**1.** U memoriji inicijalizirati četiri riječi te ih učitati u registre. Također, inicijalizirati četiri okteta koje je potrebno pročitati i staviti na stog.

.equ adresa, 0x2000 /\* adresa bloka u memoriji \*/

.equ stack\_top, 0x1000 /\* vrh stoga na adresi 0x1000\*/

.global \_start

.text

\_start: /\* Početak programa \*/

MOV sp, #stack\_top /\* postavlja se stack pointer na vrh stoga \*/

adr r0, pod /\* stavlja se u ro adresa podataka \*/

LDR r1, [r0], #4 /\* u r1 prvi podatak iz memorije 0x000F \*/

LDR r2, [r0], #4 /\* u r2 drugi podatak iz memorije 0x00FF \*/

LDR r3, [r0], #4 /\* u r3 treci podatak iz memorije 0x0FFF \*/

LDR r4, [r0], #4 /\* u r4 cetvrti podatak iz memorije 0xFFFF \*/

LDRB r5, [r0], #1 /\* u r6 prvi byte podataka 0x01 \*/

STR r5, [sp], #-4 /\* na stog se stavlja r5 \*/

LDRB r5, [r0], #1

STR r5, [sp], #-4

LDRB r5, [r0], #1

STR r5, [sp], #-4

LDRB r5, [r0], #1

STR r5, [sp], #-4

STR r4, [sp], #-4

stop:

B stop /\* program end and get into endless loop \*/

pod:

.word 0x000f, 0x00ff, 0x0fff, 0xffff

.byte 0x01, 0x02, 0x03,0x04

.end

**2.** Ucitati broj iz memorije te zamijeniti prvih i zadnjih 16 bita. Koristiti stog za prijenos parametara

.equ stack\_top, 0x1000 /\* Define \*/

.global \_start

.text

\_start:

ADR r0, pod /\*Adreasa podataka u r0\*/

LDR r1, [r0] /\* Broj u r1 \*/

MOV sp, #stack\_top

STR r1, [sp], #-4 /\* Stavi r1 na stog i oduzmi sp \*/

BL okreni /\* Skoci na funkciju \*/

LDR r2, [sp] /\* 0x56781234 stavi u u r2 \*/

STR r2, [r0] /\* Vrati r2 natrag u memoriju \*/

stop:

B stop

okreni:

LDR r1, [sp, #4]! /\* Dohvati podatak sa stoga. Prije dohvata zbroji sp \*/

MOV r2, r1, LSL#16 /\* LSB 16 => MSB 16 \*/

MOV r3, r1, LSR#16 /\* MSB 16 => LSB 16 \*/

ADD r1, r3, r2 /\*Zbroji ta 2 dijela \*/

STR r1, [sp, #-4]! /\* Stavi podatak na stog, ali nemoj prepisat r1 \*/

MOV PC, LR

pod:

.word 0x12345678

.end

**VAŽNO - 3 načina adresiranja**

* [r0], #x - S adrese r0 uzmi podatak i napravi nešto s njim, zatim r0 = r0 + x
* [r0, #x] - S adrese r0+x uzmi podatak i napravi nešto s njim. r0 se ne minja. Ovo je klasično bazno adresiranje.
* [r0, #x]! – Predindeksiranje. Prvo je: r0= r0 + x. Zatim s nove r0 uzmi podatak i napravi nešto.

**3.** Pročitati iz memorije riječ od 32 bita. Napisati funkciju koja ce zamijeniti dva bajta u sredini. Predati parametre preko stoga.

.global \_start

.text

\_start:

ADR r0, pod

LDR r1, [r0]

MOV sp, #0x1000

STR r1, [sp], #-4 /\* Stavi podatak na stog \*/

BL okreni /\* Poziv procedure \*/

LDR r2, [sp], #4 /\* Dohvati novu vrijednost sa stoga \*/

LDR r1, [sp] /\* Dohvati originalni r1 sa stoga i stavi ga u 1\*/

STR r2, [r0] /\* Spremi novu vrijednost na stog \*/

stop:

B stop

okreni:

LDR r1, [sp, #4]! /\* Ucitaj podatak sa stoga pomocu predindeksiranja \*/

MOV r2, #0xFF /\* U r2 se nalazi 0x0000 00ff \*/

MOV r3, r2, LSL #24 /\* U r3 se nalazi 0xff00 0000 \*/

ORR r2, r2, r3 /\* U r2 se nalazi 0xff00 00ff \*/

AND r3, r2, r1 /\* U r3 se nalazi broj 0x1200 0078 \*/

MOV r2, #0xFF /\* Ponovno iskoristimo r2 za maske \*/

MOV r2, r2, LSL #16 /\* U r2 se nalazi 0x00ff 0000 \*/

AND r4, r2, r1 /\* U r4 se nalazi broj 0x0034 0000 \*/

MOV r4, r4, LSR #8 /\* U r4 se nalazi broj 0x0000 3400 \*/

MOV r2, r2, LSR #8 /\* U r2 se nalazi 0x0000 ff00 \*/

AND r5, r2, r1 /\* U r5 se nalazi broj 0x0000 5600 \*/

MOV r5, r5, LSL #8 /\* U r5 se nalazi broj 0x0056 0000 \*/

ADD r1, r3, r4 /\* U r1 se nalazi broj 0x1200 3478 \*/

ADD r1, r1, r5 /\* U r1 se nalazi broj 0x1256 3478 \*/

STR r1, [sp, #-4]! /\* Stavi r1 na stog, ali BEZ prepisivanja stare vrijednosti \*/

MOV PC, LR

pod:

.word 0x12345678

.end

**4.** U memoriji sustava nalazi se blok 32-bitnih podataka nepoznate duljine u formatu dvojnog komplementa. Blok podataka počinje s podatkom koji se nalazi na adresi 0x1000 a završava sa podatkom 0xffffffff. Napisati program za koji će zbrojiti sve podatke iz bloka. Koristiti potprogram za zbrajanje, a parametre prenijeti preko stoga. Zbroj mora biti u 64-bitnom formatu dvojnog komplementa. Zbroj treba spremiti od adrese 0x2000 u big endiannu.

