GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC334

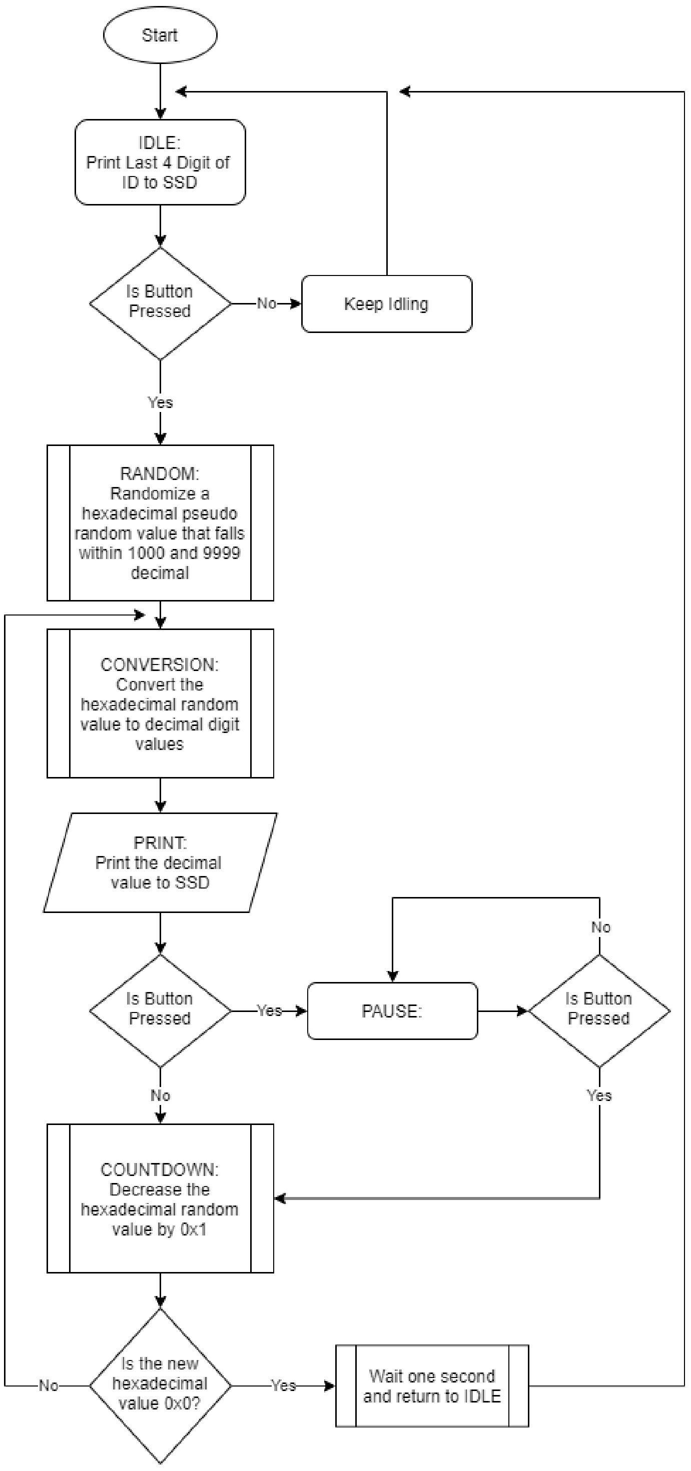
MICROPROCESSORS

Project 1 Report

Prepared by
171024086 Berat KIZILARMUT

1. Introduction

In this project I will be designing a randomized counter that counts down, from a pseudo random value that falls within 1k and 10k, to zero, in increments of one which corresponds to a millisecond. This project utilizes many previous functions we've tasked to design before, such as delay functions, button interrupt functions etc. but also requires brand new functions and parameters to learn, such as SSD displays and hexadecimal decoders.


Start
IDLE: Print Last 4 Digit of ID to SSD
Is Button Pressed
No
Keep Idling
Yes
RANDOM: Randomize a hexadecimal pseudo random value that falls within 1000 and 9999 decimal
CONVERSION: Convert the hexadecimal random value to decimal digit values
PRINT: Print the decimal value to SSD
Is Button Pressed
Yes
PAUSE:
Is Button Pressed
No
Yes
No
COUNTDOWN: Decrease the hexadecimal random value by 0x1
Is the new hexadecimal value 0x0?
No
Yes
Wait one second and return to IDLE

2. Approach

Visualized the simplified requirements on a flowchart, given left. Previous tasks designed to use in previous homeworks and labs will be utilized when they can. For example, I will be using the same pseudo random algorithm I've used on Homework 2.

