GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

ELECTRONICS ENGINEERING

ELEC 334

MICROPROCESSORS

HW2

Name - Surname	Yağmur DERYA
Student ID	171024011

(main.c in problem 1 and problem 2 is named as "main 2.c" because of the "Teams". I edited it twice and teams renamed it in this way. In original project it was named as "main.c").

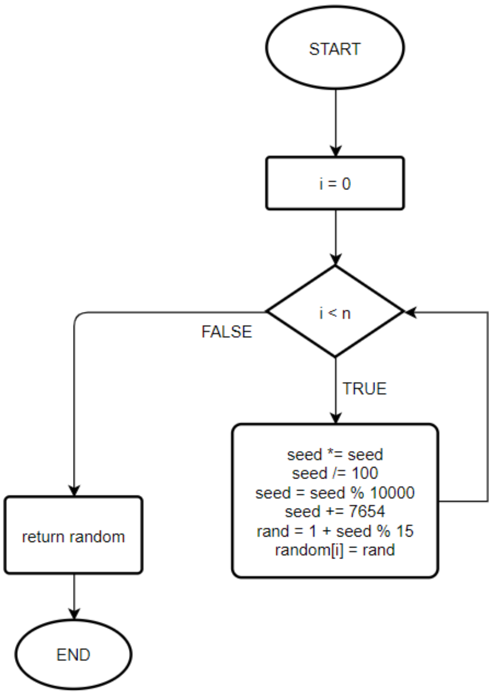
Problem 1

I inspired by middle-square method which is a method of generating pseudo-random numbers.

To generate a sequence of n-digit random numbers, an n-digit starting value is created and squared, producing a 2*n-digit number. The middle n digits of the result would be the next number in the sequence and returned as the result.

To avoid repetition, a random number is added to the seed in for loop.

Flowchart:



C Code:

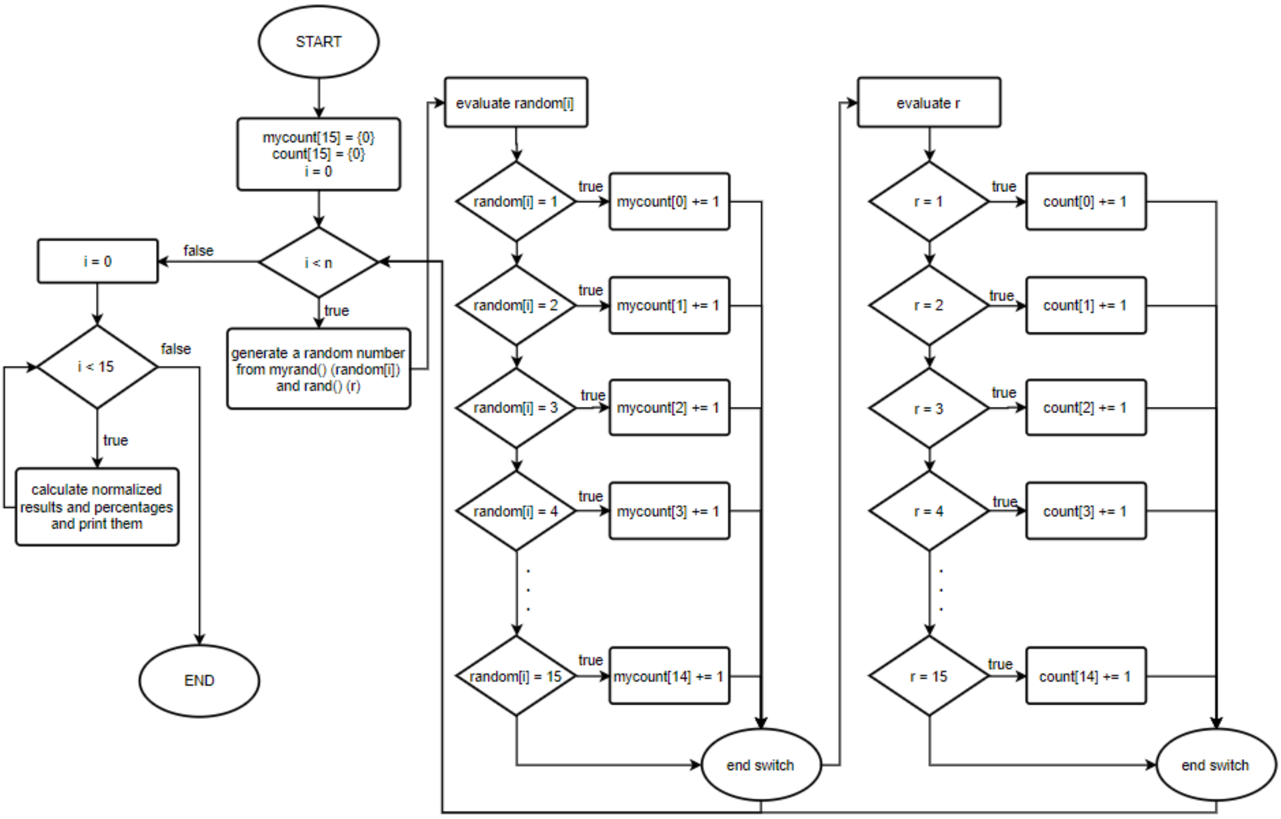
```
//myrand.h

#ifndef _MYRAND_H
#define _MYRAND_H

long int* myrand(int, int, long int*);

#endif
```

```
//myrand.c

#include <stdio.h>
#include "myrand.h"

long int* myrand(int seed, int n, long int* random){

        int i, rand;

        /* inspired by mid square method */
        for(i = 0; i < n; ++i){
                seed *= seed; //8-digit number
                seed /= 100; //deletes last 2 digits
                seed = seed % 10000; //deletes first 2 digits
                seed += 7654; //to make it more random, 4-digit random number is added
                rand = 1 + seed % 15; //to generate numbers between [1, 15]
                random[i] = rand;
        }

        return random;
}
```

```
//main.c

#include <stdio.h>
#include "myrand.h"

int main(){

        int x; //loop number to generate final random number
        long int* random = (long int*) malloc (x * sizeof(long int)); //random array to get all
random numbers from myrand()

        printf("enter how many loops to generate the random number: ");
        scanf("%d", &x);
        int seed = 7394; //random seed value which is given by me

        random = myrand(seed, x, random);
        printf("random number: %d\n", random[x-1]);
        free(random);
        return 0;
}
```