.global \_start

.text

\_start:

/\*stavljanje testnih podataka u memoriju\*/

ADR r0, pod

MOV r1, #0x1000 /\* Adresa bloka gdje trebaju bit podaci u memoriji \*/

punjenje\_bloka:

LDR r2, [r0], #4 /\* Uzmi podatak sa adrese r0 \*/

STR r2, [r1], #4 /\* Stavi ga na adresu r1 \*/

CMP r2, #0xFFFFFFFF /\* Kad dodje do 0xffff ffff stani\*/

BNE punjenje\_bloka

/\*ucitavanje prvog broja\*/

MOV sp, #0x1000

MOV r4, #0x1000

LDR r0, [r4],#4

MOV r1, #0

CMP r0, #0

BGE glavna\_petlja

MVN r1, r1

glavna\_petlja:

LDR r2, [r4],#4

MOV r3, #0

CMP r2,#0

BGE r1\_pozitivan

MVN r3, r3

r1\_pozitivan:

/\*spremanje maina\*/

STR r4, [sp],#-4

STR r3, [sp],#-4

STR r2, [sp],#-4

ADD sp, sp, #-8

/\*parametri+funkcija+povrat\*/

STR r3, [sp],#-4

STR r2, [sp],#-4

STR r1, [sp],#-4

STR r0, [sp],#-4

BL zbroj

LDR r0, [sp,#4]!

LDR r1, [sp,#4]!

/\*main restore\*/

LDR r2, [sp,#4]!

LDR r3, [sp,#4]!

LDR r4, [sp,#4]!

CMP r2,#0xFFFFFFFF

BNE glavna\_petlja

/\*spremanje rezultata u big endianu\*/

spremanje:

MOV r4, #0x2000 /\*adresa za pohranu\*/

MOV r2, r1

r1\_store: /\*ROR rotira bitove udesno\*/

MOV r1, r1, ROR #24 /\*dovedemo prvi bajt na zadnje mjesto\*/

STRB r1, [r4],#1 /\*i spremimo ga\*/

CMP r1, r2 /\*ponavljamo dok ne spremimo sve bajtove\*/

BNE r1\_store

MOV r2, r0

r0\_store:

MOV r0, r0, ROR #24

STRB r0, [r4],#1

CMP r0, r2

BNE r0\_store

stop:

B stop

zbroj:

LDR r0, [sp, #4]!

LDR r1, [sp, #4]!

LDR r2, [sp, #4]!

LDR r3, [sp, #4]!

ADDS r0, r0, r2 /\*pribrajamo donje bitove u r0\*/

ADC r1, r1, r3 /\*pribrajamo gornje bitove+carry bit u r3\*/

STR r0, [sp,#4]

STR r1, [sp,#8]

MOV PC, LR

pod:

.word 0x8FF00000, 0x8FF05500, 0x8F1C00C0, 0x8AA0A000, 0x8FA00106, 0xffffffff

.end

**5.** arm7 koristi little endian za pohranu podataka u memoriji. To znači da se najznačajniji bajt nalazi na najnižoj adresi, a najmanje značajan bajt na najvišoj adresi. Tako će se podatak koji u registru ima oblik 0x12345678, spremiti kao78 56 34 12, i obrnuto.

Ako se koristi little endian logika, 05 00 u memoriji je zapravo broj 0x0005. Kad se to prebaci iz memorije u registar imat će takav oblik. To pošaljemo funkciji koja izračuna 2'k. Dobije se 0xFFFB. Kad bi ga takvog išli spremiti, u memoriji bi se bajtovi pohranili obrnutim redoslijedom FB FF (little endian). Zato treba prvo zamijeniti bajtove da se dobije FB FF. Nakon spremanja u memoriju to je FF FB (big endian)

.global \_start

.equ stack\_top, 0x1000

.text

\_start:

MOV sp, #stack\_top

ADR r0, podaci

/\* Stvaranje r10 = 0xffff koji nam sluzi za ispitivanje kraja petlje\*/

MOV r9, #0xFF /\* U r9 stavi 0xff \*/

ORR r10, r9, r9, LSL #8 /\* U r10 stavi 0xffff \*/

ORR r10, r10, r10, LSL #16 /\* U r10 stavi 0xffffffff \*/

petlja:

LDRSH r1, [r0], #2 /\* Ucitaj podatak u r1 \*/

CMP r1, r10 /\* Usporedi broj i 0xffffffff (znak za kraj) \*/

BEQ stop /\* Ako su isti zavsi program \*/

STRH r1, [sp], #-4 /\* Stavi podatak na stog \*/

STR r0, [sp], #-4 /\* Stavi registar r0 na stog \*/

ADD sp, sp, #-4 /\* Ostavi slobodnog mjesta za rezultat \*/

STRH r1, [sp], #-4 /\* Stavi podatak ponovo na stog \*/

BL procedura /\* Poziv procedure \*/

LDRSH r2, [sp, #4]! /\* Stavi obradjeni broj u r2\*/

LDR r0, [sp, #4]! /\* Vrati r0 \*/

LDRH r1, [sp, #4]! /\* Vrati r1 \*/

STRH r2, [r0, #-2] /\* Spremanje rezultata u memoriju \*/

B petlja /\* Ponovi petlju \*/

procedura:

LDRSH r0, [sp, #4]! /\* Dohvati podatak koji treba invertirat \*/

MVN r0, r0 /\* Invertiranje bitova \*/

ADD r0, r0, #1 /\* Kad dodamo 1, dobijemo dvojni komplement \*/

/\* Zamjena MSB i LSB bajtova. Pazi, format broja je 0xabcd pa je lako \*/

MOV r1, r0, LSR #8 /\* r1 = 0x00 MSB \*/

MOV r0, r0, LSL #8 /\* r0 = 0xLSB 00 \*/

ORR r0, r0, r1 /\* Zbroji obrnute brojeve \*/

STRH r0, [sp, #4] /\* Stavi obradjeni broj na slobodno mjesto na stogu \*/

MOV PC, LR

stop:

B stop

podaci:

.hword -5, 5, 10, 234, -541, 0xFFFF

.end

**VAŽNO – Općenito zamjena bilo kojeg broja u zapis sa dvojnim komplementom**

LDR / LDRSH r0, **[sp, #4]!** / **[sp]** / **[sp], #4** /\* Dohvati podatak koji treba invertirat \*/

MVN r0, r0 /\* Invertiranje bitova \*/

ADD r0, r0, #1 /\* Kad dodamo 1, dobijemo dvojni komplement \*/

**VAŽNO – Petlje za učitavanje brojeva**

1. Prvo kreirati broj (tj. znak za prekid) koji označava kraj petlje (pomoću maski, LSL i ORR)
2. Svaki učitani broj usporedit sa znakom za prekid

**Primjer.**

/\* kreiranje znaka za prekid 0xFFFF FFFF pomocu jednog registra \*/

MOV r2, #0xFF /\* U r1 stavi 0x0000 00FF \*/

ORR r2, r2, r2, LSL #8 /\* U r2 stavi 0x0000 FFFF \*/

ORR r2, r2, r2, LSL #16 /\* U r2 stavi 0xFFFF FFFF \*/

petlja\_za\_upis:

LDR r1, [r0], #4 /\* Ucitaj podatak u r1 \*/

CMP r1, r2 /\* Usporedi broj i znak za kraj \*/

BEQ stop /\* Ako su isti zavsi program \*/

...

B petlja\_za\_upis

stop

B stop

**6.** Isto kao 5. zadatak ali big endian. Ako cemo koristiti big endian logiku, 05 00 u memoriji stvarno oznacava broj 0x0500. Ali kad se to prebaci iz memorije u registar imat će oblik 0x0005. To pošaljemo funkciji. Ona prvo okrene bajtove da dobijemo 0x0500. Zatim se računa 2'k. Dobije se FB 00. Kad ovo spremimo u memoriju, pohranit će se kao 00 FB (default je little-endian).

.global \_start

.equ stack\_top, 0x1000

.text

\_start:

MOV sp, #stack\_top

ADR r0, podaci

petlja:

LDRSH r1, [r0], #2 /\*u r1 podatak\*/

STRH r1,[sp],#-4 /\* Spremanje stanja r0 i r1, tj. spremanje maina\*/

STR r0,[sp],#-4

ADD sp, sp, #-4

STRH r1,[sp],#-4 /\* Stavi r1 na stog s kojim ces radit\*/

BL procedura

LDRSH r2, [sp,#4]! /\* Vrati izmjenjenu vrijednost \*/

LDR r0,[sp, #4]! /\* Vrati stanje main-a \*/

LDRH r1,[sp, #4]!

STRH r2, [r0, #-2] /\*spremanje rezultata\*/

CMP r2,#0x0001 /\* 0xFFFF se prvo okrene (nema promjene) pa se \*/

BEQ stop /\*racuna dvojni komplement koji iznosi #0x0001\*/

B petlja

procedura:

LDRSH r0, [sp, #4]! /\*Ucitaj podatak \*/

/\*Zamjena bajtova\*/

AND r2, r0, LSR#8 /\*r2 = 0xLSB 00\*/

MOV r0, r0, LSL#8 /\*r1 = 0x00 MSB\*/

ORR r0, r0, r2 /\* Zbroji \*/

MVN r0, r0 /\* invertiranje bitova \*/

ADD r0, r0, #1 /\*+1, da se dobije 2. komplement\*/

STRH r0, [sp, #4] /\* Spremi rezultat \*/

MOV PC, LR

stop:

B stop

podaci:

.hword -5, 5, 10, 234, -541, 0xFFFF

.end

**7.** Napisati program koji filtrira ulazni niz 16 bitovnih podataka nepoznate duljine spremljene počevši od adrese 0x1000. Niz završava 0xFFFF. Rezultat se sprema počevši od adrese 0x2000. Funkcija filtra je y[n] = 0,5 x[n] + 0,125 x[n-1] - 0,25 x[n-2] + 0,25 x[n-3]. Funkciju filtra napisati kao potprogram i parametre kao i rezultat prenositi preko stoga.

.equ vrh\_stoga, 0x1000

.equ novi\_podaci, 0x2000

.global \_start

.text

\_start:

ADR r0, pod

MOV r1, #0x1000

punjenje:

LDRH r2, [r0], #2 /\* Ocitaj podatak \*/

STRH r2, [r1], #2 /\* Spremi ga na adresu 0x1000 (kao tu pocinju podaci) \*/

CMP r2, #0x0 /\* Usporedi sa krajem \*/

BNE punjenje /\* Ako nisi na kraju, nastavi punit \*/

MOV sp, #vrh\_stoga /\* inicijalizacija stoga\*/

MOV r6, #novi\_podaci /\* U r6 ce se stavljat rezultati\*/

MOV r0, #0x1000 /\* Na ovoj adresi pociinju podaci \*/

LDRH r1, [r0], #2 /\* Ucitaj prvi podatak \*/

MOV r2, #0 /\* Inicijalizacija r2, r3 i r4 na nula \*/

MOV r3, #0

MOV r4, #0

petlja:

