

UDŽBENICI SVEUČILIŠTA U ZAGREBU  
MANUALIA UNIVERSITATIS STUDIORUM ZAGRABIENSIS



**Recenzenti:**

prof. dr. sc. Danko Basch  
prof. dr. sc. Željko Hocenski

**Crteži, slog i prijelom:**

dr. sc. Martin Žagar

**Nakladnik 1. izdanja:**

Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

**Tisak 1. izdanja:**

AKD d.o.o.

© dr. sc. Martin Žagar, doc. dr. sc. Josip Knezović, Ivana Bosnić, dipl. ing., prof. dr. sc. Mario Kovač, 2015.

**ISBN:** 978-953-184-181-8

CIP zapis dostupan u računalnome katalogu Nacionalne i sveučilišne knjižnice u Zagrebu pod brojem 839203

Uporabu ovog sveučilišnog priručnika odobrio je Senat Sveučilišta u Zagrebu na sjednici održanoj 17. travnja 2013. (Klasa 032-01/13-01/7, Ur. broj 380-061-117-13-3)

**Ovaj dokument namijenjen je isključivo studentima Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu kao pomoć pri savladavanju gradiva iz predmeta Arhitektura računala 1 i za druge svrhe se ne smije koristiti.**

**Nijedan dio ove knjige ne smije se umnožavati, fotokopirati niti reproducirati na bilo koji način bez pismene dozvole nakladnika.**

dr. sc. Martin Žagar  
doc. dr. sc. Josip Knezović  
Ivana Bosnić, dipl. ing.  
prof. dr. sc. Mario Kovač

# **Zbirka programskih zadataka za procesor ARM 7**

2. prepravljeno izdanje

Zagreb, 2015.



# Sadržaj

<b>1. UVOD .....</b>	<b>1</b>
<b>2. PROGRAMIRANJE PROCESORA ARM 7 .....</b>	<b>3</b>
2.1. ZAPISIVANJE PODATAKA U RAČUNALU .....	3
2.1.1. Pretvorba iz zapisa 1'k u zapis 2'k (FRISC 2.2.2.) .....	3
2.1.2. Pretvorba iz zapisa 1'k u zapis s bitom za predznak (FRISC 2.2.3.) .....	5
2.1.3. Pretvorba iz 14-bitnog zapisa 1'k u 32-bitni 1'k (FRISC 2.2.4.) .....	6
2.1.4. Pretvorba iz 32-bitnog zapisa s bitom za predznak u 64-bitni 2'k (FRISC 2.2.7.) .....	7
2.1.5. Pretvorba iz 32-bitnog zapisa 2'k u 48-bitni zapis s bitom za predznak (FRISC 2.2.9.) .....	8
2.1.6. Pretvorba 16-bitnog zapisa s bitom za predznak u 16-bitni 2'k u zapisu <i>big-endian</i> (FRISC 2.2.11.) .....	10
2.1.7. Pretvorba iz zapisa NBC u pakirani BCD (FRISC 2.2.13.) .....	11
2.1.8. Pretvorba iz 32-bitnog zapisa u pakiranom BCD-u u 24-bitni NBC (FRISC 2.2.14.) .....	13
2.2. OPERACIJE S PODATCIMA U RAČUNALU .....	15
2.2.1. Zbrajanje parova brojeva (FRISC 2.3.1.) .....	15
2.2.2. Zbrajanje NBC-brojeva i prekoračenje opsega (FRISC 2.3.2.) .....	16
2.2.3. Oduzimanje 2'k-brojeva i prekoračenje opsega (FRISC 2.3.3.) .....	17
2.2.4. Ispitivanje podataka u zapisu 2'k (FRISC 2.3.4.) .....	18
2.2.5. Preslikavanje pozitivnih brojeva u novi blok (FRISC 2.3.5.) .....	19
2.2.6. Usporedba 2'k-brojeva i upis većeg u novi blok (FRISC 2.3.6.) .....	20
2.2.7. Prebrajanje negativnih 2'k brojeva u bloku (FRISC 2.3.7.) .....	21
2.2.8. Zbrajanje 16-bitnih 2'k-brojeva i pretvorba u 32-bitni zapis s bitom za predznak (FRISC 2.3.8.) .....	22
2.2.9. Proširivanje brojeva u zapisu s bitom za predznak sa 16 na 32 bita (FRISC 2.3.9.) .....	23
2.2.10. Zbroj po bajtovima i zamjena podataka u bloku (FRISC 2.3.10.) .....	25
2.2.11. Zbrajanje više podataka u zapisu s bitom za predznak (FRISC 2.3.11.) .....	26
2.2.12. Množenje NBC-broja s konstantom (FRISC 2.3.12.) .....	28
2.2.13. Množenje NBC-broja s konstantom i spremanje u dvostruku preciznost (FRISC 2.3.13.) .....	29
2.2.14. Množenje NBC-brojeva pomoću metode uzastopnog pribrajanja (FRISC 2.3.14.) .....	32
2.2.15. Množenje 2'k-brojeva pomoću metode uzastopnog pribrajanja (FRISC 2.3.15.) .....	33
2.2.16. Cjelobrojno dijeljenje 2'k-brojeva s konstantom (FRISC 2.3.16.) .....	34
2.2.17. Cjelobrojno dijeljenje NBC-brojeva metodom uzastopnog oduzimanja (FRISC 2.3.17.) .....	35
2.2.18. Izračun površine jednakokračnog trokuta (FRISC 2.3.18.) .....	36
2.2.19. Srednja vrijednost 2'k-brojeva u bloku podataka (FRISC 2.3.19.) .....	36
2.3. POTPROGRAMI .....	37
2.3.1. Prijenos parametara i rezultata u potprogram pomoću registara (FRISC 2.4.2.) .....	38
2.3.2. Prijenos parametara i rezultata u potprogram pomoću fiksnih lokacija (FRISC 2.4.3.) .....	40
2.3.3. Prijenos parametara i rezultata u potprogram pomoću memorijskih lokacija iza poziva potprograma .....	41
2.3.4. Prijenos parametara pomoću stoga i vraćanje rezultata registrom (FRISC 2.4.5) .....	42
2.3.5. Prijenos parametara u potprogram i vraćanje rezultata pomoću stoga (FRISC 2.4.7.) .....	44
2.3.6. Kombinirani načini prijenosa parametara i vraćanja rezultata iz potprograma (FRISC 2.4.8.) .....	46
2.4. OPĆI PROGRAMSKI ZADATCI .....	48
2.4.1. Usporedba i množenje 32-bitnih brojeva (ZI10) .....	48
2.4.2. Potprogram za računanje srednje vrijednosti četiriju brojeva (FRISC 2.5.4.) .....	49
2.4.3. Zamjena dva 16-bitna broja njihovim 32-bitnim zbrojem (FRISC 2.5.7.) .....	50
2.4.4. Premještanje podataka unutar bloka (FRISC 2.5.8.) .....	51
2.4.5. Potprogram za prebrajanje završnih ničtica (FRISC 2.5.9.) .....	54
2.4.6. Računanje sume niza (ZI06) .....	55
2.4.7. Izračunavanje svih djelitelja 32-bitnog broja .....	56
2.4.8. Provjera točke na paraboli .....	57
2.4.9. Izbacivanje podataka iz bloka .....	59

2.4.10.	Prebrajanje elemenata liste (ZI07) .....	60
2.4.11.	Predznačno proširivanje 2'k brojeva (ZI08) .....	62
2.4.12.	Provjera je li broj savršen (ZI12) .....	63
2.4.13.	Težinska suma niza (ZI09) .....	65
2.4.14.	Izračun funkcije $16 \cdot A + B/4$ .....	67
2.4.15.	Usporedba brojeva i računanje apsolutne vrijednosti (ZI11) .....	68
2.4.16.	Kvadrat 16-bitnog broja u formatu 2'k .....	70
2.4.17.	Dijeljenje uzastopnim oduzimanjem i rastavljanje na proste faktore .....	71
2.4.18.	Oduzimanje i dijeljenje brojeva u dvostrukoj preciznosti .....	71
2.4.19.	Potprogram za izračun funkcije sa četiri parametra .....	71
2.4.20.	Zrcalni bitovi i provjera palindroma .....	72
2.4.21.	Potprogram za pretvorbu iz zapisa BCD u NBC .....	72
2.4.22.	Brojenje pojavljivanja 2-bitnog uzorka u broju .....	72
2.4.23.	Pronalazak podniza u nizu bitova .....	73
2.4.24.	Zbrajanje brojeva u zapisu nepomičnog zareza .....	73
2.4.25.	Izračun cjelobrojnog dijela broja u zapisu IEEE .....	74
<b>3.</b>	<b>PROGRAMIRANJE VANJSKIH JEDINICA .....</b>	<b>75</b>
3.1.	VANJSKE JEDINICE GPIO I RTC .....	75
3.1.1.	Brzi prekid FIQ .....	76
3.1.2.	Prijenos podatka pomoću GPIO i RTC (ZI08) .....	77
3.1.3.	Ispitivanje podatka na vratima sklopa GPIO (ZI09) .....	80
3.1.4.	Kontinuirano mijenjanje bita na sklopu GPIO (ZI06) .....	83
3.1.5.	Alarm ostvaren sklopovima GPIO i RTC .....	84
3.1.6.	Prijenos s vrata B na vrata A sklopa GPIO (ZI07) .....	86
3.1.7.	Termostat s dva sklopa GPIO .....	89
3.1.8.	Ispis teksta na LCD .....	91
3.1.9.	Upravljanje pružnim prijelazom pomoću sklopovima GPIO i RTC .....	93
3.1.10.	Pješački semafor (ZI10) .....	97
3.1.11.	Mikrovalna pećnica upravljana sklopovima GPIO i RTC (ZI11) .....	100
3.1.12.	Upravljanje inkubatorom pomoću sklopova GPIO i RTC (ZI12) .....	103
3.1.13.	Očitavanje i ispisivanje temperature .....	106
3.1.14.	Rampa za parkiralište .....	106
<b>4.</b>	<b>PRILOG - TABLICE PROGRAMIRANJA PROCESORA ARM 7 .....</b>	<b>108</b>

# Popis zadataka sa završnih ispita

## Završni ispiti

ZI06 .....	55, 82
ZI07 .....	60, 85
ZI08 .....	62, 76
ZI09 .....	65, 79
ZI10 .....	48, 96
ZI11 .....	68, 99
ZI12 .....	63, 102

# Predgovor

Tijekom pisanja ovog predgovora zazvonio je mobilni telefon i skrenuo nam je tijek misli. Isto tako je smetao zvuk televizije u pozadini pa nismo mogli napisati ovaj predgovor. Zato smo sjeli u automobil i otišli na posao. Pitate se kakve veze ima ova zbirka s mobilnim telefonom, televizorom i automobilom? Odgovor je jednostavan - povezuje ih procesor ARM. Budući da je danas ARM najčešće korišteni procesor u mobilnim i različitim drugim uređajima, logično je koristiti ga u nastavi na kolegiju Arhitektura računala 1, koji se predaje na prvoj godini studija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Ova zbirka zadataka predstavlja nadopunu predavanjima i prvenstveno je namijenjena studentima, kako bi im olakšala svladavanje gradiva. Međutim, zbog raširenosti primjene procesora ARM, zbirka može poslužiti i svima onima koji žele na primjerima usavršiti načine programiranja ovog procesora. Budući da je Arhitektura računala 1 jedan od temeljnih kolegija preddiplomskog studija na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, želja nam je bila pokazati primjere programiranja porodice procesora ARM 7. Ova zbirka nastavlja se na *Zbirku programskih zadataka za procesor FRISC*, odakle je dio zadataka namjerno preuzet i riješen za procesor ARM 7, kako bi se pojedina rješenja mogla uspoređivati.

Za nastanak ove zbirke zaslužni su i prof.dr.sc. Danko Basch, dr.sc. Branko Mihaljević, dr.sc. Marin Orlić, doc.dr.sc. Hrvoje Mlinarić, Tomislav Sečen, dipl.ing., doc.dr.sc. Igor Čavrak i Daniel Hofman, dipl.ing. Za kraj, nadamo se da smo vam ovom zbirkom barem malo približili svijet programiranja procesora ARM.

Zagreb, 2013.

Autori



# 1. Uvod

Ova zbirka zadataka sastoji se od većeg broja programskih zadataka za procesor ARM 7 riješenih u programskom okruženju ATLAS, grupiranih po poglavljima koja slijede predavanja iz predmeta Arhitektura računala 1. Time se ova zbirka i nastavni materijali za predavanja međusobno nadopunjuju.

Zadatci su podijeljeni u dvije glavne cjeline: općenito programiranje procesora ARM 7 (poglavlje 2) i programiranje vanjskih jedinica (poglavlje 3). Gradivo drugog poglavlja dodatno je razvrstano po potpoglavljima na: zapisivanje podataka u računalu, operacije s podacima u računalu te rad s potprogramima. Na kraju poglavlja nalazi se niz zadataka koji u sebi objedinjuju gradivo prethodnih potpoglavlja. U trećem poglavlju nalazi se gradivo koje obuhvaća iznimke i programiranje vanjskih jedinica GPIO i RTC, te termostata i LCD-a spojenih na GPIO.

Zadatci nisu razvrstani samo po gradivu, nego su dodatno poredani od jednostavnijih ka kompliciranijima, kako bi proces učenja bio olakšan. Težina svakog zadatka označena je oznakama ★, ★★ i ★★★ redom od lakših ka težim zadatcima. Ocjena težine ne uključuje samo duljinu zadatka, nego i složenost upotrijebljenog algoritma, težinu implementacije odabranog algoritma, težinu razumijevanja zadatka itd. U zbirci ne postoji ni jedan zadatak koji bi zaista bio težak ili opsegom velik, jer se takvi zadatci ne koriste u nastavi niti se pojavljuju na ispitima. Zato ocjene težine treba uzimati samo kao približne i relativne. Iako je najveći dio zadataka riješen, dio zadataka je bez rješenja i oni služe studentima za samostalno rješavanje i vježbu. Naravno, svatko može pokušati samostalno riješiti bilo koji zadatak iz zbirke i onda ga usporediti s predloženim rješenjem. Budući da u programiranju ne postoje jedinstvena rješenja, ne očekuje se da će se ona podudarati.

Zadatci preuzeti iz završnih ispita također su posebno označeni. Oznaka ZI označava završni ispit: na primjer, ZI06 znači završni ispit iz 2006. godine. Na taj način studenti mogu otprilike vidjeti vrstu i težinu zadataka kakve mogu očekivati na nadolazećim ispitima. Dodatno, s obzirom da je znatan dio zadataka preuzet iz *Zbirke programskih zadataka za procesor FRISC*, kraj takvih zadataka nalaze se oznake broja tog zadatka u Zbirci programskih zadataka za procesor FRISC: na primjer FRISC 2.2.4. Ovo je učinjeno da bi se pojedina rješenja mogla uspoređivati s rješenjima za procesor FRISC.

Svi zadatci u zbirci izloženi su na jednolik način, koliko god je to bilo moguće. Nakon teksta kojim se zadaje pojedini zadatak slijedi predloženo moguće rješenje zadatka opisano s općenitog stajališta. Drugim riječima, ukratko se objašnjava osnovno idejno rješenje bez ulaženja u detalje programske izvedbe. Iza toga slijedi programsko rješenje zadatka u obliku pogodnom za izvođenje na simulatoru ATLAS. Unutar samog teksta programa ukratko su komentirani svi najvažniji dijelovi kako bi se olakšalo praćenje rada programa i razumijevanje programske izvedbe. Na kraju su, tamo gdje je to potrebno, u dodatnom tekstu podrobnije komentirani dijelovi programa koji su teži za razumijevanje, ili čije objašnjenje nije bilo praktično umetnuti u sam programski tekst. U dijelu zadataka predlaže se čitatelju da razmisli kakvo će biti ponašanje programa pod određenim okolnostima, ili da pokuša riješiti

zadatak na drugačiji način, ili da ga promijeni tako da rješava nešto drugačiju zadaću nego je prvobitno zadano. Ovo bi trebalo potaknuti studente na razmišljanje i stvaranje vlastitih rješenja, a ne samo na analiziranje postojećih.

Programi u ovoj zbirci nisu napisani s težnjom da budu optimalni što se tiče brzine izvođenja, zauzeća memorije ili da imaju „najelegantniji“ oblik. Umjesto toga težilo se jednostavnosti, razumljivosti i konzistentnosti te, koliko je to bilo moguće, korištenju dobrih programerskih praksi (modularnost, čuvanje stanja registara, korištenje općeprihvaćenih konvencija itd.). Ponekad rješenja zadataka služe da bi se, kroz primjer, pokazao pristup rješavanju problema na način koji u prethodnim zadacima nije bio korišten.

Kada se navode u tekstu, brojevi su pisani različitim bazama. Dekadski brojevi najčešće su pisani bez posebne oznake baze, a binarni i heksadekadski s oznakom baze iza broja (npr. 2100<sub>16</sub>). Od ovog pravila se povremeno odstupa, ako se u zadatku spominje više brojeva u različitim bazama: tada se oznake baze koriste i iza dekadskih brojeva kako ne bi došlo do zabune. Također se koji puta oznaka baze ispušta iza heksadekadskog broja, ako se u njemu nalaze znamenke koje ne mogu pripadati drugim bazama, kao na primjer u broju 41AA00DF. U programskom kodu se bez oznaka pišu brojevi u heksadekadskoj bazi (podrazumijevana baza), a želi li se napisati binarni ili dekadski broj, to se mora naznačiti prefiksom **%B** odnosno **%D** ispred broja. To je u skladu sa pisanjem brojeva prema pravilima programskog okruženja ATLAS u kojem se od gore navedenog pravila odstupa u dva slučaja. Prvi slučaj je kod adresiranja neposrednog pomaka u polju `<Oprnd2>` (vidi Prilog) gdje se podrazumijeva dekadski baza za pomak. Na primjer, u naredbi **MOV R2,R0,ASR #12** broj 12 je zapisan dekadski. Drugi slučaj je dekadski zapis broja kod zadavanja rotacije ulijevo prilikom upisa neposredne vrijednosti u naredbama za obradu podataka. Na primjer, u naredbi **MOV R0,#3A<8** broj 8 je zapisan dekadski i tom naredbom se u registar **R0** zapisuje vrijednost 3A00<sub>16</sub>.

U programskom okruženju ATLAS, procesor ARM 7 započinje s izvođenjem programa čija prva naredba je na adresi 0. Zato se pretpostavlja da glavni program započinje na toj adresi što je u programima obično naznačeno pseudonaredbom **ORG 0**. Ako bi se ovaj redak i ispustio, programi bi i dalje bili ispravni jer asemblerski prevoditelj u ATLAS-u pretpostavlja da punjenje programa počinje od adrese 0. U simulatoru ATLAS na raspolaganju je memorija kapaciteta 64 Kb koja zauzima adrese od 0 do FFFF. Samo u pojedinim zadacima zadano je da se koriste memorijske lokacije izvan navedenog opsega i tada se pretpostavlja da je na ARM 7 spojena memorija većeg kapaciteta.

Program, njegovi potprogrami i potrebni podatci mogu „na papiru“ biti označeni pseudonaredbama **ORG** i napisani u bilo kojem redoslijedu. Takav proizvoljan redoslijed može biti pregledniji ili lakše razumljiv, na primjer zbog redoslijeda kojim se potprogrami međusobno pozivaju. Međutim, u stvarnom programu koji pišemo za ATLAS, adrese iza pseudonaredaba **ORG** moraju biti u rastućem redoslijedu pa su programi u ovoj zbirci pisani na takav način.

Na kraju zbirke nalazi se prilog s tablicom naredaba procesora ARM 7 i tablicama za programiranje sklopova GPIO i RTC. Ove tablice služe kao podsjetnik prilikom programiranja i smiju se koristiti na ispitima. Istovjetne tablice su raspoložive studentima na internetskim stranicama predmeta Arhitektura računala 1, pri čemu je jedina razlika u načinu oblikovanja koji je prikladniji za ispis i korištenje na ispitima.

## 2. Programiranje procesora ARM 7

Ovo poglavlje pokriva prvi dio gradiva vezanog za procesor ARM 7 – programiranje procesora u asemblerskom ili mnemoničkom jeziku. U programima se pretpostavlja da se računalni sustav sastoji samo od procesora i memorije. Drugim riječima, ne koriste se vanjske jedinice za komunikaciju s okolinom, nego se pretpostavlja da se svi potrebni podatci za izvođenje već nalaze u memoriji ili registrima procesora te da se svi rezultati ponovno spremaju u memoriju ili registre.

Gradivo ovog poglavlja obuhvaća osnovne zadatke koji se javljaju u svakom asemblerskom programiranju, kao što su rad s bitovima i podacima pohranjenima u različitim zapisima (prikazima ili formatima) te pretvorbe između njih, kao i izvođenje raznih operacija s podacima, bilo da se radi o aritmetičkim ili logičkim operacijama. Nakon toga je pokazano korištenje potprograma s posebnim naglaskom na prijenos parametara u potprogram i vraćanje rezultata iz potprograma. U zadnjem potpoglavlju dan je niz zadataka u kojima se kombinira gradivo iz prethodnih potpoglavlja.

### 2.1. Zapisivanje podataka u računalu

Prilikom programiranja u višim programskim jezicima (na primjer C, C++, Java, C# itd.) programeri imaju na raspolaganju različite tipove podataka kao što su *int*, *float*, *unsigned int* i najčešće ne trebaju poznavati kako su interno ovi podatci zapisani u računalu. Međutim, u asemblerskom programiranju ne postoje tipovi podataka, nego se programer mora sam brinuti za zapisivanje brojeva, njihovu interpretaciju i za operacije koje će nad njima izvoditi. U ovom je potpoglavlju objašnjeno nekoliko najčešće korištenih zapisa brojeva i načini pretvorbe između njih:

- zapis cjelobrojnih podataka bez predznaka (NBC, pakirani i nepakirani BCD)
- zapis cjelobrojnih podataka s predznakom (zapis s bitom za predznak, jedinični komplement ili skraćeno 1'k, dvojni komplement ili skraćeno 2'k)
- pretvorbe između različitih zapisa
- promjena broja bitova u zapisu: smanjivanje broja bitova, povećavanje broja bitova proširivanjem podataka (proširivanje ničticama, predznačno proširivanje)
- redoslijed zapisa bajtova u memoriji ili *endianness* (*little-endian*, *big-endian*).

#### 2.1.1. Pretvorba iz zapisa 1'k u zapis 2'k (FRISC 2.2.2.)

Riješen: DA    Težina: ★

Napisati program koji iz bloka čija se adresa nalazi u memoriji na lokaciji označenoj labelom **POD** čita 32-bitne podatke u zapisu 1'k dok ne pročita podatak 0. Pročitane podatke treba

pretvoriti u 32-bitni zapis 2'k i spremiti ih na lokacije iz kojih su pročitani. U blok podataka upisati proizvoljno nekoliko pozitivnih i negativnih brojeva u zapisu 1'k.

### Prijedlog rješenja:

Na početku petlje se prvo provjerava je li učitani zadnji broj (naredbe **CMP** i **BEQ**). Pretvorba iz zapisa 1'k u 2'k jednostavna je jer su ovi zapisi vrlo slični. Za pozitivne brojeve zapisi su isti. Zato se prilikom pretvorbe prvo provjerava predznak broja (**BPL**); ako je broj pozitivan, pretvorba i spremanje se preskaču. Negativni brojevi u zapisu 2'k razlikuju se od zapisa 1'k samo po tome što su uvećani za jedan. Zato se za slučaj negativnog broja prvo broj poveća za jedan i onda spremi na lokaciju iz koje je pročitani. U rješenju su u bloku zapisani brojevi: -1, -2, 2, -16, 17 i 0.