First of all, I will explain my approaches to required functions in this project.

Idle function, this function will continuously print last 4 digits of my ID, which is 4086. This function will also continuously check for a button press to initialize the random function.

Random function, I will utilize my pseudo random algorithm from HW2 Problem 1 and 2. To utilize my previous function first I must implement my own modulo operator. I will set the random function to random between 1 and 8999, then add 1000 after that.

Conversion function, this function will convert the hexadecimal values to decimal values. The function will decrease the random value by decimal 1000 thousand and increase the decimal digit counter by one. After this it will compare the remaining to 1000. If it's lower than 1000 It will skip to hundreds digit, if not it will loop. Exact same operation will be implemented to hundreds, tens and ones.

Print function, this function will print the decimal values digit by digit to 4 Digit Seven Segment Display. It will utilize a lookup table that correspond to the correct A B C D E F G values for a decimal value. Switch case will used compare the decimal values against switch and choose the correct set of A B C D E F G values.

Countdown function, this function will decrease the random value by one and loop back to conversion to start the process back again.

Pause function, this function will stop the ongoing countdown process with a button press, until another button press.

3. Assembly Code

In this section, I will be explaining my code function by function. Full code will be put on the end of the report to prevent mess.

First things first, the preparation phase where peripherals are clocked, SSD digits and decimal encoding are set. GPIOA and GPIOB are clocked by setting bits 0 and as 1 on IOPENR. After the clocking, port modes will be initialized on GPIOA and GPIOB MODDER's respectively. Setting the port corresponding bits 00 will set the port as input, setting the bits as 01 will set the port as output. I've set the pins PB0 to PB7 and PA4 to PA7 as output. PB0 to PB6 are used as SSD A B C D E F G pins. PB7 is used as external indicator led. PA4 to PA7 is used as SSD D1 D2 D3 and D4. PA0 is set as input to connect the interrupt button. I've avoided using PA2-3 and jumped to PA4 from PA0 because these pins are absent on the board, guessing they're used internally.

After the pins are set correctly, correct digit values are set to correct pins to send the decimal encodings to correct decimal point. These said decimal encodings are set, for example encoding of the number 2 which is "0 1 0 0 1 0 0" G to A on common anode.

In this code I've utilized Branch Link operation when I needed it by linking to a branch, pushing LR at first, and popping PC when the function is completed to return to branching position. I had to reposition functions, change their order or split their subfunctions up in the code because of the imm8 reach range of branch link and branch conditional operations. I've tried to utilize non conditional branching operation as often as possible, especially when imm8 is out of reach, because non conditional branching operation has imm11 reach, which is significantly larger than imm8. Since I've utilized branching operations and had to change order of the functions, functions explained in this section might not be in this exact order on the assembly file itself.

Random function, which utilizes the same algorithm I've used in HW2 Problem 1 and 2 which was "next = (a * next) % b" and the result will be modulus operated with the desired window from 1, which is 8999 in this case. After the random value is created which falls within the window of 1 to 8999, 1000 decimal will be added to make it fall withing the margin of 1000 to 9999. Modulus operation is absent in our microcontroller function, so I had to create my own modulus operator.