ADD sp, sp, #-2 /\* Prvo ostavi mjesto na stogu za rezultat \*/

STRH r4, [sp], #-2 /\* Postavljanje parametara funkcije na stog \*/

STRH r3, [sp], #-2

STRH r2, [sp], #-2

STRH r1, [sp], #-2

BL filter /\* Pozovi funkciju \*/

LDRH r5, [sp, #2]! /\* U r5 stavimo rezultat \*/

/\*pohrana, priprema za sljedecu iteraciju\*/

STRH r5, [r6], #2

MOV r4, r3 /\* x[n-3] postaje prijasnji x[n-2] \*/

MOV r3, r2 /\* x[n-2] postaje prijasnji x[n-1] \*/

MOV r2, r1 /\* x[n-1] postaje prijasnji x[n] \*/

LDRH r1, [r0], #2 /\* x[n] citamo iz memorije \*/

CMP r1, #0 /\* Ako je r1 tj. zadnji clan nula prekini \*/

BNE petlja /\* Za njega ne tribas racunat y[n] \*/

stop:

B stop

filter:

LDRH r7, [sp, #2]! /\* Uzimanje parametra sa stoga \*/

LDRH r8, [sp, #2]!

LDRH r9, [sp, #2]!

LDRH r10, [sp, #2]!

MOV r7, r7, LSR#1 /\* r1 = r1 / 2 \*/

ADD r7, r7, r8, LSR#3 /\* r1 = r1 + r2 / 8 \*/

SUB r7, r7, r9, LSR#2 /\* r1 = r1 - r3 / 4 \*/

ADD r7, r7, r10, LSR#2 /\* r1 = r1 + r3 / 4 \*/

STRH r7, [sp,#2] /\* Stavljanje rezultata na stog \*/

MOV PC, LR /\* Povrat \*/

pod:

.hword 2, 8, 4, 32, 16, 12, 7, 8, 9, 0x0

.end

**LOGIKA ZADATKA:** bitno je naglasit da zadatak ide sljedećim smjerom

1) Prvo je x[n] = 2, x[n-1], x[n-2], x[n-3] su nula

2) Zatim je x[n] = 8, x[n-1] =2, ostali su nula

3) Zatim je x[n] = 4, x[n-1]= 8, x[n-2] = 2, x[n-3] = 0

Itd.

Kad se dođe do kraja, tj. postane x[n] = 0x0 (tj nula) program se odmah prekida tj. za to se ne računa. Ovo je zapravo preinaka zadatka jer bi kraj trebao bit 0xffff, a to se može postić korištenjem petlji u 5. zadatku.

**TIPOVI GRANANJA**

* BNE - Branch if Not Equal to
* BEQ - Branch if Equal to
* BLT - Branch if Less than
* BLE - Branch if Less or Equal to
* BGT - Branch if Greater than
* BGE - Branch if Greater or Equal to
* BMI - Branch if Negative
* BPL - Branch if Positive or Zero

**8.** U memoriji od adrese 0x1000 nalazi se blok sa 32-bitnim podacima u formatu dvojnog komplementa. Blok je nepoznate duljine, ali se zna da je zaključen podatkom F000 0000. Napišite program za koji treba u svim pozitivnim podacima u bloku zamijeniti dvije najniže skupine od 4 bita. Negativni podaci se ne mijenjaju.

.global \_start

.text

\_start:

ADR r0, podaci

MOV r1, #0x1000

punjenje:

LDR r2, [r0], #4 /\* Citaj podatke iz memorije \*/

STR r2, [r1], #4 /\* Spremaj ih na lokaciju 0x1000 \*/

CMP r2, #0xF0000000

BNE punjenje

MOV r0, #0x1000 /\* Stavi podatke u r0 \*/

petlja:

LDR r1, [r0], #4 /\* Ucitaj podatak \*/

CMP r1, #0xF0000000 /\* Ako je doslo do zadnjeg podatka \*/

BEQ stop /\* prekini izvrsavanje petlje \*/

CMP r1, #0 /\* Ako je broj negativan \*/

BLT petlja /\* nemoj radit nikakve promjene \*/

MOV r2, #0x0F /\* Maska r2 = 0x0000 000f \*/

AND r3, r1, r2 /\* r3 = r2 AND r1 \*/

MOV r2, #0xF0 /\* Maska r2 = 0x0000 00f0 \*/

AND r4, r1, r2 /\* r4 = r2 AND r1 \*/

EOR r5, r1, r3 /\* pomocu 2 ekskluzivno ili naredbe \*/

EOR r5, r5, r4 /\* izdvoji prva 3 bajta\*/

MOV r3, r3, LSL #4 /\* pomakni zadnja 4 bita lijevo \*/

MOV r4, r4, LSR #4 /\* pomakni zadnja 4 bita udesno \*/

ADD r1, r5, r3 /\* Formiranje nove vrijednosti \*/

ADD r1, r1, r4

B petlja

stop:

B stop

podaci:

.word 0x12345678, 0xf2312341, 0x22222256, 0x03451125, 0xeabcdefa, 0xF0000000

.end

**9.** U memoriji sustava nalazi se blok 32-bitnih podataka u formatu dvojnog komplementa.Blok podataka počinje s podatkom koji se nalazi na adresi 0x1000 a završava sa podatkom koji se nalazi na adresi 0x1100. Napisati program za koji će zbrojiti sve podatke iz bloka. Zbroj mora biti u 64-bitnom formatu dvojnog komplementa. Zbroj treba spremiti od adrese 0x2000 u big endiannu.

