PURS – rješenja oglednog primjera ZI

(rješenja su kopirana s fer2, znači moguće je da nešto nije točno)

1.

Dan je programski odsječak u asembleru. Registar R7 sadrži adresu na kojoj se u memoriji nalazi niz riječi čija je duljina sadržana u registru R8. Potrebno je napisati funkciju u C jeziku, koja pronalazi riječ 1 u nizu. Povratne vrijednosti funkcije su broj pojavljivanja riječi 1, te adresa posljednje riječi 1 u nizu. Ukoliko znak nije pronađen vratiti 0 za broj pojavljivanja i posljednju adresu niza.

```
                    AREA    asm_zadatak, CODE, READONLY
                    ARM
                    EXPORT asm_zad
        asm_zad
                    IMPORT c_zadatak
                    ; . . .
                    LDR R7, =adresa_niza
                    MOV R8, #12
                    STMDB R13!, {R7, R8}
                    MOV R1, R13
                    BL c_zadatak
                    NOP
                    NOP
                    ; . . .
        adresa_niza DCD 1,2,3,0,1,1,0,0,9,1,0,2
                    END
```

Rješenje:

```
int c_zadatak (int arg0, int *args){
    int brojac=0;
    int *adresa = (int *)(*args);
    int duljina = *(args+1);
    int adresa_zadnji;

    while(duljina >0){
        if(*adresa == 1){
            brojac ++; adresa_zadnji= adresa;
        }
        adresa ++;
        duljina --;
    }


    if(brojac)
        *args=adresa_zadnji;
    else
        *args=adresa-1;

    return brojac;
}
```

Objašnjenje, kolegica fucaboy's girl:

Pitanje: Jel dobro riješen 1. zad ovdje, zašto nigdje ne koriste drugi argument funkcije c_zadatak? I da li je R7 iz zadatka arg0, a R8 *args, ako da zar ne bi trebalo biti obrnuto?

Odgovor: Da, u redu je riješen. Poanta je u tome da u ovom asembleru vidiš da se stog pusha u R1, a na stogu su redom R7 i R8. E sad, funkcija prenosi najprije R0-R3 kao svoje argumente, a ako ima dodatnih argumenata oni će dalje preko stoga, UVIJEK ovim redoslijedom. Zato ti moraš prvo 'preskočit' R0 argument. To je ovaj arg0 i njega ignoriramo jer nam treba samo da dođemo do R1. Kako je u R1 stack pointer (to je R13, jel), onda to treba bit pointer i pointa na redom R7 i R8. Znači ovo što ona dalje uzme adresu sa *args, to je R7, a duljinu sa (*args+1), to je R8.

2.

Dan je skup funkcija koje tokom izvršavanja daju zahtjev za programskim prekidom, pri čemu je pripadajući SWI broj 0x12, odnosno 0x654321. Pretpostaviti da postoji programski odsječak za početnu obradu programskog prekida (ASM_SWI_handler) koji poziva odgovarajuću funkciju za posluživanje prekida kojoj prosljeđuje SWI broj i adresu vrha stoga. Funkcija koja je zatražila programski prekid proslijedila je četiri argumenta. Potrebno je napisati C kod funkcije za posluživanje prekida koja vraća broj neparnih brojeva među tim argumentima ukoliko je prekid zatražila funkcija koja koristi ARM skup instrukcija. Ukoliko je prekid zatražila funkcija koja koristi THUMB skup instrukcija, potrebno je vratiti broj parnih brojeva među argumentima. Rezultat je potrebno vratiti funkciji koja je zatražila programski prekid.