Rješenje:			
	ORG	0	
	LDR	R0, POD	; učitavamo adresu u R0 = 1000
PETLJA	LDR	R1, [R0]	; učitavamo 1'k podatak iz bloka
	CMP	R1, #0	; provjera kraja podataka
	BEQ	KRAJ	
PRETVORBA	BPL	DALJE	; ako je broj pozitivan, nema pretvorbe
NEGATIVAN	ADD	R1, R1, #1	; iz 1'k u 2'k
DALJE	STR	R1, [R0], #4	; spremanje rezultata, povećanje adrese
	B	PETLJA	
KRAJ	HALT		
POD	DW	1000	
	ORG	1000	
	DW	0FFFFFFFE, 0FFFFFFFD, 2, 0FFFFFFEF, 11, 0	

### Komentar rješenja:

Na memorijsku lokaciju označenu labelom **POD** upisali smo početnu adresu bloka ( $1000_{16}$ ) a od adrese  $1000_{16}$  upisali smo podatke iz bloka (-1, -2, 2, 16, 17, 0) u formatu jediničnog komplementa. Učitavanje adrese na koju pokazuje **POD** ostvarili smo naredbom **LDR R0, POD**. Ova naredba nije striktno procesorska naredba, nego ju assembler prevede u naredbu oblika: **LDR R0, [R15, #<odmak>]**, pri čemu se koristi registarsko adresiranje s odmakom. Kao bazni registar koristi se programsko brojilo **PC(R15)**, a potreban odmak će se automatski izračunati od strane asemblerskog prevoditelja koristeći vrijednost programskog brojila pri izvođenju naredbe te vrijednosti adrese označene labelom **POD**. Ekvivalentno ovoj naredbi, postoji i (pseudo)naredba **STR Rn, LABELA** koja sprema sadržaj registra **Rn** na memorijsku lokaciju označenu labelom.

Učitavanje podataka iz bloka i zapisivanje rezultata ostvareno je naredbama **LDR R1, [R0]**, odnosno **STR R1, [R0], #4**. Prilikom spremanja koristimo oblik naredbe sa postindeksiranjem pri čemu će se podatak iz registra **R1** spremiti na adresu na koju pokazuje **R0**, a nakon toga će se adresni registar **R0** uvećati za 4 kako bi u sljedećem prolazu petlje pokazivao na sljedeći podatak u bloku.

**2.1.2. Pretvorba iz zapisa 1'k u zapis s bitom za predznak (FRISC 2.2.3.)****Riješen: DA    Težina: ★**

Napisati program koji iz memorijske lokacije  $1000_{16}$  čita podatke u zapisu 1'k dok ne pročita podatak 0. Pročitane podatke treba pretvoriti iz zapisa 1'k u zapis s bitom za predznak i spremiti na lokacije iz kojih su pročitani. Od lokacije  $1000_{16}$  upišite proizvoljno nekoliko pozitivnih i negativnih brojeva u zapisu 1'k.

**Prijedlog rješenja:**

U oba prikaza negativan broj prepoznaje se po najvišem bitu pa je za ispitivanje predznaka broja dovoljno ispitati zastavicu N. Zbog toga se prilikom pretvorbe prvo provjerava predznak broja (naredba **BPL**). Ako je broj pozitivan, pretvorba i spremanje se preskaču jer su brojevi u oba zapisa u tom slučaju isti. Negativni brojevi u zapisu s bitom za predznak razlikuju se od zapisa 1'k u tome što imaju jedinicu na najvišem bitu, a u nižim bitovima se nalazi apsolutni iznos broja, dok zapis 1'k ima u svim bitovima jedinični komplement apsolutnog iznosa broja (u najvišem bitu također će se nalaziti jedinica, jer apsolutni iznos broja na najvišem bitu ima ničticu, koju će operacija jediničnog komplementa promijeniti u jedinicu). Pretvorba negativnog broja može se obaviti na nekoliko načina. Najjednostavnije je prvo cijeli broj jedinično komplementirati (naredbom **MVN**), čime se dobije apsolutni iznos broja, a nakon toga se postavi jedinica na najvišem bitu. Konačno, dobiveni rezultat sprema se na lokaciju iz koje je pročitani polazni broj.

**Rješenje:**

	ORG	0	
	MOV	R0, #10<8	
PETLJA	LDR	R1, [R0]	
	CMP	R1, #0	; ako smo došli kraja
	BEQ	KRAJ	; skoči na KRAJ
PRETVORBA BPL	DALJE		; ako je broj pozitivan, nema pretvorbe
NEGATIVAN	MVN	R1, R1	; jedinično komplementiranje
	ORR	R1, R1, #80<24	; postavljanje najvišeg bita - predznak
DALJE	STR	R1, [R0], #4	; spremanje rezultata
	B	PETLJA	
KRAJ	HALT		
	ORG	1000	
	DW	0FFFFFFFE, 0FFFFFFFD, 2, 0FFFFFFEF, 1671, 0	

**Komentar rješenja:**

Naredbom **MOV R0, #10<8** u registar **R0** se upisuje početna adresa  $1000_{16}$ . U nastavku programa ovaj registar ćemo koristiti kao adresni registar za dohvat i spremanje podatka (**LDR R1, [R0]** i **STR R1, [R0], #4**).

Pretvorba negativnih brojeva ostvarena je komplementiranjem (**MVN R1, R1**), a nakon toga upisivanje jedinice na najviši bit rezultata ostvaruje se naredbom **ORR R1, R1, #80<24**. Rezultat se sprema u registar **R1** koji se konačno sprema u memoriju na mjesto originalnog podatka.

Alternativni način pretvorbe negativnog broja iz zapisa 1'k u zapis s bitom za predznak je da se prvo obriše jedinica na najvišem bitu, a nakon toga se cijeli broj jedinično komplementira, čime se vrati jedinica u najviši bit koji je prethodno obrisan, a u nižim bitovima se dobije apsolutni iznos broja:

NEGATIVAN BIC	R1, R1, #80<24	; brisanje najvišeg bita
MVN	R1, R1	; komplementiranje - postavljanje najvišeg bita
		; i upis apsolutne vrijednosti u ostale bitove

Može li se i ovaj način pretvorbe primijeniti na pretvorbu u suprotnom smjeru (iz zapisa s bitom za predznak u 1'k)? Pretvorba se može obaviti i pomoću naredbe **EOR** s odgovarajućom maskom. Može li se ovaj način primijeniti i na pretvorbu u suprotnom smjeru?

Osim u prikazanim zapisima 1'k i s bitom za predznak, predznak nekog broja zapisanog i u zapisu 2'k se također ispituje na isti način provjeravanjem najvišeg bita, odnosno zastavice N.

### 2.1.3. Pretvorba iz 14-bitnog zapisa 1'k u 32-bitni 1'k (FRISC 2.2.4.)

Riješen: DA Težina: ★

Napisati program koji čita podatke iz izvorišnog bloka u memoriji i zapisuje ih u odredišni blok. Izvorišni blok počinje na adresi 2100<sub>16</sub> i sadrži 50<sub>10</sub> podataka, a odredišni blok je na adresi 3100<sub>16</sub>. Podatci u izvorišnom bloku prikazani su u 14-bitnom zapisu 1'k tako da su podatci pohranjeni kao 32-bitne riječi, a gornjih 18 bita podatka je popunjeno ničicama. Podatke u odredišni blok treba spremati u 32-bitnom zapisu 1'k.

#### Prijedlog rješenja:

Za proširivanje sa 14 na 32 bita iskoristit će se naredbe pomaka. Prvo će se 14-bitni broj naredbom **MOV R3, R3, LSL #18** pomaknuti logički ulijevo tako da najznačajniji (četрнаести) bit originalnog broja dođe u najznačajniji bit, a zatim će se naredbom **MOV R3, R3, ASR #18** izvesti aritmetički pomak udesno natrag na originalno mjesto u registru. Aritmetički pomak čuva najviši bit (bit predznaka) i njime popunjava bitove koji ulaze s lijeva. Kod procesora ARM ne postoje izdvojene naredbe za pomake i rotacije, nego se ove operacije mogu izvesti u sklopu registarskih i aritmetičko-logičkih naredaba. Zbog toga će se u programu iskoristiti naredba **MOV**. Dodatno, pomak će u naredbi biti zadan kao neposredni broj koji je podrazumijevano zapisan u dekadskoj bazi.

#### Rješenje:

	ORG	0	
	MOV	R0, #21<8	; R0 adresa izvorišta podataka
	MOV	R1, #50	; R1 brojač podataka
	MOV	R2, #31<8	; R2 adresa odredišta podataka
PETLJA	LDR	R3, [R0], #4	; učitavanje podatka
	MOV	R3, R3, LSL #18	; pomak ulijevo za 18 mjesta
	MOV	R3, R3, ASR #18	; pomak udesno za 18 mjesta, predznak se čuva!
	STR	R3, [R2], #4	; spremanje podatka
	SUBS	R1, R1, #1	; smanjivanje brojača
	BNE	PETLJA	; skok na novi podatak
IZVOR	ORG	2100	
	DW	0, 1, 3414, 69	; ...i još 46 podataka

**Komentar rješenja:**

Naredbom **MOV R0, #21<8** upisujemo adresu odredišnog bloka u registar **R0**. U zapisu neposredne vrijednosti broj 21 je podrazumijevano u heksadekadskoj bazi, dok je broj 8 u dekadskoj bazi tako da je konačna vrijednost podatka  $2100_{16}$ . U sljedećoj naredbi **MOV R1, #D50** podatak  $50_{10}$  se može zapisati unutar jednog bajta pa ga je moguće zapisati bez navođenja vrijednosti za rotaciju nakon znaka „<“. Proširenje broja u zapisu 1'k sa 14 na 32 bita izvedeno je naredbama **MOV R3, R3, LSL #18** i **MOV R3, R3, ASR #18**. Naredba **SUBS R1, R1, #1** na kraju petlje smanjuje brojač podataka u bloku i određuje je li potrebno nastaviti petlju. Sufiks **S** u naredbi označava da naredba treba postaviti i zastavice, što nam je ovdje potrebno jer nakon naredbe ispitujemo zastavicu **Z** u naredbi **BNE**. Bez sufiksa **S**, naredba **SUB** ne bi postavila zastavice pa bi time izvođenje programa bilo pogrešno. Budući da se podatci u zapisu 1'k pretvaraju u istovjetan zapis s većim brojem bitova, svi podatci iz izvorišnog bloka sigurno se mogu prikazati u odredišnom bloku. U općenitom rješenju, gdje broj podataka u bloku ne bi bio unaprijed poznat, bolje bi bilo provjeriti brojač na početku petlje, za slučaj da je veličina bloka 0.

### 2.1.4. Pretvorba iz 32-bitnog zapisa s bitom za predznak u 64-bitni 2'k (FRISC 2.2.7.)

Riješen: DA    Težina: ★

Napisati program koji čita podatke iz izvorišnog bloka u memoriji i zapisuje ih u odredišni blok. Izvorišni blok počinje na adresi  $8000_{16}$ , a odredišni blok je na adresi  $6000_{16}$ . Podatci u izvorišnom bloku su u 32-bitnom zapisu s bitom za predznak, a podatke u odredišni blok treba spremiti u 64-bitnom zapisu 2'k. Brojeve koji se ne mogu prikazati treba zamijeniti podatkom 0. Izvorišni blok sadrži  $200_{16}$  podataka.

**Prijedlog rješenja:**

Zapis s bitom za predznak (BZP) može prikazati manji raspon brojeva (uz isti broj bitova) od zapisa 2'k, tako da će se u odredišnom bloku sigurno moći prikazati svi brojevi. Kako nije drukčije zadano, 64-bitni broj zapisat će se u poretku *little-endian*.

**Rješenje:**

	ORG	0	
GLAVNI	MOV	R2, #8<12	; izvorište podataka
	MOV	R3, #6<12	; odredište podataka
	LDR	R4, PREDZNAK	; maska za učitavanje predznaka
	MOV	R5, #2<8	; broj podataka u bloku = $200_{16}$
PETLJA	LDR	R0, [R2], #4	; pročitati podatak i povećati adresu izvora
	MOV	R1, #0	; obrisati gornji dio 64-bitnog broja
	ORRS	R0, R0, #0	; provjeriti bit predznaka broja (najviši)
	BPL	POZIT	; pozitivan broj
NEGAT	AND	R0, R0, R4	; maskom obrisati bit predznaka
	RSB	R0, R0, #0	; $R0 = 0 - R0 = -R0$ - donji dio
	MVN	R1, #0	; gornji dio 64-bitnog broja = sve jedinice
POZIT	STR	R0, [R3], #4	; spremiti donji dio broja
	STR	R1, [R3], #4	; spremiti gornji dio broja

(nastavak na sljedećoj stranici)

GOTOV	SUBS	R5, R5, #1	; smanjiti brojač podataka
	BNE	PETLJA	; ako nije gotovo nastaviti
	HALT		; kraj
PREDZNAK	DW	07FFFFFF	; maska za brisanje predznaka 32-bitnog BZP
ODRED	ORG	6000	
	DW	0	
	ORG	8000	
IZVOR	DW	0, 12345A, 80005670, 0FFFFFFF	; ...i još 1FC podataka

### Komentar rješenja:

Za provjeru bita predznaka podatka u 32-bitnom zapisu s bitom za predznak može se iskoristiti zastavica N jer ona nakon naredbe **ORRS R0, R0, #0** odražava bit 31 registra **R0**, što se poklapa s bitom za predznak. Istovjetni rezultat bi dobili i naredbom **CMP R0, #0**. Nakon što je određen predznak broja, u slučaju negativnog podatka potrebno ga je dvojno komplementirati, čime se dobije nižih 32 bita rezultata, dok za pozitivan podatak nižih 32 bita ostaje isto. Prije komplementiranja za negativne brojeve potrebno je ukloniti bit za predznak naredbom **AND R0, R0, R4**. Primijetite da prilikom učitavanja maske u registar **R4**, konstantu nismo mogli napisati neposredni broj (8 bita s parnim pomakom ulijevo), pa ju nije moguće navesti unutar naredbe (npr. ~~MOV R4, #7FFFFFFF~~), već je maska učitana iz memorije naredbom **LDR R4, PREDZNAK**.

Komplementiranje broja ostvareno je naredbom **RSB** (*reverse subtract*) budući da se radi o brojevima u dvojnog komplementu. Viših 32 bita mogu biti ili sve ničice (za pozitivni podatak) ili sve jedinice (za negativni podatak), te se tako i postavljaju, ovisno o predznaku originalnog podatka. Nakon pohrane rezultata treba pripaziti na pomak adrese izvora (za 4 bajta) i odredišta (za 8 bajta). U općenitom rješenju bolje bi bilo provjeriti brojač na početku petlje, za slučaj da je veličina bloka 0. Ovaj program neće raditi ispravno za pretvaranje broja -0 (takozvana negativna ničica, odnosno podatka  $80000000_{16}$  u BZP zapisu). Što će se dogoditi ako se ovakav broj nađe u izvorišnom bloku? Izmijenite program tako da radi ispravno i za ovaj slučaj?

### 2.1.5. Pretvorba iz 32-bitnog zapisa 2'k u 48-bitni zapis s bitom za predznak (FRISC 2.2.9.)

Riješen: DA    Težina: ★

Napisati program koji čita podatke iz izvorišnog bloka u memoriji i zapisuje ih u odredišni blok. Izvorišni blok počinje na adresi  $10000_{16}$ , a odredišni blok je na adresi  $20000_{16}$ . Podatci u izvorišnom bloku su u 32-bitnom zapisu 2'k, a podatke u odredišni blok treba spremiti u 48-bitnom zapisu s bitom za predznak, ali tako da broj bude zapisan u 64 bita (podatak je zapisan u nižih 48 bitova, a u viših 16 bitova trebaju biti ničice). Izvorišni blok sadrži  $100_{16}$  podataka.

### Prijedlog rješenja:

Zapis 2'k uz isti broj bitova može prikazati veći raspon brojeva od zapisa s bitom za predznak. Međutim, u ovom slučaju se broj bitova u odredišnom zapisu povećava, pa će se svi podatci iz izvornog bloka sigurno moći prikazati u odredišnom bloku.



**Rješenje:**

```

ORG      0
GLAVNI   MOV    R4, #10<12      ; R4 - izvoriste podataka
          MOV    R5, #20<12      ; R5 - odredište podataka
          MOV    R6, #1<8        ; R6 - broj podataka
PETLJA   CMP    R6, #0          ; ima li još podataka u bloku
          BEQ    KRAJ           ; gotovo

          LDR    R3, [R4], #4     ; učitavanje podatka i uvećanje adrese izvorišta
          MOV    R2, #0          ; upisati ničticu u gornjih 32 bita rezultata
          TST    R3, R3          ; postaviti zastavice (N)

          BPL    SPREMI

          ; rad s negativnim brojevima
NEGAT    MOV    R2, #8<12       ; najviši bit je 1, postavi 48. bit
          RSB    R3, R3, #0      ; R3 = -R3

SPREMI   STR    R3, [R5], #4     ; spremanje donjih 32 bita
          STR    R2, [R5], #4     ; spremanje gornjih 16 bitova i 16 ničtica

          SUB    R6, R6, #1      ; smanjiti brojač
          B      PETLJA

KRAJ     HALT

IZVOR    ORG    10000
          DW     0FFFFFFF, 0FFFFFFC0, 80000000, 123456 ; ... i još FC podataka

ODRED    ORG    20000
          DS     800

```

**Komentar rješenja:**

Postupak pretvorbe razlikuje se za pozitivne i negativne podatke u bloku. Pozitivni podatci mogu se direktno prepisati u odredišni blok, uz gornjih 16 bitova postavljenih na ničticu. Prilikom spremanja rezultata treba obratiti pažnju da se zapisuju dvije 32-bitne riječi. Od toga, prva riječ sačinjava donjih 32 bita podatka, a gornjih 16 bitova podatka spremaju se naredbom **STR R2, [R5], #4** i to kao 32-bitna riječ koju sačinjava korisnih 16 bitova podatka zajedno sa 16 ničtica koji dopunjavaju konačni podatak do širine od 64 bita. Za negativne podatke potrebno je izračunati apsolutnu vrijednost (naredba **RSB**), a predznak u gornjoj riječi sačuvati postavljanjem najvišeg bita 48-bitnog rezultata naredbom **MOV R2, #8<12** (gornjih 16 bitova imaju vrijednost  $8000_{16}$ ). Primijetite da je nakon spremanja ovakvog 48-bitnog rezultata adresa odredišta uvećana za 8. Za razliku od nekoliko prethodnih programa, ovdje se brojač petlje ispituje na početku petlje.

Provjerite radi li program ispravno za najveći negativni broj u 32-bitnom zapisu  $2^k$  ( $80000000_{16}$ ). Kako se taj broj zapisuje u 48-bitom zapisu s bitom za predznak? Kako bi trebalo promijeniti program da je bilo zadano da 48-bitni brojevi trebaju biti zapisani u 48 bitova (tj. u 6 uzastopnih bajtova)?

### 2.1.6. Pretvorba 16-bitnog zapisa s bitom za predznak u 16-bitni 2'k u zapisu *big-endian* (FRISC 2.2.11.)

Riješen: DA    Težina: ★

U bloku podataka koji počinje od adrese  $1000_{16}$ , nalaze se 16-bitni brojevi u zapisu s bitom za predznak, zapisani u poretku *little-endian*. Blok sadrži  $100_{16}$  podataka. Sve podatke u bloku treba pretvoriti u 16-bitni zapis 2'k, ali u redosljedu *big-endian*.

#### Prijedlog rješenja:

Za rješenje ovog zadatka treba poznavati građu zapisa s bitom za predznak i 2'k te njihov međusobni odnos, a također i značenje *endianness*-a. U petlji se učitava jedan po jedan podatak, pretvara se i upisuje natrag u blok. Ovo se ponavlja  $100_{16}$  puta.

Rješenje:			
	ORG	0	
	MOV	R0, #1<12	; R0 - adresa početka bloka
	MOV	R1, #1<8	; R1 - brojač za petlju
	LDR	R4, MASKA	; maska za brisanje 16. bita
PETLJA	LDRSH	R3, [R0]	; učitavanje podatka u R3, predzn.proširivanje
	CMP	R3, #0	; ispitaj predznak
	BPL	SPREMI	; pozitivni brojevi -> nema pretvorbe
			; negativni brojevi -> pretvorba
NEGAT	AND	R3, R3, R4	; brisanje bita predznaka i proširenja
DVA_K	RSB	R3, R3, #0	; R3 = -R3
SPREMI	STRB	R3, [R0, #1]	; nižih 8 bitova broja u viši bajt u memoriji
	MOV	R3, R3, ROR #8	; rotiranje za dobivanje viših 8 bitova
	STRB	R3, [R0]	; viših 8 bitova broja na niži bajt u memoriji
	ADD	R0, R0, #2	; povećavanje pokazivača na adrese za 2
	SUBS	R1, R1, #1	; smanjivanje brojača za 1
	BNE	PETLJA	
KRAJ	HALT		
MASKA	DW	07FFF	
	ORG	1000	
	DH	1, 8001, 0, 34, 8034, 8000	

#### Komentar rješenja:

U petlji se koriste registar **R0** kao pokazivač na podatke u bloku, **R1** kao brojač petlje i registar **R3** u koji se u svakom koraku petlje učitava podatak koji se obrađuje. Na početku petlje **PETLJA** učitava se 16-bitni podatak u registar **R3**. Učitani podatak ima normalni *endianness* (*little-endian*) procesora ARM 7 pa ga se izravno učitava i koristi, bez potrebe da mu se zamjenjuju bajtovi. Prilikom učitavanja, podatak se odmah predznačno proširuje sa 16 na 32 bita korištenjem naredbe **LDRSH**. Zatim se ispituje predznak proširenog podatka korištenjem naredbe **CMP**.

Ako je broj pozitivan, onda je njegov zapis 2'k jednak zapisu s bitom za predznak i skače se na kraj petlje (labela **SPREMI**). Ako je broj negativan, nastavlja se izvođenje od labela **NEGAT**. Ovdje prvo treba obrisati bit predznaka, a zatim provesti operaciju dvojnog komplementa (**DVA\_K**). Predznak se nalazi na bitu 15, a možemo ga obrisati maskom 07FFF, pri čemu se

briše i 16 viših bitova dobivenih predznačnim proširenjem. Masku je potrebno učitati iz memorije, jer se ne može upisati direktno. To je učinjeno na početku programa naredbom **LDR R4, MASKA**. Nakon toga je broj pretvoren u zapis 2'k korištenjem naredbe **RSB**. Prije naredbe u registru **R3** bila je apsolutna vrijednost broja pa smo ovom naredbom oduzeli njegovu vrijednost od 0 te smo u registru dobili negativnu vrijednost broja.

Program nastavlja spremanjem rezultatnog 16-bitnog broja (poluriječi) u memoriju, pri čemu se okreće redoslijed bajtova u *big-endian* (labela **SPREMI**). Budući da naredba **STRH** podrazumijeva *little-endian* redoslijed, ne može se koristiti u ovom zadatku, nego je potrebno „ručno“ zapisati bajtove korištenjem naredbi **STRB**. Zato se prvo niži bajt broja zapisuje na višu adresu (**R0+1**), a zatim se pomoću rotacije dovodi viši bajt broja na mjesto nižeg bajta i zapisuje na nižu adresu (**R0**).

Nakon zapisivanja broja u memoriju, pomiče se pokazivač podataka za 2 bajta, odnosno adresa u **R0** uvećava se tako da pokazuje na sljedeću poluriječ u memoriji. Naredba **BNE PETLJA** vraća izvođenje na početak petlje, čime se postupak nastavlja za sljedeći podatak u bloku.