**big-endian: najznačajniji bajt sprema se na najnižu adresu**

**little-endian: najmanje značajan bajt sprema se na najnižu adresu (default za ARM)**

.global \_start

.text

\_start:

MOV r0, #0x1000 /\* Adresa bloka u memoriji u r0 \*/

LDR r1, pod /\* Ubacimo adresu brojeva u r1 \*/

punjenje:

STR r1, [r0], #4 /\* Spremamo r1 u memoriju \*/

ADD r1, r1, #-50 /\* Stvori neku novu vrijednost r1 \*/

CMP r0, #0x1100 /\* Ispitaj jesmo li dosli do \*/

BLE punjenje /\* memorijske lokacije 0x1000 \*/

/\* Zbrajamo po principu r1r0 + r3r2 gdje se r1 i r3 postavljani od strane nas u jedinice ili nule \*/

MOV r4, #0x1000 /\* Adresa bloka u memoriji u r4 \*/

LDR r0, [r4], #4 /\* Ucitavamo donjih 32 bita prvog broja \*/

MOV r1, #0 /\* gornja 32-bita postavimo u 0 \*/

CMP r0, #0 /\* Ako je r0 pozitivan gornjih 32 bita \*/

BGE zbrajanje /\* mora bit 0. Inace moraju bit 1 \*/

MVN r1, r1 /\* Ako su 1 radi se invertiranje bitova \*/

zbrajanje:

LDR r2, [r4], #4 /\* Ucitat donjih 32-bita drugog broja \*/

MOV r3, #0 /\* Postavit gornje bitove drugog broja u 0 \*/

CMP r2, #0 /\* Ako je r2 pozitivan gornjih 32 bita \*/

BGE gornji\_dio\_pozit /\* mora bit 0. Inace moraju bit 1 \*/

MVN r3, r3 /\* Ako su 1 radi se invertiranje bitova \*/

gornji\_dio\_pozit:

ADDS r0, r0, r2 /\* Pribrajamo donje bitove u r0 \*/

ADC r1, r1, r3 /\* Pribrajamo gornje bitove+carry bit u r1 \*/

CMP r4, #0x1100 /\* Provjera je li se doslo do kraja \*/

BLE zbrajanje /\* Ako nije zbroji sljedeci broj \*/

/\* Spremanje gornja 32-bita u big endianu \*/

MOV r4, #0x2000 /\* U r4 adresu bloka za pohranu podataka \*/

MOV r2, r1 /\* U r2 gornja 32 bita \*/

gornji\_dio\_store: /\* ROR rotira bitove udesno \*/

MOV r1, r1, ROR #24 /\* dovedemo prvi bajt na zadnje mjesto \*/

STRB r1, [r4], #1 /\* i spremimo ga \*/

CMP r1, r2 /\* Ponavljamo dok ne spremimo sve bajtove \*/

BNE gornji\_dio\_store /\* Odnosno dok se r1 ne vrati u prvobitni polozaj \*/

/\* Spremanje donja 32-bita u big endianu \*/

MOV r2, r0

donji\_dio\_store:

MOV r0, r0, ROR #24 /\* Rotacija udesno \*/

STRB r0, [r4], #1 /\* Spremi bajt \*/

CMP r0, r2 /\* Usporedi je li se broj vratio u prvobitni \*/

BNE donji\_dio\_store /\* polozaj kroz niz rotacija. Ako je, stani \*/

stop:

B stop

pod:

.word 0x8FF00000

.end

**VAŽNO - Spremanje broja u big-endian formatu**

MOV r2, r1 /\* U r1 je broj. Spremi kopiju u r2 \*/

spremanje:

MOV r1, r1, ROR #24 /\* ROR rotira bitove u desno \*/

STRB r1, [r0], #1 /\* Spremimo rotirani bajt broja u memoriju \*/

CMP r1, r2 /\* Ponavljamo dok ne spremimo sve bajtove, \*/

BNE spremanje /\* tj. dok se r1 ne vrati u prvobitni polozaj \*/

**ISTAKNUTE NAREDBE**

* MVN rx, rx – radi invertiranje bitova u registru rx
* ADDS – ADD sa opcijom da se eventualni pretek čuva u dodatnim zastavicama.
* ADC – zbrajanje s pretekom. Pretek se čuva u zastavici C (Carry).

**10.** U bloku memorije koji počinje od adrese 0x1000, nalaze se 16-bitni brojevi u dvostrukom komplementu. Oni su zapisani u normalnom FRISC-ovom little-endian redoslijedu. Nije poznato koliko blok sadrži podataka, ali se zna da je zaključen 0xFFFF.

a. Sve podatke u bloku treba pretvoriti u 2'k format, ali zapisane u big-endiannu. Nakon transformacije podataka u bloku, zaključni podatak u bloku treba biti 8000 (16).

b. Za pretvorbu podataka upotrijebiti potprogram kojemu se kao parametar šalje 16-bitni podatak u registru R3 (u nižih 16 bitova registra). Potprogram pretvara podatak i vraća ga pozivatelju u registru R3. Rezultat mora imati "okrenute" bajtove tako da bude u big-endianu i da ga pozivatelj može izravno spremiti u memoriju.

**Isti kao 5. zadatak, ali bez stoga.**

.global \_start

.text

\_start:

ADR r0, podaci

MOV r9, #0xFF /\* r9 = 0x0000 00ff \*/

ORR r9, r9, r9, LSL #8 /\* r9 = 0x0000 ffff \*/

ORR r9, r9, r9, LSL #16 /\* Maska r9 = 0xffff ffff \*/

petlja:

MOV r1, #0 /\* Resetiraj r1 \*/

LDRSH r1, [r0], #2 /\* U r1 ucitaj podatak \*/

CMP r1, r9 /\* Usporedi r1 sa uvjetom zaustavljanja \*/

BEQ stop /\* Ako je r1 = 0xffff ffff stani \*/

MOV r3, r1 /\* Stavi kopiju u r3 \*/

BL procedura /\* Poziv procedure \*/

STRH r3, [r0, #-2] /\* Spremanje rezultata \*/

B petlja /\* Sljedeci broj... \*/

procedura:

MVN r3, r3 /\* Invertiranje bitova te dodavanje \*/

ADD r3, r3, #1 /\* +1 => dvojni komplement \*/

MOV r2, r3, LSL#8 /\* Pomak LSB za 1 bajt ulijevo \*/

MOV r3, r3, LSR#8 /\* Pomak MSB za 1 bajt udesno \*/

ORR r3, r3, r2 /\* Zbrajanje \*/

MOV PC, LR /\* Kraj procedure \*/

stop:

B stop

podaci:

.hword -5, 5, 10, 234, -12, 0xFFFF

.end

**11.** Napisati potprogram koji množi dva broja. Parametre predati potprogramu preko stoga, te vratiti

rezultat isto preko stoga. Nakon završetka potprograma „očistiti“ stog.