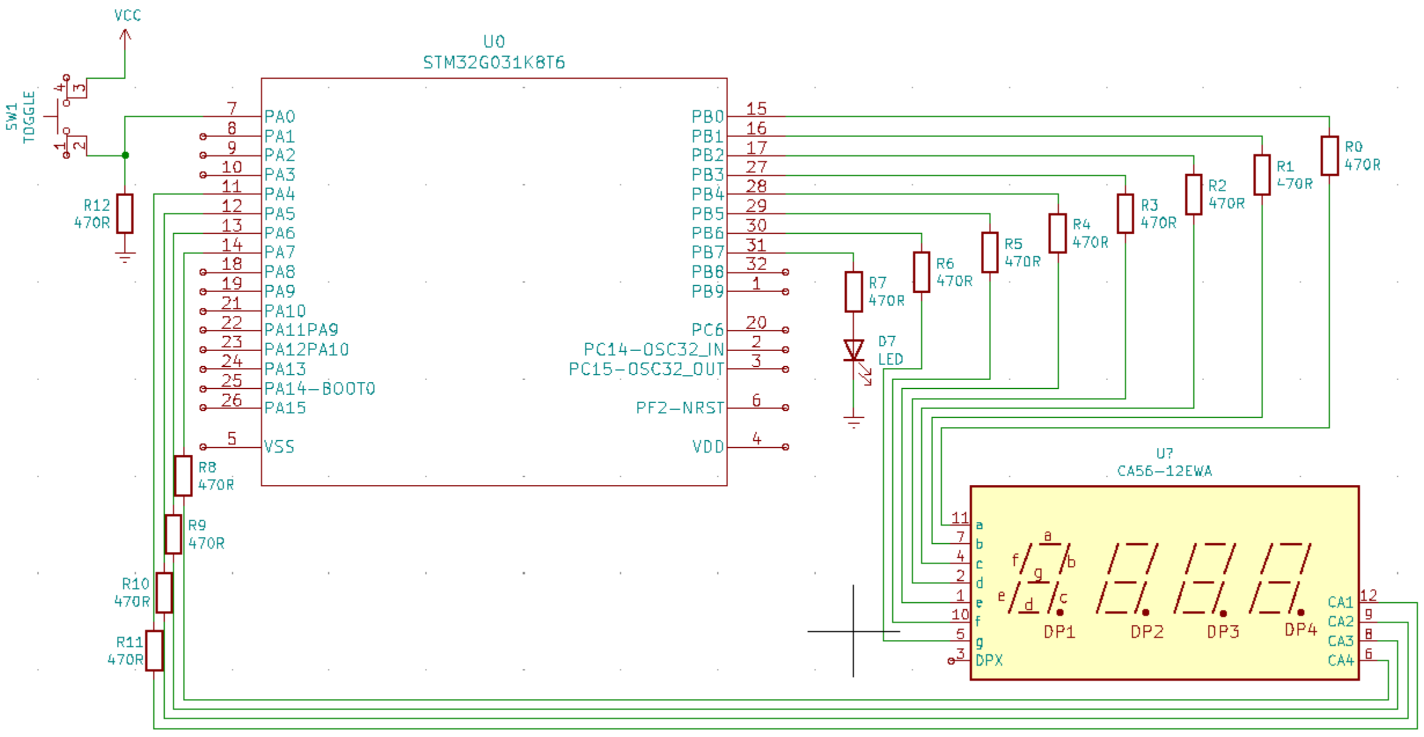
Decimal conversion function takes the value from countdown or random function and decodes it in to it’s decimal digits. This function is very simple, a counter is set for the said for the decimal point, for example a counter is set to zero for thousand digits. Functions subtracts 1000 decimal from the value until the value is lower than decimal 1000. Every time the function subtracts, the digit counter is increased by one. Using this method for thousands, hundreds, tens and ones digits, number is decoded in to decimal value.

Countdown function subtracts the random value which had to be pushed to memory because there were not enough low registers to use. Random value is popped back to use in this function. This function subtracts one from the value, checks if it’s zero. If it is zero, function sends us to idle, if it’s not zero it loops back to decimal conversion.

When the countdown is over or at the start of the whole operation, Idle function prints last 4 digits of my ID to SSD and checks for the button press continuously.

Lastly pause function is implemented to stop the countdown process until another process lets it resume counting down.