Problem 2

Flowchart:



C Code:

```
//myrand.h

#ifndef _MYRAND_H
#define _MYRAND_H

long int* myrand(int, int, long int*);

#endif
```

```
//test_random.h

#ifndef _TESTRAND_H
#define _TESTRAND_H

void test_random(int, int, long int*);

#endif
```

```
//test_random.c

#include <stdio.h>
#include "test_random.h"
#include <time.h>
#include "myrand.h"

void test_random(int seed, int n, long int* random){
        int i, r;
        int mycount[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
        int count[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

        for(i=0; i<n; ++i){
                r = 1 + rand() % 15;
//              /* to check numbers*/
//              printf("%d - my rand: %d\t\trand: %d\n", i+1, random[i], r);

                /* count numbers 1-15 which are generated by myrand() and rand() */
                switch (random[i]){
                        case 1:
                                mycount[0] +=1;
                                break;
                        case 2:
                                mycount[1] +=1;
                                break;
                        case 3:
                                mycount[2] +=1;
                                break;
                        case 4:
                                mycount[3] +=1;
                                break;
                        case 5:
                                mycount[4] +=1;
                                break;
                        case 6:
                                mycount[5] +=1;
                                break;
                        case 7:
                                mycount[6] +=1;
                                break;
                        case 8:
                                mycount[7] +=1;
                                break;
                        case 9:
                                mycount[8] +=1;
                                break;
                        case 10:
                                mycount[9] +=1;
                                break;
                        case 11:
                                mycount[10] +=1;
                                break;
                        case 12:
                                mycount[11] +=1;
                                break;
                        case 13:
                                mycount[12] +=1;
                                break;
                        case 14:
                                mycount[13] +=1;
                                break;
                        case 15:
                                mycount[14] +=1;
                                break;
                }

                switch (r){
                        case 1:
                                count[0] +=1;
                                break;
                        case 2:
                                count[1] +=1;
                                break;
                        case 3:
                                count[2] +=1;
                                break;
                        case 4:
                                count[3] +=1;
                                break;
                        case 5:
                                count[4] +=1;
                                break;
                        case 6:
                                count[5] +=1;
                                break;
                        case 7:
                                count[6] +=1;
                                break;
                        case 8:
                                count[7] +=1;
                                break;
                        case 9:
                                count[8] +=1;
                                break;
                        case 10:
                                count[9] +=1;
                                break;
                        case 11:
                                count[10] +=1;
                                break;
                        case 12:
                                count[11] +=1;
                                break;
                        case 13:
                                count[12] +=1;
                                break;
                        case 14:
                                count[13] +=1;
                                break;
                        case 15:
                                count[14] +=1;
                                break;
                } //count ends

                if(i == (n-1))
                        printf("\n");

        } //for ends
        float mynorm, norm, myperc, perc;

        /* calculate normalized results of the generated numbers */
        printf("*results from myrand()*\t\t*results from rand()*\n");
        for(i=0; i<15; ++i){

                myperc = mycount[i] * 100;
                myperc /= n;
                perc = count[i] * 100;
                perc /= n;
                mynorm = myperc / 100;
                norm = perc / 100;

                printf("%d = %f (%%%.3f)\t\t%d = %f (%%%.3f)\n", i+1, mynorm, myperc, i+1, norm,
perc);
        }
}
```

```
//myrand.c

#include <stdio.h>
#include "myrand.h"

long int* myrand(int seed, int n, long int* random){

        int i, rand;

        /* inspired by mid square method */
        for(i = 0; i < n; ++i){
                seed *= seed; //8-digit number
                seed /= 100; //deletes last 2 digits
                seed = seed % 10000; //deletes first 2 digits
                seed += 7654; //to make it more random, 4-digit random number is added
                rand = 1 + seed % 15; //to generate numbers between [1, 15]
                random[i] = rand;
        }

        return random;
}
```

```
//main.c

#include <stdio.h>
#include "myrand.h"
#include "test_random.h"


int main(){

        int x; //number of the tested random numbers
        long int* random = (long int*) malloc (x * sizeof(long int)); //random array to get all
random numbers from myrand()

        printf("enter how many numbers you generate: ");
        scanf("%d", &x);
        int seed = 7394; //random seed value which is given by me

        random = myrand(seed, x, random);
        test_random(seed, x, random);
        free(random);

        return 0;
}
```