.equ x, 9 /\* Prvi broj \*/

.equ y, 7 /\* Drugi broj \*/

.equ stack\_top, 0x1000 /\* Vrh stoga na adr 0x1000 \*/

.global \_start

.text

\_start:

MOV sp, #stack\_top /\* Vrh stoga na sp \*/

MOV r0, #x /\* Ucitavanje brojeva u registre \*/

MOV r1, #y

ADD sp, sp, #-4 /\* Za povratnu vrijednost \*/

STR r0, [sp], #-4 /\* Stavljanje parametara na stog \*/

STR r1, [sp], #-4

BL mnozenje /\* Poziv funkcije - Branch and Link \*/

LDR r5, [sp, #4]! /\* Povratna vrijednost \*/

stop:

B stop

mnozenje:

LDR r3, [sp, #4]! /\* Dohvat prvog operatora sa stoga \*/

LDR r4, [sp, #4]! /\* Dohvat drugog operatora sa stoga \*/

MUL r5, r3, r4 /\* Mnozenje \*/

STR r5, [sp, #4] /\* spremanje na rezervirano mjesto \*/

MOV PC, LR /\* Povrat \*/

.end

**VAŽNO – Upis brojeva u određeni memorijski blok**

ADR r0, podaci /\* Adresa trenutnih podataka \*/

MOV r1, #0x1000 /\* Adresa bloka u memoriji \*/

punjenje:

LDR r2, [r0], #4 /\* Dohvati broj \*/

STR r2, [r1], #4 /\* stavi ga u blok u mem. \*/

CMP r2, #0xUVJ\_ZAUST /\* Usporedi sa uvjetom zaustavljanja \*/

BNE punjenje

**12.** Napisati program koji će u bloku 32 bitovnih podataka nepoznate duljine sve negativne brojeve postaviti na početak bloka, a sve pozitivne na kraj. Blok je zaključen podatkom 00000000. Početna adresa bloka je 0x1000.

.global \_start

.text

\_start:

adr r0, podaci /\* Adresa trenutnih podataka \*/

MOV r1, #0x1000 /\* Adresa bloka u memoriji \*/

punjenje:

LDR r2, [r0], #4 /\* Dohvati broj \*/

STR r2, [r1], #4 /\* stavi ga u blok u mem. \*/

CMP r2, #0

BNE punjenje

MOV r0, #0x1000 /\* Postavi se na blok u mem. \*/

MOV r1, r0 /\* r1 je pokazivac na pocetak i bit cepomocni registar \*/

petlja:

LDR r2, [r0], #4 /\* Dohvati broj \*/

CMP r2, #0 /\* Usporedba sa nulom \*/

BEQ stop

CMP r2, #0 /\* Ako je broj pozitivan ajde na sljedeci \*/

BGT petlja

LDR r3, [r1] /\* Dohvati broj sa pocetka \*/

STR r2, [r1], #4 /\* Na pocetak spremi negativni broj i pomakni se sa pocetka \*/

STR r3, [r0, #-4] /\* Vrati se natrag pomocu baznog adresiranja i spremi broj \*/

B petlja

stop:

B stop

podaci:

.word -2, 1, -2, 10, -7, -6, -5, 14, 0x00000000

.end

**13.** Napisati program koji će bloku 32 bitovnih podataka nepoznate duljine izračunati srednju vrijednost. Blok je zaključen podatkom 00000000. Početna adresa bloka je 0x1000. Ovo je rijesno na isti nacin ko u onim lanjskim primjerima. Dakle, nista pametno, rezultat je cijeli broj.

.global \_start

.text

\_start:

adr r0, pod /\* Adresa trenutnih podataka \*/

MOV r1, #0x1000 /\* Adresa bloka u memoriji \*/

punjenje:

LDR r2, [r0], #4 /\* Dohvati broj \*/

STR r2, [r1], #4 /\* stavi ga u blok u mem. \*/

CMP r2, #0

BNE punjenje

MOV r0, #0x1000 /\* Pocetna adresa bloka \*/

MOV r2, #0 /\* Suma, inicijalizacija na 0 \*/

MOV r3, #0 /\* Brojac elemenata \*/

MOV r4, #0 /\* Rezultat dijeljenja \*/

ucitavanje:

LDR r1, [r0], #4 /\* Ucitaj broj \*/

CMP r1, #0 /\* Je li se doslo do kraja? \*/

BEQ dijeljenje /\* Ako je, ajde na dijeljenje \*/

ADD r2, r2, r1 /\* Suma = Suma + r2 \*/

ADD r3, r3, #1 /\* r3++ \*/

B ucitavanje

dijeljenje:

SUB r2, r2, r3 /\* Oduzmi r2 = r2 - r3 \*/

CMP r2, #0 /\* Usporedi r2 sa nulom \*/

BLT stop /\* Ako je r2 < 0 stani \*/

ADD r4, r4, #1 /\* Povecaj brojac r4 \*/

B dijeljenje

stop:

B stop

pod:

.word 3, 5, 12, 7, 35, 1, 0

.end

**14.** Napisati program koji filtrira ulazni niz 16 bitovnih podataka nepoznate duljine spremljene počevši od adrese 0x1000. Niz završava 0xFFFF. Rezultat se sprema počevši od adrese 0x2000. Funkcija filtra je y[n] = 0,6 x[n] + 0,25 x[n-1] + 0,05 x[n-2].

a. Funkciju filtra napisati kao potprogram i parametre kao i rezultat prenositi preko stoga.