4. Hardware

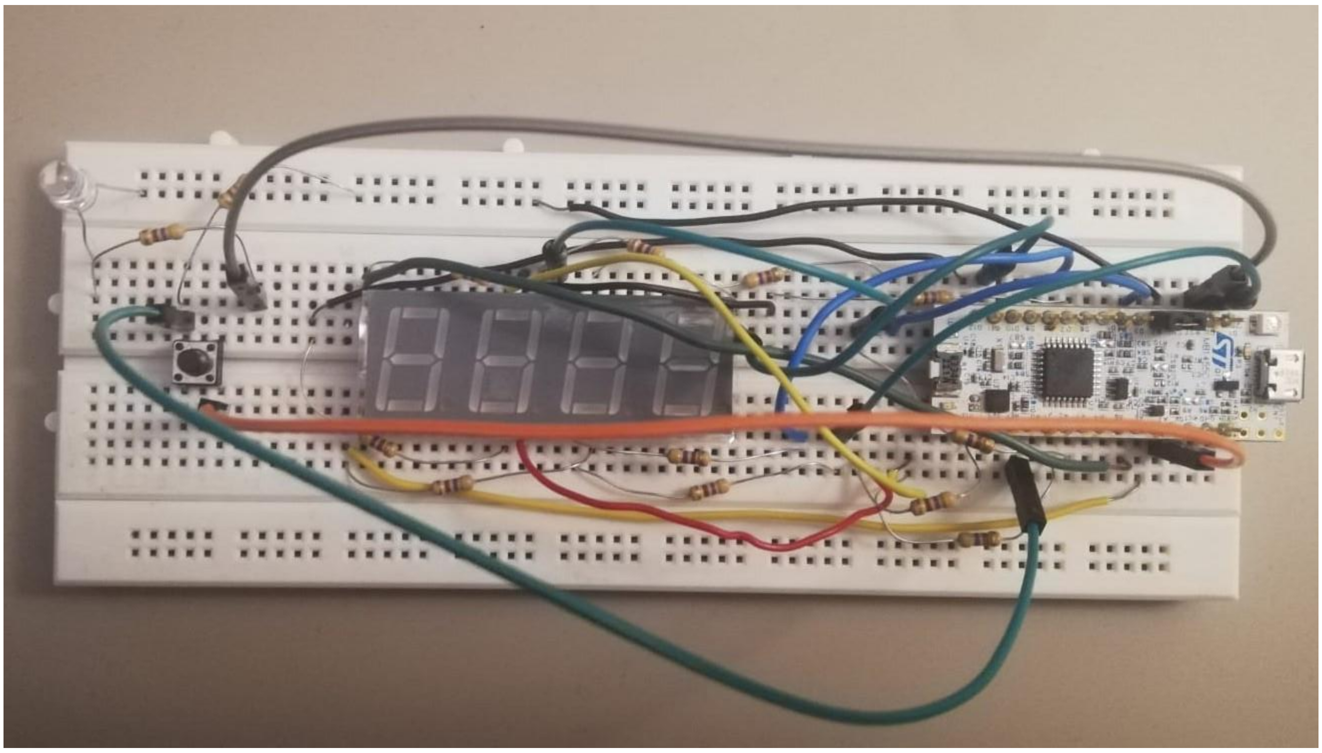


Set the components on a breadboard exactly same as the hardware diagram, breadboard figure given below. Connected pins PB0 to PB6 to corresponding A B C D E F G pins of the 4 Digit SSD each through one 470 ohm resistor. Connected pin PB7 to external indicator led, passing through a current limiting resistor with the value of 470 ohm, leading to ground since this led is working as active high.

On the GPIOA Pins; Connected the button, PA0 pin leading to leg 2 of the button, parallel to a 470 ohm resistor going to ground. Leg 3 of the button is connected to Vcc. Pins PA4 to PA7 are connected to D1 to D4 pins of the SSD in order.

Total component list is;

#	Part Name	Link	Amount	Unit Price	Total Price
1	STM32 G031K8T Development Board	https://www.empastore.com/stm32-islemci-kiti-nucleo-g031k8	1	137.59 TL	137.59 TL
2	4-Digit Seven Segment Display Common Anode	https://www.direnc.net/14mm-4lu-anot-display	1	7.45 TL	7.45 TL
3	470 Ohm Resistor	https://www.robotistan.com/14w-470r-direnc-paketi-10-adet	13	0.046 TL	0.6 TL
4	4-Pin Tactile Button	https://www.direnc.net/6x6-42mm-tach-buton-4-bacak	1	0.28 TL	0.28 TL
5	5mm Red Led	https://www.direnc.net/5mm-led-seffaf-kirmizi	1	0.19 TL	0.19 TL
6	Breadboard	https://www.direnc.net/tekli-breadboard	1	8.84 TL	8.84 TL
7	Jumper Cable M-M	https://www.direnc.net/40-adet-erkek-erkek-jumper-20cm	20	0.093 TL	1.86 TL
				Total	156.81 TL



5. Conclusion

In this project, I've utilized most of my previous acquired skills on assembly and had to learn many new skills. I've learned about Push and Pop operations, what they do exactly and how I can use them when I need to. I've utilized this said new skill to implement primitive function calls to my code. I've learned the limits of branch operations and how to get around of those said limitations, in many cases I used non conditional branch operations to increase its range.