Kako biste riješili ovaj zadatak da su početni podatci bili široki 32 bita, a pretvoreni podatci su i dalje široki 16 bitova? Kako biste riješili ovaj zadatak da je širina početnih podataka bila 16 bitova, a da su pretvoreni podatci trebali biti široki 32 bita (★★★)?

### 2.1.7. Pretvorba iz zapisa NBC u pakirani BCD (FRISC 2.2.13.)

Riješen: DA Težina: ★★

U registru **R0** je NBC-broj koji treba pretvoriti u pakirani BCD i spremiti ga u registar **R2**.

#### Prijedlog rješenja:

U rješenju prvo treba broj u NBC-u rastaviti na dekadске znamenke uzastopnim dijeljenjem sa bazom, brojem 10. Ostatci dijeljenja su dekadске znamenke koje treba spremati u skupine od dva 4-bitna para koji sačinjavaju dvije znamenke upisane u jedan bajt kako to propisuje pakirani BCD.

Rješenje:			
	ORG	0	
	MOV	R0, #12<4	; proizvoljno odabrani broj u NBC-u
	MOV	R5, #0	; brojač znamenki
	MOV	R6, #0	; pomoćni registar
POC	MVN	R2, #0	; inicijalizacija rezultata na -1
	; dijeljenje R0/10 metodom uzastopnog oduzimanja		
ZN	ADD	R2, R2, #1	; povećanje rezultata dijeljenja
	SUBS	R0, R0, #D10	
	BHI	ZN	
	ADD	R0, R0, #D10	; ostatak dijeljenja
	ADD	R5, R5, #1	; uvećavanje brojača znamenki
	MOV	R6, R6, LSL #4	; spremanje zadnje znamenke u pakirani BCD
	ORR	R6, R6, R0	
	MOV	R0, R2	; traženje iduće znamenke
	CMP	R0, #0	
	BNE	POC	

(nastavak na sljedećoj stranici)

```

MOV    R2, #0
REV    MOV    R2, R2, LSL #4    ; znamenke u R6 su upisane obrnutim
      AND    R1, R6, #0F      ; redoslijedom pa ih treba preokrenuti
      ORR    R2, R1, R2
      MOV    R6, R6, LSR #4
      SUBS   R5, R5, #1
      BNE    REV
      HALT

```

### Komentar rješenja:

Pretvorbu u BCD najlakše je napraviti tako da se pojedine dekadске znamenke početnog broja izdvajaju pomoću dijeljenja sa 10, pri čemu je ostatak pri dijeljenju pojedina znamenka. Na primjer, broj  $1234/10 = 123$  i ostatak 4. Ostatak 4 je najniža dekadská znamenka. Postupak se nastavlja s rezultatom dijeljenja:  $123/10 = 12$  i ostatak 3 (sljedeća dekadská znamenka). Zatim se računa  $12/10 = 1$  i ostatak 2 (sljedeća dekadská znamenka). Konačno, dijeli se  $1/10 = 0$  i ostatak 1 (najviša dekadská znamenka). Postupak se ovdje prekida jer smo kao rezultat dobili 0. Svaku od dobivenih znamenaka 4, 3, 2 i 1 treba u ispravnom redoslijedu spremiti u zasebna 4 bita u rezultatu.

Postupak dijeljenja ostvaren je metodom uzastopnog oduzimanja. U petlji se broj u registru **R0** dijeli sa 10, a rezultat se računa u registru **R2** koji se u svakom koraku petlje uveća za jedan. Rezultat se na početku postavlja na -1 jer pri prvom uvećavanju njegova vrijednost postaje 0. Primjerice, za broj 39 su u tablici prikazane vrijednosti registara prilikom izvođenja naredbe **BHI ZN** u sljedećem programskom odsječku:

```

POC    MVN    R2, #0          ; inicijalizacija rezultata na -1
ZN     ADD    R2, R2, #1      ; povećanje rezultata dijeljenja
      SUBS   R0, R0, #D10     ; dijeljenje uzastopnim oduzimanjem
      BHI    ZN

```

R2 (rezultat)	R0 (djeljenik)	Komentar
-1	39	Početno stanje
0	29	(39 >= 10) skače na ZN
1	19	(29 >= 10) skače na ZN
2	9	(19 >= 10) skače na ZN
3	-1	(9 < 10) ne skače na ZN

Postupak dijeljenja daje dekadské znamenke od nižih ka višima. Nakon svakog dijeljenja nova znamenka se stavlja na najniža 4 bita registra **R6**, a prethodna vrijednost **R6** se prije toga pomakne za 4 mjesta u lijevo. Tako u registru **R6** dobivamo broj u kojemu se nalaze pojedine znamenke zapisa BCD, ali u obrnutom redoslijedu. Zadnji dio programa je petlja (**REV**) koja okreće znamenke u ispravan redoslijed i sprema ih u registar **R2**.

Znamenke se okreću tako da se u svakom koraku petlje izdvoje 4 najniža bita iz **R6** (to je viša znamenka konačnog zapisa u BCD-u) i stave u **R1**, a zatim se prebace u najniža 4 bita od **R2**. Registar **R6** se pomiče u desno tako da se u sljedećem koraku petlje dohvati sljedeća znamenka manje težine, a **R2** se pomiče u lijevo, tako da prethodne prebačene više znamenke dođu na više pozicije što je upravo ono mjesto na koje ih se želi dovesti okretanjem. Petlja se ponavlja sve dok u **R6** ima znamenaka, odnosno sve dok **R6** ne postane ništica. U registru **R5** nalazi se broj znamenki. Ovaj registar se inkrementira u prvoj petlji

programa prilikom računanja svake od znamenki, a onda se u petlji za spremanje u točan redoslijed umanjuje za jedan i koristi za određivanje kraja prolaza petlje.

### 2.1.8. Pretvorba iz 32-bitnog zapisa u pakiranom BCD-u u 24-bitni NBC (FRISC 2.2.14.)

Riješen: DA Težina: ★★★

Napisati program koji čita podatke iz izvorišnog bloka u memoriji i zapisuje ih u odredišni blok. Izvorišni blok počinje na adresi  $100_{16}$ , a odredišni blok na adresi  $200_{16}$ . Podatci u izvorišnom bloku zapisani su kao 32-bitni pakirani BCD-ovi, a podatke u odredišni blok treba pohraniti u 24-bitnom zapisu NBC. Brojeve koji se ne mogu prikazati zamijeniti podatkom 0. Izvorišni blok zaključen je podatkom 0, a odredišni blok treba zaključiti podatkom  $FFFFFF_{16}$ .

#### Prijedlog rješenja:

Za pretvorbu iz pakiranog BCD-a u NBC potrebno je iz broja zapisanog BCD-om odvojiti svaku znamenku, zapisanu u četiri bita podatka, i dodati je rezultatu pomnoženu s odgovarajućom težinom (na primjer:  $358 = 3 \cdot 100 + 5 \cdot 10 + 8 \cdot 1$ ). Umjesto množenja svake znamenke težinom, jednostavnije je svaki put prije dodavanja znamenke rezultat pomnožiti sa 10, pri čemu je početni rezultat jednak ničiti i potrebno je znamenke uzimati od većih težina prema nižima (na primjer:  $358 = (((0 \cdot 10 + 3) \cdot 10 + 5) \cdot 10 + 8)$ ). Množenje se kod procesora ARM 7 ostvaruje naredbom **MUL**. Nakon što je obrađeno svih 8 znamenaka, pretvorba je gotova i rezultat se može pohraniti u memoriju.

Pakirani 32-bitni BCD može prikazati 8 dekadskih znamenaka, odnosno može prikazati brojeve u rasponu od 0 do  $99999999_{10}$ . Za prikaz broja  $99999999_{10}$  pomoću NBC-a treba 27 bitova, pa se dio raspona ne može prikazati u 24 bita, na što je potrebno obratiti pažnju.

#### Rješenje:

	ORG	0	
PROGRAM	MOV	R1, #1<8	; staviti adresu izvora u R1
	MOV	R2, #2<8	; staviti adresu odredišta u R2
	LDR	R3, MASKA	; učitati masku u R3
PETLJA	LDR	R4, [R1], #4	; učitati podatak iz bloka u R4
	CMP	R4, #0	; je li pročitana oznaka kraja bloka?
	BEQ	KRAJ	; ako je, onda je gotovo
	MOV	R7, #0	; postaviti rezultat na 0
	MOV	R5, #8	; ponovi za svih 8 znamenaka pakiranog BCD-a
	MOV	R6, #0A	; broj 0A=10 za množenje
OSAM	MUL	R7, R7, R6	; pomnožiti rezultat s 10
	MOV	R4, R4, ROR #28	; pomaknuti na sljedeću najvišu znamenku
	AND	R8, R4, #0F	; odvojiti znamenku iz najniža 4 bita
	ADD	R7, R7, R8	; dodati znamenku rezultatu
	SUBS	R5, R5, #1	; svih 8 znamenaka obrađeno?
	BNE	OSAM	; nastaviti petlju
ISPITAJ	ANDS	R4, R3, R7	; odrediti je li broj moguće prikazati
	BEQ	SPREMI	; greška ako ima više od 24 bita

(nastavak na sljedećoj stranici)

	MOV	R7, #0	; ne može se prikazati, zamijeniti ničicom
SPREMI	STRB	R7, [R2], #1	; pohraniti prvi bajt rezultata
	MOV	R7, R7, ROR #8	; pomaknuti na sljedeći bajt
	STRB	R7, [R2], #1	; pohraniti drugi bajt rezultata
	MOV	R7, R7, ROR #8	; pomaknuti na sljedeći bajt
	STRB	R7, [R2], #1	; pohraniti treći bajt rezultata
	B	PETLJA	; nastaviti po bloku
KRAJ	MVN	R0, #0	; staviti oznaku kraja odredišnog bloka u R0
	STRB	R0, [R2], #1	; zaključiti odredišni blok
	STRB	R0, [R2], #1	; nije potrebno rotirati podatak...
	STRB	R0, [R2], #1	; ...jer se ništa time ne dobije
	HALT		
MASKA	DW	0FF000000	; pojava ovih bitova signalizira prijenos
IZVOR	ORG	100	
	DW	12345678, 99999999, 3344, 16777215, 16777216, 0	
ODRED	ORG	200	
	DW	0	

### Komentar rješenja:

Vanjska petlja u kojoj se obrađuju pojedini podatci iz bloka nalazi se na labeli **PETLJA**, na čijem početku se odmah provjerava je li učitana oznaka kraja bloka. Nakon toga se u unutrašnjoj petlji na labeli **OSAM** dohvaćaju znamenke BCD-broja (ima ih osam) te se množe brojem 10 i pribrajaju rezultatu. Na labeli **ISPITAJ** se ispituje je li dobiveni broj unutar 24-bitnog opsega. Na labeli **SPREMI** se dobiveni broj sprema u memoriju. Konačno, nakon što su obrađeni svi podatci, na labeli **KRAJ** se upisuje oznaka kraja u odredišni blok.

Ispravnost rezultata ispituje se provjerom najvišeg bajta rezultata: ako u njemu ima jedinica, rezultat je izvan opsega od 24 bita.

Pohranjivanje 24-bitnog rezultata u memoriju izvodi se pohranjivanjem svakog bajta posebno naredbom **STRB** – od nižih ka viših bajtovima. Svaki sljedeći bajt dobiva se rotacijom rezultata u desno za 8 mjesta, tako da se na najnižem bajtu registra dobije sljedeći viši bajt koji treba spremiti u memoriju.

Potrebno je obratiti dodatnu pažnju na naredbu **MOV R4, R4, ROR #28**. Tom naredbom se želimo pomaknuti za 4 mjesta ulijevo na sljedeću znamenku, ali budući da kod ARM 7 postoji samo rotacija u desno, pomak na sljedeću znamenku ćemo obaviti rotacijom za 28 mjesta udesno (32 – 4 jer je rotacija ulijevo za N mjesta ekvivalentna rotaciji udesno za 32 – N mjesta). Broj rotacija 28<sub>10</sub> se u naredbi zapisuje u dekadskoj bazi jer se pomaci i rotacije u ATLAS-u zapisuju kao dekadski brojevi. Prilikom prvog prolaska kroz petlju ta naredba priprema buduću najvišu znamenku s kojom algoritam treba započeti.

Na labeli **KRAJ** zapisuje se oznaka kraja u odredišnom bloku, podatak 0xFFFFFFFF. Ovaj podatak se u registar **R0** upisuje pomoću naredbe **MVN R0, #0** koja komplementiranu vrijednost drugog operanda upisuje u odredišni registar, što znači da će nakon ove naredbe u registru **R0** biti upisane sve jedinice.

## 2.2. Operacije s podacima u računalu

U prethodnom potpoglavlju pokazano je kako se brojevi mogu prikazati u različitim zapisima. Ovo potpoglavlje pokazuje kako se izvode jednostavnije aritmetičke operacije na podacima prikazanim pomoću različitih zapisa. ARM 7, kao i drugi procesori, ima aritmetičke naredbe koje rade s brojevima u zapisima NBC i 2'k. Ako je broj u nekom drugom zapisu, treba ga ili pretvoriti u jedan od ova dva zapisa, ili programski ostvariti potrebnu operaciju. Potpoglavlje uključuje sljedeće gradivo:

- aritmetičke operacije (zbrajanje, oduzimanje) i greške kod aritmetičkih operacija
- ispitivanje i usporedba brojeva
- višestruka preciznost i proširivanje brojeva
- množenje i dijeljenje brojeva.

### 2.2.1. Zbrajanje parova brojeva (FRISC 2.3.1.)

Riješen: DA    Težina: ★

Napisati program koji zbraja 32-bitne parove podataka koji su slijedno zapisani u bloku memorije od lokacije  $1000_{16}$  do lokacije  $1020_{16}$  (ova lokacija nije uključena u blok). Prvi par brojeva nalazi se na adresama  $1000_{16}$  i  $1004_{16}$ . Zbrojevi parova spremaju se u blok memorije od lokacije  $1020_{16}$ .

#### Prijedlog rješenja:

U zadatku nije zadano u kojem zapisu su brojevi pa možemo pretpostaviti da su u NBC-u ili u zapisu 2'k (rješenje je jednako za oba slučaja). Također nije zadano što se događa u slučaju prekoračenja 32 bitnog opsega pa pretpostavljamo da do njega neće doći. Ako bi se željelo prepoznati prekoračenje opsega, to bi se za NBC radilo drugačije nego za 2'k, pa bi u tom slučaju trebalo točno znati u kojem od ova dva zapisa su brojevi koje zbrajamo.

U programu se registar **R0** koristi kao bazni adresni registar pa se u njega stavlja početna adresa bloka ( $1000_{16}$ ). Program koristi sintaksu programskog paketa ATLAS pa je u zapisu neposredne vrijednosti  $1000_{16}$  potrebno navesti broj (prije znaka "<") te vrijednost pomaka koja je podrazumijevana u dekadskoj bazi (poslije znaka "<"). Prvi i drugi podatak iz para se adresiraju sa postindeksiranjem te se zbrajaju i spremaju u određeni blok adresiran pomoću baznog registra **R1**. Petlja završava kad adresni registar **R0** dosegne graničnu lokaciju bloka podataka spremljenu u **R4** ( $1020_{16}$ ).

#### Rješenje:

	ORG	0	
	MOV	R0, #1<12	; R0= $1000_{16}$ adresni registar za prvi blok
	ADD	R1, R0, #20	; R1= $1020_{16}$ adresni registar za drugi blok
	MOV	R4, R1	; R4 će poslužiti za ispitivanje kraja
PETLJA	LDR	R2, [R0], #4	; učitavanje prvog broja, povećanje adrese
	LDR	R3, [R0], #4	; učitavanje drugog broja, povećanje adrese
	ADD	R3, R2, R3	; zbrajanje dva broja
	STR	R3, [R1], #4	; spremi zbroj u drugi blok, povećanje adrese

(nastavak na sljedećoj stranici)

	CMP	R0, R4	; ispitivanje kraja petlje
	BLT	PETLJA	; dok je R0 < 1020, petlja se ponavlja
KRAJ	HALT		
	ORG	1000	
	DW	1, 2, 3, 4, 5, 6, 7, 8	

**Komentar rješenja:**

U programu smo za dohvat podataka i spremanje rezultata koristili registarsko indirektno adresiranje s postindeksiranim odmakom što znači da se adresni registar (u uglatim zagradama) nakon dohвата podatka mijenja za odmak specificiran kao neposredna vrijednost. U konkretnom programu, registri **R0** i **R1** su se uvećavali za 4 nakon svake naredbe dohвата ili spremanja rezultata.

**2.2.2. Zbrajanje NBC-brojeva i prekoračenje opsega (FRISC 2.3.2.)****Riješen: DA    Težina: ★**

Na adresama 3300<sub>16</sub> i 4400<sub>16</sub> pohranjena su dva bloka sa po 512<sub>10</sub> podataka u 32-bitnom zapisu NBC. Napisati program koji redom čita po jedan podatak iz svakog od dva bloka, zbraja ta dva podatka i njihov zbroj zapisuje u odredišni blok na adresi 5500<sub>16</sub>. U slučaju prekoračenja opsega kod zbrajanja, u odredišni blok treba zapisati podatak FFFFFFFF<sub>16</sub>.

**Prijedlog rješenja:**

Rješenje je slično kao u prethodnom zadatku s dodatnom provjerom prekoračenja opsega nakon operacije zbrajanja. Prekoračenje opsega kod zbrajanja dva broja u zapisu NBC označeno je postavljenom zastavicom **C** nakon izvedene operacije zbrajanja.

**Rješenje:**

	ORG	0	
GLAVNI	MOV	R4, #33<8	; staviti adresu prvog bloka u R4
	MOV	R5, #44<8	; staviti adresu drugog bloka u R5
	MOV	R6, #55<8	; staviti adresu bloka rezultata u R6
	LDR	R3, BROJ	; staviti broj podataka u R3
	MOV	R0, #0	; inicijalizirati zbroj
PET	LDR	R1, [R4], #4	; podatak iz prvog bloka, povećati adresu
	LDR	R2, [R5], #4	; podatak iz drugog bloka, povećati adresu
	ADDS	R0, R1, R2	; zbrojiti podatke u R0
GRESKA	BCC	OK	; ako je došlo do greške, postavljen je C
	MVN	R0, #0	; zamijeniti rezultat s 0FFFFFFF
OK	STR	R0, [R6], #4	; pohraniti rezultat, povećati adresu
	SUBS	R3, R3, #1	; smanjiti brojač
	BNE	PET	; vratiti se na početak petlje
KRAJ	HALT		
BROJ	DW	%D 512	

(nastavak na sljedećoj stranici)



BLOK1	ORG	3300	; prvi blok
	DW	1, 5, 7, 13	; ... i još 508 <sub>10</sub> podataka
BLOK2	ORG	4400	; drugi blok
	DW	90, 2901, 0FFFFFFF9, 0FABD	; ... i još 508 <sub>10</sub> podataka
BLOK3	ORG	5500	; odredišni blok
	DS	%D 2048	

**Komentar rješenja:**

Rješenje u petlji koja se ponavlja 512 puta čita podatke iz prvog i drugog bloka, zbraja ih, te nakon zbrajanja provjerava zastavicu **C** (naredba **BCC**). U slučaju greške (postavljena zastavica, nije ispunjen uvjet **CC** - *Carry Clear*), rezultat nije valjan i zamjenjuje se brojem FFFFFFFF korištenjem naredbe **MVN R0, #0**. Registar **R3** nam služi kao brojač i u svakom prolazu petlje ga umanjujemo za 1 naredbom **SUBS**. Primijetite da naredba ima sufiks **S** na kraju čime se zadaje da nakon oduzimanja treba osvježiti zastavice što je u ovom slučaju potrebno jer se zastavice ispituju sljedećom naredbom **BNE**. Kako bi trebalo promijeniti program da se u odredišni blok umjesto zbroja želi staviti razlika podataka (uz provjeru prekoračenja opsega)?

**2.2.3. Oduzimanje 2'k-brojeva i prekoračenje opsega (FRISC 2.3.3.)**

Riješen: DA Težina: ★

U memoriji su na adresama 1000<sub>16</sub> i 4000<sub>16</sub> pohranjena dva bloka od po 200<sub>16</sub> 32-bitnih podataka u zapisu 2'k. Napisati program koji čita po jedan podatak iz svakog bloka, oduzima ih i rezultat zapisuje u odredišni blok od adrese 2000<sub>16</sub>. U slučaju prekoračenja opsega kod oduzimanja, u odredišni blok treba zapisati podatak 0.

**Prijedlog rješenja:**

U petlji će se učitavati podatci iz blokova, oduzimati i spremati u odredišni blok. Prije spremanja treba provjeriti prekoračenje opsega kod oduzimanja. U zapisu 2'k se prekoračenje kod oduzimanja (ili zbrajanja) može prepoznati ako dođe do preljeva (*overflow*), što se kod procesora ARM 7 vidi po postavljenoj zastavici **V** (ovu zastavicu ispitujemo uvjetima **VS** i **VC**).

Rješenje:			
GLAVNI	ORG	0	
	MOV	R0, #1<12	; staviti adresu prvog bloka u R0
	MOV	R1, #4<12	; staviti adresu drugog bloka u R1
	MOV	R2, #2<12	; staviti adresu bloka rezultata u R2
	MOV	R3, #2<8	; staviti broj podataka u R3
PET	LDR	R4, [R0], #4	; učitati prvi podatak, uvećati adresu
	LDR	R5, [R1], #4	; učitati drugi podatak, uvećati adresu
	SUBS	R6, R4, R5	; oduzeti podatke u R6
	BVC	OK	; ako nije došlo do greške, skok na OK
GRESKA	MOV	R6, #0	; greška: zamijeniti rezultat s 0
OK	STR	R6, [R2], #4	; pohraniti rezultat, uvećati adresu

(nastavak na sljedećoj stranici)

	SUBS	R3, R3, #1	; smanjiti brojač
	BNE	PET	; vratiti se na početak petlje
KRAJ	HALT		
	ORG	1000	; prvi blok
BLOK1	DW	1, 19, 27, %D13	; ...
	ORG	2000	; mjesto za odredišni blok
BLOK3	DS	800	
	ORG	4000	; drugi blok
BLOK2	DW	90, %D2901, %D 14, %D 1200430	; ... i još 1FC podataka

**Komentar rješenja:**

Naredba za oduzimanje podataka treba osvježiti zastavice kako bi se ispitala greška u naredbi **BVC OK**. Zato naredba oduzimanja **SUBS R6,R4,R5** ima nastavak **S**. Bez njega program ne bi ispravno radio.

Riješite zadatak ako se u odredišni blok trebaju spremati zbrojevi, a ne razlike podataka, pri čemu i dalje treba prepoznati grešku kod operacije. Koliko naredaba ste promijenili u odnosu na ponuđeno rješenje?

**2.2.4. Ispitivanje podataka u zapisu 2'k (FRISC 2.3.4.)****Riješen: DA    Težina: ★**

Napisati program koji čita podatke iz bloka pohranjenog na adresi 10000<sub>16</sub>. Podatke koji su pozitivni treba zamijeniti podatkom 1, podatke koji su negativni podatkom -1, a podatak 0 ostaviti nepromijenjenim. Izvorišni blok zaključen je podatkom 1234567<sub>16</sub>.

**Prijedlog rješenja:**

Budući da su podaci u bloku i pozitivni i negativni, možemo pretpostaviti da su u zapisu 2'k jer nije drugačije zadano.

