GEBZE TECNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 334 - Project #1

Randomized Counter

Project #1 EXPERIMENT REPORT

Preparer:
1) 171024027 – Mehmet Akif GÜMÜŞ

1. INRODUCTION:

Our objective for this project is to implement a randomized counter in Assembly. A 4-digit SSD should be connected that will display your ID (last 4 digits) when your code is not counting (idle state). When an external button is pressed, it will generate a 4-digit random number, and start counting down to 0. The generated random number should be between 1000 - 9999. When the counter reaches 0, the number 0000 should be displayed for a second, then the code should go back to idle state waiting for the next button press. Pressing the button while counting down should pause counting, and pressing again should resume counting.

2. Technical requirements:

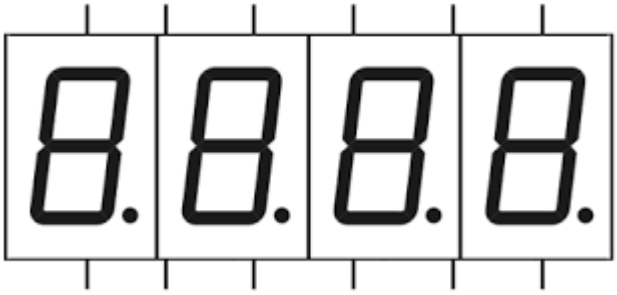
- Written in Assembly.
- An external button should be attached that will generate a random number when pressed
    - Generated random number should be between 1000 - 9999.
- An external LED that will be turned off when the counting is in progress, and will be turned on as soon as the number reaches to 0000
- A 4-digit seven segment display that will show the 4-digit number
    - the first digit should display the seconds, the last three digits should display the milliseconds, so it will count down from 9 at most.
- When the system is idle, your school ID will be shown (last 4-digits)
    - Number 0000 should be displayed at least 1 second before going to ID mode
- Pressing the button should pause the counting and pressing again should resume counting.

3. THEORETICAL RESEARCH

- ### Seven-segment display

A **seven-segment display** is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.



Şekil 1. example of 4-digit 7 segment display

- **STM32**

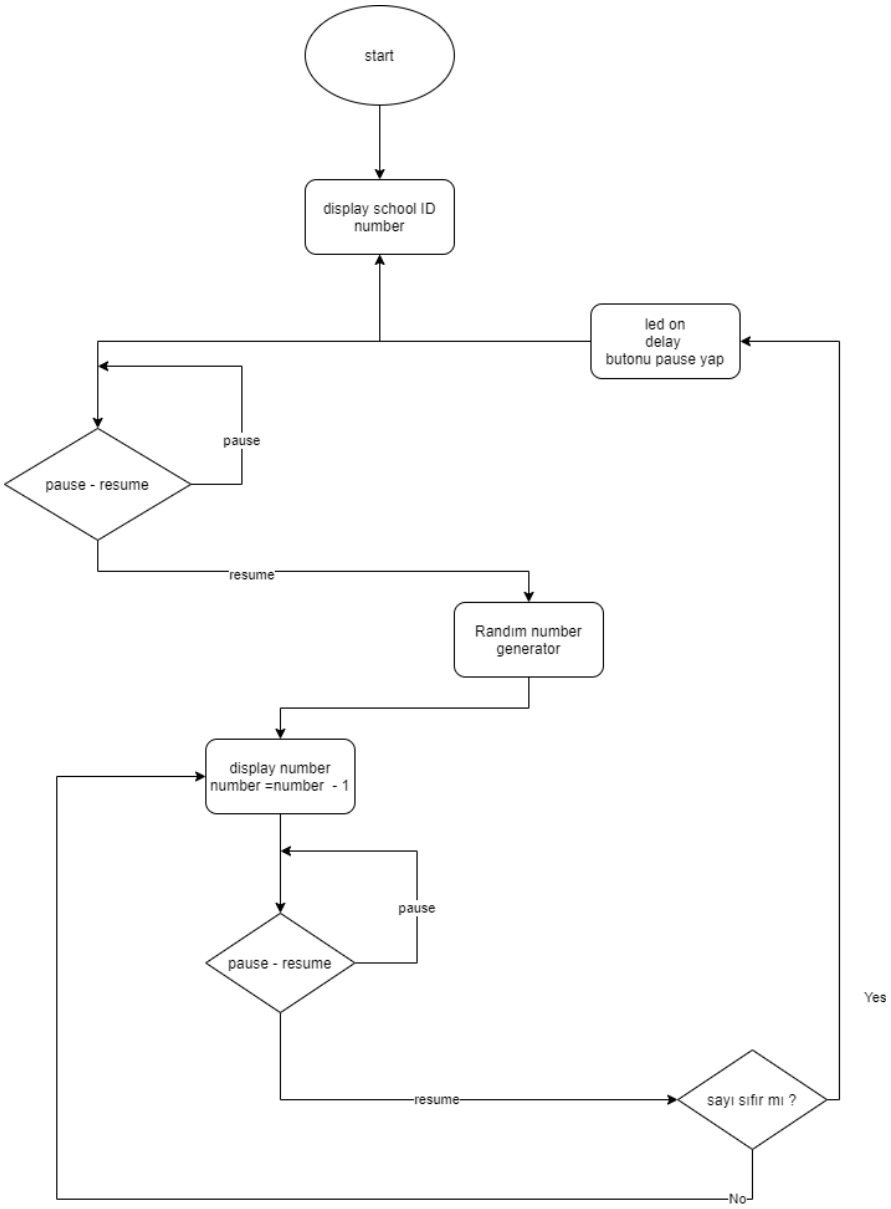
STM32 is a family of 32-bit microcontroller integrated circuits by STMicroelectronics. The STM32 chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M33F, Cortex-M7F, Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM, flash memory, debugging interface, and various peripherals.