5.1 Problems

Where I failed; I've failed to implement necessary delays and time operating windows. Decreasing number is not visible and delay functions that I've added does not suffice. I need to implement a correct push pop operation that saves seed to use on the next random operation.

I've had many problems with my development board, caused by a faulty micro USB cable that might or might not have damaged my board. ST-Link Utility Software would only detect ST-Link Debugger, not the MCU. If I were try to debug using my board STM32 would not detect my development board. I had to follow these instructions every time I wanted to debug. I had to connect the board normally via USB, update the ST-Link Debugger Firmware. Set the connecting mode under Target -> Settings -> Mode -> Connect Under Reset. Plug in the board pushing the reset button. Release the reset button when I see the "Core held under reset" prompt on ST-Link Utility.

Special thanks to our Research Assistant Erdem KÖSE, who helped me with my technical issues and provided me with a backup STM32 G031K8T Development Board to help me complete my project.

I had to write everything in VSCode without debugging and only had 5 hours to actually debug my code, which was not enough unfortunately.

6. References

1. ARM v6-M Architecture Reference Manual, ST Microelectronics
2. UM2591 User Manual STM32G0 Nucleo-32 board (MB1455), ST Microelectronics
3. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics
4. Using a button with Arduino, programmingelectronics, https://www.programmingelectronics.com/tutorial-17-using-a-button-old-version/
5. Designing a 7-segment hex decoder, Ben Eater, https://www.youtube.com/watch?v=7zffjsXqATg
6. Using an EEPROM to replace combinational logic, Ben Eater, https://www.youtube.com/watch?v=BA12Z7gQ4P0