Rješenje:			
	ORG	0	
GLAVNI	MOV	R1, #10<12	; staviti adresu bloka u R1
	LDR	R2, OZNAKA	; pročitati oznaku kraja u R2
PETLJA	LDR	R0, [R1], #4	; pročitati podatak iz bloka u R0 i povećati R1
	CMP	R0, R2	; usporediti s oznakom kraja
	BEQ	GOTOVO	; završiti program
	CMP	R0, #0	; usporediti podatak s 0
	BGT	POZIT	; strogo veći od 0 (signed greater than)
	BLT	NEGAT	; strogo manji od 0 (signed less than)
	B	PETLJA	; nastaviti petlju
POZIT	MOV	R0, #1	; staviti 1 u R0
	STR	R0, [R1, #-4]	; spremi na staro mjesto, R1 se ne mijenja
	B	PETLJA	; nastaviti petlju
NEGAT	MVN	R0, #0	; staviti 1 u R0
	STR	R0, [R1, #-4]	; spremi na staro mjesto, R1 se ne mijenja
	B	PETLJA	; nastaviti petlju

(nastavak na sljedećoj stranici)

GOTOVO	HALT		; kraj
OZNAKA	DW	1234567	; oznaka kraja, ne može se direktno ; upisati u registar!
	ORG	10000	
BLOK	DW	0, 1, 165, 0FA9, 56AF, 1A4B, 123456	

**Komentar rješenja:**

Rješenje se temelji na naredbama koje postavljaju zastavice kao rezultat uspoređivanja dva broja (**CMP**) i zatim ispituju stanje zastavica (**B{uvjet}**). Primijetite da se uz naredbe **B** koriste uvjeti za usporedbu brojeva u zapisu 2'sk (signed), obzirom da je u zadatku zadano da podaci mogu biti pozitivni i negativni brojevi. Oznaka kraja 1234567<sub>16</sub> ne može se napisati kao neposredni broj (8 bita s parnim pomakom ulijevo), pa je nije moguće navesti direktno unutar naredbe (npr. **CMP R0, #1234567**), već se treba učitati kao konstanta iz memorije (naredba **LDR R2, OZNAKA**).

**2.2.5. Preslikavanje pozitivnih brojeva u novi blok (FRISC 2.3.5.)****Riješen: DA    Težina: ★**

Napisati program koji čita 32-bitne podatke iz bloka pohranjenog na adresi 800<sub>16</sub>. Podatci u bloku su u zapisu s bitom za predznak. Pozitivne podatke treba prepisati u odredišni blok od adrese 1000<sub>16</sub>, a negativni podatci se preskaču. Blok je zaključen podatkom -1. Oznaka kraja bloka se prepisuje u odredišni blok.

**Prijedlog rješenja:**

Za određivanje je li broj pozitivan ili negativan može se upotrijebiti zastavica **N** (uvjeti **MI** i **PL**) jer je bit za predznak na najvišem mjestu u registru. Program u petlji čita podatke iz bloka i provjerava je li pročitana oznaka kraja bloka (to je uvjet za kraj petlje). Zatim se provjerava predznak broja. Ako je broj pozitivan (ispunjen uvjet **PL**), uvjetno se izvodi naredba **STRPL R1, [R5], #4** koja sprema podatak u odredišni blok i povećava adresu odredišta. U slučaju da uvjet **PL** nije ispunjen, odnosno u slučaju negativnog podatka, ova naredba se neće izvesti. Potrebno je pripaziti kako se podatak -1 prikazuje u zapisu s bitom za predznak (labela **MINUS1**).

**Rješenje:**

	ORG	0	
GLAVNI	MOV	R4, #8<8	; staviti adresu izvorišnog bloka u R4
	MOV	R5, #1<12	; staviti adresu odredišnog bloka u R5
	LDR	R6, MINUS1	; učitati oznaku kraja bloka u R6
PET	LDR	R1, [R4], #4	; učitati podatak, uvećati adresu izvorišta
	CMP	R1, R6	; je li učitana oznaka kraja?
	BEQ	KRAJ	
	CMP	R1, #0	; je li učitani broj pozitivan ; ako je negativan, preskočiti
	STRPL	R1, [R5], #4	; prepisati podatak, uvećati adresu odredišta
	B	PET	; nastaviti petlju
KRAJ	STR	R6, [R5]	; zaključiti odredišni blok
	HALT		
(nastavak na sljedećoj stranici)			
MINUS1	DW	80000001	; -1 zapisan kao 32-bitni BZP

IZVOR	ORG	800
	DW	5, 99, 80000011, 80000001
ODRED	ORG	1000
	DW	0

Specifičnost procesora ARM je da se sve naredbe (osim naredbe **NOP**) mogu izvoditi uvjetno (za razliku od procesora FRISC kod kojega se uvjetno mogu izvoditi samo upravljačke naredbe). To svojstvo je iskorišteno u naredbi **STRPL R1, [R5], #4**.

Riješite zadatak tako da su brojevi u zapisu 1'k, blok je zaključen „negativnom ničicom“, a pozitivni podatci se ne zanemaruju nego se i oni zapisuju u odredišni blok, ali promijenjenog predznaka.

### 2.2.6. Usporedba 2'k-brojeva i upis većeg u novi blok (FRISC 2.3.6.)

Riješen: DA Težina: ★

Napisati program koji čita i uspoređuje podatke iz dva bloka pohranjena na adresama 6000<sub>16</sub> i 8000<sub>16</sub>. Veći od dva podatka treba zapisati u odredišni blok na adresi 10000<sub>16</sub>. Blokovi sadrže 100<sub>10</sub> podataka zapisanih u obliku 32-bitnog zapisa 2'k.

#### Prijedlog rješenja:

Dva broja u zapisu 2'k usporedit će se naredbom **CMP**, te će se provjeriti odgovarajući uvjet za usporedbu brojeva s predznakom (**GT**).

#### Rješenje:

GLAVNI	ORG	0	
	MOV	R1, #6<12	; staviti adresu prvog bloka u R1
	MOV	R2, #8<12	; staviti adresu drugog bloka u R2
	MOV	R3, #10<12	; staviti adresu odredišnog bloka u R3
	MOV	R0, #D 100	; staviti broj podataka u R0
PET	LDR	R4, [R1], #4	; pročitati prvi, uvećati adresu bloka
	LDR	R5, [R2], #4	; pročitati drugi, uvećati adresu bloka
	CMP	R4, R5	; usporediti podatke
	STRGT	R4, [R3], #4	; ako je prvi veći, spremiti ga
	STRLE	R5, [R3], #4	; inače, spremiti drugi
	SUBS	R0, R0, #1	; smanjiti brojač
	BNE	PET	; nastaviti ako nije kraj
KRAJ	HALT		
BLOK1	ORG	6000	
	DW	14, -267B, 19, -100 ; ...i još 96 <sub>10</sub> podataka	
BLOK2	ORG	8000	
	DW	15, -267A, 19, -100 ; ...i još 96 <sub>10</sub> podataka	
REZ	ORG	10000	
	DW	0	

**Komentar rješenja:**

Rješenje u petlji čita po jedan podatak iz svakog bloka i uspoređuje ih naredbom **CMP**. Naredba **STRGT** će se izvesti u slučaju da je podatak u **R4** strogo veći od podatka u **R5**, dok će se naredba **STRLE** izvesti u obratnom slučaju, čime smo postigli da se u odredišni blok upisuje veći od dva dohvaćena podatka. U slučaju da su podaci jednaki, upisat će se podatak iz drugog bloka (budući da u zadatku nije drugačije zadano, svejedno je koji ćemo podatak upisati ako su jednaki). U svakom prolazu petlje izvest će se samo jedna od ove dvije naredbe, ovisno o tome koji od uvjeta će biti ispunjen. Obje naredbe **STR**, nakon spremanja većeg broja, uvećavaju i adresu odredišta koja se nalazi u registru **R3**.

Promijenite program tako da se u slučaju jednakosti brojeva u odredišni blok ništa ne upisuje. Promijenite program tako da se ne koristi uvjetno izvođenje memorijskih naredbi.

Da je u originalnom zadatku bilo zadano da su podatci u zapisu NBC, kako bi morali promijeniti ponuđeno rješenje?

**2.2.7. Prebrajanje negativnih 2'k brojeva u bloku (FRISC 2.3.7.)**

**Riješen: DA    Težina: ★**

U bloku memorije koji počinje od adrese  $1000_{16}$  prebrojiti koliko ima podataka manjih od 0 i taj broj treba upisati od adrese  $300_{16}$ . Blok ima  $200_{16}$  podataka u zapisu 2'k.

**Prijedlog rješenja:**

Pretpostavlja se da su podatci širine 32-bita, obzirom da nije zadano drugačije. Najjednostavnije rješenje je da se u petlji ispituju svi podatci u bloku i da se za svaki negativni podatak poveća brojač negativnih brojeva.

Rješenje:			
	ORG	0	
	MOV	R0, #1<12	; R0 adresa početka bloka
	MOV	R1, #2<8	; R1 brojač za petlju
	MOV	R2, #0	; R2 brojač negativnih
PETLJA	LDR	R3, [R0], #4	; dohvatiti podatak u R3
	CMP	R3, #0	; usporediti ga s ničticom
NEGAT	ADDLT	R2, R2, #1	; ako je negativan, povećati brojač negativnih
	SUBS	R1, R1, #1	; smanjiti brojač petlje
	BNE	PETLJA	; ako nije kraj, onda ponoviti
	MOV	R0, #3<8	
	STR	R2, [R0]	; spremiti rezultat
	HALT		

**Komentar rješenja:**

Registar **R0** služi kao pokazivač na podatke u bloku i na početku se postavlja da pokazuje na početak bloka, odnosno na prvi podatak u bloku. Registar **R1** služi kao brojač prolazaka kroz petlju, a **R2** kao brojač negativnih podataka.

Početak petlje označen je labelom **PETLJA** i ovdje se dohvaća podatak iz bloka te se uspoređuje s ničticom. Naredba **ADDLT R2,R2,#1** izvodi se ako je ispunjen uvjet da je

podatak u **R3** strogo manji od ničice. Inače, naredba se neće izvesti pa će program nastaviti s izvođenjem sljedeće naredbe za smanjivanje brojača prolazaka kroz petlju (**SUBS R1, R1, #1**). Ova naredba ima nastavak **S** kojim se specificira osvježavanje zastavice, što je potrebno jer sljedeća naredba **BNE** ispituje da li je brojač došao do 0 ili nije. Petlja se ponovno izvodi ako brojač prolazaka još nije dosegao 0.

Da li bi morali promijeniti program ako bi podatci u bloku bili u zapisu s bitom za predznak? Promijenite program tako da se ne prebrajaju negativni brojevi nego se prebraja u koliko slučajeva je broj u bloku veći od svojega sljedbenika (dakle, uspoređuju se prvi i drugi broj, zatim drugi i treći i tako dalje do predzadnjeg i zadnjeg broja). Dodatno, neka su podatci u zapisu s bitom za predznak (★★).

### 2.2.8. Zbrajanje 16-bitnih 2<sup>k</sup>-brojeva i pretvorba u 32-bitni zapis s bitom za predznak (FRISC 2.3.8.)

Riješen: DA    Težina: ★★

U memoriji se nalazi blok 16-bitnih podataka u zapisu dvojnog komplementa. Početna adresa bloka zapisana je u 4 memorijske lokacije od memorijske adrese  $1000_{16}$ . Veličina bloka podataka zapisana je od memorijske adrese  $1004_{16}$  u 4 sljedeće memorijske lokacije. Treba izračunati 32-bitni zbroj svih podataka u bloku. Nakon toga, taj zbroj treba pretvoriti u 32-bitni zapis s bitom za predznak i spremi ga u memoriju od adrese  $1008_{16}$ . Pretpostaviti da pri zbrajanju podataka i pretvorbe u zapis BZP neće doći do prekoračenja opsega.

Napišite program koji rješava zadani zadatak za blok memorije koji počinje od adrese  $30000_{16}$  i ima  $250_{16}$  podataka.

#### Prijedlog rješenja:

Budući da se početna adresa bloka i broj podataka nalaze na adresama  $1000_{16}$  i  $1004_{16}$ , prvo ih valja učitati u registre (**R0** i **R1**) za korištenje u petlji. Prethodno ćemo u registar **R4** upisati vrijednost  $1000_{16}$  te ćemo ga koristiti za adresiranje u naredbama **LDR**. 16-bitni podatci će se čitati naredbom **LDRSH** koja predznačno proširuje učitano poluriječ (16-bitni podatak). Konačni zbroj treba iz zapisa 2<sup>k</sup> pretvoriti u zapis s bitom za predznak. Ako je zbroj pozitivan, nije potrebna nikakva pretvorba jer su oba zapisa u tom slučaju jednaka. Ako je zbroj negativan, pretvorba je potrebna, a provodi se tako da se izračuna apsolutna vrijednost zbroja, a nakon toga se u najviši bit postavi jedinica.

#### Rješenje:

```

ORG      0
MOV      R4, #1<12      ; R4 = 100016
LDR      R0, [R4], #4    ; R0 je pokazivač na blok, zatim R4=100416
LDR      R1, [R4], #4    ; R1 je brojač za petlju, zatim R4=100816
MOV      R2, #0          ; u R2 će biti zbroj
LOOP     LDRSH R3, [R0], #2 ; učitaj 16-bitni 2k broj, povećaj adresu
        ADD     R2, R2, R3 ; pribrajanje proširenog broja u zbroj
        SUBS    R1, R1, #1 ; smanjivanje brojača za petlju
        BNE     LOOP      ; ispitivanje kraja petlje

        ; pretvori sumu u R2 iz 2k u zapis s bitom za predznak
BZP      ORRS    R2, R2, R2 ; postaviti zastavicu N
        BPL     POZIT     ; ako je pozitivan, ne treba pretvarati

```

(nastavak na sljedećoj stranici)

; ako je negativan, treba pretvoriti rezultat			
NEGAT	RSB	R2, R2, #0	; dvojni komplement
	MOV	R2, R2, LSL #1	; najviši bit će se upisati na najniže mjesto
	ORR	R2, R2, #1	; upis najvišeg bita na najniže mjesto
	MOV	R2, R2, ROR #1	; poravnavanje
POZIT	STR	R2, [R4]	; spremanje rezultata na 1008
	HALT		
	ORG	1000	
	DW	30000	; adresa bloka
	DW	250	; broj podataka u bloku
	DW	0	; mjesto za rezultat

**Komentar rješenja:**

Nakon što je pribrajanje završeno, ispitujemo predznak sume te u slučaju negativnog podataka vršimo pretvorbu u format s bitom za predznak. Postavljanje negativnog predznaka u zapisu s bitom za predznak je zapravo upisivanje jedinice u najviši bit zajedno sa upisom apsolutne vrijednosti podatka u ostale bitove. U rješenju je to ostvareno tako da se podatak prvo oduzme od ničice (naredba **RSB**), čime se zapravo izvede operacija dvojnog komplementa, te se u **R2** dobije apsolutna vrijednost broja. Nakon toga se registar **R2** pomakne za jedno mjesto ulijevo pri čemu se u najniži bit upiše ničica, a zatim se naredbom **ORR** s maskom 1 u najniži bit postavi jedinica. Konačno se desnom rotacijom najniži bit s upisanom jedinicom dovede na položaj najvišeg bita, tj. na položaj bita za predznak. Postavljanje najvišeg bita se moglo jednostavnije ostvariti i naredbom **ORR** s maskom  $80000000_{16}$ .

### 2.2.9. Proširivanje brojeva u zapisu s bitom za predznak sa 16 na 32 bita (FRISC 2.3.9.)

Riješen: DA    Težina: ★★

U memoriji se nalazi blok 16-bitnih brojeva u zapisu s bitom za predznak. Adresa početka bloka zapisana je na adresi  $1000_{16}$ . Blok podataka završava podatkom  $8000_{16}$  (ovaj podatak se ne smatra brojem u bloku).

Napisati program koji pretvara 16-bitne brojeve u 32-bitne te ih sprema istim redoslijedom u novi blok, počevši od adrese  $5000_{16}$ . Brojeve treba pretvarati na sljedeći način: svaki negativni broj treba pretvoriti u 32-bitni broj u zapisu s bitom za predznak. Umjesto svakog pozitivnog broja, program u novi blok treba zapisati 32-bitni broj 0.

Novi blok treba zaključiti brojem  $80000000_{16}$ . Prilikom pretvorbe brojeva, program također treba prebrajati koliko je parnih 16-bitnih brojeva bilo u početnom bloku i to treba zapisati na memorijsku lokaciju **PARNI**.

**Prijedlog rješenja:**

Predznak izvorišnog podataka može se odrediti ispitivanjem najvišeg bita na poziciji 15. Ako je u tom bitu jedinica, broj je negativan i potrebno ga je proširiti na 32 bita. Proširivanje se u ovom slučaju obavlja tako da se prvo obriše bit 15, a zatim se u bit 31 upiše jedinica. Ispitivanje parnosti svodi se na ispitivanje najnižeg bita u podatku, jer parni brojevi u tom bitu imaju ničicu, a neparni jedinicu.

Rješenje:				
	ORG	0		
	MOV	R7, #1<12		
	LDR	R0, [R7]		; R0 - početak izvorišnog bloka
	MOV	R3, #5<12		; R3 - početak odredišnog bloka
	MOV	R6, #0		; R6 - brojač parnih brojeva
	LDR	R5, MASKA1		; maska za postavljanje 31. bita(i oznaka kraja)
	LDR	R8, MASKA2		; maska za brisanje 15. bita
POC	LDRH	R1, [R0], #2		; učitavanje 16-bitnih brojeva, uvećaj adresu
	CMP	R1, #8<12		; ako je učitani broj 8000...
	BEQ	KRAJ		; ... pročitan je cijeli blok
PREDZ	ANDS	R2, R1, #8<12		; ispitati bit za predznak (bit 15)
POZIT	MOVEQ	R2, #0		; ako je rezultat 0, pozitivne brojeve brisati
NEGAT	ANDNE	R2, R1, R8		; a neg. brojevima brisati bit 15 (maska u R8)
	ORRNE	R2, R2, R5		; ... i postaviti bit 31 pomoću maske (R5)
PARNOST	ANDS	R1, R1, #1		; provjeriti parnost ispitivanjem bita 0
PARAN	ADDEQ	R6, R6, #1		; ako je rez. ništica, povećaj brojač parnih
SPREMI	STR	R2, [R3], #4		; spremi broj u odredište, uvećaj adresu
	B	POC		; povratak na učitavanje
KRAJ	STR	R5, [R3]		; zaključiti blok sa 80000000
	STR	R6, PARNI		; na lokaciju PARNI staviti broj parnih
	HALT			
MASKA1	DW	80000000		
MASKA2	DW	7FFF		
PARNI	DW	0		; mjesto za broj parnih (može i DS 4)

### Komentar rješenja:

U ovom zadatku smo koristili uvjetno izvođenje naredaba (slično kao u zadatku 2.2.5.) koje nam omogućuje da program napišemo pomoću manjeg broja naredaba. Kao primjer možemo vidjeti odsječak programa u kojem ispitujemo predznak naredbom **ANDS R2,R1,#8<12**. Nakon ove naredbe slijede naredbe koje koriste uvjetno izvođenje: **MOVEQ** (izvodi se ako je postavljena zastavica **Z**, odnosno rezultat prethodne naredbe **ANDS** ništica, što znači da je podatak pozitivan), a naredbe **ANDNE** i **ORRNE** (izvode se ako je zastavica **Z** obrisana, odnosno u slučaju da je rezultat naredbe **ANDS** različit od ništice, što znači da je podatak negativan). Na isti način se uvjetno izvode naredbe za uvećavanje brojača parnih podataka (**ADDEQ R6,R6,#1**) u odnosu na naredbu ispitivanja najnižeg bita **ANDS R1,R1,#1**. Pri tome, nužno je da ova naredba osvježi zastavice, kako bi se uvjetno izvođenje sljedećih naredbi ispravno izvelo.

Za dohvat podataka maske (**MASKA1**, **MASKA2**) koristili smo naredbe **LDR** koje u adresnom polju imaju labelu (bez uglatih zagrada). Ove naredbe su specifične po tome što će ih asemblerski prevoditelj prevesti u oblik u kojem kao bazni adresni registar koristi registar **PC**



uvećan za odgovarajući odmak kojeg će prevoditelj sam izračunati. Isti slučaj imamo sa naredbom za spremanje broja parnih podataka na labelu **PARNI** (**STR R6,PARNI**).

### 2.2.10. Zbroj po bajtovima i zamjena podataka u bloku (FRISC 2.3.10.)

Riješen: DA    Težina: ★

Napisati program koji čita 32-bitne podatke iz memorije počevši od lokacije  $1000_{16}$ , sve dok ne pročita oznaku kraja – podatak  $12345678_{16}$ . Svaki od pročitanih podataka zamijeniti s novom 32-bitnom vrijednošću koja se dobije na sljedeći način: svaki od 4 bajta pročitano podatka sadrži broj u zapisu  $2^k$  (svaki broj je širine 8 bita). Ta četiri broja treba zbrojiti u 32-bitnu vrijednost, te rezultat pohraniti u memoriju umjesto pripadnog originalnog podataka. Oznaku kraja nije potrebno zamijeniti novom vrijednošću.

Primjer: u memoriji na lokaciji  $1020_{16}$  nalazi se podatak  $010203FF_{16}$ . Nakon izvođenja programa na lokaciji  $1020_{16}$  bit će upisana vrijednost  $00000005_{16} = 1+2+3+(-1)$ . Analogno za ostale memorijske lokacije.

#### Prijedlog rješenja:

Iako bi se iz memorije mogli učitavati 32-bitni podatci naredbom **LDR** i zatim rastavljati na bajtove te nakon toga predznačno proširivati, u ovom rješenju će se svaki bajt učitavati zasebno naredbom **LDRSB** koja automatski predznačno proširuje dohvaćeni bajt.

Rješenje:			
GLAVNI	ORG	0	
	LDR	R1, OZNAKA	; učitati oznaku kraja
	MOV	R0, #1<12	; početak bloka
PETLJA	LDR	R2, [R0]	; učitati podatak
	CMP	R1, R2	; provjeriti oznaku kraja
	BEQ	KRAJ	
ZBROJ	LDRSB	R2, [R0]	; učitati prvi bajt
	LDRSB	R3, [R0, #1]	; učitati drugi bajt, ne mijenjati adresu
	ADD	R2, R3, R2	; zbrojiti
	LDRSB	R3, [R0, #2]	; učitati treći bajt, ne mijenjati adresu
	ADD	R2, R3, R2	; zbrojiti
	LDRSB	R3, [R0, #3]	; učitati četvrti bajt, ne mijenjati adresu
	ADD	R2, R3, R2	; zbrojiti
	STR	R2, [R0], #4	; spremanje zbroja, uvećati adresu
	B	PETLJA	; nastavak petlje
KRAJ	HALT		
OZNAKA	DW	12345678	
	ORG	1000	
	DW	1111334, 0FE004, 11006D5E, 12345678	

#### Komentar rješenja:

Svaki od četiri bajta obrađuje se na jednak način. Prvo se bajt učitava naredbom **LDRSB**, pri čemu je adresa dobivena pomoću baznog registra **R0** i odmak zadano neposrednom

vrijednošću (0, 1, 2 i 3). Pri tome se vrijednost baznog registra ne mijenja. Budući da su podatci u bajtovima u zapisu  $2^k$ , moraju se predznačno proširiti na 32 bita, jer naredba zbrajanja radi sa 32-bitnim podatcima (zbog toga se koristi naredba **LDRSB**, a ne **LDRB**).

Svaki broj dobiven proširivanjem pribraja se u registar **R2** koji se na kraju petlje sprema u blok memorije preko originalnog podatka. U naredbi spremanja koristimo postindeksirano adresiranje baznog registra **R0** neposrednom vrijednošću 4, tako da će se bazni registar nakon spremanja uvećati za 4, te pokazivati na sljedeći podatak u bloku koji se obrađuje.