Problem 3

- `ldr r5, [r6, #4]`

0 1 1	0	1	0 0 0 0 1	1 1 0	1 0 1
			imm5(<<2)	Rn	Rt

Hexadecimal representation: 6875

- `mvns r4, r4`

0 1 0 0 0 0	1 1 1 1	1 0 0	1 0 0
		Rm	Rd

Hexadecimal representation: 43E4

- `ands r5, r5, r4`

0 1 0 0 0 0	0 0 0 0	100	101
		Rm	Rd

Hexadecimal representation: 4025

- `adds r0, r0, r1`

0 0 0	1 1	0	0	0 0 1	0 0 0	0 0 0
				Rm	Rn	Rd

Hexadecimal representation: 1840

- `subs r2, r4, #2`

0 0 0	1 1	1	1	0 1 0	1 0 0	0 1 0
				imm3	Rn	Rd

Hexadecimal representation: 1EA2

- `asrs r2, r4, #21`

0 0 0	1 0	1 0 1 0 1	1 0 0	0 1 0
		imm5	Rm	Rd

Hexadecimal representation: 1562

- `str r5, [r6, r1]`

0 1 0 1	0 0 0	0 0 1	1 1 0	1 0 1
		Rm	Rn	Rt

Hexadecimal representation: 5075

- `bx lr (= bx R14)`

0 1 0 0 0 1	1 1	0	1110	0 0 0
			Rm	

Hexadecimal representation: 4770

- `bne 0x12`

1 1 0 1	0 0 0 1	0 0 0 1 0 0 1 0
	cond	imm8

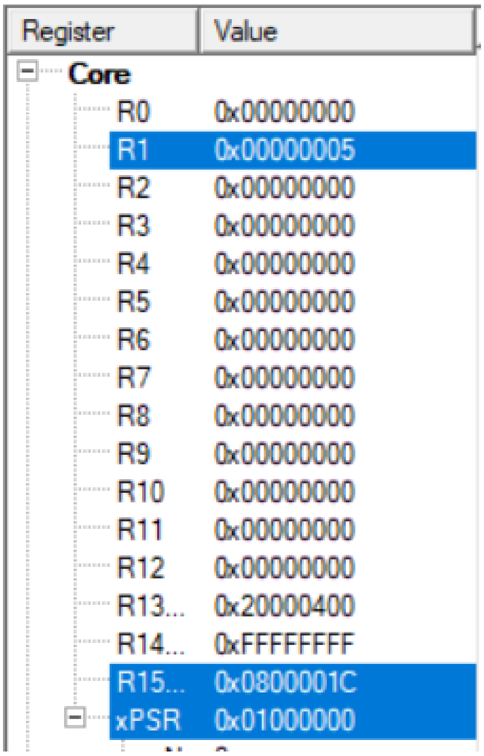
Hexadecimal representation: D112

Problem 4

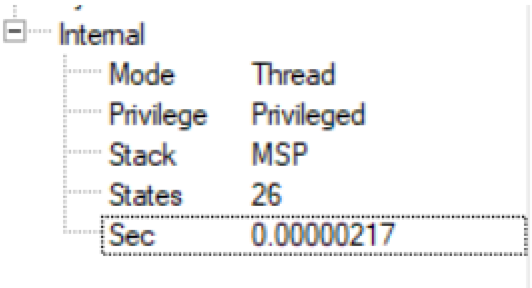
instruction	cycles
LDR	2 or 1
MVNS	1
ANDS	1
ADDS	1
ADD	1
SUBS	1
ASRS	1
STR	2 or 1
BX	2
BNE	2 or 1

Problem 5

```
       MOVS R0, #5
       MOVS R1, #0 ; counter to test delay

    delay
       SUBS R0, #1
