ELEC 334 – MICROPROCESSORS

PROJECT 1 – RANDOMIZED COUNTER

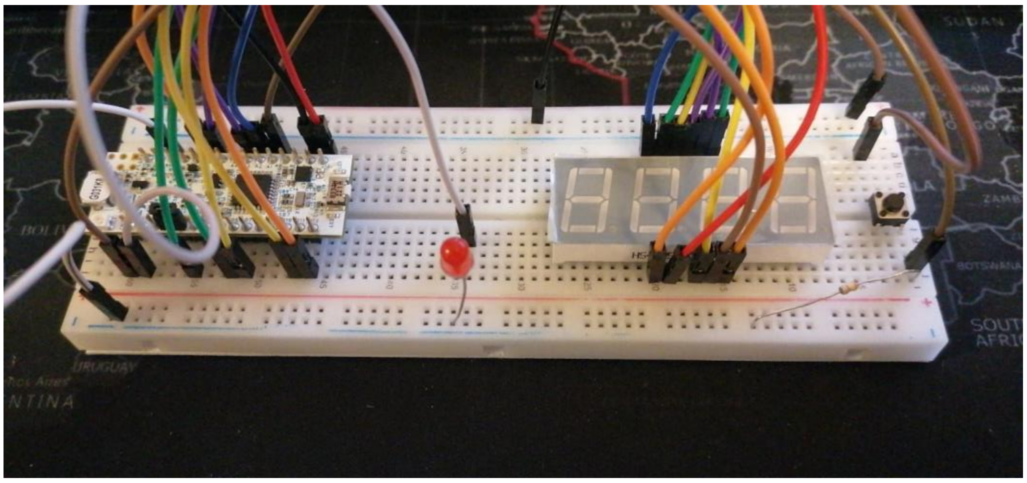
AZİZ CAN AKKAYA 171024005

Contents

1 Intorduction

1.1 Briefing

Document Version	: 1.1
Document Publised Date	: 24/11/2020
Project Version	: v4.7_D



1.2 Objective

This project is developed on a STM32G031K8T NUCLEO BOARD. The Arm Cortex-M0+ architecture gained functionality with "asm.s" file developed with assembly language. The aim of the Project is to understand and gain a brief knowledge about ARM Cortex-M0+, how the pins work and put a general knowledge how instructions work.

2 Technical Aspects

2.1 Materials

Material	Price
Breadboard x1	8,84 TL
Jumper x40	3.72 TL
390R Resistor x1	0,03 TL
4x Seven Segment Display x1	7,45 TL
STM32G031K8 x1	137,59 TL
5mm LED Red x1	0,19 TL
Tactile Switch x1	0,28 TL

2.2 Method

The system supports 3 different button press type, which we will cover further into the section. Program will display 0000 on the 4 digit 7 segment display for a second. After that program will be on a loop. In this loop board will display 4005 on 4 digit 7 segment display and check for button interrupt. If the button is pressed. It will start the countdown from 9000 for the first countdown. In every step of countdown the program will generate a random number on the r7 register. After the countdown the random number will be displayed and will be saved into the responsible registers. Lastly if the pressed again in the countdown the program will paused until a further interrupt.

- Display 0000 on the segment display for a second.
- Display 4005 on the segment display until the button pressed.
- If the button pressed start the countdown from 9000 and display every number on the segment display.
- If the button pressed stop the countdown and display the number that stopped. After that wait for an another interrupt fort o contiune.
- After the countdown display the random number for 1 second.
- Display 0000 for a second and wait for an interrupt again and this the countdown starts from the random number generated by the board.