Uočite koje se slične naredbe ponavljaju u petlji **ZBROJ** i riješite zadatak tako da izbjegnute ovo ponavljanje.

### 2.2.11. Zbrajanje više podataka u zapisu s bitom za predznak (FRISC 2.3.11.)

Riješen: DA    Težina: ★★

U memoriji se nalazi blok 32-bitnih podataka u zapisu s bitom za predznak. Blok podataka počinje podatkom koji se nalazi na adresi  $1000_{16}$ , a završava podatkom na adresi  $1100_{16}$ . Napisati program koji će zbrojiti sve podatke iz bloka. Zbroj treba biti u 64-bitnom zapisu  $2^k$  i treba ga spremiti od adrese  $2000_{16}$  u poretku *big-endian*.

#### Prijedlog rješenja:

Podatci zauzimaju  $104_{16}$  bajta (prvi podatak počinje na  $1000_{16}$ , a zadnji na  $1100_{16}$ ), pa je u bloku ukupno  $104_{16}/4_{16} = 41_{16}$  podatak širine 32 bita. Prije zbrajanja potrebno je napraviti pretvorbu podataka u zapis  $2^k$ .

Zbog zbrajanja više brojeva može se lako dogoditi da 32 bita nisu dovoljna za zapis rezultata pa je zadano da rezultat mora biti 64 bitni. Zato ćemo zbrajanje provoditi u dvostrukoj preciznosti. Nakon pretvorbe u zapis  $2^k$  broj ćemo pomoću naredaba **ADD** i **ADC** pribrojiti trenutačnom 64 bitnom rezultatu.

#### Rješenje:

	ORG	0	
	MOV	R6, #0	; mjesto za zbroj, niži dio
	MOV	R7, #0	; mjesto za zbroj, viši dio
	MOV	R0, #1<12	; pokazivač podataka
	MOV	R1, #41	; brojač za petlju
	MVN	R8, #0	; R8 = -1
	LDR	R5, MASKA	; maska za brisanje bita predznaka
PETLJA	LDR	R3, [R0], #4	; učitati broj, uvećati pokazivač
	CMP	R3, #0	; ispitati predznak broja
NEG	ANDMI	R3, R3, R5	; ako je negativan, obrisati bit predznaka
	EORMI	R3, R3, R8	; i pretvoriti u $2^k$ : komplementirati pa
	ADDMI	R3, R3, #1	; dodati 1
	ADDNIS	R6, R6, R3	; zbrojiti niži dio broja, osvježava zastavice
	ADCMIS	R7, R7, R8	; viši dio broja sa 0xFFFFFFFF iz R8 i
			; prijenosom iz nižeg dijela
	BMI	NEXT	; nastaviti sa sljedećim brojem
POZIT	ADDS	R6, R6, R3	; ako je broj pozitivan zbrojiti niži dio broja
	ADC	R7, R7, #0	; viši dio s 0 i prijenosom iz nižeg dijela

(nastavak na sljedećoj stranici)

NEXT	SUBS BNE	R1, R1, #1 PETLJA	; smanjivanje brojača petlje
KRAJ	MOV MOV	R8, #20<8 R9, #8	; spremanje u poretku big endian ; brojač petlje
L1	MOV STRB SUB CMP MOVEQ CMP BNE HALT	R7, R7, ROR #24 R7, [R8], #1 R9, R9, #1 R9, #4 R7, R6 R9, #0 L1	; spremanje bajt po bajt ; provjera jesu li obrađena prva 32 bita ; ako da, obrađuju se druga 32 bita
MASKA	DW	7FFFFFFF	; maska za brisanje bita predznaka

### Komentar rješenja:

Program počinje inicijalizacijom potrebnih podataka: 64-bitnog zbroja koji se sprema u registre **R6** (nižih 32 bita) i **R7** (viših 32 bita), pokazivača bloka **R0**, brojača podataka **R1** i maske za brisanje bita za predznak **R5**.

Ispitivanje predznaka ostvareno je naredbom usporedbe podatka sa ničicom (**CMP R3, #0**) te uvjetnim izvođenjem sljedećih 5 naredaba uz uvjet **MI**. Ako je podatak u zapisu s bitom za predznak pozitivan (na najvišem bitu je ničica), on će se pribrojiti zbroju i to naredbom **ADDS** u nižih 32 bita (**ADDS R6, R6, R3**), te naredbom zbrajanja s prijenosom **ADC** na viših 32 bita (**ADC R7, R7, #0**). Ovaj odsječak je ostvaren kod labela **POZIT**. Ukoliko je dohvaćeni podatak negativan, potrebno ga je pretvoriti u dvojni komplement, što se ostvaruje brisanjem najvišeg bita i izvođenjem operacije dvojnog komplementa na tako dobivenom podatku (labela **NEG**). Pribrajanje u zbroj se u nižih 32 bita ostvaruje identično kao i kod pozitivnih podataka, dok je pribrajanje u viših 32 bita potrebno ostvariti s predznačno proširenim podatkom, odnosno brojem koji na viših 32 bita ima sve jedinice (broj -1 u registru **R8**).

Nakon što su na ovaj način obrađeni svi podatci iz bloka, na labeli **KRAJ** slijedi spremanje 64 bitnog zbroja u poretku *big endian*. Kod *big endiana*, se na prvu (najnižu) adresu sprema najviši bajt kojeg sačinjavaju podatci od bita 63 do bita 56, a oni se nalaze u najvišem bajtu registra **R7**. To je ostvareno u petlji kombinacijom rotacije podatka udesno za 24 mjesta, čime zapravo postizemo željenu rotaciju ulijevo za 8 ( $32 - 24 = 8$ ) mjesta (1 bajt). To činimo jer kod procesora ARM ne postoji rotacija ulijevo. Time najviši bajt (kojega prvo treba spremati) dolazi na položaj najnižeg bajta kojega se zatim naredbom **STRB R7, [R8], #1** sprema u memoriju od adrese  $2000_{16}$ . Zatim se sprema sljedeći bajt iz **R7** i tako dalje do najnižeg bajta – bajtovi se spremaju na slijedne lokacije u memoriji.

Umjesto da se nakon toga izvede još jedna petlja u kojoj se umjesto bajtova iz **R7** spremaju bajtovi iz **R6**, cijelo spremanje izvodi se u samo jednoj petlji od 8 koraka, ali se u tijelu petlje provjerava trenutak kad je gotovo spremanje višeg dijela rezultata, odnosno kad treba početi sa spremanjem nižeg dijela. To je učinjeno ispitivanjem brojača (**CMP R9, #4**) i uvjetnim prebacivanjem nižeg dijela rezultata iz registra **R6** u registar **R7** (**MOVEQ R7, R6**).

Prvo pokušajte preraditi tijelo glavne petlje programa (od labela **PETLJA**) tako da zamijenite redoslijed odsječaka za obradu pozitivnih brojeva (od labela **POZ**) i negativnih brojeva (od labela **NEG**). Zatim pokušajte napisati ovaj dio petlje bez uvjetnog izvođenja naredaba,

odnosno korištenjem obične naredbe uvjetnog skoka. Usporedite brzinu izvođenja sve tri inačice petlje!

### 2.2.12. Množenje NBC-broja s konstantom (FRISC 2.3.12.)

Riješen: DA    Težina: ★

Napisati program koji čita  $210_{16}$  podataka iz bloka pohranjenog na adresi  $10000_{16}$ . Svaki podatak treba pomnožiti brojem  $9_{10}$  uporabom naredaba za pomak i zbrajanje, te rezultat zapisati na isto mjesto u početni blok. Podatci u bloku su u 32-bitnom zapisu NBC. U slučaju prekoračenja opsega pri množenju, u bloku treba zadržati originalni podatak.

#### Prijedlog rješenja:

Množenje pomoću pomaka općenito je brže od množenja uzastopnim pribrajanjem (koje će biti pokazano kasnije), a izvedba je vrlo jednostavna u slučaju množenja s poznatim brojem (tj. s konstantom). Množenje brojem 9 izvodi se u više koraka – tri množenja sa 2 pomnožit će podatak sa 8, a nakon toga umnošku treba još dodati originalni podatak.

Budući da se množenje izvodi u nekoliko koraka, u svakom od njih treba provjeriti moguće prekoračenje opsega. Za NBC-brojeve se prekoračenje opsega prepoznaje postavljenom zastavicom prijenosa C, koju treba ispitati nakon svakog pomaka i zbrajanja.

#### Rješenje:

GLAVNI	ORG	0	
	MOV	R3, #10<12	; staviti adresu bloka u R3
	MOV	R1, #21<4	; staviti broj podataka u R1
PETLJA	LDR	R2, [R3], #4	; podatak iz bloka pročitati u R2, uvećaj adresu
			; množenje konstantom 9 (rezultat se akumulira u registru R4)
	MOVS	R4, R2, LSL #1	; pomaknuti za jedno mjesto ulijevo (R2*2)
	BCS	GRESKA	; ako je došlo do greške, skoči na GRESKA
	MOVS	R4, R4, LSL #1	; pomaknuti za jedno mjesto ulijevo (R2*4)
	BCS	GRESKA	; ako je došlo do greške, skoči na GRESKA
	MOVS	R4, R4, LSL #1	; pomaknuti za jedno mjesto ulijevo (R2*8)
	BCS	GRESKA	; ako je došlo do greške, skoči na GRESKA
	ADDS	R4, R2, R4	; R4 = R2 + 8*R2
	BCS	GRESKA	; ako je došlo do greške, nastaviti
	STR	R4, [R3, #-4]	; pohraniti na isto mjesto u bloku
GRESKA	SUBS	R1, R1, #1	; smanjiti brojač podataka
	BNE	PETLJA	; nastaviti petlju ako nismo gotovi
GOTOV	HALT		; kraj
BLOK	ORG	10000	
	DW	0, 0B69, 136DA	; ...i još 20D podataka

#### Komentar rješenja:

Ako se prekoračenje smije zanemariti, množenje se može pojednostavniti na dvije naredbe:

```
MOV    R4, R2, LSL #3
ADD    R4, R2, R4
```

U ovakvom rješenju se prekoračenje opsega može provjeriti prethodnim ispitivanjem tri najviša bita u podatku, odnosno onih bitova koji prilikom pomicanja izlaze iz registra. Ako je na bilo kojem od navedenih bitova jedinica, doći će do greške zbog prekoračenja. Greška prekoračenja nakon naredbe **ADD** provjerila bi se jednako kao u ponuđenom rješenju zadatka. Modificirajte rješenje tako da se za množenje koriste samo dvije naredbe, ali da se provjerava prekoračenje.

Promijenite zadatak tako da su ulazni podaci u dvojnog komplementu (★★).

### 2.2.13. Množenje NBC-broja s konstantom i spremanje u dvostruku preciznost (FRISC 2.3.13.)

Riješen: DA    Težina: ★★★

Napisati program koji iz bloka na adresi  $7000_{16}$  čita  $100_{16}$  podataka u obliku 32-bitnog zapisa NBC. Svaki podatak treba pomnožiti sa  $81920_{10}$  uporabom naredaba za rotaciju ili pomak. Rezultate zapisivati u odredišni blok na adresi  $10000_{16}$  u obliku 64-bitnog zapisa NBC. Broj  $81920_{10}$  zapisan binarno je  $1\ 0100\ 0000\ 0000\ 0000_2$ , odnosno  $2^{16} + 2^{14}$ , pa vrijedi da je  $X \cdot 81920 = X \cdot 2^{16} + X \cdot 2^{14}$ .

#### Prijedlog rješenja:

Množenje nekog broja (X) potencijom broja dva ( $2^N$ ) ekvivalentno je pomicanju broja X za N mjesta u lijevo. Dakle, množenje podatka može se izvesti zbrajanjem dvaju pomaknutih početnih podataka, za 14 i za 16 mjesta.

Budući da umnožak mora biti u dvostrukoj preciznosti, moramo pretpostaviti da pri pomicanju početnog podatka za 14 i 16 mjesta ulijevo dolazi do izlaženja viših bitova početnog podatka iz opsega od 32 bita. Ovi izlazni bitovi ne smiju se izgubiti jer ih treba pribrojiti u konačni 64-bitni rezultat i to na mjesto viših 32 bita. Ako bi se upotrijebilo pomicanje za 14 i 16 mjesta, viši izlazni bitovi bi se izgubili. Zato se umjesto pomicanja koristi rotacija ulijevo kod koje će izlazni bitovi doći na niže položaje u rezultatu. S nižih položaja se izlazni bitovi mogu izdvojiti nakon čega se trebaju pribrojiti višem dijelu rezultata. Bitovi preostali nakon izdvajanja izlaznih bitova sačinjavaju niže dijelove podataka koje također treba pribrojiti i to nižem dijelu rezultata. Izdvajanja bitova lako se ostvaruju pomoću naredbe **AND** i maskiranja. Prilikom zbrajanja nižih i viših dijelova rezultata koristi se zbrajanje u dvostrukoj preciznosti, tj. prvo se naredbom **ADDS** zbroje niži dijelovi, a zatim se viši dijelovi zbroje naredbom **ADC**.

Još valja napomenuti da se kod procesora ARM 7 pomaci i rotacije izvode naredbom **MOV** kod koje se drugi operand može pomicati i rotirati za željeni broj mjesta. Međutim, ARM 7 nema rotaciju ulijevo, jer se umjesto nje uvijek može napraviti odgovarajuća rotacija udesno. Želi li se ostvariti rotacija ulijevo za N bitova, isti rezultat dobiva se rotacijom udesno za  $32-N$  bitova.

#### Rješenje:

	ORG	0	
GLAVNI	MOV	R1, #7<12	; staviti adresu bloka podataka u R1
	MOV	R2, #10<12	; staviti adresu bloka rezultata u R2
	MOV	R3, #1<8	; staviti broj podataka u R3

(nastavak na sljedećoj stranici)

	LDR	R9, M1	; maske za izdvajanje podnizova bitova
	LDR	R10, M2	
	LDR	R11, M3	
	LDR	R12, M4	
PETLJA	LDR	R4, [R1], #4	; pročitati podatak iz bloka, uvećaj adresu
	MOV	R5, R4, ROR #18	; ulijevo za 14 mjesta = udesno za 18 (32-14)
	MOV	R7, R4, ROR #16	; ulijevo za 16 mjesta = udesno za 16
			; razdvajanje podatka iz R5 na viših 32 bita i nižih 32 bita
	AND	R6, R9, R5	; viši dio pomaknutog podatka staviti u R6
	AND	R5, R10, R5	; niži dio pomaknutog podatka staviti u R5
			; razdvajanje podatka iz R7 na viših 32 bita i nižih 32 bita
	AND	R8, R11, R7	; viši dio pomaknutog podatka u R8
	AND	R7, R12, R7	; niži dio pomaknutog podatka u R7
			; zbrajanje viših i nižih dijelova broja
	ADDS	R5, R7, R5	; R5 - niži dio 64-bitnog rezultata
	ADC	R6, R8, R6	; R6 - viši dio 64-bitnog rezultata
	STR	R5, [R2], #4	; pohraniti niži dio
	STR	R6, [R2], #4	; pohraniti viši dio
DALJE	SUBS	R3, R3, #1	; smanjiti brojač podataka
	BNE	PETLJA	; nastaviti petlju ako nismo gotovi
GOTOV	HALT		; kraj
M1	DW	000003FFF	; nižih 14 bitova (viši bitovi 64-bitnog broja)
M2	DW	0FFFC000	; viših 18 bitova (niži bitovi 64-bitnog broja)
M3	DW	0000FFFF	; nižih 16 bitova (viši bitovi 64-bitnog broja)
M4	DW	0FFF0000	; viših 16 bitova (niži bitovi 64-bitnog broja)
	ORG	7000	
BLOK	DW	14, 250, 169A27, ...	
	ORG	10000	
REZ	DS	800	

### Komentar rješenja:

Množenje podatka u registru **R4** konstantama  $2^{14}$  i  $2^{16}$  izvodi se rotacijom i maskiranjem bitova. Podatak u registru **R4** rotira se ulijevo za 14 mjesta i sprema u registar **R5** odnosno rotira se ulijevo za 16 mjesta te se sprema u registar **R7**, čime u **R5** i **R7** dobivamo sljedeći raspored bitova:

R5: 

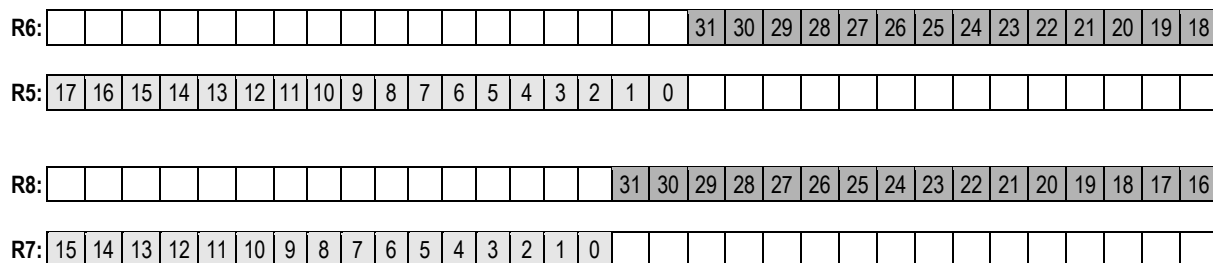
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18
----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----

R7: 

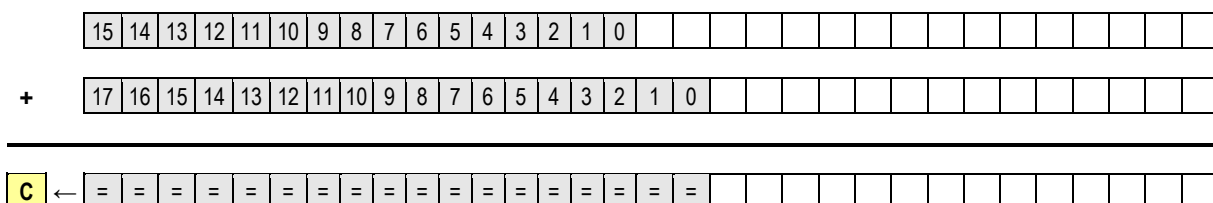
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Rotacija čuva više bitove koji bi se izgubili pomakom (označeni tamnijom sivom bojom na slici). Sljedeći korak je razdvojiti više (tamno sivi bitovi) i niže (svijetlo sivi bitovi) 32-bitne

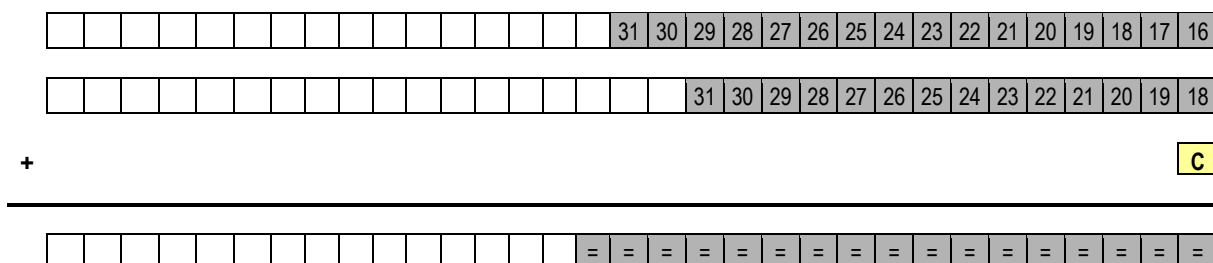
riječi svakog od ovih dvaju brojeva tako da ti brojevi budu zapisani u 64-bita. To se ostvaruje maskiranjem pomoću naredaba **AND** koje će broj pomnožen sa  $2^{14}$  spremi u **R6** (viša 32 bita) i **R5** (niža 32 bita), a broj pomnožen sa  $2^{16}$  će spremi u **R8** (viša 32 bita) i **R7** (niža 32 bita):



U razdvojenim dijelovima su bitovi bez oznaka ispunjeni ničesticama. Tako razdvojene dijelove treba zbrojiti u dvostrukoj preciznosti (u 64 bita). Prvo zbrajamo niže riječi iz registara **R7** i **R5** naredbom **ADDS** (rezultat se sprema u R5):



Naredba **ADD** može prouzročiti prijenos koji treba uračunati pri zbrajanju viših riječi iz registara **R8** i **R6**, za što služi naredba **ADC** (rezultat se sprema u **R6**):



Zbog mogućnosti pojave prijenosa kod zbrajanja naredbom **ADC**, ukupno mogući broj bitova rezultata je za jedan veći od broja bitova u podacima (označeno tamnije sivom bojom).

Množenje se moglo izvesti i znatno jednostavnije. Proučite sami sljedeći odsječak:

```
LDR    R4, [R1], #4      ; učitati broj u R4 - isto kao i prije
; R4 * 2^14 staviti u R5 (niži dio) i R6 (viši dio)
MOV     R5, R4, LSL #14
MOV     R6, R4, LSR #18

; R4 * 2^16 staviti u R7 (niži dio) i R8 (viši dio)
MOV     R7, R4, LSL #16
MOV     R8, R4, LSR #16
```

(nastavak na sljedećoj stranici)

```

; zbrajanje viših i nižih dijelova broja - isto kao i prije
ADDS    R5, R7, R5
ADC     R6, R8, R6

```

U ovakvom rješenju prednosti su brojne: manje zauzeće memorije i registara, brži rad i jednostavniji program. Dodatno, vrlo je lagano promijeniti ovo rješenje tako da radi s brojevima u zapisu 2<sup>k</sup>. Koje naredbe bi trebalo promijeniti?

## 2.2.14. Množenje NBC-brojeva pomoću metode uzastopnog pribrajanja (FRISC 2.3.14.)

Riješen: DA    Težina: ★

Napisati program za množenje dva 8-bitna broja zapisana NBC-om. Brojevi su na lokacijama 1000<sub>16</sub> i 1001<sub>16</sub>, a rezultat treba spremiti na 1002<sub>16</sub>. Radi jednostavnosti pretpostavlja se da neće doći do prekoračenja 8-bitnog opsega. Zadatak riješiti metodom uzastopnog pribrajanja.

### Prijedlog rješenja:

Metoda uzastopnog pribrajanja temelji se na pribrajanju jednog operanda onoliko puta koliko iznosi drugi operand (npr.  $5 \cdot 2 = 2 + 2 + 2 + 2 + 2$ ). Prvi operand se pribraja u rezultat dok se drugi operand koristi kao brojač u petlji koji se u svakom prolazu umanjuje za 1.

### Rješenje:

```

ORG      0
MOV      R3, #1<12
LDRB     R0, [R3]          ; dohvat prvog broja
LDRB     R1, [R3, #1]      ; dohvat drugog broja
MOV      R2, #0            ; rezultat množenja je početno 0

PETLJA   ADD     R2, R2, R0  ; pribrojiti prvi operand rezultatu
          SUBS   R1, R1, #1  ; drugi operand je brojač za petlju
          BNE    PETLJA
GOTOVO   STRB    R2, [R3, #2] ; spremiti rezultat
          HALT

```

### Komentar rješenja:

U programu se prvo dohvaćaju operandi u registre **R0** i **R1**. Nakon toga se briše registar **R2** u kojemu će se pribrajanjem akumulirati rezultat. Operand iz registra **R1** upotrebljava se kao brojač u petlji za pribrajanje. Petlja je vrlo jednostavna jer se u njoj samo pribraja **R0** rezultatu u **R2** i smanjuje se brojač **R1**.

Uočite da će se za drugi operand jednak ništici, petlja izvesti 2<sup>32</sup> puta. Hoće li konačni rezultat biti ispravan?