7. Full Assembly Code

```
/*
 * project1.s
 *
 * author: Berat Kizilarmut
 *
 * description: Countdown clock which utilizes pseudo random to count
down 10 to 1 seconds at random.
 *
 */


.syntax unified
.cpu cortex-m0
.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 o
ffset

.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 of
fset


/* 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:
    /* Clocking the peripherals */
    /* Enabling GPIOA and GPIOB clock, Setting bits 0 and 1 on IOPENR
 as 1 */
```

```
    LDR r3, =(0x40021000 + 0x34)
    LDR r2, [r3]
    LDR r1, =0x3
    ORRS r2, r2, r1
    STR r2, [r3]

    /* Setting 8 pins as output on GPIOB modder, 7 of them for SSD A,
B,C... legs, 1 for output led */
    LDR r3, =(0x50000400 + 0x00) /* GPIOB Base Address + Modder Offse
t */
    LDR r2, [r3]
    LDR r1, =0xFFFF5555 /* Binary 11111111111111110101010101010101 */
    ANDS r2, r2, r1 /* Setting odd bits zero */
    LDR r1, =0x5555 /* Binary 101010101010101 */
    ORRS r2, r2, r1 /* Setting even bits one */
    STR r2, [r3]

    /* Setting PA0 for input, And PA4-
7 for output on GPIOA Modder (Because PA2-
3 is absent on board pinout) */
    LDR r3, =(0x50000000 + 0x00) /* GPIOA Base Address + Modder Offse
t */
    LDR r2, [r3]
    LDR r1, =0xFFFFFFFC /* Binary 11111111111111111111111111111100 */
    ANDS r2, r2, r1 /* Setting PA0 as input to connect a Push Button
*/
    LDR r1, =0xFFFF55FF /* Setting PA4-
7 as output to use as SSD Digit Inputs */
    ANDS r2, r2, r1 /* Setting odd bits zero */
    LDR r1, =0x5500 /* Binary 101010101010101 */
    ORRS r2, r2, r1 /* Setting even bits one */
    STR r2, [r3]

/* IDLE FUNCTION */
Idle: /* My ID is 171024086 */
    BL Clear
    /* Turning on the indicator led */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* Turning on the 8th bit, PB7 */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    BL Clear
    /* Printing Decimal 4 */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x99 /* Setting 1 1 0 0 1 1 0, G to A */
    ORRS r6, r6, r7 /* Set ones */
    ANDS r6, r6, r7 /* Set zeros */
    STR r6, [r5] /* Store */
    /* Setting Digit 1 */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x10 /* PA4, Binary 10000 to Hex 10 */
    ORRS r6, r6, r7 /* Set 5th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    BL Clear
    /* Printing Decimal 0 */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xC0 /* Setting 0 1 1 1 1 1 1, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    /* Setting Digit 2 */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x20 /* PA5, Binary 100000 to Hex 20 */
    ORRS r6, r6, r7 /* Set 6th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    BL Clear
    /* Printing Decimal 8 */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* Setting 1 1 1 1 1 1 1, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    /* Setting Digit 3 */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x40 /* PA6, Binary 1000000 to Hex 40 */
    ORRS r6, r6, r7 /* Set 7th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    BL Clear
    /* Printing Decimal 6 */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x2 /* Setting 1 1 1 1 1 0 1, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    /* Setting Digit 4 */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* PA7, Binary 10000000 to Hex 80 */
    ORRS r6, r6, r7 /* Set 8th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    BL Clear
    /* Checking for Button Press */
    LDR r3, =(0x50000000 + 0x10) /* GPIOA Base Address + IDR Offset *
/
    LDR r2, [r3]
    LDR r1, =0x1
    ANDS r2, r2, r1
    CMP r2, r1
    BNE Idle /* Reach imm8 */
    B Random /* Reach imm11 */

Clear: /* Clearing the screen before writing */
    PUSH {LR}
    LDR r7, =0x2555
Delayfunc3: /* Delaying */
    SUBS r7, #0x1
    BNE Delayfunc3

    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r7, =0xFFFF0000 /* Setting 0 0 0 0 0 0 0, G to A */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

/* PRINT FUNCTIONS */
Print: /* Print function that has many switch cases to determine the
correct way to decode the decimal info to SSD */
    LDR r7, =0x12555
Delayfunc4: /* Delaying */
    SUBS r7, #0x1
    BNE Delayfunc4
PrintD1: /* Thousands Decimal Point, Switch Case to compare counter n
umbers */
    BL Clear
    CMP r1, #0x0 /* Compare the value */
    BNE Skip11 /* If it's not the same, skip the Branch Link */
    BL Print0 /* If it's same, go to Branch Link and Print out the va
lue */
Skip11: /* Skip to Digit 1, Number 1 */

    CMP r1, #0x1
    BNE Skip12
    BL Print1
Skip12:

    CMP r1, #0x2
    BNE Skip13
    BL Print2
Skip13:

    CMP r1, #0x3
    BNE Skip14
    BL Print3
Skip14:

    CMP r1, #0x4