Şekil 2

4. TASK

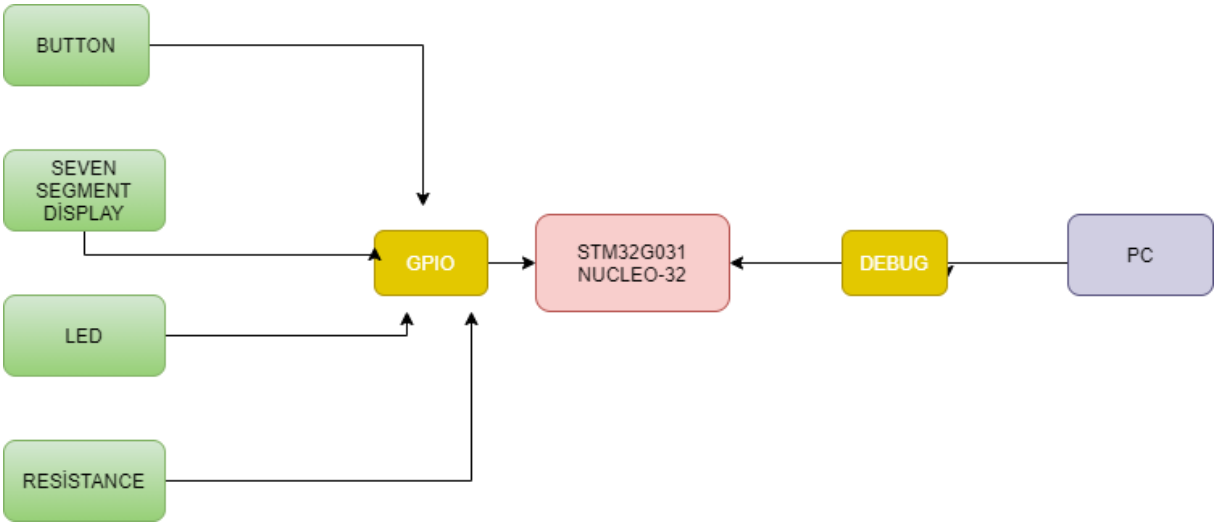
- nucleo-32 card schematıcs examıned
- The seven segment display block diagram was examined
- Created flowchart
- Created blockdiagram
- Necessary connections are made on the breadboard
- Button, controller and seven segment display connection has been made
- code compiled