       ADDS R1, #1
       CMP R0, #0
       BNE delay
```



Register	Value
Core	
R0	0x00000000
R1	0x00000005
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13...	0x20000400
R14...	0xFFFFFFFF
R15...	0x0800001C
xPSR	0x01000000



Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	26
Sec	0.00000217

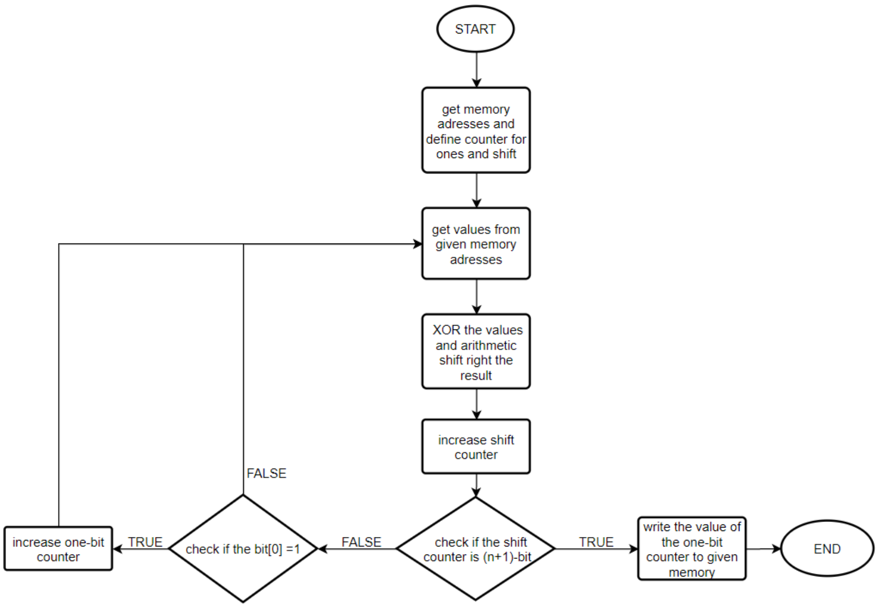
The delay is 0.00000217 seconds.

Problem 6

```
/* 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 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

/* main function */
.section .text
main:
       /* enable GPIOB clock, bit1 on IOPENR */
       ldr r6, =RCC_IOPENR
       ldr r5, [r6]
       /* movs expects imm8, so this should be fine */
       movs r4, 0x2
       orrs r5, r5, r4
       str r5, [r6]

       /* setup PB12 for led 01 for bits 25-24 in MODER */
       ldr r6, =GPIOB_MODER
       ldr r5, [r6]
       ldr r4, =0x3000000
       mvns r4, r4
       ands r5, r5, r4
       ldr r4, =0x1000000
       orrs r5, r5, r4
       str r5, [r6]

       /* turn on led connected to B12 in ODR */
       ldr r6, =GPIOB_ODR
       ldr r5, [r6]
       ldr r4, =0x1000
       orrs r5, r5, r4
       str r5, [r6]
       b .
       nop
```