    BNE Skip15
    BL Print4
Skip15:

    CMP r1, #0x5
    BNE Skip16
    BL Print5
Skip16:
    CMP r1, #0x6
    BNE Skip17
    BL Print6
Skip17:

    CMP r1, #0x7
    BNE Skip18
    BL Print7
Skip18:

    CMP r1, #0x8
    BNE Skip19
    BL Print8
Skip19:
    BL Digit1 /* Set the correct digit */
    CMP r1, #0x9
    BNE PrintD2  /* Move on to Hundrends Decimal Point */
    BL Print9

PrintD2: /* Hundreds Decimal Point, Switch Case to compare counter nu
mbers */
    BL Clear
    CMP r2, #0x0 /* Compare the value */
    BNE Skip21 /* If it's not the same, skip the Branch Link */
    BL Print0 /* If it's same, go to Branch Link and Print out the va
lue */
Skip21: /* Skip to Digit 2, Number 1 */

    CMP r2, #0x1
    BNE Skip22
    BL Print1
Skip22:

    CMP r2, #0x2
    BNE Skip23
    BL Print2
Skip23:

    CMP r2, #0x3
    BNE Skip24
    BL Print3
Skip24:

    CMP r2, #0x4
    BNE Skip25
    BL Print4
Skip25:

    CMP r2, #0x5
    BNE Skip26
    BL Print5
Skip26:

    CMP r2, #0x6
    BNE Skip27
    BL Print6
Skip27:
    CMP r2, #0x7
    BNE Skip28
    BL Print7
Skip28:

    CMP r2, #0x8
    BNE Skip29
    BL Print8
Skip29:
    BL Digit2 /* Set the correct digit */
    CMP r2, #0x9
    BNE PrintD3  /* Move on to Tens Decimal Point */
    BL Print9

PrintD3: /* Tens Decimal Point, Switch Case to compare counter number
s */
    BL Clear
    CMP r3, #0x0 /* Compare the value */
    BNE Skip31 /* If it's not the same, skip the Branch Link */
    BL Print0 /* If it's same, go to Branch Link and Print out the va
lue */
Skip31: /* Skip to Digit 3, Number 1 */

    CMP r3, #0x1
    BNE Skip32
    BL Print1
Skip32:

    CMP r3, #0x2
    BNE Skip33
    BL Print2
Skip33:

    CMP r3, #0x3
    BNE Skip34
    BL Print3
Skip34:

    CMP r3, #0x4
    BNE Skip35
    BL Print4
Skip35:

    CMP r3, #0x5
    BNE Skip36
    BL Print5
Skip36:

    CMP r3, #0x6
    BNE Skip37
    BL Print6
Skip37:

    CMP r3, #0x7
    BNE Skip38
    BL Print7
Skip38:

    CMP r3, #0x8
    BNE Skip39
    BL Print8
Skip39:
    BL Digit3 /* Set the correct digit */
    CMP r3, #0x9
    BNE PrintD4  /* Move on to Ones Decimal Point */
    BL Print9

PrintD4: /* Ones Decimal Point, Switch Case to compare counter number
s */
    BL Clear
    CMP r4, #0x0 /* Compare the value */
    BNE Skip41 /* If it's not the same, skip the Branch Link */
    BL Print0 /* If it's same, go to Branch Link and Print out the va
lue */
Skip41: /* Skip to Digit 4, Number 1 */

    CMP r4, #0x1
    BNE Skip42
    BL Print1
Skip42:

    CMP r4, #0x2
    BNE Skip43
    BL Print2
Skip43:

    CMP r4, #0x3
    BNE Skip44
    BL Print3
Skip44:

    CMP r4, #0x4
    BNE Skip45
    BL Print4
Skip45:

    CMP r4, #0x5
    BNE Skip46
    BL Print5
Skip46:

    CMP r4, #0x6
    BNE Skip47
    BL Print6
Skip47:

    CMP r4, #0x7
    BNE Skip48
    BL Print7
Skip48:

    CMP r4, #0x8
    BNE Skip49
    BL Print8
Skip49:
    BL Digit4 /* Set the correct digit */
    CMP r4, #0x9
    BNE Skip50
    BL Print9
Skip50:
    /* Checking Button Input */
    LDR r3, =(0x50000000 + 0x10) /* GPIOA Base Address + IDR Offset *
/
    LDR r2, [r3]
    LDR r1, =0x1
    ANDS r2, r2, r1
    CMP r2, r1
    BEQ Pause
    B Countdown

/* RANDOM FUNCTIONS */
Random: /* Utilizing pseudo random number algorithm from HW2, Problem
1 */
    /* Turning off the indicator led */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* Turning off the 8th bit, PB7 */
    MVNS r7, r7
    ORRS r6, r6, r7
    ANDS r6, r6, r7
    STR r6, [r5] /* Store */

    LDR r0, =0x1B67 /* Set seed */
    LDR r1, =0x41A7 /* a = 16807 */
    LDR r2, =0x7FFFFFFF /* b = 2147483647 */
    MULS r0, r1, r0 /* next = (seed * a) */
    BL Modulo /* random = next % b */

    LDR r2, =0x2327 /*  random = random %8999 */
    BL Modulo

    LDR r3, =0x3E8
    ADDS r0, r0, r3 /* Add 1000 to make the random value fall within
1000 and 9999 */
    PUSH {r0} /* Push to use on Countdown */
    B Decimal_Conversion /* Branch to Decimal Conversion Function */

Modulo: /* Modulo Op to use in Random Op */
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
Modulo_Loop:
    CMP r0, r2
    BLT Leave /* If r0 is lower than r2, modulo op is completed, leav
e function */
    SUBS r0, r0, r2
    BGE Modulo_Loop /* Loop if greater or equal */
Leave: /* Leave the function and return back */
    POP {PC}

Countdown:
    BL Clear
    LDR r7, =0x12555
Delayfunc1: /* Delaying */
    SUBS r7, #0x1
    BNE Delayfunc1
    /* Checking Button Input */
    LDR r5, =(0x50000000 + 0x10) /* GPIOA Base Address + IDR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x1
    ANDS r6, r6, r7
    CMP r6, r7
    BEQ Pause
    CMP r0, #0x0
    BEQ Idle_Jump /* Cannot branch to "Idle", need another operation
*/