5. FLOWCHART

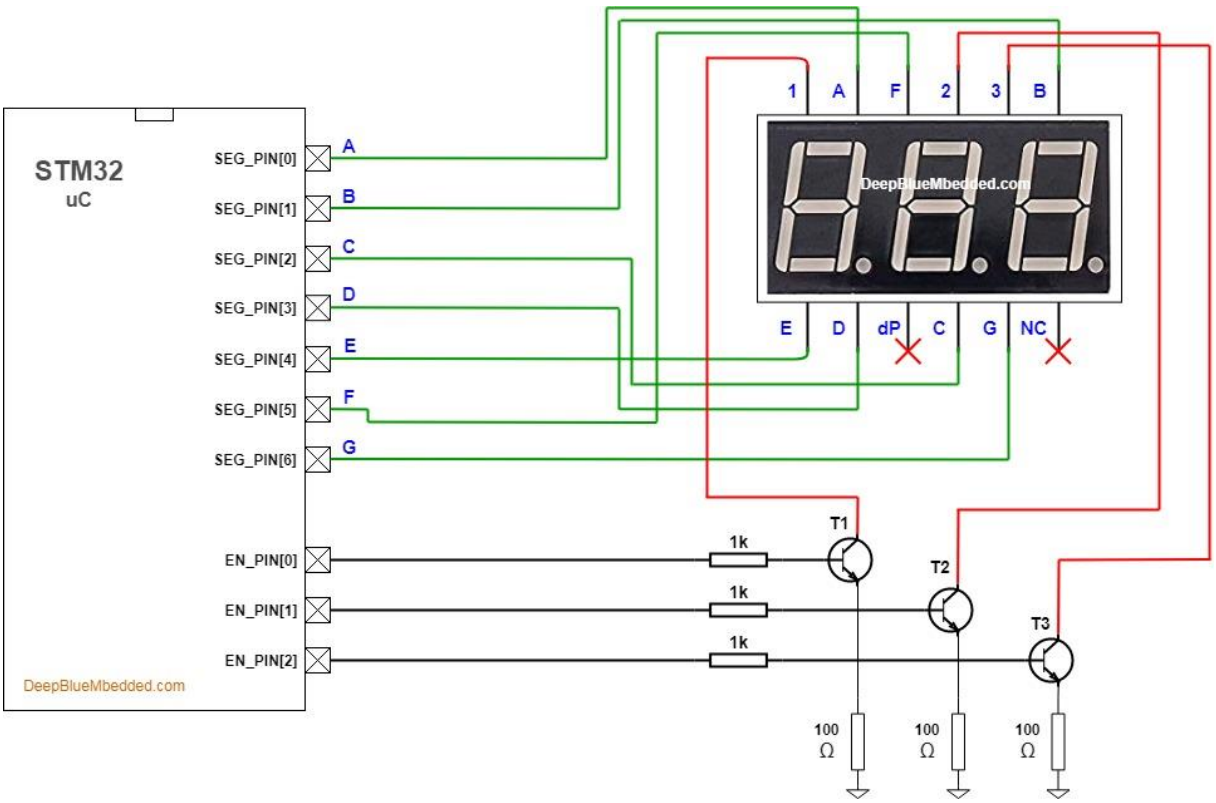


Şekil 3

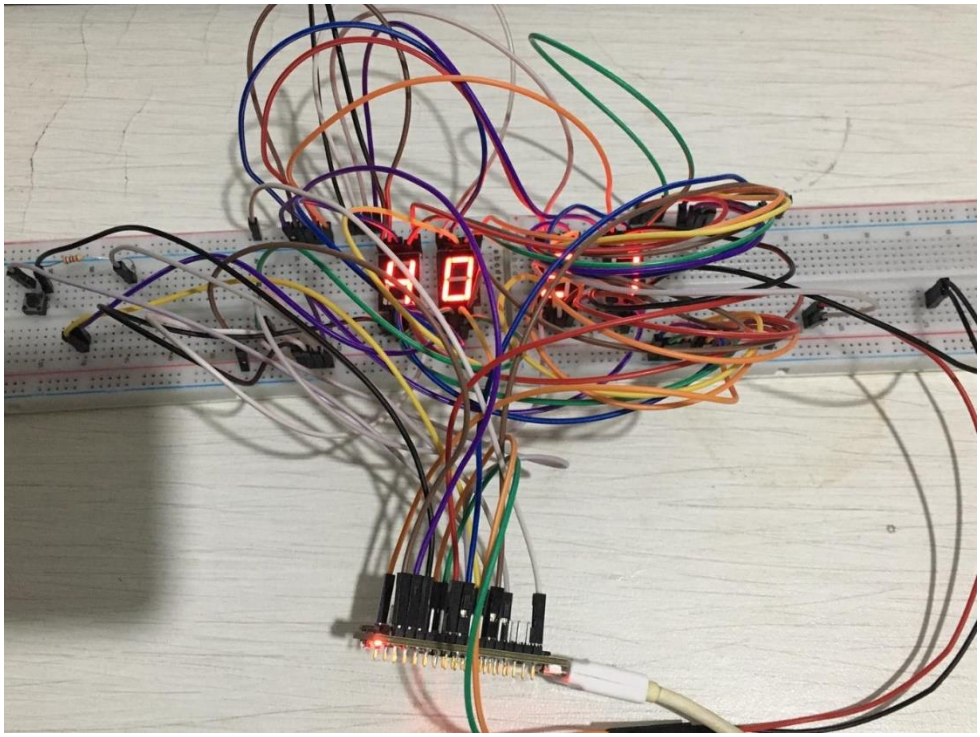
6. BLOCK DIAGRAM

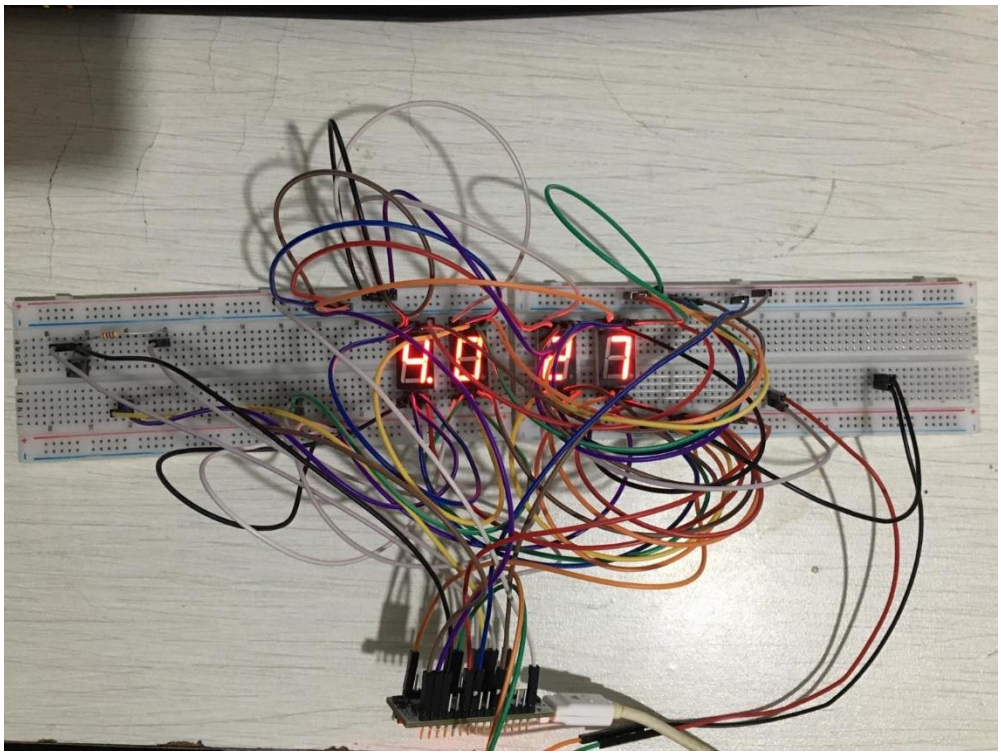


Şekil 4



7. PICTURES:





8. CODE:

```
/*
 * project1.s
 *
 * author: Mehmet Akif GÜMÜŞ
 *
 * description:
           My objective for this project is to implement a
randomized counter in Assembly. A 4-digit SSD
should be connected that will display your ID (last 4 digits) when
your code is not counting (idle
state). When an external button is pressed, it will generate a 4-
digit random number, and start
counting down to 0. The generated random number should be between
1000 - 9999. When the
counter reaches 0, the number 0000 should be displayed for a second,
then the code should go
back to idle state waiting for the next button press. Pressing the
button while counting down
should pause counting, and pressing again should resume counting.
 */

.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 GPIOA_BASE,       (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register
offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR input
offset

.equ GPIOB_BASE,       (0x50000400)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register
offset
.equ GPIOB_IDR,        (GPIOB_BASE + (0x10)) // GPIOB IDR input
offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR output
```

```
offset



/* 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:

      /* sent clock for enable GPIOC, bit2 on IOPENR */
      ldr r6, =RCC_IOPENR
      ldr r5, [r6]
      movs r4, 0x3
      orrs r5, r5, r4
      str r5, [r6]

      /* the MODER's bits (19-0) setup */
      /* PB0-PB7 SSD pins are enable for "01" as output*/
      /* PB9 pushbutton pin is enable for "00" as input*/
      /* PB8 LED pin is enable for "01"  as outpu*/

      ldr r6, =GPIOB_MODER
      ldr r5, [r6]
    ldr r4, = #0x000FFFFF
    bics r5, r5, r4             // it clears bits from r4 which is
'1' to '0' zero on r5
    ldr r4, = #0xFFF15555 // 1111 1111 1111 0001 0101 0101 0101 0101
    orrs r5, r5, r4
    str r5, [r6]

      /* the MODER's bits (15-8) setup*/