Iako je vrlo jednostavna za programsko ostvarenje, metoda uzastopnog pribrajanja nije učinkovita u pogledu brzine izvođenja jer broj prolazaka kroz petlju može biti velik – ovisno o iznosu operanda koji se upotrebljava kao brojač. Promijenite program tako da se kao brojač u petlji koristi manji operand.

Dodatni nedostatak je ovisnost trajanja množenja o iznosima operanada, što je nepoželjno. Bolja je metoda pomaka i zbrajanja koja je općenito brža i uvijek jednakog trajanja –



neovisno o operandima. Ova metoda odgovara algoritmu što se primjenjuje za množenje „na papiru“.

Procesor ARM 7 inače ima ugrađene naredbe za množenje. Umjesto pisanja vlastitih odsječaka za množenje, bolje je koristiti postojeće naredbe. Izuzetak su jednostavna množenja s konstantama (npr. 2, 5) koja se ponekad mogu izvoditi brže nego ugrađene naredbe.

## 2.2.15. Množenje 2<sup>k</sup>-brojeva pomoću metode uzastopnog pribrajanja (FRISC 2.3.15.)

Riješen: DA    Težina: ★★

Napisati program za množenje dva 32-bitna broja u zapisu 2<sup>k</sup>. Brojevi su na lokacijama 1000<sub>16</sub> i 1004<sub>16</sub>, a rezultat treba spremiti na 1008<sub>16</sub>. Radi jednostavnosti pretpostavlja se da neće doći do prekoračenja opsega. Zadatak treba riješiti metodom uzastopnog pribrajanja.

### Prijedlog rješenja:

Množenje se obavlja s pozitivnim operandima što daje pozitivan rezultat. Prije množenja treba negativne operande pretvoriti u pozitivne. Također treba zapamtiti predznake operandi, jer rezultat treba biti negativan samo ako su oba operanda imala različite predznake.

Na kraju se konačnom zbroju, koji je sigurno pozitivan, po potrebi mijenja predznak na temelju zapamćenih predznaka operandi.

### Rješenje:

```

ORG      0
MOV      R6, #1<12
LDR      R0, [R6]          ; dohvati prvog broja
LDR      R1, [R6, #4]      ; dohvati drugog broja
EOR      R3, R0, R1        ; zapamtiti predznake obaju operandi u R3
                                ; (važan je najviši bit)
                                ; pretvaranje operandi u pozitivne brojeve (ako su negativni)
TEST_1   ORRS      R0, R0, R0 ; provjeriti predznak prvog operandi
TEST_2   RSBMI     R0, R0, #0 ; ako je negativan, promijeniti mu predznak
TEST_2   ORRS      R1, R1, R1 ; provjeriti predznak drugog operandi
TEST_2   RSBMI     R1, R1, #0 ; ako je negativan, promijeniti mu predznak

                                ; množenje dva pozitivna broja
MOV      R2, #0            ; rezultat množenja
PETLJA   ADD      R2, R0, R2 ; pribrojiti operand rezultatu
SUBS     R1, R1, #1
BNE      PETLJA

                                ; postaviti ispravan predznak rezultata
PREDZNAK CMP      R3, #0    ; jesu li operandi imali različite predznake
RSBMI    R2, R2, #0        ; ako da, rezultat je negativan
STR      R2, [R6, #8]      ; spremiti rezultat
HALT

ORG      1000
PRVI     DW      8
DRUGI    DW      -5
REZ      DS      4

```

**Komentar rješenja:**

Predznaci obaju operandata se ne pamte zasebno, nego se pamti samo jesu li bili isti ili različiti. To se jednostavno ostvaruje korištenjem logičke operacije ekskluzivno-ILI (naredba **EOR**) na operandima i spremanjem rezultata u **R3**. To će u najvišem bitu od **R3** postaviti 1 ako su predznaci bili različiti, odnosno 0 ako su predznaci bili isti. Taj najviši bit se ispituje tako što se ispituje predznak podatka u **R3** u dijelu programa označenom labelom **PREDZNAK** (predznak 2'k brojeva određen je najvišim bitom, odnosno iskoristit ćemo zastavicu **N**). Ukoliko je nakon izvođenja naredbe **CMP R3, #0** postavljena zastavica **N** (najviši bit u **R3** je 1), izvodi se sljedeća uvjetna naredba **RSBMI R2, R2, #0** čime se konačnom umnošku postavlja predznak.

Na labelama **TEST\_1** i **TEST\_2** se pojedinačno ispituju predznaci operandata i ako su negativni, pretvaraju se u pozitivne brojeve pomoću naredbe **RSBMI** gdje je drugi operand ničica. Time se dobivaju pozitivni brojevi koji će se koristiti u petlji uzastopnog pribrajanja. Slijedi petlja za množenje uzastopnim pribrajanjem, a nakon obavljenog množenja provjerava se najviši bit u **R3** usporedbom podatka s nulom kako je prethodno opisano.

**2.2.16. Cjelobrojno dijeljenje 2'k-brojeva s konstantom (FRISC 2.3.16.)****Riješen: DA    Težina: ★**

Napisati program koji čita  $300_{16}$  podataka iz bloka pohranjenog na adresi  $7000_{16}$ . Svaki podatak treba cjelobrojno podijeliti sa  $8_{10}$  uporabom naredaba za pomak i rezultat zapisati na isto mjesto u početnom bloku. Podatci u bloku su u 32-bitnom zapisu 2'k.

**Prijedlog rješenja:**

Dijeljenje brojem koji je potencija broja 2 izvodi se jednostavnim pomakom za odgovarajući broj mjesta udesno (u ovom slučaju za 3 mjesta, jer je  $2^3 = 8$ ). Za pomak će se upotrijebiti aritmetički pomak, jer on čuva predznak broja zapisanog u obliku 2'k.

Rješenje:			
GLAVNI	ORG	0	
	MOV	R1, #7<12	; staviti adresu bloka u R1
	MOV	R2, #3<8	; staviti broj podataka u R2
PETLJA	LDR	R3, [R1]	; pročitati podatak iz bloka u R3
	MOV	R3, R3, ASR #3	; podijeliti sa 8
	STR	R3, [R1], #4	; pohraniti na isto mjesto, uvećati adresu
DALJE	SUBS	R2, R2, #1	; smanjiti brojač podataka
	BNE	PETLJA	; nastaviti petlju ako nije gotovo
GOTOVO	HALT		; kraj
BLOK	ORG	7000	
	DW	100, %D 200, -9AF8F, %D -124	; ...i još 3FC podataka

**Komentar rješenja:**

Prilikom dijeljenja naredbom **MOV R3, R3, ASR #3** gube se bitovi koji čine ostatak dijeljenja – u ovom slučaju niža 3 bita. Bitove je moguće sačuvati ako je potrebno znati ostatak dijeljenja. Da je u zadatku bilo zadano da se dijele brojevi u zapisu NBC, tada bi umjesto

aritmetičkog pomaka (**ASR**) trebalo upotrijebiti logički pomak **LSR**, koji ulazne bitove s lijeve strane puni ničticama.

### 2.2.17. Cjelobrojno dijeljenje NBC-brojeva metodom uzastopnog oduzimanja (FRISC 2.3.17.)

Riješen: DA    Težina: ★

Napisati program za cjelobrojno dijeljenje dva 32-bitna broja u zapisu NBC. Brojevi su na lokacijama  $1000_{16}$  i  $1004_{16}$ , a rezultat treba spremiti na  $1008_{16}$ . Zadatak treba riješiti metodom uzastopnog oduzimanja.

#### Prijedlog rješenja:

Metoda uzastopnog oduzimanja zasniva se na oduzimanju djelitelja od djeljenika sve dok je rezultat pozitivan. Pri svakom uspješnom oduzimanju rezultat (koji se početno postavi na ničticu) se povećava za jedan. Pod uspješnim oduzimanjem misli se na oduzimanje koje neće dati negativan rezultat. Na primjer  $11/2$  računa se kao  $11-2-2-2-2$  što daje rezultat 5 jer je bilo 5 uspješnih oduzimanja, a šesto oduzimanje bi dalo rezultat -1. Nakon 5 uspješnih oduzimanja dobiva se broj 1 koji je ujedno i ostatak dijeljenja.

Rješenje:			
	ORG	0	
	MOV	R3, #1<12	
	LDR	R0, [R3], #4	; dohvat prvog broja, povećati adresu
	LDR	R1, [R3], #4	; dohvat drugog broja, povećati adresu
	MOV	R2, #0	; početna vrijednost rezultata dijeljenja
PETLJA	SUBS	R0, R0, R1	; oduzeti djelitelj od djeljenika
	ADDGE	R2, R2, #1	; povećati rezultat ako je uspjelo
	BGE	PETLJA	; i vratiti se natrag na petlju
GOTOVO	STR	R2, [R3]	; inače, spremiti rezultat
	HALT		

#### Komentar rješenja:

U programu se prvo dohvaćaju operandi u registre **R0** i **R1**. Nakon toga se briše registar **R2** u kojemu se kasnije izračunava rezultat. U petlji se oduzima **R1** od **R0** i ispituje je li rezultat oduzimanja pozitivan ili negativan. Ako je broj dobiven oduzimanjem negativan, onda se sljedeće dvije naredbe (**ADDGE** i **BGE**) neće izvesti te će program izaći iz petlje, a u **R2** se nalazi rezultat koji se sprema na lokaciju  $1008_{16}$ . Ako je broj dobiven oduzimanjem pozitivan, povećava se rezultat u **R2** za jedan i ponavlja se petlja.

Što će se dogoditi ako je djelitelj jednak 0, a što ako je djeljenik jednak 0? Proširite program tako da izračuna i ostatak dijeljenja koji treba spremiti na lokaciju  $1012_{16}$ .

Što se tiče jednostavnosti i učinkovitosti, za metodu uzastopnog oduzimanja vrijedi sve što je rečeno za metodu uzastopnog zbrajanja. Ponovno je bolje upotrijebiti metodu koja odgovara dijeljenju „na papiru“ – metodu pomaka i oduzimanja.

**2.2.18. Izračun površine jednakokračnog trokuta (FRISC 2.3.18.)****Riješen: DA    Težina: ★**

Napisati program koji izračunava površinu jednakokračnog trokuta čija su visina i duljina osnovice zapisane u memoriji na adresama  $1000_{16}$  i  $1002_{16}$  u 16-bitnom NBC-u. 32-bitni rezultat treba spremiti u registar R2. Kao konkretne vrijednosti za računanje treba zadati visinu  $17_{10}$  i duljinu osnovice  $18_{10}$ . Formula za površinu je  $(visina * duljina\_osnovice) / 2$ .

**Prijedlog rješenja:**

U ovom jednostavnom zadatku kombinirano je dijeljenje s konstantom te množenje, što je već pokazano u prethodnim zadatcima. Budući da nije drugačije zadano, za množenje je upotrijebljena naredba **MUL**, radi jednostavnije programske izvedbe. Učitane podatke u NBC-u treba proširiti ničticama sa 16 na 32 bita (naredba **LDRH**).

**Rješenje:**

```

ORG      0
MOV      R3, #1<12

LDRH     R0, [R3], #2    ; visina trokuta
LDRH     R1, [R3], #2    ; duljina osnovice trokuta
MOV      R2, #0          ; inicijalizacija rezultata množenja u R2

MUL      R2, R0, R1      ; množenje

MOV      R2, R2, LSR #1  ; dijeljenje sa 2 pomakom u desno za 1 bit
HALT

ORG      1000
DH       %D 17,  %D 18   ; visina i osnovica
REZ      DS       4

```

**Komentar rješenja:**

Ukoliko nije zadano drukčije, kod procesora ARM 7 je, za razliku od procesora FRISC, na raspolaganju čitav niz naredaba za množenje (u ovom primjeru koristimo naredbu **MUL**), čime je množenje znatno olakšano. Podatci, zapisani kao poluriječi u memoriji, učitani su naredbama **LDRH** u registre **R0** i **R1**. Naredba **LDRH** popunjava ničticama viših 16 bitova registra, čime se podatak učitava i ujedno proširuje ničticama. Budući da su početni brojevi 16-bitni, sigurno neće doći do prekoračenja 32-bitnog opsega prilikom množenja. Dalje se normalno množe i dijele 32-bitni brojevi.

**2.2.19. Srednja vrijednost 2<sup>k</sup>-brojeva u bloku podataka (FRISC 2.3.19.)****Riješen: DA    Težina: ★**

Napisati program koji računa srednju vrijednost podataka u bloku memorije. U bloku se nalaze 64 podatka koji su u 32-bitnom zapisu 2<sup>k</sup>. Blok je zapisan od početne adrese  $500_{16}$ . Srednju vrijednost potrebno je zapisati na lokaciju  $400_{16}$ . Pretpostavlja se da zbroj podataka u bloku ne prelazi 32-bitni opseg.

**Prijedlog rješenja:**

Pribrajanje podatka obavlja se u petlji koja se izvodi 64 puta. Dobiveni zbroj treba podijeliti sa 64, koji je potencija broja 2 ( $64 = 2^6$ ) pa se dijeljenje može ostvariti pomakom u desno za 6 mjesta. Brojevi su u zapisu 2<sup>k</sup> pa će se koristiti aritmetički pomak kako bi se sačuvao predznak broja (za NBC-brojeve dijeljenje bi trebalo ostvariti logičkim pomakom).

**Rješenje:**

	ORG	0	
	MOV	R0, #0	; inicijalizacija brojača i adresa
	MOV	R1, #5<8	
	MOV	R2, #4<8	
	MOV	R4, #0	; zbroj
PETLJA	LDR	R3, [R1], #4	; učitavanje podatka
	ADD	R4, R3, R4	; zbrajanje
	SUBS	R0, R0, #1	; smanjiti brojač podataka
	BNE	PETLJA	; ispitati je li zbrojeno svih 64
	MOV	R4, R4, ASR #6	; dijeljenje sa 64 – zapis 2'k
	STR	R4, [R2]	; spremanje na adresu 400
	HALT		; zaustavljanje procesora

Riješite zadatak uz pretpostavku da pri zbrajanju brojeva iz bloka može doći do prekoračenja 32-bitnog opsega. To znači da zbrajanje treba obaviti u dvostrukoj preciznosti. Uočite da će konačni rezultat sigurno stati u 32-bita. Napravite inačicu programa u kojemu će konačni rezultat biti spremljen kao 32-bitni broj i inačicu u kojoj će biti spremljen u dvostrukoj preciznosti. Što morate promijeniti u ovim rješenjima, ako su brojevi u bloku u zapisu NBC?

## 2.3. Potprogrami

U ovom potpoglavlju pokazani su zadatci u kojima se koriste potprogrami. Za rad s potprogramima potrebno je razumjeti kako radi stog i kako ARM pri pozivanju i povratku iz potprograma rukuje s povratnom adresom u registru **R14**. Da bi se moglo raditi sa stogom, potrebno je odrediti na kojim memorijskim adresama će se nalaziti prostor stoga, što se zadaje upisom početne vrijednosti u pokazivač stoga, tj. u registar **R13** (drugo ime mu je **SP**). Uobičajeno mjesto za stog su memorijske lokacije s najvišim adresama, a stog se puni prema nižim adresama.

Za razliku od procesora FRISC, kod procesora ARM 7 stog možemo popunjavati i prema višim i prema nižim adresama u odnosu na trenutni vrh stoga, ovisno o tome koje nastavke za rad sa stogom koristimo u naredbama kojima spremamo blok registara na stog. Kada na stog spremamo jedan ili više registara, stog će rasti prema nižim adresama ako koristimo naredbe **STMED** ili **STMFD** (pripadne naredbe za učitavanje sa stoga su **LDMED** i **LDMFD**). Ako prilikom spremanja na stog koristimo naredbe **STMEA** i **STMFA**, stog će rasti prema višim adresama (pripadne naredbe za učitavanje sa stoga su **LDMEA** i **LDMFA**).

Na početku potprograma **treba sačuvati sve one registre koje potprogram mijenja** (ne računajući registre pomoću kojih se eventualno vraća rezultat). Ako će se iz potprograma pozivati isti ili neki drugi potprogram, potrebno je na stog spremiti i registar **R14** (**LR**) u koji se sprema povratna adresa. Registri se uobičajeno spremaju na stog, a ovaj postupak naziva se još i spremanjem konteksta. Naravno, na kraju potprograma treba obnoviti vrijednosti spremljenih registara. Spremanje konteksta vrlo je važno i treba ga raditi u svakom potprogramu. Može se reći da je to jedna od praksi dobrog programiranja, jer bi bez toga bilo puno teže koristiti potprogramme i oni bi bili podložniji greškama.

Svaki puta kad se potprogram poziva obično mu se šalju drugačije vrijednosti parametara i na temelju njihovih vrijednosti dobivamo neki rezultat. Prenošnje parametara i vraćanje rezultata mogu se ostvariti na razne načine koji su pokazani u ovom potpoglavlju. Potpoglavlje sadrži sljedeće gradivo:

- poziv i povratak iz potprograma, spremanje povratne adrese
- spremanje i obnova konteksta
- prijenos parametara i vraćanje rezultata pomoću registara
- prijenos parametara i vraćanje rezultata pomoću fiksnih memorijskih lokacija
- prijenos parametara i vraćanje rezultata pomoću stoga
- prijenos parametara preko memorijskih lokacija iza naredbe poziva potprograma
- kombinirani načini prijenosa parametara i vraćanja rezultata.

### 2.3.1. Prijenos parametara i rezultata u potprogram pomoću registara (FRISC 2.4.2.)

Riješen: DA    Težina: ★

Napisati potprogram **POTP** koji prima parametre preko registara **R0** i **R1** kao podatke u zapisu 2'k. Potprogram ispituje vrijednost registra **R0** te ako je vrijednost manja ili jednaka od 0, potrebno je u podatku iz registra **R1** postaviti bitove 0, 2, 7, 14, 25 i 31 te rezultat vratiti u registru **R2**. Ako je vrijednost registra **R0** veća od ničice, potprogram vraća vrijednost 0 u registru **R2**.

Glavni program treba pozvati potprogram **POTP** pomoću kojega će na temelju podatka na memorijskoj lokaciji **ISPITAJ** (parametar koji se ispituje) promijeniti bitove podatku na lokaciji **BROJ** (parametar koji se mijenja).

#### Prijedlog rješenja:

Potprogram u svom radu može mijenjati neke registre, ali ne može znati je li pozivatelj potprograma u tim registrima imao neke korisne podatke. Zato potprogram na početku svog rada mora spremiti vrijednosti svih registara koje mijenja što se naziva **spremanjem konteksta**. Izuzetak su registri preko kojih potprogram vraća rezultate, jer pozivatelj zna da će se oni promijeniti pa u njima ne smije čuvati korisne podatke koji bi mu kasnije mogli zatrebati. Kontekst se uobičajeno sprema na stog naredbom za spremanje bloka podataka u memoriju **STM**.

Prilikom izlaska iz potprograma se stanja spremljenih registara moraju obnoviti naredbom za učitavanje bloka podataka iz memorije **LDM**, što se naziva **obnovom konteksta**. Proučite nastavke za rad sa stogom za naredbe **LDM** i **STM**. Bitno je naglasiti da se koriste isti nastavci i za spremanje i za obnovu konteksta.

U primjerima iz ove zbirke najčešće se koriste nastavci **FD** (*Full Descending*) koji imaju različito značenje za naredbe **STM** i **LDM**. Kod naredbe **STMFD** prvo se umanjuje vrijednost registra **R13** (pokazivač stoga) za 4, potom se na tu memorijsku lokaciju sprema sadržaj registra s najvećim indeksom. Ovo se ponavlja za sve registre iz navedenog bloka. Kod naredbe **LDMFD** prvo se sprema sadržaj memorijske lokacije na koju pokazuje **R13** u registar s

najmanjim indeksom, potom se vrijednost registra **R13** uvećava za 4. Ovo se ponavlja za sve registre iz navedenog bloka.

**Rješenje:**

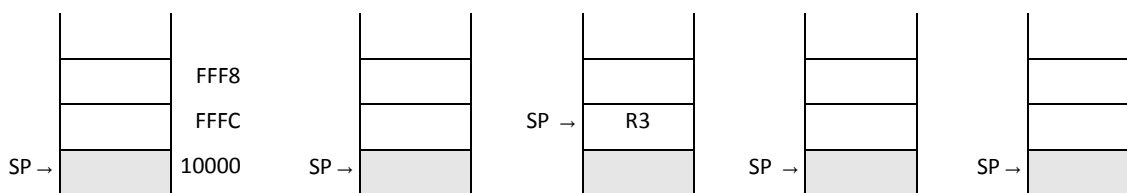
GLAVNI	ORG	0	
	MOV	R13, #10<12	; inicijalizacija stoga
	LDR	R0, ISPITAJ	; dohvat prvog parametra u R0
	LDR	R1, BROJ	; dohvat drugog parametra u R1
	BL	POTP	; poziv potprograma
	STR	R2, BROJ	; spremi rezultat iz R2 natrag u memoriju
	HALT		
ISPITAJ	DW	-2	
BROJ	DW	12345678	
POTP			; potprogram
	STMFD	R13!, {R3}	; spremanje R3 (tj. spremanje konteksta)
	CMP	R0, #0	; ispitivanje R0
	BLE	NEGAT	; ako R0 <= 0 skok na NEGAT
	MOV	R2, #0	; R0 > 0, brisanje R2
	LDMFD	R13!, {R3}	; obnavljanje R3
	MOV	PC, LR	; povratak iz potprograma
NEGAT	LDR	R3, MASKA	; postavi bitove 0, 2, 7, 14, 25 i 31
	ORR	R2, R1, R3	; ...u R1 i stavi rezultat u R2
	LDMFD	R13!, {R3}	; obnavljanje R3 (tj. obnavljanje konteksta)
	MOV	PC, LR	; povratak iz potprograma
MASKA	DW	%B 10000010000000000100000010000101	

**Komentar rješenja:**

Potprogram prvo ispituje parametar prenesen registrom **R0** što se ostvaruje naredbom **CMP R0, #0**. Ako je vrijednost u **R0** manja ili jednaka 0, potprogram skače na labelu **NEGAT** gdje se dohvaća maska u registar **R3**, a nakon toga se postavljaju zadani bitovi i sprema rezultat u registar **R2** naredbom **ORR R2, R1, R3**. Ako je podatak u **R0** veći od 0, briše se registar **R2** preko kojeg se vraća povratna vrijednost. Budući da potprogram mijenja sadržaj registra **R3**, onda ga se na početku izvođenja potprograma treba spremiti na stog naredbom **STMFD R13!, {R3}**, a pred sam povratak obnoviti sa stoga naredbom **LDMFD R13!, {R3}**.

Budući da potprogram prima parametre kao vrijednosti u zapisu 2'k, u naredbi uvjetnog skoka korišten je sufiks **LE** (*signed less than or equal*). Da se radilo o NBC brojevima, koristio bi se sufiks **LS** (*unsigned less or same*), odnosno naredba **BLS NEGAT**.