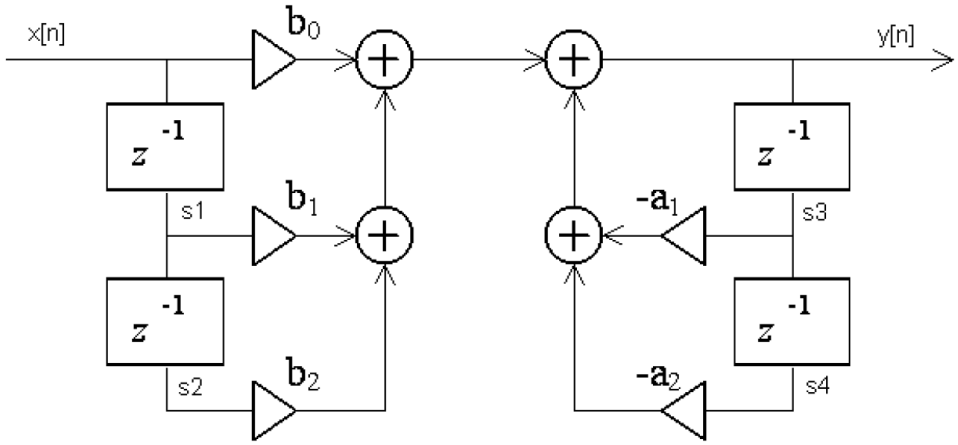
Rješenje:

```
void C_SWI_Handler(int SWI_num, int *args) {
    switch(SWI_num){
        case(0x654321): Broji_neparne(args);
        case(0x12): Broji_parne(args);
    }
}


void Broji_neparne(int *args){
    int *niz= (int*)(*args);
    int brojac=0;

    for (int i=0; i<4; i++){
        if (*niz%2 ==1 )
            brojac++;
        niz++;
    }
    *args=brojac;
}

//... isto i za parne samo u if (*niz%2==0)....
```

9.

Napišite C funkciju po uzoru na FIR filtar s labosa, koja implementira filtarsku strukturu danu slikom. Koristite generičku cjelobrojnu implementaciju (dakle tipove short (16 bitova) i int (32 bita)). Pretpostavite da su koeficijenti tipa *short* globalno dostupni i inicijalizirani na točne vrijednosti prije prvog poziva funkcije. Stanja filtra s1, s2, s3 i s4 su također globalno dostupne varijable tipa *short* te su inicijalizirana na 0 prije prvog poziva funkcije. Prototip funkcije neka je *short obrada_integer(short xn);*

Napomena: Iako se radi o IIR filtru koji se može ostvariti na više načina, implementacija mora
odgovarati slici.



Rješenje:

```
short obrada_integer (short xn){
    long rez=0;
    short novo_st;
    short izl;

    rez=-a1*s3*2 -a2*s4*2;  // zbrojimo desnu stranu (množenje s 2 ->
                               emul. frakt. množenja)

    rez=rez+0x8000; // emulacija frakt. množenja
    novo_st=rez>>16;

    rez=xn*b0*2 +s1*b1*2 +s2*b2*2; //zbrojimo lijevu stranu
    rez=rez+0x8000;
    novo_st+=rez>>16; // to sve zajedno zbrojimo

    s2=s1; //upis u sklopove za kašnjenje
    s1= xn;

    s4=s3;
    s3= novo_st;

    return novo_st;
}
```

10.

Otkrijte kako treba podesiti registre I0, L0 odnosno M0 adresnog generatora (DAG) da bi čitali odnosno mijenjali memorijske lokacije sljedećim redoslijedima. Također, napišite 3 adrese koje bi slijedile iza navedenih.

Rješenje (boldano je zadano u zadatku)**:**

ADR1	ADR2	ADR3	ADR4	ADR5	ADR6	ADR7	I0	L0	M0
100	**103**	**100**	**103**	100	103	100	100	6	3
100	**103**	**106**	**102**	105	101	104	100	7	3
100	**103**	**102**	**101**	100	103	102	100	4	3
100	**105**	**101**	**106**	102	107	103	100	9	5
101	**104**	**107**	**103**	106	102	105	101	7	3

Neko objašnjenje (kolega karlo5):

Potrebno je odrediti registre: I, L i M.

I je adresa koju DAG daje na sabirnicu.
L je širina odnosno broj mogućih elemenata u cirkularnom bufferu.
M je broj za koji se pomičem kad prolazimo kroz buffer.

I0 je početna adresa pa je prepisujemo, gledamo koju vrijednost trebamo dodati na prvu adresu da dobijemo drugu i to je M.

E sad za L. Cirkularni buffer je zatvoren krug, odnosno iza neke adrese vraćamo se na početnu. I sad gledamo nakon koje to adrese dodavanjem sadrzaja M registra se vraćamo na tu adresu koja je manja od trenutne.

