EE306 - Microprocessors

Laboratory Exercise 4
I/O

March 17, 2020

Erdal Sidal Dogan
#041702023

1 Task1

Listing 1: Assembly code of Task-I

```
.global _start
_start:
    LDR R0, =0xFF200020 // 7-segment address
    LDR R1, =0xFF200050 // push button address
    LDR R2, =SEQUENCE // sequential digits
    MOV R3, #0 // array index

MAIN:
    LDR R8, [R1] // push button configuration
    LDRB R5, [R2, R3]
    STR R5, [R0]
    CMP R8, #1 // check if push button config is 1
    BLEQ ZERO // 1 corresponds showing single 0
    CMP R8, #2 // check if push button config is 2
    BLEQ INCREMENT // 2 corresponds incrementing the digits continuously
    CMP R8, #4 // check if push button config is 4
    BLEQ DECREMENT // 4 corresponds decrementing the digits continuously
    CMP R8, #8 // check if push button config is 8
    BLEQ BLNK // 1 corresponds showing blank
    BL DELAY // apply delay
    B MAIN // loop

INCREMENT: // subroutine for incrementing the digits on the display continuously
    ADD R3, R3, #4 /* increment by 4 since our memory is word-addressable and digits
        are stored as array */
    CMP R3, #0x28 /* 0x24 is the index of the array element where the hex correspondence
    of 9 in 7-segment is being held. check if the number has passed 9, if yes; equalize
    it 0 zero to make circular loop. */
    MOVEQ R3, #0
    BX LR

DECREMENT: // subroutine for decrementing the digits on the display continuously
    CMP R3, #0x00 /* 0x00 is the first index of the array element where the hex
    correspondence of 0 in 7-segment is being held. check if the number has decremented
    below 0, if yes; equalize it to last index -where the hex correspondence of 9 in
    7-segment is being held- to make circular loop. */
    MOVEQ R3, #0x28
    SUB R3, R3, #4 /* decrement by 4 since our memory is word-addressable and digits
        are stored as array */
    BX LR

ZERO: // subroutine for displaying 0 on the display
    MOV R3, #0
    BX LR

BLNK: // delete everything from 7-segment leds.
    MOV R12, #0x00
    STR R12, [R0]
    BX LR
```

```
DELAY:
    LDR R7, =400000 // delay counter
    SUB_LOOP:
        SUBS R7, R7, #1 // perform 400000 subtraction for delay
        BNE SUB_LOOP
    BX LR

end: B end
// 7 segment correspondences of digits from 0-9 stored in order
SEQUENCE: .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0xFF, 0xEF
.end
// 0 = 0x3F, Array Index: 0
// 1 = 0x06, Array Index: 4
// 2 = 0x5B, Array Index: 8
// 3 = 0x4F, Array Index: C
// 4 = 0x66, Array Index: 10
// 5 = 0x6D, Array Index: 14
// 6 = 0x7D, Array Index: 18
// 7 = 0x07, Array Index: 1C
// 8 = 0xFF, Array Index: 20
// 9 = 0xEF, Array Index: 24
```

2 Task2

Listing 2: Assembly code of Task-II

```
.global _start
_start:
       LDR R0, =0xFF200020 // firt chunk of 7-segment address
       LDR R1, =0xFF200030 // second chunk of 7-segment address
       LDR R2, =GROUPID // characters beginning address
       LDR R3, [R2] // load characters from mem to reg
       STR R3, [R0] // store to mem location where the 7-segment reads
       LDR R3, [R2, #4] /* our memory is word-addressable, increment the adress by
       4 to get the remaining parts of the GROUPID */
       STR R3, [R1] // store the remaining parts of the GROUPID text
       B END

END: B END
GROUPID: .byte 0x66, 0x73, 0x1C, 0x5C, 0x50, 0x7D
.end
// G = 0x7D
// r = 0x50
// o = 0x5C
// u = 0x1C
// p = 0x73
// 4 = 0x66
```

3 Task3

Listing 3: Assembly code of Task-III

```
.global _start
_start:
    LDR R0, =0xFF200020 // 7-segment address
    LDR R1, =0xFF200030 // 7-segment address
    LDR R4, =0xFF200050 // pushbutton address

LOOP:
    LDR R2, =SET // set text to "set"
    BL DISPLAY_WORD // call subroutine for updating the display
    LDR R2, =ALARM // set text to "alarm"
    BL DISPLAY_WORD
    LDR R2, =SET // set text to "set"
    BL DISPLAY_WORD
    LDR R2, =TIME // set text to "time"
    BL DISPLAY_WORD
    B LOOP // loop

DISPLAY_WORD:
    LDR R10, [R4]
    CMP R10, #0 // check if any keys are pressed
    BEQ DISPLAY_WORD // if no keys are pressed, check again.
    LDR R3, [R2] // load the values from mem to reg
    STR R3, [R0] // store to mem location where the 7-segment reads
    LDR R3, [R2, #4] /* our memory is word-addressable, increment the adress by 4
    to get the remaining parts of the text */
    STR R3, [R1] // store the remaining parts of the text
    PUSH {LR} /* save the content of the link register before branchin to 'delay'
    subroutine */
    B DELAY // perform delay

DELAY: // perform delay
    LDR R12, =1200000 // delay counter
       SUB_LOOP:
        SUBS R12, R12, #1
        BNE SUB_LOOP
    POP {PC} // go back where the 'display_word' subroutine was first called


END: B END
SET: .byte 0x78, 0x79, 0x6D, 0x00, 0x00, 0x00, 0x00, 0x00
//extra zeros are for misalignment issues, notice they complement 8 bytes.
ALARM: .byte 0x37, 0x37, 0x50, 0x77, 0x38, 0x77, 0x00, 0x00
TIME: .byte 0x79, 0x37, 0x37, 0x06, 0x78, 0x00
.end
```