Iako potprogram osim registra **R3** koristi i registre **R0**, **R1** i **R2**, jedino se **R3** sprema i obnavlja sa stoga. To je zato što potprogram ne mijenja sadržaj registara **R0** i **R1**, a registar **R2** se koristi za povratak rezultata. Stanje na stogu za vrijeme izvođenja programa prikazano je sljedećom slikom:



Na slici su prikazani podatci na stogu, a ne pojedine memorijske lokacije od jednog bajta, jer se na stog uvijek stavljaju (ili uzimaju) 32-bitni podatci. Uz početno stanje stoga, prikazano na krajnje lijevoj strani slike, naznačene su i memorijske adrese podataka na stogu. Uz svako stanje na stogu prikazano je i trenutno mjesto na koje pokazuje pokazivač stoga **SP**. Na slici su s lijeva na desno redom prikazana stanja: prije poziva potprograma (neposredno prije naredbe **BL POTP**); neposredno nakon poziva potprograma (naredba **BL** stavila je povratnu adresu u **R14**, to je adresa naredbe **STR R2,BROJ**); nakon spremanja konteksta (naredba **STMFD R13!,{R3}**) i za vrijeme izvođenja potprograma; neposredno nakon obnavljanja konteksta (naredba **LDMFD R13!,{R3}**); neposredno nakon povratka iz potprograma (naredba **MOV PC,LR** je povratnu adresu iz **R14** stavila u programsko brojilo tj. skočila je na povratnu adresu). Izvođenje se nastavlja naredbom na povratnoj adresi **STR R2,BROJ**. Uočite da je stanje na stogu jednako prije i poslije poziva potprograma, na što treba paziti, jer se sa stoga uvijek mora skinuti onoliko podataka koliko je na njega stavljeno. Usporedite niz stanja na stogu sa stanjima stoga u odgovarajućem programu za FRISC.

### 2.3.2. Prijenos parametara i rezultata u potprogram pomoću fiksnih lokacija (FRISC 2.4.3.)

Riješen: DA Težina: ★

Napisati potprogram **POTP** koji parametre prima preko tri 32-bitne memorijske lokacije **P0**, **P1** i **P2** kao brojeve u zapisu 2'k. Potprogram ispituje sadržaj podatka na lokaciji **P0**, te u slučaju negativnog podatka oduzima broj iz lokacije **P2** od broja iz lokacije **P1**, a inače zbraja brojeve iz lokacija **P1** i **P2**. Potprogram sprema rezultat na memorijsku lokaciju **REZ**. Napisati i glavni program koji poziva potprogram **POTP** sa parametrima: 1 (**P0**), 321<sub>16</sub> (**P1**) i 222<sub>16</sub> (**P2**).

#### Prijedlog rješenja:

Potprogram prima parametre preko memorijskih lokacija **P0**, **P1** i **P2**, tako da vrijednosti iz tih lokacija naredbama **LDR** učitava u registre **R0**, **R1** i **R2**. Prije toga, potprogram treba sačuvati te registre spremanjem na stog naredbom **STMFD R13!,{R0,R1,R2}**. Ispitivanje predznaka može se ostvariti naredbom **ORRS R0,R0,R0**. S obzirom da je za svako grananje potrebno izvršiti samo jednu naredbu, nije korištena naredba grananja **B**, već uvjetno izvođenje naredaba (nastavci **MI** i **PL**). Prije povratka, potprogram će spremiti rezultat na labelu **REZ**, te obnoviti registre **R0**, **R1** i **R2** naredbom **LDMFD**.

#### Rješenje:

```

ORG      0
GLAVNI   MOV    R13, #10<12      ; inicijalizacija stoga
MOVE     BL     POTP             ; poziv potprograma
         HALT
         ; potprogram
POTP     STMFD   R13!, {R0, R1, R2} ; spremanje registara koji se mijenjaju

```

(nastavak na sljedećoj stranici)



	LDR	R0, P0	; dohvati parametara
	LDR	R1, P1	
	LDR	R2, P2	
	ORRS	R0, R0, R0	; ispitaj R0
	SUBMI	R0, R1, R2	; ako je R0 < 0, oduzeti R1 i R2 u R0
	ADDPL	R0, R1, R2	; ako je R0 >= 0, zbrojiti R1 i R2 u R0
KRAJ	STR	R0, REZ	; spremanje povratne vrijednosti na REZ
	LDMFD	R13!, {R0, R1, R2}	; obnavljanje registara
	MOV	PC, LR	; povratak iz potprograma
P0	DW	1	; tri parametra za potprogram
P1	DW	321	
P2	DW	222	
REZ	DW	0	; mjesto za rezultat

**Komentar rješenja:**

Ispitivanje predznaka moguće je ostvariti i drugim kombinacijama naredaba, na primjer pomakom ulijevo za jedno mjesto te potom ispitivanjem zastavice prijenosa.

### 2.3.3. Prijenos parametara i rezultata u potprogram pomoću memorijskih lokacija iz poziva potprograma

Riješen: DA    Težina: ★★

Napisati potprogram **POTP** koji prima 32-bitni 2'k parametar pomoću memorijske lokacije koja se nalazi neposredno iza poziva potprograma. Potprogram ispituje vrijednost parametra te ako je vrijednost manja ili jednaka od 0, potrebno je na memorijsku lokaciju koja se nalazi iza lokacije parametra kao rezultat zapisati 32-bitnu vrijednost -1. Ako je vrijednost prvog parametra veća od ničice, potrebno je kao rezultat zapisati 32-bitnu vrijednost 1.

Glavni program treba zamijeniti 32-bitni podatak na labeli **POD** sa rezultatom poziva potprograma **POTP**.

**Prijedlog rješenja:**

U glavnom programu potrebno je prvo učitati podatak sa lokacije **POD**, a potom ga spremi na lokaciju iza poziva potprograma. Tu lokaciju ćemo označiti labelom **PARAM** u cilju lakšeg adresiranja. Također ćemo rezervirati i sljedeću memorijsku lokaciju (označit ćemo ju labelom **REZ**) za zapis rezultata izvođenja potprograma **POTP**. U potprogramu ispituje vrijednost parametra i u ovisnosti o vrijednosti zapisujemo -1/1 na lokaciju **REZ**.

Za dohvati parametra i spremanje rezultata u potprogramu iskoristit ćemo registar **LR (R14)** u kojeg se sprema povratna adresa odnosno adresa memorijske lokacije iza poziva potprograma **BL POTP**.

Rješenje:			
	ORG	0	
GLAVNI	MOV	R13, #10<12	; inicijalizacija stoga
	LDR	R0, POD	; učitavanje podatka
	STR	R0, PARAM	; spremanje na lokaciju iza poziva potprograma

(nastavak na sljedećoj stranici)

	BL	POTP	; poziv potprograma
PARAM	DW	0	; inicijalno je parametar 0
REZ	DW	0	; inicijalno je rezultat 0
	LDR	R0, REZ	; učitavanje rezultata
	STR	R0, POD	; i spremanje na lokaciju POD
	HALT		
			; potprogram
POTP	STMFD	R13!, {R3}	; spremanje R3 (tj. spremanje konteksta)
	LDR	R3, [LR], #4	; učitavanje parametra sa PARAM, pomak LR na REZ
	CMP	R3, #0	; ispitivanje R0
	MVNLE	R3, #0	; ako je R0 ≤ 0 upisuje se FFFFFFFF u R3
	MOVGT	R3, #1	; ako je R0 > 0 upisuje se 1 u R3
	STR	R3, [LR], #4	; spremanje rezultata i pomak LR na ispravnu ; povratnu adresu
	LDMFD	R13!, {R3}	; obnavljanje R3
	MOV	PC, LR	; povratak iz potprograma
POD	DW	5	

**Komentar rješenja:**

Potrebno je uočiti da prilikom prijenosa/vraćanja parametara putem memorijskih lokacija koje se nalaze iza poziva potprograma, te lokacije ne smijemo „izvoditi“ jer se na njima ne nalazi program nego podaci. Budući da nakon poziva potprograma ne možemo koristiti naredbu skoka da bi preskočili parametre (jer se oni nalaze odmah nakon poziva potprograma), potrebno je povratnu adresu u LR prije povratka iz potprograma uvećati za broj memorijskih lokacija koje su zauzeli parametri.

### 2.3.4. Prijenos parametara pomoću stoga i vraćanje rezultata registrom (FRISC 2.4.5)

Riješen: DA    Težina: ★★

Napisati potprogram **POTP** koji preko stoga prima tri parametra u zapisu 2'k. Potprogram ispituje sadržaj prvog parametra na stogu (ovdje se pod prvim parametrom misli na parametar najbliži vrhu stoga), te ako je negativan oduzme drugi parametar od trećeg parametra na stogu, a inače ih zbraja. Potprogram vraća rezultat u registru **R0**. Parametre sa stoga uklanja pozivatelj.

Napisati glavni program koji iz memorijskih lokacija od adrese na labelama **A**, **B** i **C** uzima tri parametra koje šalje u potprogram **POTP**, rezultat sprema na adresu 200<sub>16</sub> i uklanja parametre sa stoga.

**Prijedlog rješenja:**

U glavnom programu potrebno je inicijalizirati stog te dohvatiti podatke sa labela **A**, **B** i **C**. Nakon toga, dohvaćene riječi stavljaju se na stog kao parametri za potprogram (naredbom **STMFD R13!, {R0, R1, R2}**). Nakon poziva i izvođenja potprograma (naredba **BL POTP**), potrebno je ukloniti parametre sa stoga što je najjednostavnije napraviti uvećanjem registra **R13** za 12<sub>10</sub>. Konačno, glavni program sprema rezultat izvođenja potprograma na lokaciju 200<sub>16</sub>.

Potprogram prvo sprema na stog registre koje mijenja, a zatim sa stoga treba učitati parametre u registre. Međutim, na vrhu stoga se nalazi kontekst kojega pri učitavanju parametara treba „preskočiti“. To se ostvaruje uvećavanjem pokazivača stoga za odmak 12<sub>10</sub>. Kako kasnije ne bi bilo potrebno vraćati pokazivač stoga na pravu adresu, uvećana adresa vrha stoga se sprema u **R0**. Tada se pomoću registra **R0** dohvaćaju parametri, naredbom **LDMFD R0, {R3, R4, R5}**. Povratna adresa se sprema u registar **LR**, tako da se ona ne uzima u obzir prilikom računanja odmaka (za razliku od procesora FRISC). Ostatak potprograma istovjetan je potprogramu iz zadatka 2.3.2.

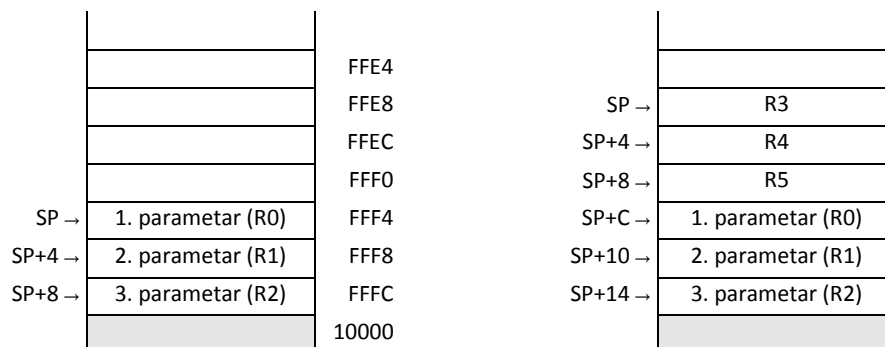
Rješenje:			
GLAVNI	ORG	0	
	MOV	R13, #10<12	; inicijalizacija stoga
	LDR	R0, A	; učitaju se parametri u R0, R1 i R2
	LDR	R1, B	
	LDR	R2, C	
	STMFD	R13!, {R0, R1, R2}	; spremi parametre na stog
	BL	POTP	; poziv potprograma
	ADD	R13, R13, #0C	; ukloni parametre sa stoga (3 parametra)
	MOV	R2, #2<8	
	STR	R0, [R2]	; spremi rezultat na lokaciju 200
	HALT		
POTP	STMFD	R13!, {R3, R4, R5}	; spremanje registara (konteksta)
	ADD	R0, R13, #0C	; preskakanje spremljenog konteksta na stogu
	LDMFD	R0, {R3, R4, R5}	; učitavanje parametara sa stoga
	ORRS	R3, R3, R3	; ispitaj prvi
	SUBMI	R0, R5, R4	; ako je prvi < 0, treci - drugi -> R0
	ADDPL	R0, R5, R4	; ako je prvi >= 0, drugi + treci -> R0
KRAJ	LDMFD	R13!, {R3, R4, R5}	; obnavljanje registara (konteksta)
	MOV	PC, LR	; povratak iz potprograma
A	DW	123	; podatci koji se zbrajaju ili oduzimaju
B	DW	-456	
C	DW	-1	; oznaka zbrajanja ili oduzimanja

### Komentar rješenja:

Ovakav način prijenosa parametara (stogom) i vraćanja rezultata (registrom) je vrlo praktičan i uobičajen, jer omogućava rekurzivne pozive i ne ograničava broj parametara koji se mogu prenijeti potprogramu. Prevoditelji za više programske jezike daju sličan asemblerski program (iako se obično zbog optimizacije parametri prenose registrima kad god je to moguće).

Lijeva slika prikazuje sadržaj stoga pri samom ulasku u potprogram **POTP**, prije izvođenja prve naredbe potprograma. Pokazivač stoga pokazuje na lokaciju gdje je spremljen sadržaj registra **R0** (jer se on sprema na najnižu memorijsku lokaciju). Na višim lokacijama se nalaze redom registri **R1** i **R2**. Sadržaj stoga nakon spremanja konteksta **STMFD R13!, {R3, R4, R5}**

prikazan je na desnoj slici – vrijednost registara s manjim rednim brojem stavlja se na manje adrese. Naredbom **ADD R0,R13,#0C** u **R0** se upisuje adresa na stogu gdje se nalazi 1. parametar. Izvođenjem naredaba **LDMFD R0,{R3,R4,R5}**, ne mijenja se sadržaj registra **SP** pa tako ni okvir stoga, a parametri se učitavaju u registre **R3,R4** i **R5**. Nakon obnavljanja konteksta, stanje stoga je ponovno kao na lijevoj slici. Naposljetku, glavni program će ukloniti parametre čime stog postaje prazan.



### 2.3.5. Prijenos parametara u potprogram i vraćanje rezultata pomoću stoga (FRISC 2.4.7.)

Riješen: DA    Težina: ★★★

Napisati potprogram **POTP** koji prima tri parametra preko stoga. Potprogram ispituje sadržaj prvog parametra na stogu, te ako je negativan oduzme drugi parametar od trećeg parametra na stogu, a inače ih zbraja. Potprogram vraća rezultat preko stoga. Parametre i rezultat sa stoga uklanja pozivatelj.

Napisati glavni program koji čita  $300_{10}$  podataka koji počinju od adrese  $1000_{16}$  i šalje tri po tri podatka u potprogram **POTP**. Rezultati se redom spremaju od adrese  $5000_{16}$ .

#### Prijedlog rješenja:

Za razliku od procesora FRISC gdje vraćanje rezultata preko stoga predstavlja problem s obzirom na to da se i povratna adresa sprema na stog, procesor ARM 7 povratnu adresu sprema u registar **R14 (LR)** što značajno olakšava rješenje.

#### Rješenje:

```

ORG      0
GLAVNI   MOV      SP, #10<12      ; inicijalizacija stoga

        MOV      R0, #1<12        ; početna adresa izvorišnog bloka
        MOV      R1, #D 100        ; brojač podataka (3*100)
        MOV      R3, #5<12        ; početna adresa odredišnog bloka
PETLJA   LDMIA    R0!, {R4, R5, R6} ; dohvat podataka i povećavanje pokazivača
        STMFD    SP!, {R4, R5, R6} ; slanje parametara na stog

        BL       POTP              ; poziv potprograma

(nastavak na sljedećoj stranici)

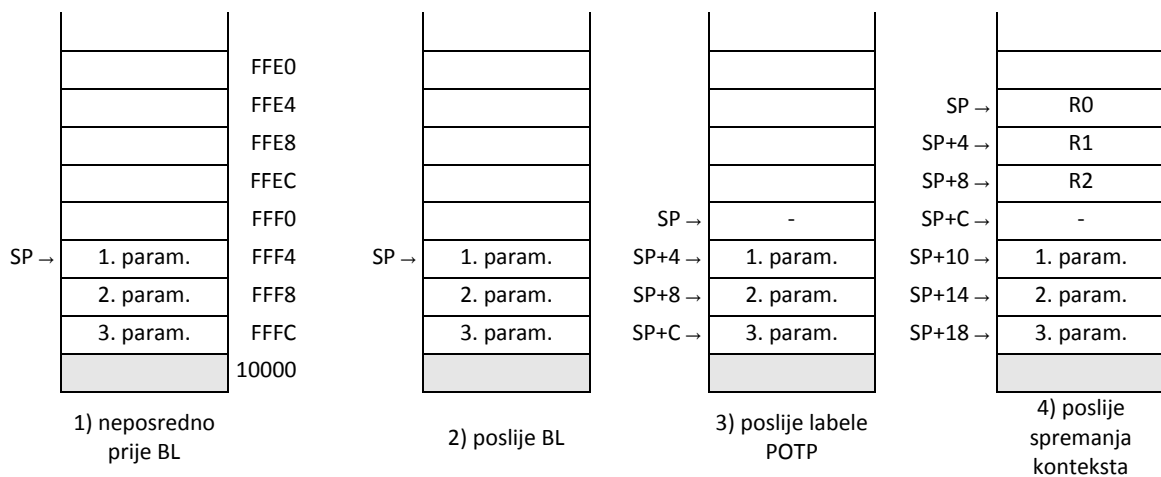
REZULT   LDR      R2, [SP], #10     ; skidanje rezultata (povratne vrijednosti)...
        ; ...i uklanjanje parametara
        STR      R2, [R3], #4      ; spremanje rezultata i povećavanje pokazivača
        SUBS     R1, R1, #1        ; brojač
        BNE      PETLJA           ; ako nije gotovo, vrati se natrag na petlju

```

	HALT		
POTP	SUB	SP, SP, #4	; predviđeno mjesto za rezultat
	STMFD	SP!, {R0, R1, R2}	; čuvanje konteksta
	LDR	R0, [SP, #10]	; prvi parametar
	LDR	R1, [SP, #14]	; drugi parametar
	LDR	R2, [SP, #18]	; treći parametar
	ORRS	R0, R0, R0	; postavljanje zastavica
UPIS	SUBMI	R0, R2, R1	
	ADDPL	R0, R1, R2	
	STR	R0, [SP, #0C]	; upis rezultata na predviđeno mjesto na stogu
KRAJ	LDMFD	SP!, {R0, R1, R2}	; obnovi kontekst
VAN	MOV	PC, LR	; izađi iz potprograma

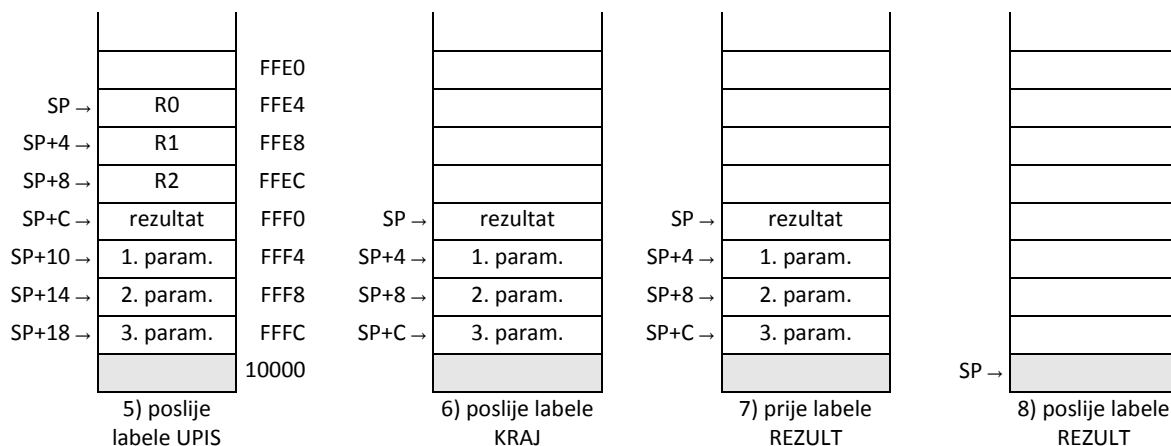
### Komentar rješenja:

Sljedeće slike prikazuju niz stanja stoga tijekom izvođenja programa (u rješenju i na slici odmaci su u heksadekaskoj bazi). Na početku izvođenja stog je prazan, što nije prikazano slikom. Osim što glavni program u petlji 100<sub>10</sub> puta poziva potprogram, samo pozivanje potprograma je slično zadatku 2.3.4. Svakom pozivu potprograma prethodi učitavanje po tri podataka u registre **R4, R5 i R6** (**LDMIA R0!, {R4,R5,R6}**). Znak **!** iza registra **R0** označava da će se vrijednost registra **R0** nakon naredbe promijeniti (u ovom slučaju uvećati za 12<sub>10</sub>). Slijedi stavljanje po tri parametra na stog (stanje 1) naredbom **STMFD SP!, {R4,R5,R6}**, a nakon poziva potprograma naredbom **BL** neće se na stog spremati povratna adresa kao kod procesora FRISC, nego se povratna adresa kod procesora ARM 7 sprema u registar **R14 (LR)** (stanje 2). To je u ovom slučaju adresa naredbe na labeli **REZULT** (naredba **LDR R2, [SP], #10**)



U potprogramu se prvo naredbom **SUB SP, SP, #4** na vrhu stoga rezervira mjesto za rezultat (stanje 3). Nakon toga treba spremati kontekst naredbom **STMFD SP!, {R0,R1,R2}** (stanje 4). Naredbom **STR R0, [SP, #0C]** na labeli **UPIS** sprema se rezultat na predviđeno mjesto na stogu, a SP je nakon naredbe ostao nepromijenjen (stanje 5). Nakon labele **KRAJ** stanje na stogu se mijenja jer je obnovljen kontekst, odnosno spremljeni registri **R0, R1 i R2** vraćeni su u originalno stanje (stanje 6). Prilikom izlaska iz potprograma nema promjena na stogu.

Povratak naredbom **MOV PC, LR** će uzeti povratnu adresu iz **R14** i vratiti izvođenje u glavni program (stanje 7). Budući da se rezultat nalazi na vrhu stoga, naredbom **LDR R2, [SP], #10** učitava se rezultat u registar R2 i istodobno sa stoga uklanjamo rezultat i sva tri parametra (stanje 8) jer se **SP** nakon naredbe uveća za 16<sub>10</sub>.



### 2.3.6. Kombinirani načini prijenosa parametara i vraćanja rezultata iz potprograma (FRISC 2.4.8.)

Riješen: DA Težina: ★★

Napisati potprogram **NAJMANJI** koji u bloku 32-bitnih brojeva u zapisu 2'k pronalazi najmanji broj. Glavni program šalje dva parametra potprogramu: adresu početka bloka šalje preko stoga, a veličinu bloka preko registra **R0**. Potprogram preko fiksne memorijske lokacije **REZ** vraća iznos najmanjeg broja u bloku. Potprogram treba čuvati stanja registara. Parametre sa stoga treba ukloniti pozivatelj.

Napisati glavni program koji treba pozvati potprogram **NAJMANJI** za blok na adresi 4000<sub>16</sub> koji sadrži 100<sub>16</sub> podataka. Nakon izvođenja potprograma, glavni program treba spremiti vrijednost najmanjeg broja na memorijsku lokaciju zadanu labelom **MIN**, koja se u memoriji nalazi odmah iza glavnog programa. Također, glavni program treba u bloku obrisati sve podatke različite od najmanjeg broja.

#### Prijedlog rješenja:

Kao što pokazuje ovaj zadatak, jedan potprogram može kombinirati različite načine prijenosa parametara i vraćanja rezultata. Glavni program treba inicijalizirati stog, postaviti parametre za potprogram, pozvati potprogram i spremiti rezultat izvođenja na labelu **MIN**. Nakon toga, treba ponovo prijeći po cijelom bloku i sve podatke koji su različiti od dobivenog minimuma zamijeniti ničicama.

Potprogram prima jedan parametar preko stoga (adresu početka bloka), a drugi parametar preko registra **R0** (broj podataka u bloku). Spremanje konteksta i dohvrat parametra sa stoga odvija se jednako kao u prethodnim primjerima. Kontekst sačinjavaju svi registri opće namjene koje potprogram mijenja, uključujući i registar **R0** preko kojeg se šalje parametar u potprogram (jer se i **R0** mijenja).