    SUBS r0, #0x1
    B Decimal_Conversion

Idle_Jump: /* Added a jump branch to solve the problem*/
    B Idle

Pause:
    LDR r7, =0x12555
Delayfunc2: /* Delaying */
    SUBS r7, #0x1
    BNE Delayfunc2

    LDR r5, =(0x50000000 + 0x10) /* GPIOA Base Address + IDR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x1
    ANDS r6, r6, r7
    CMP r6, r7
    BEQ Countdown
    B Pause

/* DECIMAL CONVERSION FUNCTIONS */
Decimal_Conversion: /* Function to convert the random hex value to de
cimal */
    LDR r1, =0x0 /* Thousands Counter */
    MOVS r2, r1 /* Hundreds Counter */
    MOVS r3, r1 /* Tens Counter */
    MOVS r4, r1 /* Ones Counter */

Thousand:
    LDR r5, =0x3E8 /* Loading 1000 decimal to compare to */
    CMP r0, r5 /* Comparing the random value to 1000 */
    BLT Hundred /* If the value is lower than thousand, skip to check
ing hundreds */
    SUBS r0, r0, r5 /* Subtract a thousand */
    ADDS r1, #0x1 /* Increasing the Thousands Counter */
    B Thousand /* Loop */

Hundred:
    LDR r5, =0x64 /* Loading 100 decimal to compare to */
    CMP r0, r5 /* Comparing the random value to 100 */
    BLT Ten /* If the value is lower than hundred, skip to checking t
ens */
    SUBS r0, r0, r5 /* Subtract a hundred */
    ADDS r2, #0x1 /* Increasing the Hundreds Counter */
    B Hundred /* Loop */

Ten:
    LDR r5, =0xA /* Loading 10 decimal to compare to */
    CMP r0, r5 /* Comparing the random value to 10 */
    BLT One /* If the value is lower than ten, skip to checking ones
*/
    SUBS r0, r0, r5 /* Subtract ten */
    ADDS r3, #0x1 /* Increasing the Tens Counter */
    B Ten /* Loop */

One:
    LDR r5, =0x1 /* Loading 1 to compare to */
    CMP r0, r5 /* Comparing the random value to 1 */
    BLT Print_Jump /* If the value is lower than one, jump to printin
g. Can't because of imm8 need another branch to assist in jump */
    SUBS r0, r0, r5 /* Subtract one */
    ADDS r4, #0x1 /* Increasing the Ones Counter */
    B One /* Loop */

Print_Jump: /* Branch to extend the reach */
    B Print

/* Setting Seven Segment Display Digits */
Digit1:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x10 /* PA4, Binary 10000 to Hex 10 */
    ORRS r6, r6, r7 /* Set 5th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Digit2:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x20 /* PA5, Binary 100000 to Hex 20 */
    ORRS r6, r6, r7 /* Set 6th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Digit3:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x40 /* PA6, Binary 1000000 to Hex 40 */
    ORRS r6, r6, r7 /* Set 7th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Digit4:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
```

```
    LDR r5, =(0x50000000 + 0x14) /* GPIOA Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* PA7, Binary 10000000 to Hex 80 */
    ORRS r6, r6, r7 /* Set 8th bit 1 */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

/* Seven Segment Display Number Encoding */
/* Using Pins PB0 to PB6 as A to G on Seven Segment Display */
Print0:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xC0 /* Setting 1000000, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print1:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xF9 /* Setting 1111001, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print2:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xA4 /* Setting 0100100, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print3:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xB0 /* Setting 0110000, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */
```

```
Print4:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x99 /* Setting 0011001, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print5:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x92 /* Setting 0010010, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print6:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x82 /* Setting 0000010, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print7:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0xF8 /* Setting 1111000, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

Print8:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x80 /* Setting 0000000, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */
```

```
Print9:
    PUSH {LR} /* Pushing lr to return to branching point on the futur
e */
    LDR r5, =(0x50000400 + 0x14) /* GPIOB Base Address + ODR Offset *
/
    LDR r6, [r5]
    LDR r7, =0x90 /* Setting 0010000, G to A */
    ORRS r6, r6, r7 /* Set the value */
    ANDS r6, r6, r7 /* Set everything else zero */
    STR r6, [r5] /* Store */
    POP {PC} /* Return back */

    /* for(/* /* )/*  */
    b .

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