2.2.1 Random Number Algorithm

We used the Pseudo Code Random Number Generator algorithm which depends on several elements. The formula we will use:

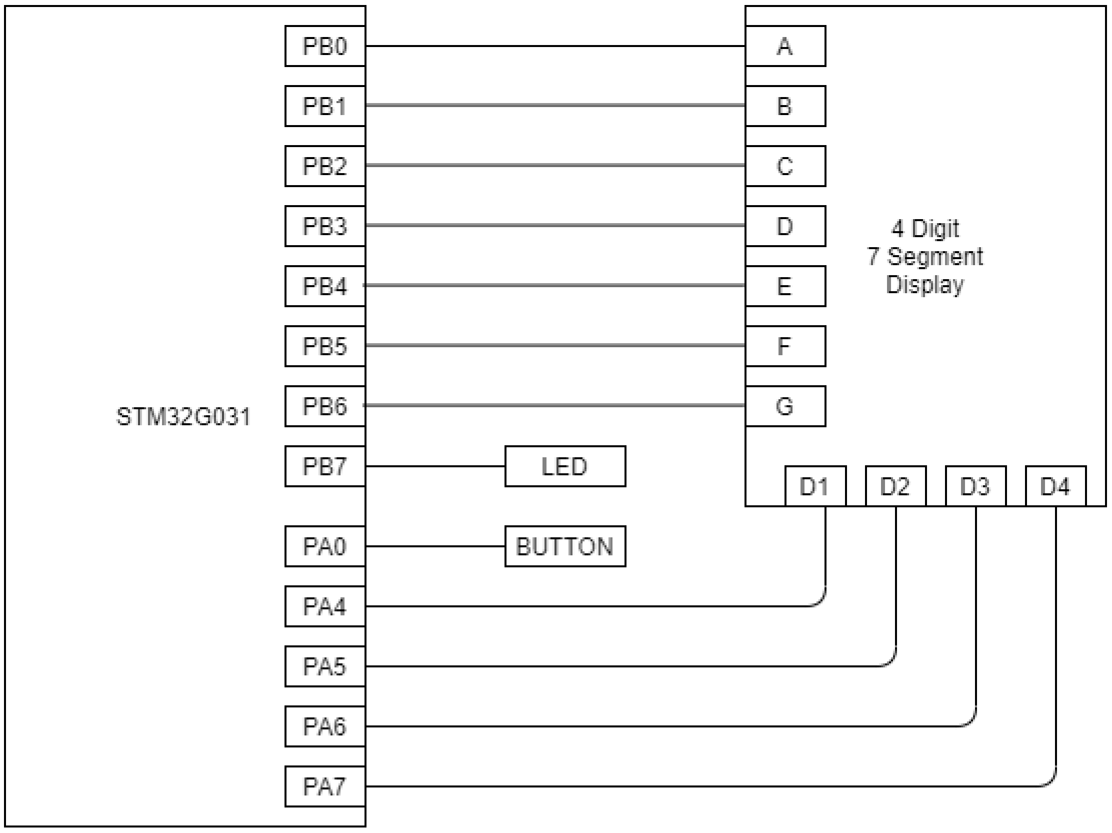
$$next = (next * a + c) \bmod m$$

a = 11035, c = 123456, m = 9000.