/\*Buduci da ne znam kako radit sa float brojevima, ovdje je koristen filter y[n] = 3x[n] + 4x[n-1] + 5x[n-2]. Najveci dio koda otpada na spremanje i dizanje podataka sa stoga. Funkciji se salju 3 parametra, ona ih mnozi s koef, zbraja ih i vraca. Za podatke koji slijede iza koda radi ispravno.\*/

.equ vrh\_stoga, 0x1000

.equ novi\_podaci, 0x2000

.global \_start

.text

\_start:

MOV sp, #vrh\_stoga

MOV r5, #novi\_podaci

ADR r0, pod

LDRH r1, [r0], #2

MOV r2, #0

MOV r3, #0

petlja:

/\*spremanje maina i ostavljanje mjesta na stogu\*/

STRH r5, [sp], #-2

STRH r3, [sp], #-2

STRH r2, [sp], #-2

STRH r1, [sp], #-2

STRH r0, [sp], #-2

ADD sp, sp, #-2

/\*slanje parametara funkciji i poziv\*/

STRH r3, [sp], #-2

STRH r2, [sp], #-2

STRH r1, [sp], #-2

BL filter

/\*povrat rezultata i maina\*/

LDRH r4, [sp,#2]!

LDRH r0, [sp, #2]!

LDRH r1, [sp, #2]!

LDRH r2, [sp, #2]!

LDRH r3, [sp, #2]!

LDRH r5, [sp, #2]!

/\*pohrana, priprema za sljedecu iteraciju\*/

STRH r4, [r5],#2

MOV r3, r2 /\*x[n-2] postaje prijasnji x[n-1]\*/

MOV r2, r1 /\*x[n-1] postaje prijasnji x[n]\*/

LDRH r1, [r0], #2 /\*x[n] citamo iz memorije\*/

CMP r1,#0

BNE petlja

stop:

B stop

filter:

/\*load koeficijenata\*/

ADR r5,koef

LDR r0,[r5],#4

LDR r1,[r5],#4

LDR r2,[r5],#4

/\*racunanje i spremanje u r3\*/

LDRH r4, [sp, #2]! /\*uzimanje parametra sa stoga\*/

MLA r3, r0, r4 /\*r3=r0\*r4+r3\*/

LDRH r4, [sp, #2]!

MLA r3, r1, r4, r3

LDRH r4, [sp, #2]!

MLA r3, r2, r4, r3

/\*stavljanje na stog i povratak\*/

STRH r3, [sp,#2]

MOV PC, LR

pod:

.hword 1, 2, 3, 4, 5, 6, 7, 8, 9, 0x0000

koef:

.word 3,4,5

/\*.float 0.25, 0.6, 0.05, 0x00000000\*/

.end

**15.** Napisati program koji čita niz znakova – string (zero terminated string) proizvoljne dužine zapisan na adresi 0x1000 i koji pretvara sva velika slova u mala.

.global \_start

.text

\_start:

ADR r0, podaci

MOV r1, #0x1000

punjenje:

LDRB r2, [r0], #1 /\* Slova su u memoriju upisana kao BYTE \*/

STRB r2, [r1], #1

CMP r2, #0

BNE punjenje

MOV r0, #0x1000

petlja:

LDRB r1, [r0], #1 /\* Ucitaj slovo u r1 \*/

CMP r1, #0 /\* Provjera je li kraj niza \*/

BEQ stop /\* Ako je, skoci na stop \*/

CMP r1, #65 /\* Provjera je li znak veliko slovo \*/

BLT petlja /\* Velika slova su ASCII 65-90 \*/

CMP r1, #90 /\* Provjera gornje granice velikih slova \*/

BGT petlja /\* Ako je vece od 90 nije veliko slovo \*/

ADD r1, r1, #32 /\* malo\_slovo = veliko\_slovo + 32 \*/

STRB r1, [r0, #-1] /\* Pohrana znanak natrag u memoriju \*/

B petlja /\* Dohvati sljedeci znak \*/

stop:

B stop

podaci:

.byte 'A', 'b', 'C', 'd', 'E', 0x00000000

.end

**VAŽNO – TIPOVI PODATAKA**

* BAJTOVI – Razmak je #1, a deklariraju se kao .byte 'a', 'b', 0x22, 0x1, 0xff
* HALF-WORD – Razmak je #2, a deklariraju se kao .hword 0x1111, 0xffff, 0x2222, 0x22
* WORD – Razmak je #4, a deklariraju se kao .word 0x11111111, 0xffffffff, 0x0, 0x1234

**16.** U bloku memorije koji slijedi u području iza koda prebrojiti koliko ima neparnih podatka. Brojanje završava prvim podatkom 0.

.global \_start

.text

\_start:

adr r0, podaci /\* r0 je adresa podataka \*/

MOV r2, #0 /\* brojac r2 inicijalizirat na 0 \*/

petlja:

LDR r1, [r0], #4 /\* Podaci se prebacuju u r1 \*/

CMP r1, #0 /\* Jesmo li dosli do kraja? \*/

BEQ stop

MOV r3, #0x1 /\* Maska 0x0000 0001 u r3 \*/

AND r4, r3, r1 /\* Izvuci zadnji bit \*/

CMP r4, #0 /\* Ako je zadnji bit 0, onda nista \*/

BEQ petlja

ADD r2, r2, #1 /\* ELSE dio: Povecaj brojac za 1 \*/

B petlja

stop:

B stop

podaci:

.word -2, 1, -2, 10, -7, -6, -5, 14, 0

.end

**17.** U bloku memorije koji slijedi u području iza koda prebrojiti koliko ima podatka do prvog podatka 0.

.global \_start

.text

\_start:

adr r0, pod /\* U r0 adresa podataka \*/

MOV r1, #0 /\* Brojac \*/

petlja:

