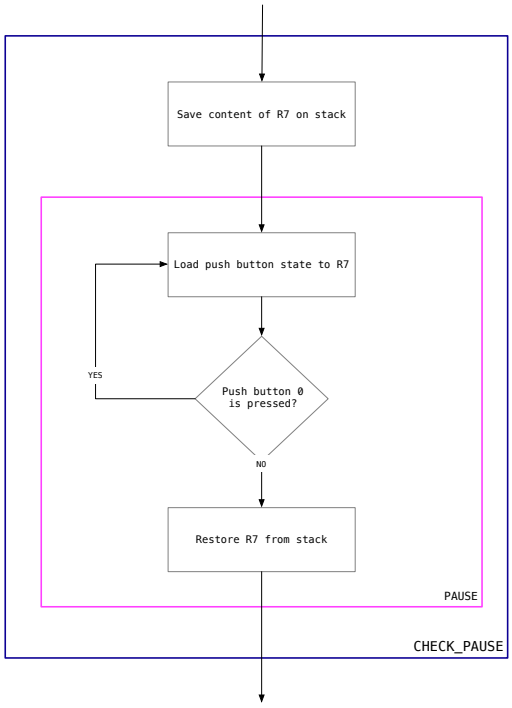

Figure 7: Flowchart of Check Pause Subroutine

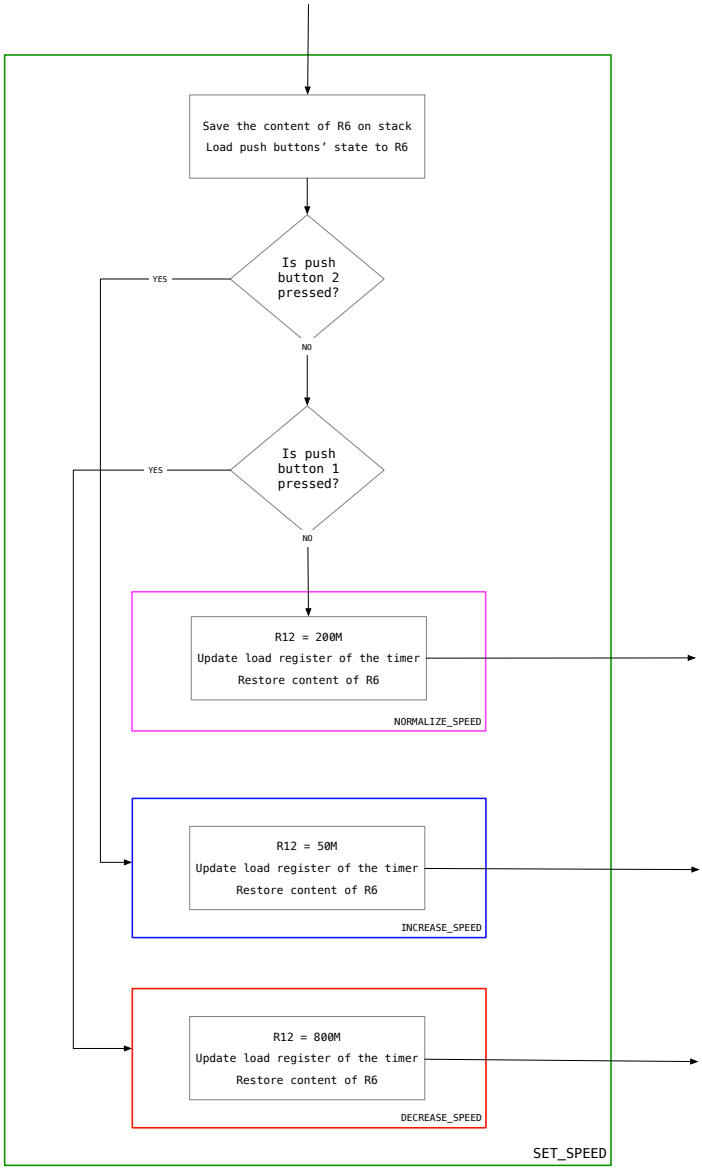

Figure 8: Flowchart of Set Speed Subroutine

4.4 PRINT_STRING & DELAY & FIND_LOC

4.4.1 DELAY

The A9 Private Timer is being used for delays. This subroutine checks the timer configuration continuously until it finishes its countdown, after it finished this subroutine resets it and start over again when called next time.

4.4.2 PRINT_STRING

This subroutine determines which letter to print relying on the input and which 7segment to print it.

4.4.3 FIND_LOC

The representation of possible inputs, which are capital and lowercase letters along with numbers, are stored sequentially in the memory. This subroutine is used for locating the correct 7-segment representation in the memory.

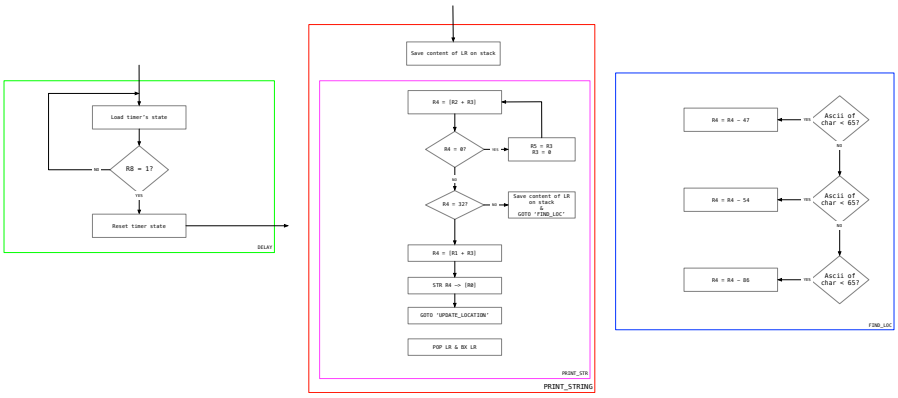

Figure 9: Flowchart of Print String Method

4.4.4 UPDATE_LOCATION

This subroutine determines which 7-segment to write next character. Given that there are jumps between memory locations of 7-segments and circular loop, it has been decided that its best to handle these operations in a different subroutine. First we set it to be the 7-segment that is right next to the previous one. Then we check if has exceeded any limits, if it did, reset the position accordingly.

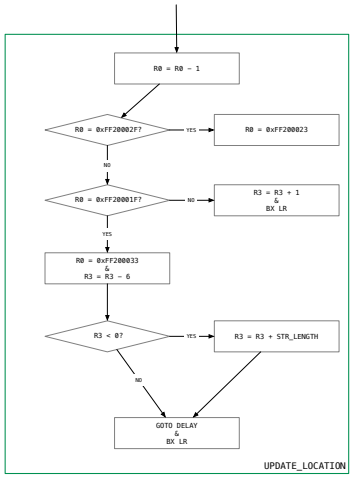

Figure 10: Birds