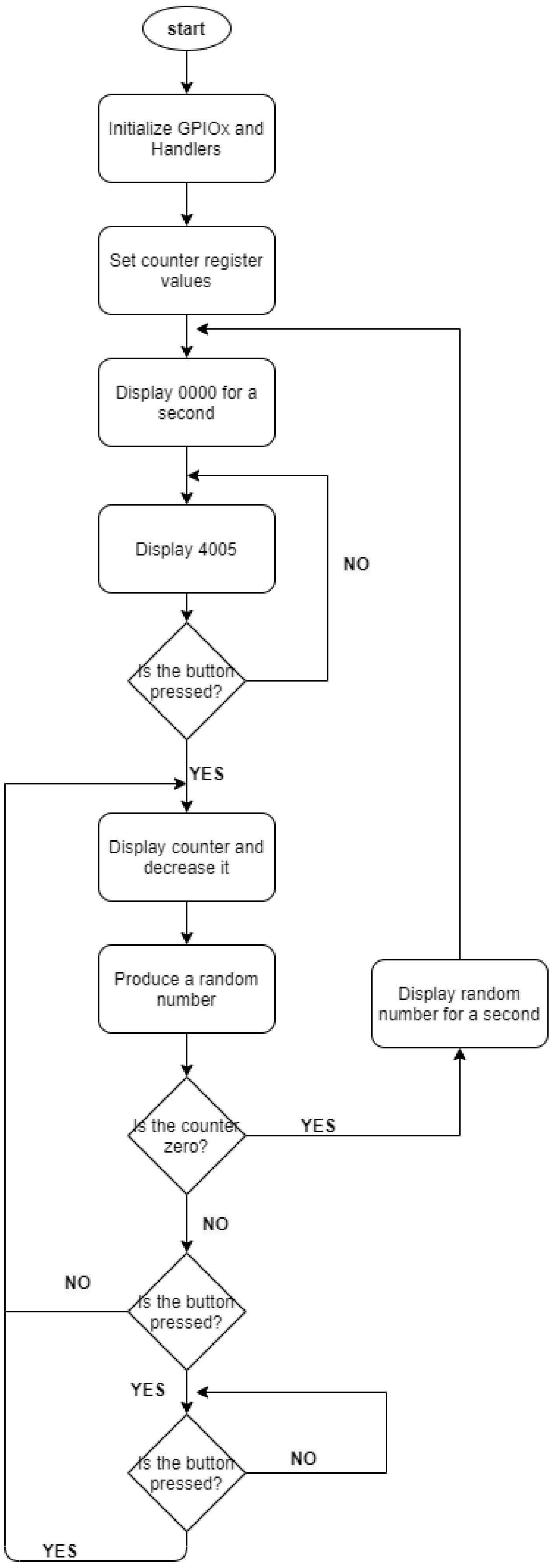
This formula will produce a number from 0 to 8999. After the countdown we add 1000 to number to maket this a number between 1000 and 9999.

2.3 Diagrams

2.3.1 Hardware Block Diagram



2.3.2 Software Block Diagram


start
Initialize GPIOx and Handlers
Set counter register values
Display 0000 for a second
Display 4005
NO
Is the button pressed?
YES
Display counter and decrease it
Produce a random number
Display random number for a second
Is the counter zero?
YES
NO
NO
Is the button pressed?
YES
Is the button pressed?
NO
YES

3 Output

https://youtu.be/DPQuOiU-mz8

4 Conclusion

This project showed us a crucial points such as code length, data handling, register managment, bitwise module handling and problem solving.

Lets break it down and pieces shall we? First of all it was a great way to understand our C code that we programed on previous assignments. We had a chance to inspect how our code will work on machine wise and how it can be handled in different ways.

On the register handling part we learned how to preserve and manage our data for our usage. Since it is so easy to lose data that could be used in further instructions, it is important to store them in a good manner.

For code length we found clever methods fill the stack completely by shortening some parts but sacrificing readability. It can backfire for different readers but it efficient to shorten them.

Last setback that we faced is bitwise module activation. Putting right addresses and values into the registers played an important role on this process.