Problem 7

Flowchart:



START
get memory adresses and define counter for ones and shift
get values from given memory adresses
XOR the values and arithmetic shift right the result
increase shift counter
FALSE
increase one-bit counter
TRUE
check if the bit[0] =1
FALSE
check if the shift counter is (n+1)-bit
TRUE
write the value of the one-bit counter to given memory
END

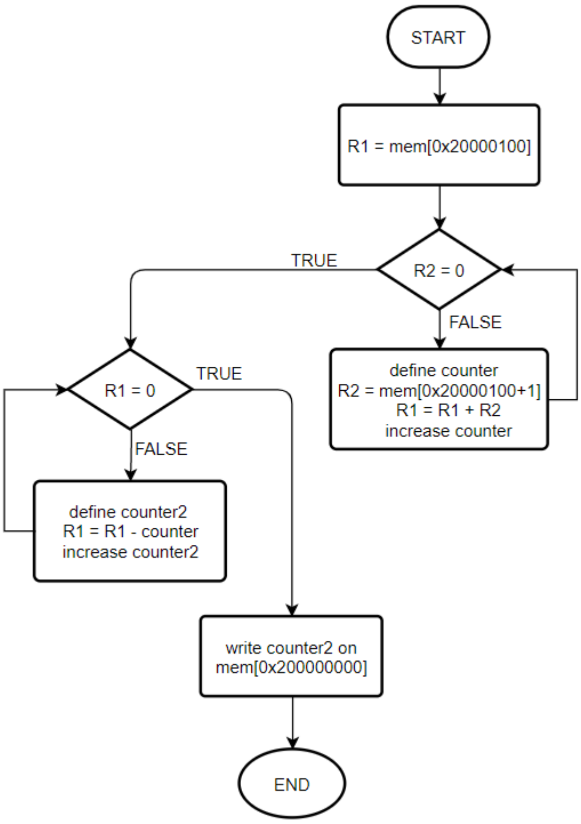
Assembly Code:

```
;the values are assumed as 16-bit
       LDR R0,=0x14224
       LDR R1,=0x14228
       MOVS R2, #0x0 ; to count ones
       MOVS R3, #0x0 ; to shift
loop
       LDR R4,[R0] ; R4 = mem[0x14224]
       LDR R5,[R1] ; R5 = mem[0x14228]
       EORS R4, R4, R5 ; R4 = R4 ^ R5 ; to set different bits
       ASRS R4, R4, R3 ; R4 = R4 >> R3
       MOVS R7, #0x1
       ADDS R3, #1 ; increase R3 to shift one more right in every loop
       CMP R3, #17
       BEQ finish ; if R3 = 17 leave the loop and go to <finish>
       ANDS R7, R7, R4 ; if R3 != 17, R7 = R7 & R4
       CMP R7, #1
       BNE loop ; if R7 != 1, go back to loop
       ADDS R2, R2, #1 ; if R7 = 1, increase counter
       B loop ; go back to loop
finish
       LDR R6, =0x1422C
       STR R2, [R6]
```

Normally, the address should start from 0x20000000 but it is assumed that we can do memory operations in these addresses.

Problem 8

Flowchart:



Assembly Code:

```
        LDR R0, =0x20000100
        LDR R1, [R0] ;R1 = 12
        MOVS R7, #0 ; To count elements of the array
        MOVS R6, #0 ; Counter for divider
loop
        ADDS R7, #1
        LDR R2, [R0, R7] ;R2 = 27 .. 30 .. 29
        ADDS R1, R1, R2 ;R1 = R1 + R2
        CMP R2, #0
        BNE loop

div
        SUBS R1, R1, R7
        ADDS R6, #1
        CMP R1, #0
        BNE div

        LDR R0, =0X20000000
        STR R6, [R0]
```

References

[1] https://en.wikipedia.org/wiki/Middle-square_method

[2] ARMv6-M Architecture Reference Manual

[3] https://developer.arm.com/documentation/ddi0484/b/Programmers-Model/Instruction-set-summary

[4] RM0444 Reference Manual - STM32G0x1 advanced ARM®-based 32-bit MCUs