      /* seperate digit PB8-PB5 SSD print control pins are enable
for "01" as output*/

      ldr r6, =GPIOA_MODER
      ldr r5, [r6]
    ldr r4, = #0x0000FF00 // 0000 0000 0000 0000 1111 1111 0000 0000
    bics r5, r5, r4             // it clears bits from r4 which is
```

```
'1' to '0' zero on r5
    ldr r4, = #0xEBFF55FF // 1110 1011 1111 1111 0101 0101 1111 1111
    orrs r5, r5, r4
    str r5, [r6]

// my ID is 171024027
//(last 4 digits) 4027

ID_print:

      /* print  first index */
      ldr r6, =GPIOA_ODR
    ldr r4, =#0b10000
      str r4, [r6]
      movs r7, #7                                   //7
      bl print_units

      ldr r1, =#10000
      bl delay
      /*////////////////***********////////////*/
      /* print  second index */
      ldr r6, =GPIOA_ODR
    movs r4,  #0b100000
      str r4, [r6]                                  //2
      movs r7, #2
      bl print_units

      ldr r1, =#10000
      bl delay
      /*////////////////***********////////////*/
      /* print  third index */
      ldr r6, =GPIOA_ODR
    ldr r4, =#0b1000000
      str r4, [r6]                                  //0
      movs r7, #0
      bl print_units

      ldr r1, =#10000
      bl delay
      /*////////////////***********////////////*/
      /* print  fourth index */
      ldr r6, =GPIOA_ODR
    ldr r4, =#0b10000000
      str r4, [r6]
      movs r7, #4                                   //4
      bl print_units

      ldr r1, =#10000
      bl delay
      /*////////////////***********////////////*/

      movs r3, #0 // r3 holds working status and und
```

```
      b bcontrol_1

print_units:                    //comparing the number which is on
binary with its decimal type

      ldr r6, = GPIOB_ODR

      cmp r7, #9
      beq digit9

      cmp r7, #8
      beq digit8

      cmp r7, #7
      beq digit7

      cmp r7, #6
      beq digit6

      cmp r7, #5
      beq digit5

      cmp r7, #4
      beq digit4

      cmp r7, #3
      beq digit3

      cmp r7, #2
      beq digit2

      cmp r7, #1
      beq digit1

      cmp r7, #0
      beq digit0

digit0:
      movs r4, #0b1000000
      str r4, [r6]
      bx lr

digit1:
      movs r4, #0b1111001