5 Appendix

```
/*
 * project1.s
 *
 * author: Aziz Can Akkaya
 * number: 171024005
 */

.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOC_BASE,       (0x50000800)          // GPIOC base address
.equ GPIOC_MODER,      (GPIOC_BASE + (0x00)) // GPIOC MODER register offset
.equ GPIOC_ODR,        (GPIOC_BASE + (0x14)) // GPIOC ODR register offset

.equ GPIOA_BASE,     (0x50000000)              //GPIOA base address
.equ GPIOA_MODER,    (GPIOA_BASE + (0x00))    //GPOIA MODER reigster offset
.equ GPIOA_ODR,            (GPIOA_BASE + (0x14))      //GPIOA ODR register offset
.equ GPIOA_IDR,            (GPIOA_BASE + (0X10))      //GPIOA IDR register offset

.equ GPIOB_BASE,     (0x50000400)              //GPIOB base address
.equ GPIOB_MODER,    (GPIOB_BASE + (0x00))      //GPIOB MODER register offset
.equ GPIOB_ODR,            (GPIOB_BASE + (0x14))      //GPIOB ODR register offset

.equ leddelay,             (1000) //delay value

.equ a,       (11035)                   //a value for random code generator
.equ c,       (12345)                   //c value for random code generator
.equ m,       (9000)             //m value for random come generator

.equ ZERO,    (0x40)             //4-digit SSD pin set for GPIOA, nummber 0
.equ ONE,     (0x79)             //4-digit SSD pin set for GPIOA, nummber 1
.equ TWO,     (0x24)             //4-digit SSD pin set for GPIOA, nummber 2
.equ THREE,   (0x30)             //4-digit SSD pin set for GPIOA, nummber 3
.equ FOUR,    (0x19)             //4-digit SSD pin set for GPIOA, nummber 4
.equ FIVE,    (0x12)             //4-digit SSD pin set for GPIOA, nummber 5
```

```
.equ SIX,      (0x02)              //4-digit SSD pin set for GPIOA, nummber 6
.equ SEVEN,    (0x78)              //4-digit SSD pin set for GPIOA, nummber 7
.equ EIGHT,    (0x00)              //4-digit SSD pin set for GPIOA, nummber 8
.equ NINE,     (0x10)              //4-digit SSD pin set for GPIOA, nummber 9


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
  .word _estack              /*      Stack pointer */
  .word Reset_Handler +1     /*      Reset handler */
  .word Default_Handler +1   /*        NMI handler */
  .word Default_Handler +1   /* HardFault handler */
  /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
  /* set stack pointer */
  ldr r0, =_estack
  mov sp, r0

  /* initialize data and bss
   * not necessary for rom only code
   * */
  bl init_data
  /* call main */
  bl main
  /* trap if returned */
  b .


/* initialize data and bss sections */
.section .text
init_data:

  /* copy rom to ram */
  ldr r0, =_sdata
  ldr r1, =_edata
  ldr r2, =_sidata
  movs r3, #0
  b LoopCopyDataInit

  CopyDataInit:
        ldr r4, [r2, r3]
        str r4, [r0, r3]
        adds r3, r3, #4

  LoopCopyDataInit:
        adds r4, r0, r3
        cmp r4, r1
        bcc CopyDataInit

  /* zero bss */
  ldr r2, =_sbss
  ldr r4, =_ebss
  movs r3, #0
  b LoopFillZerobss
```

```

 FillZerobss:
        str  r3, [r2]
        adds r2, r2, #4

 LoopFillZerobss:
        cmp r2, r4
        bcc FillZerobss

 bx lr


/* default handler */
.section .text
Default_Handler:
 b Default_Handler


/* main function */
.section .text
main:
 /* enable GPIOC clock, bit2 on IOPENR */
 ldr r6, =RCC_IOPENR
 ldr r5, [r6]
 /* movs expects imm8, so this should be fine */
 movs r4, 0x3
 orrs r5, r5, r4
 str r5, [r6]
 //GPIOA PIN SET FOUR OUTPUT
 ldr r6, =GPIOA_MODER
 ldr r5, [r6]

 movs r4, 0xFF
 lsls r4, r4, #8
 bics r5, r5, r4

 movs r4, 0x55
 lsls r4, r4, #8
 orrs r5, r5, r4
 ldr r4, =0xFFFFFFFC
 ands r5, r5, r4
 str r5,[r6]
 //GPIOB PIN SET FOR OUTPUT
 ldr r6, =GPIOB_MODER
 ldr r5, [r6]
 ldr r4, =0x5555
 ands r5, r5, r4
 str r5, [r6]
 //GPIOA PIN SET FOR INPUT
 ldr r6, =GPIOA_IDR
 ldr r5, [r6]
 ldr r4, =0x1
 ands r5, r5, r4
 str r5, [r6]
 //COUNTER REGISTER SET
 movs r0, #9
 movs r1, #0
 movs r2, #0
 movs r3, #0
```

```

//Setting delay
set1:
  ldr r7, =1000000
