Table of Contents

1. FIC Probleme

1.1. Exemplul 1

```
unsigned int mask=12
unsigned int seed=15
unsigned int lfsr_shift(unsigned int lfsr)
{
 unsigned int feedback=0;
 feedback = lfsr & 1;
 lfsr >>= 1;
 if ( feedback == 1)
        lfsr ^=mask;
 return lfsr;
}

void main(void)
{
   unsigned int lfsr = seed;
   do {
     lfsr = lfsr_shift(lfsr);
     print(lfsr);
   }while(lfsr!=seed)
}
```

```
             LDR R0,SEED      //incarc in registrul R0 valoarea seed-ului
L1           JMS shift_lfsr   //apelez subrutina shift_lfsr
             OUT R0,4         //afisez valoarea din R0
             LDR R2,SEED      //incarc valoarea seed-ului si in R2
             CMP R0,R2        //compar R0 cu R2
             BNE L1           //si daca sunt diferite, salt la L1
             HLT
shift_lfsr   PSH {LR}         //salvez registrul LR pe stiva
             LDR R1,MASK      //incarc in registrul R1 valoarea mastii
             MOV R2,#1        //initializez registrul R2 cu 1
             AND R3,R0,R2     //R3 = R0 & R2
             LSR R0,R0,#1     //R0 = R0 >> 1
             CMP R3,#1        //compar R3 cu 1
             BNE end_s        //si daca nu sunt egale, salt la end_s
             XOR R0,R0,R1     //R0 = R0 ^ R1
end_s        POP {PC}         //iau valoarea LR-ului de pe stiva si o scriu in PC
            RET               //return din subrutina
SEED    DAT 15
MASK    DAT 12
```

1.2. Exemplul 2

Implementati in limbajul de asamblare al procesorului RISC (simulat online) o subrutina ce va permite citirea unei valori K si a unor numere pana cand se introduce 0. Pentru fiecare valoare citita se va seta bitul K (bitul K se pune pe 1) si noua valoare va fi salvata in memorie incepand cu adresa 200. Pentru setarea bitului K aveti nevoie de shiftare la stanga si OR.

```
start LDR R0, arrayaddr
        INP R1, 2       //k bit
      JMS read_fct   //R0= arrayaddr, R1=k
      HLT
arrayaddr DAT 200

read_fct PSH {LR}
         MOV R4, #1
         LSL R4,R4,R1 //R4=R4<<R1 = 1<<k
         MOV R2, #0 //used to count the number of values
L1       INP R3, 2 //read value
         CMP R3, #0
         BEQ L2
         ORR R3, R3, R4
         STR R3, [R0]
         ADD R0, #1
         ADD R2, #1
         BRA L1
L2       POP {PC}
         RET
```

1.3. Exemplul 3

Se citeste o valoare N>0 ce va reprezenta numarul de elemente ale unui array. Sa se implementeze urmatoarele functii: a) functie care va citi valori care vor fi salvate intr-un array incepand cu adresa 180. b) functie care va afisa valorile din array de la adresa 180. c) functie care va calcula valoarea minima din array. d) functie care va construi un nou array la adresa 200, scazand din array-ul initial, valoarea minima. e) functie care va sorta cel de-al doilea array (bubble-sort).

```
start   INP R0, 2 // input N                                                    Assembler
                LDR R1, arrayaddr
                JMS input_fct
                JMS output_fct
                PSH {R0}
                JMS min_fct
                MOV R2, R0
                POP {R0}
                //R0=n, R1, arrayaddr, R2=min value
                LDR R3, new_addr
                JMS create_fct
                LDR R1, new_addr
                //R0=n, R1=addr
                JMS sort
                HLT
arrayaddr DAT 180
new_addr  DAT 200

input_fct       PSH {LR}
                        MOV R2,#0 //index
                        MOV R5, #0
L1          INP R4, 2 //input value
            ADD R5, R2, R1
            STR R4, [R5]
            ADD R2, #1
            CMP R2,R0
```

```
            BLT L1
                        POP {PC}

output_fct  PSH {LR}
                        MOV R2,#0 //index
                        MOV R5, #0
L2          ADD R5, R2, R1
                        LDR R3, [R5]
                        OUT R3, 4
                        ADD R2,#1
                        CMP R2,R0
                        BLT L2
                        POP {PC}
//R0=N, R1=arrayaddr
min_fct     PSH {LR}
                        MOV R2, #1 //index i
                        LDR R4, [R1]
                        MOV R5,#0
L3          ADD R5, R2, R1 // R5=addr R1+index i
                        LDR R3, [R5]
                        CMP R4, R3
                        BLT end
                        MOV R4,R3
end         ADD R2,#1
                        CMP R2, R0
                        BLT L3
                        OUT R4, 4
                        MOV R0, R4
                        POP {PC}
                        //R0=n, R1, arrayaddr, R2=min value, R3=new_addr
create_fct  PSH {LR}
                        MOV R4,#0
                        MOV R5,#0
L4          ADD R5, R1,R4  //calculate address
                        LDR R6, [R5]
                        SUB R6, R6,R2
                        ADD R5, R3, R4
                        STR R6, [R5]
                        ADD R4,#1
                        CMP R4,R0
                        BLT L4
                        POP {PC}
//R0=N, R1=200
sort        PSH {LR}
                        MOV R7, R0
                        SUB R7, #1 //R7=N-1
                        MOV R2, #0 //i
                        MOV R3, #0 //j
                        MOV R4, #0 //used as some variable
LI          MOV R3, R2
                        ADD R3, #1 // j=i+1
LJ                      ADD R4, R2,R1 //calculate address=address+i
                        LDR R5, [R4] //load a[i] in R5
                        ADD R4, R3,R1 //calculate address=address+j
                        LDR R6, [R4] //load a[j] in R6
                        CMP R5,R6 //compare a[i] and a[j]
                        BLT L7    //if a[i]<a[j]=> jump to L7 and just increment j and i
                        //if a[i]>=a[j]
                        MOV R4, R5 //R4 = a[i]
                        MOV R5, R6 //a[i] = a[j]
                        MOV R6, R4 //a[j] = R4
                        ADD R4, R2, R1 //R4 = address + i
                        STR R5, [R4] //save a[i]
                        ADD R4, R3,R1 //R4 = address + j
                        STR R6, [R4] //save a[j]
```

```
L7          ADD R3, #1 //increment j
                       CMP R3, R0
                       BLT LJ
                       ADD R2, #1 //increment i
                       CMP R2, R7 //compare i with N-1
                       BLT LI     //if i is less than N-1, then continue looping :)
                       POP {PC}
```

1.3.1. Exemplul 4

```
#include <stdio.h>

int main()
{
    int i = 9;
    unsigned int count = 0;
    while (i) {
        count += i & 1;
        i >>= 1;
    }
    printf("%d", count);
    return 0;
}
```

```
start   INP R0,2
        MOV R1,#0
L1      MOV R2, R0
        AND R2, #1
        ADD R1,R1,R2
        LSR R0,#1
        CMP R0,#0
        BNE L1
        OUT R1,4
        HLT
```

Created: 2022-11-11 Fri 13:34
Validate