      str r4, [r6]
      bx lr

digit2:
      movs r4, #0b0100100
      str r4, [r6]
      bx lr
```

```
digit3:
      movs r4, #0b0110000
      str r4, [r6]
      bx lr

digit4:
      movs r4, #0b0011001
      str r4, [r6]
      bx lr

digit5:
      movs r4, #0b0010010
      str r4, [r6]
      bx lr

digit6:
      movs r4, #0b0000010
      str r4, [r6]
      bx lr

digit7:
      movs r4, #0b1111000
      str r4, [r6]
      bx lr

digit8:
      movs r4, #0b0
      str r4, [r6]
      bx lr

digit9:
      movs r4, #0b0010000
      str r4, [r6]
      bx lr


bcontrol_1:

      /* read button connected with PB9 addressed in IDR*/
      ldr r6, = GPIOB_IDR
      ldr r5, [r6]
      lsrs r5, r5, #9
      movs r4, #0x1
      ands r5, r5, r4


      ldr r1, =#10
      bl delay



      /*control button value*/
      cmp r5, #0x1
      beq bpress_1
```

```
      cmp r3, #0x1
      beq random_number_generate
      bne ID_print

bpress_1:

      cmp r3, #0x1
      beq resume_p
      bne pause_r

resume_p://resume to pause: pause the number while counting down
      movs r3, #0
      b bcontrol_1

pause_r://pause to pause: continues to count from where it left off
      movs r3, #1

movs r2, #0
random_number_generate:   //normally it has to be a random  number
generator but i could not find out how i can do that
      adds r2, r2, #1         //i put a data to countdown.

      ldr r2, =#2879


counter:

      push {r3}
      subs r2, r2, #1

      ldr r0, =#10

print_counter:

      movs r3, #0
      movs r4, r2
      movs r6, #10

unit0:
      cmp r6, r4
      bcs unit0_end
      subs r4, r6
      adds r3, #1
      b unit0
unit0_end:
      muls r3, r3, r6
      subs r4, r2, r3
      movs r7, r4

      ldr r6, =GPIOA_ODR
    ldr r4, =#0b10000
      str r4, [r6]
      bl print_units //1
```

```
      ldr r1, =#1000
      bl delay


      movs r3, #0
      movs r4, r2
      movs r5, #10
      movs r6, #100
unit1:
      cmp r6, r4
      bcs unit1_end
      subs r4, r6
      adds r3, #1
      b unit1
unit1_end:

      muls r3, r3, r6
      subs r4, r2, r3//
      movs r7, r4
      movs r3, #0
unit1_x:
      cmp r5, r7