//diplay 0000 on the ssd for 1 second
start_loop:
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, ZERO
  orrs r5, r5, r4
  str r5, [r6]

  subs r7, r7, #1
  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0xF0
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =0
  cmp r7, r4
  bne start_loop
//clear output values
clear:
  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, =GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]
  //b countdown
//button check
button:
  bl id_mod
  ldr r6, =GPIOA_IDR
     ldr r5, [r6]
     ldr r4, =0x1
     ands r5, r5, r4
  cmp r5, #1
  beq countdown

//id display (4005) if the button is not pressed
id_mod:
  push {lr}
  //D1 OUTPUT
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, FOUR
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x10
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay5:
  subs r4, r4, #1
  bne delay5
  //CLEAR PINS
  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, =GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]
  //D2 and D3 OUTPUT
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, ZERO
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x60
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay7:
  subs r4, r4, #1
  bne delay7
  //CLEAR PINS
  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, =GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]
  //D4 OUTPUT
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, FIVE
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x80
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay8:
  subs r4, r4, #1
  bne delay8

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, =GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]
  pop {pc}

//if the button pressed in start it will lead to this point
countdown:
  bl button_check
  bl  check
  bl led_on
  bl display
  bl random
  //check and decrase control for units digit
  cmp r3, #0
  beq decrease_ten
  subs r3, r3, #1
  b countdown

decrease_ten:
  //check and decrase control for tens digit
  movs r3, #9
  cmp r2, #0
  beq decrease_hundred
  subs r2, r2, #1
  b countdown

decrease_hundred:
  //check and decrase control for hundreds digit
  movs r2, #9
  cmp r1, #0
  beq decrease_thousand
  subs r1, r1, #1
  b countdown

decrease_thousand:
  //check and decrase control for thousands digit
  movs r1, #9
  cmp r0, #0
  beq show1
  subs r0, r0, #1
  b countdown
```

```

//button check for pause
button_check:
  push {lr}
  ldr r6, =GPIOA_IDR
     ldr r5, [r6]
     ldr r4, =0x1
     ands r5, r5, r4
  cmp r5, #1
  beq pause
  pop {pc}

//pause loop, it will wait for an another press
pause:
  bl display
  bl led_on
  ldr r6, =GPIOA_IDR
     ldr r5, [r6]
     ldr r4, =0x1
     ands r5, r5, r4
  cmp r5, #1
  bne pause
  pop {pc}

show1:
  ldr r6, =0
  ldr r4, =1000
  adds r7, r7, r4
  movs r5, r7
  cmp r5, r4
  bgt mod1

show2:
  //setting up thousands value
  movs r0, r6
  ldr r6, =0
  ldr r4, =100
  cmp r5, r4
  bgt mod2

show3:
  //setting up hundreds value
  movs r1, r6
  ldr r6, =0
  ldr r4, =10
  cmp r5, r4
  bgt mod3

show4:
  //setting up tens value
  movs r2, r6
  ldr r6, =0
  ldr r4, =1
  cmp r5, r4
  bgt mod4

show5:
  //setting up units value
  movs r3, r6
  ldr r7, =1000
  b display_result

display_result:
  //displaying result for a second
  bl led_off
  bl display
  subs r7, r7, #1
  ldr r4, =0
  cmp r7, r4
  bne display_result
  b set1

//general modding function for every digits
mod1:
  subs r5, r5, r4
  adds r6, r6, #1
  cmp r5, r4
  bge mod1
  b show2

mod2:
  subs r5, r5, r4
  adds r6, r6, #1
  cmp r5, r4
  bge mod2
  b show3


mod3:
  subs r5, r5, r4
  adds r6, r6, #1
  cmp r5, r4
  bge mod3
  b show4

mod4:
  subs r5, r5, r4
  adds r6, r6, #1
  cmp r5, r4
  bge mod4
  b show5

//random generator
random:
  push {lr}
  ldr r4, =a
  //next*a
  muls r7, r7, r4
  ldr r4, =c
  //next*a + c
  adds r7, r7, r4
  //mod m
  ldr r4 , =m
  cmp r7, r4
  bgt mod