Npr. za drugi red tablice:
100 103 106 102

Početna adresa je 100 --> I0=100, 103-100=3 --> M=3,

adrese idu: 100, 100+3=103, 103+3=106, i onda dalje 100,101,102 da se dobije 102.

Dakle, posljednja adresa je 106. 100 - 106 = 7 adresa, L=7.

Još objašnjenja, kolega bl4st3r:

Za prvi redak, očito je da je I0 = 100, M0 = 3 jer je pomak između Addr2 i Addr1 = 3. E sad kak ja to radim. Raspisujem adrese od 100 pa nadalje dok ne nabodem ili intuicijom ne zaključim da mi 'kruženje' odgovara. Za taj prvi slučaj npr.:

100

101

102

103

ovo ti nije dosta, jer je slijed 100 -> 103 -> 102 itd. Očito 'fali' još adresa:

100

101

102

103

104

ovo isto nije dosta, slijed je 100 -> 103 -> 101 -> 104.

Ali, je u 'adresnom spremniku':

100

101

102

103

104

105,

onda je slijed: 100 -> 103 -> 100 -> 103 -> 100 itd. L0 je dakle 6

Ergo, sam gledaš kolika ti mora biti duljina spremnika da se kod premotavanja nabode prava adresa.

Slična šema je za 4 red, adrese su

100

101

102

103

104

105

106

107

108

I još jedno, kolegica fucaboy's girl:

Ja radim ovak:

I0 i M0 stoje, al ja L0 odredim ovako:

- nađem prve dvije adrese koje su oblika tako da je prvo manja pa veća (znači negdje se prevrtilo preko spremnika), npr to je ADR3=106 i ADR4=102 u 2. redu

- formula (ne znam otkud sam ju izvukla) glasi: nova_adresa = (stara_adresa+M0-I0)%L0 + I0

- uvrstim sve, dobijem 102 = (106+3-100)%L0 + 100 -> 2=9%L0 iz čeg se vidi da je L0=7

11.

Simulirajte rad zadanog C programa. U tablicu unesite vrijednosti vektora *delay* i *izlbuf* nakon svakog poziva funkcije *obrada_float*.

```
#include <ccblkfn.h>
#define L 3
#define Nsamp 4
float coeffs[] = {-2, 1, 3, 4, 5};
float signal[] = {-1, 1, 2, 4, 5, 7};
float izlbuf[Nsamp] = {0};
float delay[L] = {0};
int idx_coeffs;
int idx_izlbuf = 0;
int idx_delay = 0;

void obrada_float(float uzorak){
    float acc = 1;
    int i;
    delay[idx_delay] = uzorak;
    idx_delay = circindex(idx_delay, 1, L);
    idx_coeffs = 2;
    for (i = 0; i < L; ++i){
        acc += coeffs[idx_coeffs] * delay[idx_delay];
        idx_delay = circindex(idx_delay, 1, L);
        idx_coeffs = circindex(idx_coeffs, 1, L - 1);
    }
    izlbuf[idx_izlbuf++] = acc;
}

int main(void){
    int j;
    for (j = 0; j < Nsamp; ++j){
        obrada_float(signal[j]);
    }
return 0;
}
```

Rješenje:

	delay[0]	delay[1]	delay[2]	izlbuf[0]	izlbuf[1]	izlbuf[2]	izlbuf[3]
j=0	-1	0	0	3	0	0	0
j=1	-1	1	0	3	-2	0	0
j=2	-1	1	2	3	-2	-5	0
j=3	4	1	2	3	-2	-5	-2

Djelomično objašnjenje s fer2 (kolega bl4st3r):

Vrlo jednostavno, funkcija circindex je praktički modulo operator, samo na nižoj razini upregne one cirkularne spremnike.

Ako je inicijalno idx_coeffs = 2, onda

idx_coeffs = circindex(idx_coeffs, 1, L - 1) praktički radi ovo:

idx_coeffs = (idx_coeffs + 1) % (L - 1).

I di je problem. idx_coeffs je inicijalno 2, pa je novi novi idx_coeffs:

idx_coeffs = (2 + 1) % (3 - 1) = 1 (3 modulo 2 jelte). Nakon toga počinje alterniranje 0, 1 itd.

Ergo, najobičnija modulo operacija.