      bcs unit1_x_end
      subs r7, r5
      adds r3, #1
      b unit1_x
unit1_x_end:
      movs r7, r3

      ldr r6, =GPIOA_ODR
    ldr r4, =#0b100000
      str r4, [r6]
      bl print_units //1

      ldr r1, =#1000
      bl delay


      movs r3, #0
      movs r4, r2
      movs r5, #100
      ldr r6, =#1000
unit2:
      cmp r6, r4
      bcs unit2_end
      subs r4, r6
      adds r3, #1
      b unit2
unit2_end:

      muls r3, r3, r6
      subs r4, r2, r3//
      movs r7, r4
```

```
      movs r3, #0
unit2_x:
      cmp r5, r7
      bcs unit2_x_end
      subs r7, r5
      adds r3, #1
      b unit2_x
      unit2_x_end:
      movs r7, r3

      ldr r6, =GPIOA_ODR
    ldr r4, =#0b1000000
      str r4, [r6]
      bl print_units //1

      ldr r1, =#1000
      bl delay


      movs r3, #0
      movs r4, r2
      ldr r5, =#1000
      ldr r6, =#10000
unit3:
      cmp r6, r4
      bcs unit3_end
      subs r4, r6
      adds r3, #1
      b unit3
unit3_end:

      muls r3, r3, r6
      subs r4, r2, r3//
      movs r7, r4
      movs r3, #0
unit3_x:
      cmp r5, r7
      bcs unit3_x_end
      subs r7, r5
      adds r3, #1
      b unit3_x
unit3_x_end:
      movs r7, r3

      ldr r6, =GPIOA_ODR
    ldr r4, =#0b10000000
      str r4, [r6]
      bl print_units //1

      ldr r1, =#1000
      bl delay

      subs r0, r0, #1
      bne print_counter
```

```
      pop {r3}


bcontrol_2:

      /* read button connected to PB9 addressed at IDR*/
      ldr r6, = GPIOB_IDR
      ldr r5, [r6]
      lsrs r5, r5, #9
      movs r4, #0x1
      ands r5, r5, r4

      ldr r1, =#10000
      bl delay
      /*check button value*/
      cmp r5, #0x1
      beq button_press_2

      cmp r3, #0x1
      beq compare_zero
      bne print_counter

button_press_2:

      cmp r3, #0x1
      beq resume_p_2
      bne pause_r_2

resume_p_2:
      movs r3, #0
      b bcontrol_2

pause_r_2:
      movs r3, #1


compare_zero:

      cmp r2, #0
      beq display_on
      bl counter

display_on:
      movs r3, #0
      bl ID_print

delay:
      subs r1, r1, #1
      bne delay
      bx lr

      b .

      /* this should never get executed */
      nop
```