LDR r2, [r0], #4 /\* U r2 podatak \*/

CMP r2, #0 /\* Usporedi r2 sa nulom \*/

BEQ stop /\* r2 == 0 -> kraj \*/

ADD r1, r1, #1 /\* Inace uvecaj brojac \*/

B petlja /\* Skoci na petlja \*/

stop:

B stop

pod:

.word 0x000f, 0x00ff, 0x0fff, 0xffff, 0x0

.byte 0x01, 0x02, 0x03,0x04

.end**18.** Napišite program za dijeljenje dva cijela broja! Dijeljenje raditi po principu oduzimanja

.global \_start

.text

\_start:

ADR r0, pod

LDR r1, [r0], #4

LDR r2, [r0]

petlja:

SUB r2, r2, r3

CMP r2, #0

BLE stop

ADD r3, r3, #1

B petlja

stop:

b stop

pod:

.word 930, 68

.end

**19.** Napišite program za izračun drugog krojena iz cijelog broja prema sljedećem algoritmu:

(x + 1)2 = x2 + 2x + 1 neka je x = p = 0, tada je (x + 1)2 = q = 1  
tada se korijen može računati temeljem jednadžbe:  
 q = q + 2\*p + 1;  
 p = p + 1;  
 dok je q manje ili jednako zadanom broju

.global \_start

.text

\_start:

adr r3, pod /\* adresa podatka \*/

ldr r4, [r3] /\* U r4 broj kojeg korijenujemo \*/

mov r0, #0 /\* U r0 p \*/

mov r1, #1 /\* U r1 q \*/

petlja:

add r0, r0, #1 /\* Inkrementira se p \*/

mov r2, r0, LSL#1 /\* r2 = (p+1)\*2 \*/

add r2, r2, r1 /\* r2 = q + (p+1)\*2 \*/

add r1, r2, #1 /\* r1 = q + (p+1)\*2 + 1 \*/

cmp r1, r4 /\* Usporedi s brojem \*/

BLE petlja /\* Ako je r1<r4 nastavi petlju \*/

kraj:

b kraj

pod:

.word 6154

**20.** U memoriji od adrese 0x1000 nalazi se blok sa 32-bitnim podacima u formatu dvojnog komplementa. Blok je nepoznate duljine, ali se zna da je zaključen podatkom F000 0000. Napišite program za koji treba u svim podacima u bloku zamijeniti dvije najniže skupine od 4 bita (vidi sliku). Negativni podaci se ne mijenjaju.

četvrti bajt

treći bajt

drugi bajt

.global \_start

.org 0x00001000 /\* postaviti adresu podataka na 0x00001000 \*/

.data

.word 0x5a22115a, 0x11223345, -5, 0xF0000000

.org 0x00007000

\_start:

mov r0, #0xFF /\* r0 = 0x0000 00FF \*/

mov r0, r0, LSL #8 /\* r0 = 0x0000 FF00 \*/

orr r0, r0, #0xFF /\* r0 = 0x0000 FFFF \*/

mov r0, r0, LSL #8 /\* r0 = 0x00FF FF00 \*/

orr r0, r0, #0xFF /\* r0 = 0x00FF FFFF \*/

mov r0, r0, LSL #8 /\* r0 = 0xFF00 0000 \*/

mov r1, #0xF0 /\* r1 = 0x000000F0 \*/

mov r1, r1, LSL #24 /\* r1 = 0xF0000000 \*/

mov r2, #0x1000 /\* blok pod. 0x1000 u r2 \*/

petlja:

ldr r3, [r2], #4 /\* pročitati podatak u r3 \*/

cmp r3, r1 /\* usporedi s uvjetom zaustavljanje u r1 \*/

beq kraj

and r4, r3, #0xF0 /\* odaberi gornja 4 bita zadnjeg bayta \*/

mov r4, r4, LSR #4 /\* pomakni ih u desno za 4 mjesta \*/

and r5, r3, #0x0F /\* odaberi donja 4 bita zadnjeg bayta \*/

mov r5, r5, LSL #4 /\* pomakni ih u lijevo za 4 mjesta \*/

and r3, r3, r0 /\* Filtriraj dio koji se ne mijenja \*/

orr r3, r3, r4 /\* Stvori podatak sa zarotiranih zadnjih 8 bita \*/

orr r3, r3, r5

b petlja

kraj:

b kraj

.end

**21.** Napisati potprogram koji dijeli dva cijela broja koja se prenose preko stoga. Rezultat također vratiti preko stoga.

.equ stog, 0x1000

\_start:

adr r0, podaci

mov sp, #stog

ldr r1, [r0], #4

str r1, [sp, #-4]!

ldr r1, [r0], #4

str r1, [sp, #-4]!

bl dijeli

ldr r1, [sp], #4

kraj:

b kraj

dijeli:

ldr r1, [sp], #4

ldr r2, [sp], #4

mov r3, #0

b l1

l0: add r3, r3, #1

l1: subs r2, r2, r1

bgt l0

/\* rezultat na stog \*/

str r3, [sp, #-4]!

/\* vrati se u program \*/

mov pc, r14

podaci:

.word 2456, 52

**22.** Program u C-u prebaci u asembler:

**int** x = 20, y = 35, rez = 0;  
**if** (x < y)  
 rez = y –x;  
**else**  
 rez = x – y;

.global \_start

.text

\_start:

adr r0, pod /\* stavlja se u ro adresa podataka \*/

LDR r1, [r0], #4 /\* u r1 prvi podatak iz memorije 20 \*/

LDR r2, [r0], #4 /\* u r2 drugi podatak iz memorije 35 \*/

CMP r1, r2 /\*usporedba dvaju brojeva\*/

BGE y\_manji /\*ako je x veci ili jednak y\*/

SUB r3, r2, r1

B stop

y\_manji:

SUB r3, r1, r2

stop:

B stop /\* program end and get into endless loop \*/

pod:

.word 20, 35

.end