  pop {pc}

mod:
  subs r7, r7, r4
  cmp r7, r4
  bge mod
  pop {pc}

//checking if all the digits is 0 or not for the counter
check:
  push {lr}
  cmp r0, #0
  beq check1
  pop {pc}

check1:
  cmp r1, #0
  beq check2
  pop {pc}

check2:
  cmp r2, #0
  beq check3
  pop {pc}

check3:
  cmp r3, #0
  beq show1
  pop {pc}

//led on function
led_on:
  push {lr}
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x80
  orrs r5, r5, r4
  str r5, [r6]
  pop {pc}

//led off function
led_off:
  push {lr}
  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  orrs r5, r5, r4
  str r5, [r6]
  pop {pc}

//switch case for to set r0 value
d1:
  push {lr}
     cmp r0, #0
     beq zero

     cmp r0, #1
     beq one

  cmp r0, #2
     beq two

  cmp r0, #3
     beq three

  cmp r0, #4
     beq four

  cmp r0, #5
     beq five

  cmp r0, #6
     beq six

  cmp r0, #7
     beq seven

     cmp r0, #8
     beq eight

     cmp r0, #9
     beq nine

//switch case for to set r1 value
d2:
  push {lr}
     cmp r1, #0
     beq zero

     cmp r1, #1
     beq one

  cmp r1, #2
     beq two

  cmp r1, #3
     beq three

  cmp r1, #4
     beq four

  cmp r1, #5
     beq five

  cmp r1, #6
     beq six

  cmp r1, #7
     beq seven

     cmp r1, #8
     beq eight

     cmp r1, #9
     beq nine

//switch case for to set r2 value
d3:
  push {lr}
     cmp r2, #0
     beq zero
```

```

    cmp r2, #1
    beq one

  cmp r2, #2
    beq two

  cmp r2, #3
    beq three

  cmp r2, #4
    beq four

  cmp r2, #5
    beq five

  cmp r2, #6
    beq six

  cmp r2, #7
    beq seven

    cmp r2, #8
    beq eight

    cmp r2, #9
    beq nine

//switch case for to set r4 value
d4:
  push {lr}
    cmp r3, #0
    beq zero

    cmp r3, #1
    beq one

  cmp r3, #2
    beq two

  cmp r3, #3
    beq three

  cmp r3, #4
    beq four

  cmp r3, #5
    beq five

  cmp r3, #6
    beq six

  cmp r3, #7
    beq seven

    cmp r3, #8
    beq eight

    cmp r3, #9
    beq nine

//setting values for GPIOA pin output
zero:
  movs r4, ZERO
    pop {pc}

one:
  movs r4, ONE
    pop {pc}

two:
  movs r4, TWO
    pop {pc}

three:
  movs r4, THREE
    pop {pc}

four:
  movs r4, FOUR
    pop {pc}

five:
  movs r4, FIVE
    pop {pc}

six:
  movs r4, SIX
    pop {pc}

seven:
  movs r4, SEVEN
    pop {pc}

eight:
  movs r4, EIGHT
    pop {pc}

nine:
  movs r4, NINE
    pop {pc}

//digit display function
display:
  push {lr}
  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x10
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  bl d1
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay
```

```

delay1:
  subs r4, r4, #1
  bne delay1

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x20
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  bl d2
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay2:
  subs r4, r4, #1
  bne delay2

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x40
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  bl d3 //movs r4, data
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay3:
  subs r4, r4, #1
  bne delay3

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x80
  orrs r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  bl d4
  orrs r5, r5, r4
  str r5, [r6]

  ldr r4, =leddelay

delay4:
  subs r4, r4, #1
  bne delay4

  ldr r6, = GPIOA_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]

  ldr r6, = GPIOB_ODR
  ldr r5, [r6]
  movs r4, 0x00
  ands r5, r5, r4
  str r5, [r6]
        pop {pc}
```

6 References

- STM32G031K8 Data Sheet
- RM0444 Reference Manual for STM32G0x1 Devices
- ARMv6-M Architecture Reference Manual
- https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/#:~:text=Pseudo%20Random%20Number%20Generator(PRNG)%20refers%20to%20an%20algorithm%20that,state%20using%20a%20seed%20state.
- https://www.projehocam.com/arduino-ile-4-digit-7-segment-display-uygulamasi/
- https://www.keil.com/support/man/docs/armasm/armasm_dom1361289850039.htm