9. PARTS LIST:

	PRICE LIST	
MATERIAL	PIECE	PRICE
Bread board	1	10 TL
Stm GO31K8	1	105 TL
Resistor	4	10 TL
Seven segment display	4	10 TL
Led	1	0,50 TL
Button	1	0,50 TL
jumper	50	0,25 TL

10.CONCLUSION:

The subject of this project is to write a function that generates random numbers in assembly language and display the generated number on the seven segment display. Before starting to write the code, a flowchart was created and a blockdiagram was drawn so that the flowchart was used while writing the code. While the system is idle, that is, it is not counting down from any random numbers, the last four digits of our school number were displayed on the seven segment display. When the button is pressed, it starts counting down from the number generated by the random number generator function. But here, despite my long research, I could not come to a conclusion. I've tried several ways to generate a random number, one of which is to measure how many milliseconds the button is active while the button is pressed, and generate a random number with the resulting random number. Since it would be a millisecond precision measurement, I would be able to get a random number each time. However, I encountered many problems and could not achieve the result I wanted. When I could not solve the situation, I made a fixed data entry myself to show that the rest of the code was working. The countdown starts from this number. As you can see, when the button is pressed, the number stops counting down and when the countdown is over, the last four digits of my school number appear on the screen. It took a lot of effort while establishing connections on the breadboard because: although I connected the same inputs of the seven segment displays in parallel to each other from a single input, I used a lot of jumpers. This caused a lot of wrong connections. When it comes to the final stage of the project, the last 4 digits of the school number were shown on the screen without creating a random variable, and a random number entered by me was reduced one by one so that each digit reached zero.

11. REFERANCES:

- https://www.instructables.com/Using-a-4-digit-7-segment-display-with-arduino/
- https://en.wikipedia.org/wiki/Seven-segment_display
- https://en.wikipedia.org/wiki/STM32
- https://app.diagrams.net/