Za traženje najmanjeg broja koristit će se registar **R4** u kojemu će se čuvati trenutni minimum. Kao početni minimum postaviti će se vrijednost prvog podatka u bloku. Nakon toga treba u petlji prijeći sve ostale podatke te ih uspoređivati s trenutnim minimumom. Ako

je neki podatak manji od trenutnog minimuma, on postaje trenutni minimum i nastavlja se daljnja obrada bloka sve dok se ne obradi i zadnji podatak.

**Rješenje:**

```

GLAVNI    ORG      0
          MOV      SP, #10<12      ; inicijalizacija stoga

          MOV      R2, #4<12        ; R2 - početak bloka
          STMFD    SP!, {R2}        ; adresa početka bloka na stog
          MOV      R0, #1<8         ; R0 - broj podataka

          BL       NAJMANJI         ; poziv potprograma

          ADD      SP, SP, #4        ; čišćenje stoga, jedan parametar
          LDR      R1, REZ          ; dohvat rezultata u R1...
          STR      R1, MIN          ; ...i spremanje najmanjeg na adresu MIN

POC        LDR      R6, [R2]         ; učitati podatak za usporedbu
          CMP      R1, R6           ; preskaču se isti (najmanji) podatci
          BEQ      PRESKOK

          MOV      R3, #0           ; ako nisu isti, stavlja se 0 na mjesto broja
          STR      R3, [R2]         ; ... i sprema se na isto mjesto

PRESKOK    ADD      R2, R2, #4        ; povećava se adresa za sljedeći broj
          SUBS     R0, R0, #1        ; smanjuje se brojač i vrti se petlja
          BNE      POC
          HALT

MIN        DW      0
REZ        DW      0                ; memorijska lokacija za povratnu vrijednost

          ; potprogram
NAJMANJI   STMFD    SP!, {R0, R2, R3, R4} ; spremanje konteksta

          LDR      R2, [SP, #10]     ; R2 je adresa bloka (parametar sa stoga)
          LDR      R4, [R2]         ; R4 je trenutni najmanji (proglašen prvi)

PETLJA     LDR      R3, [R2]         ; trenutni podatak (za prvi prolaz isti)
          CMP      R3, R4           ; usporedba je li manji od najmanjeg?

R3_MANJI   MOVLT    R4, R3          ; signed! ako <, postaje trenutno najmanji
          ADD      R2, R2, #4        ; pomak adrese za novi broj
          SUBS     R0, R0, #1        ; smanjiti brojač i vrti se petlja
          BNE      PETLJA

          STR      R4, REZ          ; spremanje rezultata u memorijsku lokaciju

KRAJ       LDMFD    SP!, {R0, R2, R3, R4} ; obnova konteksta
          MOV      PC, LR

```

**Komentar rješenja:**

U potprogramu se parametar sa stoga dohvaća naredbom **LDR R2,[SP,#10]**, nakon što je kontekst spremljen na stog, a stanje na stogu u tom trenutku je prikazano na slici (odmaci su u heksadekadskoj bazi).

		FFE8
SP →	R0	FFEC
SP+4 →	R2	FFF0
SP+8 →	R3	FFF4
SP+C →	R4	FFF8
SP+10 →	parametar	FFFC
		10000

## 2.4. Opći programski zadatci

U ovom potpoglavlju nalaze se zadatci koji u sebi kombiniraju gradivo iz prethodnih potpoglavlja. Često se radi o zadacima sa starih ispita, u kojima se obično traži od studenata da pokažu razumijevanje gradiva i snalaženje u primjeni stečenih znanja.

### 2.4.1. Usporedba i množenje 32-bitnih brojeva (ZI10)

Riješen: DA    Težina: ★

Napisati odsječak programa koji sa stoga učitava 64-bitni broj zapisan u formatu 2'k, zatim uspoređuje dva 32-bitna broja koji se nalaze u **R0** i **R1** te ako su isti njihov umnožak pribraja 64-bitnom broju i pohranjuje ga natrag na isto mjesto na stog. Pretpostavka: stog je inicijaliziran i podaci su već na stogu i u registrima.

#### Prijedlog rješenja:

Ovaj zadatak je najjednostavnije riješiti pomoću naredbe **SMLAL** koja radi upravo ono što se traži u zadatku.

#### Rješenje:

```

...
LDMFD    SP!, {R2, R3}    ; učitavanje 64-bitnog broja sa stoga
CMP      R0, R1            ; usporedba R0 i R1
SMLALEQ  R3, R2, R0, R1    ; množenje R0 i R1 i zbrajanje sa R2 i R3
STMEQFD  SP!, {R2, R3}    ; spremanje na ista mjesta na stog
...

```

#### Komentar rješenja:

Primijetite da se prilikom učitavanja 64-bitnog podatka sa stoga u registru **R2** nalazi nižih 32 bita, a u registru **R3** viših 32 bita. Zbog toga je potrebno obrnuti redoslijed registara **R2** i **R3** kod naredbe **SMLAL** (jer se u prvom registru nalazi viših 32 bita, a u drugom nižih 32 bita). Redoslijed navođenja registara **R0** i **R1** u istoj naredbi je proizvoljan.

Dodavanjem sufiksa **EQ**, naredbe **SMLAL** i **STMF** će se izvesti samo u slučaju kad je ispunjen uvjet da su registri **R0** i **R1** jednaki. Zastavice se postavljaju naredbom **CMP R0,R1**. Budući da naredba **SMLALEQ R2,R3,R0,R1** ne postavlja zastavice, one vrijede i za sljedeću naredbu **STMEQFD SP!,{R2,R3}**. Pažnja: kod sufiksa naredaba za višestruki prijenos podataka u/iz memorije (**LDM** i **STM**) prvo dolazi polje uvjeta, a potom način adresiranja (dakle ispravno je **STMEQFD**, a krivo je **STMFDEQ**)

Da smo htjeli da naredba **SMLALEQ R2,R3,R0,R1** postavlja zastavice, onda bi dodali sufiks **S** na kraju naredbe te bi ona glasila **SMLALEQS R2,R3,R0,R1**.



## 2.4.2. Potprogram za računanje srednje vrijednost četiriju brojeva (FRISC 2.5.4.)

Riješen: DA    Težina: ★★

Napisati potprogram **AVGB** koji treba izračunati srednju vrijednost četiriju 8-bitnih podataka u zapisu NBC. Podatci se prenose u potprogram preko stoga kao jedan 32-bitni podatak, gdje je svaki bajt jedan 8-bitni NBC podatak. Srednju vrijednost treba vratiti kao 32-bitni rezultat preko registra **R0**. Zbrajanje bajtova treba riješiti petljom. Napisati glavni program koji za 8-bitne podatke u zapisu NBC:  $3_{16}$ ,  $1_{16}$ ,  $4F_{16}$  i  $5_{16}$  računa srednju vrijednost i sprema je na lokaciju **REZ**. Pozivatelj treba ukloniti parametar sa stoga.

### Prijedlog rješenja:

Glavni program učitava jedan 32-bitni podatak iz memorije (zapisan na lokaciji **PODATAK**) koji sadrži zadana četiri 8-bitna podatka, i šalje ga potprogramu preko stoga. Nakon izvođenja potprograma, registrom **R0** prima se rezultat potprograma i sprema ga se na zadanu lokaciju u memoriju.

Potprogram za izračunavanje srednje vrijednosti treba izdvojiti pojedine bajtove i zbrojiti ih, te na kraju zbroj podijeliti sa četiri. Potprogram učitava 32-bitni podatak sa stoga u registar **R1**. Dobiveni podatak rastavlja se na bajtove pomoću petlje koja se izvodi četiri puta. Prije petlje briše se **R0** jer će mu se u svakom koraku petlje pribrojiti vrijednost jednog od bajtova. U svakom koraku petlje se pomoću maske  $FF_{16}$  izdvaja jedan bajt podatka iz **R1** i pribraja se registru **R0**. Također se **R1** rotira za 8 bitova udesno kako bi se u sljedećem koraku petlje izdvojio sljedeći, viši bajt. Kada su podatci razdvojeni i zbrojeni, dijeli ih se sa 4 logičkim pomakom za 2 bita udesno (**MOV R0, R0, LSR #2**). Rezultat se vraća registrom **R0**.

### Rješenje:

	ORG	0	
GLAVNI	MOV	SP, #10<12	; inicijalizacija stoga
	LDR	R0, PODATAK	; dohvat 4 bajta i stavljanje na stog
	STMFD	SP!, {R0}	
	BL	AVGB	; poziv potprograma
	ADD	SP, SP, #4	; čišćenje stoga
	STR	R0, REZ	; spremanje rezultata
	HALT		
PODATAK	DW	03014F05	; 4 podatka (ili: PODATAK DB 5,4F,1,3)
REZ	DW	0	; mjesto za rezultat
AVGB	STMFD	SP!, {R1, R2, R3}	; spremanje konteksta
	LDR	R1, [SP, #0C]	; učitavanje parametra tj. 32-bitnog podatka
	MOV	R2, #4	; brojač za petlju
	MOV	R0, #0	; početni zbroj bajtova

(nastavak na sljedećoj stranici)

PETLJA	AND	R3, R1, #0FF	; maskiranje bajta
	MOV	R1, R1, ROR #8	; pomak podatka (može i SHR)
	ADD	R0, R3, R0	; dodavanje zbroju
	SUBS	R2, R2, #1	; smanjivanje brojača za petlju
	BNE	PETLJA	
DALJE	MOV	R0, R0, LSR #2	; dijeljenje sa 4

LDMFD	SP!, {R1, R2, R3}	; obnova konteksta
MOV	PC, LR	; povratak

**Komentar rješenja:**

U potprogramu se, umjesto maskiranja i rotiranja, može u petlji čitati izravno bajt po bajt iz memorije, tj. sa stoga gdje se nalazi parametar. To se može postići tako da se prvo u jedan registar spremi adresa parametra, na primjer u **R4** pomoću **ADD R4, SP, #odmak** (iznos odmaka ovisit će o tome kako je napisan potprogram, odnosno ovisit će o broju registara koji se spremaju kao kontekst). U petlji bi se bajtovi učitali naredbom **LDRB Rx, [R4], #1**. Naravno, **R4** bi trebalo spremati i obnavljati s ostatkom konteksta.

### 2.4.3. Zamjena dva 16-bitna broja njihovim 32-bitnim zbrojem (FRISC 2.5.7.)

**Riješen: DA    Težina: ★**

U memoriji je blok koji se sastoji od parova 16-bitnih podataka u zapisu 2'k. Memorijski blok počinje na adresi  $2000_{16}$ , a zaključen je parom podataka  $F0F0_{16}$  i  $F0F0_{16}$ .

Napisati program koji svaki par 16-bitnih brojeva u bloku memorije zamjenjuje novim 32-bitnim podatkom na sljedeći način. Za svaki par 16-bitnih brojeva u bloku treba izračunati njihov zbroj.

Zbroj treba biti u 32-bitnom zapisu 2'k i treba ga spremati u blok na mjesto početnog para brojeva.

Prilikom pretvorbe podataka, program također treba prebrajati koliko je negativnih 16-bitnih podataka bilo u početnom bloku i to na kraju programa treba zapisati na lokaciju **BROJAC**.

Primjer (zbrajanje za jedan par podataka): U memoriji se na adresi  $2000_{16}$  nalazi podatak  $FFFB_{16}$ , a na adresi  $2002_{16}$  podatak  $0002_{16}$ , što su brojevi -5 i +2. Ovaj par treba pretvoriti u broj  $-3 = -5 + 2$  i zapisati podatak  $FFFFFFFD_{16}$  na adresu  $2000_{16}$ , a brojač treba povećati za 1.

**Prijedlog rješenja:**

Program treba u petlji učitali jedan po jedan par 16-bitnih brojeva u registre. S obzirom da je učitane brojeve potrebno usporediti s oznakom kraja  $F0F0_{16}$ , prvo se naredbom **LDRH** učitavaju brojevi kako bi se usporedili sa zaključnim parom brojeva. Ako su oba broja jednaka zaključnom podatku  $F0F0_{16}$ , petlja se završava.

Nakon provjere, za ponovno učitavanje koristimo naredbu **LDRSH**. Prednost te naredbe je predznačno proširivanje broja na 32 bita pa to nije potrebno raditi pomoću pomaka, što olakšava rješavanje. Osim pretvorbe, također treba povećati brojač negativnih brojeva koji se čuva u registru **R1**. Brojevi se zbrajaju tek nakon pretvaranja u zapis 2'k.

**Rješenje:**

	ORG	0	
GLAVNI	MOV	R0, #2<12	; adresa početka bloka je u R0
	MOV	R1, #0	; brojač negativnih je u R1
	LDR	R4, MASKA	; učitavanje maske
			; provjera kraja bloka
PETLJA	LDRSH	R2, [R0]	; učitati prvi 16-bitni broj u R2

	CMP	R2, R4	; provjeriti je li F0F0
	BNE	PRVI	; ako nije, skočiti na obradu prvog podatka
	LDRSH	R3, [R0, #2]	; učitati drugi 16-bitni broj u R3 + odmak za 2
	CMP	R3, R4	; provjeriti je li F0F0
	BEQ	KRAJ	; ako jest, skočiti na kraj
PRVI	; provjera predznaka i pretvorba prvog podatka		
	CMP	R2, #0	; provjeriti predznak broja
	ADDLT	R1, R1, #1	; ako je negativan, povećati brojač negativnih
DRUGI	; provjera predznaka i pretvorba drugog podatka		
	LDRSH	R3, [R0, #2]	; učitati drugi 16-bitni broj u R3 + odmak za 2
	CMP	R3, #0	; provjeriti predznak broja
	ADDLT	R1, R1, #1	; ako je negativan, povećati brojač negativnih
ZBROJI	; zbrajanje oba podatka i spremanje rezultata		
	ADD	R2, R2, R3	; zbrajanje podataka u zapisu 2'k
	STR	R2, [R0]	; spremiti 32-bitni rezultat preko podataka
	ADD	R0, R0, #4	; pomak adrese za 4
	B	PETLJA	; nastavak petlje
KRAJ	STR	R1, BROJAC	; spremiti brojač iz R1 u zadanu lokaciju
	HALT		
BROJAC	DW	0	
MASKA	DW	0F0F0	; oznaka kraja

Podaci se mogu odmah učitati pomoću naredbe **LDRSH** jer je podatak kojim označavamo kraj (F0F0) manji od 20 bita pa se predznačnim proširivanjem neće promijeniti. Pojednostavnite program tako da prilikom provjere kraja bloka učitajte oba 16-bitna podatka jednom naredbom **LDR** i ispitajte kraj bloka samo jednom naredbom **CMP**. Isto tako, izbacite naredbu uvećavanja adresnog registra **R0** tako što ćete iskoristiti registarsko adresiranje s postindeksiranjem u jednoj od prethodnih naredaba.

#### 2.4.4. Premještanje podataka unutar bloka (FRISC 2.5.8.)

Riješen: DA    Težina: ★★★

Napisati program koji će u bloku podataka nepoznate duljine sve negativne brojeve postaviti na početak bloka, a sve pozitivne na kraj. 32-bitni podatci u bloku su u zapisu s bitom za predznak. Blok je zaključen podatkom  $00000000_{16}$  (radi jednostavnosti pretpostavite da u bloku nema podatka  $80000000_{16}$ ). Početna adresa bloka je  $1000_{16}$ .

##### Prijedlog rješenja:

Uređivanje bloka temelji se na dva pokazivača. Prvi kreće od početka bloka ka kraju (pokazivač u **R0**), a drugi kreće od kraja bloka k njegovom početku (pokazivač u **R1**). Kad se pokazivači susretnu („sudare“), postupak je gotov. Prvi pokazivač **R0** pomiče se sve dok pokazuje na negativne podatke (jer su oni na početku bloka, odnosno na pravom mjestu) i zaustavlja se dok naiđe na prvi pozitivan podatak (koji je na krivom mjestu u bloku). Obrnuto, drugi pokazivač **R1** pomiče se dok pokazuje na pozitivne podatke (jer su oni na kraju bloka, odnosno na pravom mjestu) i zaustavlja se dok naiđe na prvi negativan podatak (koji je na krivom mjestu u bloku). Pri svakom pomicanju pokazivača mora se provjeravati jesu li se sudarili. Kad se pronađu prvi pozitivan podatak od početka bloka i prvi negativan s

kraja bloka, treba im zamijeniti mjesta i nastaviti s traženjem pomoću **R0** i **R1** od sljedećih podataka.

Budući da broj podataka u bloku nije poznat, prije opisanog postupka treba pronaći zadnji podatak u bloku i postaviti **R1** da pokazuje na njega.

Rješenje:			
GLAVNI	ORG	0	
	MOV	R1, #1<12	; adresa početka bloka
POMICI	LDR	R2, [R1], #4	; dohvatiti podatak iz bloka
	CMP	R2, #0	; je li to zaključni podatak 0?
	BEQ	R1_NA_KRAJ	; ako jest, to je kraj bloka
	B	POMICI	
R1_NA_KRAJ	SUB	R1, R1, #8	; postaviti R1 na zadnji podatak u bloku...
			; ...jer je postindeksiranjem pokazivač prošao zaključni podatak
	MOV	R0, #1<12	; postaviti R0 na početak bloka
			; postupak premještanja podataka
TRAZI_POZ	LDR	R2, [R0]	; traži prvi pozitivan broj od početka bloka
	ORRS	R2, R2, R2	
	BPL	TRAZI_NEG	; kad se nađe pozitivni, zaustaviti traženje
	ADD	R0, R0, #4	; pomaknuti pokazivač na sljedeći podatak
	CMP	R0, R1	; provjeriti je li se R0 „sudario“ s R1
	BEQ	KRAJ	; ako jest, onda je postupak gotov
	B	TRAZI_POZ	; vratiti se na provjeru sljedećeg podatka
TRAZI_NEG	LDR	R3, [R1]	; traži prvi negativan broj od kraja bloka
	ORRS	R3, R3, R3	
	BMI	ZAMJENA	; ako je negativan, zaustaviti traženje
	SUB	R1, R1, #4	; pomaknuti pokazivač na sljedeći podatak
	CMP	R0, R1	; provjeriti je li se R0 „sudario“ s R1
	BEQ	KRAJ	; ako jest, onda je postupak gotov
	B	TRAZI_NEG	; vratiti se na provjeru sljedećeg podatka
ZAMJENA	STR	R3, [R0]	; zamijeniti mjesta negativnog i pozitivnog
	STR	R2, [R1]	
			; nastavak traženja od sljedećih lokacija
			; pomaknuti pokazivače R0 i R1 uz provjeru „sudara“
	ADD	R0, R0, #4	
	CMP	R0, R1	
	BEQ	KRAJ	
(nastavak na sljedećoj stranici)			
	SUB	R1, R1, #4	
	CMP	R0, R1	
	BEQ	KRAJ	
	B	TRAZI_POZ	; nakon zamjene, ponoviti postupak
KRAJ	HALT		

**Komentar rješenja:**

Na samom početku programa treba postaviti pokazivač **R1** na zadnji podatak u bloku (misli se na zadnji korisni podatak, a ne na podatak koji označava kraja bloka). To se radi u petlji **POMICI** u kojoj se pokazivač **R1** pomiče od početka bloka do zaključnog podatka. Iza petlje (na labeli **R1\_NA\_KRAJ**) se **R1** treba vratiti za jedno mjesto unatrag jer pokazuje predaleko (odnosno pokazuje na zaključni podatak).

Postupak premještanja podataka u bloku sastoji se od tri koraka. Prvi je traženje prvog pozitivnog podatka od početka bloka. Drugi je traženje prvog negativnog podatka od kraja bloka. Treći korak je njihova zamjena. Ovaj se postupak ponavlja dok se ne pregledaju (i eventualno zamijene) svi podatci u bloku.

U petlji **TRAZI\_POZ** se prvi pokazivač **R0** pomiče prema kraju bloka, a u **R2** se dohvaća podatak na koji pokazuje **R0** i ispituje se predznak podatka. To se ponavlja sve dok se ne naiđe na prvi pozitivni podatak.

U petlji **TRAZI\_NEG** se drugi pokazivač **R1** pomiče prema početku bloka, a u **R3** se dohvaća podatak na koji pokazuje **R1** i ispituje se predznak podatka. To se ponavlja sve dok se ne naiđe na prvi negativni podatak.

Nakon što su pronađena ova dva podatka koji su na krivim mjestima, oni se zamjenjuju (dio programa od labela **ZAMJENA**) čime dolaze na prava mjesta. Pokazivači **R0** i **R1** pomiču se za po jedno mjesto dalje, odnosno na sljedeće podatke. Time se **R0** i **R1** stalno približavaju sredini bloka i jedan ka drugome. Ovaj se postupak ponavlja dok se cijeli blok ne obradi. Postupak završava kad se **R0** i **R1** „sudare“, a to treba provjeriti svaki puta kad se **R0** ili **R1** pomiču.

Ispitivanje predznaka broja je jednostavno jer se u zapisu s bitom za predznak po najvišem bitu raspoznaje je li broj pozitivan ili negativan (uz izuzetak „pozitivne ništice“ koja ovdje označava kraj bloka, te „negativne ništice“ koja se prema tekstu zadatka ne može pojaviti u bloku)

Slijedi drugačije rješenje, kod kojeg se također koriste dva pokazivača, ali oba se pomiču od početka bloka prema kraju. Samostalno proučite kako radi ovaj program.

	MOV	R0, #1<12	; pokazivač traženja pozitivnih – početak
	MOV	R1, #1<12	; pokazivač traženja negativnih – početak
PETLJA	LDR	R3, [R0]	; provjera je li kraj bloka
	CMP	R3, #0	
	BEQ	KRAJ	; ako je kraj => gotovo
	BMI	NEGAT	; inače odredi predznak broja
POZIT	ADD	R1, R1, #4	; broj je pozitivan
	LDR	R4, [R1]	; pomak R1 do prvog negativnog
	CMP	R4, #0	
(nastavak na sljedećoj stranici)			
	BEQ	KRAJ	
	BPL	POZIT	
	STR	R3, [R1]	; zamjena pozitivnog i negativnog broja
	STR	R4, [R0]	
NEGAT	ADD	R0, R0, #4	; preskakanje negativnih brojeva
	MOV	R1, R0	; oba pokazivača iza niza negativnih brojeva

B	PETLJA
KRAJ	HALT

### 2.4.5. Potprogram za prebrajanje završnih ništica (FRISC 2.5.9.)

Riješen: DA    Težina: ★★

Napisati potprogram **CTZ** (*count trailing zeroes*) koji za podatak prenesen preko stoga, prebraja završne ništice (to su sve uzastopne ništice na najnižim bitovima). Na primjer, za broj  $3FC20180_{16}$ , odnosno  $0011\ 1111\ 1100\ 0010\ 0000\ 0001\ 1000\ 0000_2$  rezultat je 7. Rezultat se vraća preko registra **R0**.

U glavnom programu treba, korištenjem potprograma **CTZ**, prebrojiti završne ništice za svaki podatak iz bloka 32-bitnih podataka koji se nalazi od adrese  $500_{16}$  pa do  $1000_{16}$  (zadnji podatak je na adresi  $FFC_{16}$ ). Podatkom  $F0F0F0F0_{16}$  treba zamijeniti one brojeve u bloku koji imaju više od 11 završnih ništica.

#### Prijedlog rješenja:

U glavnom programu potrebno je petljom proći kroz cijeli blok podataka te svaki podatak pomoću stoga proslijediti potprogramu **CTZ**. Brojač za petlju bit će adresa podataka, čija će početna vrijednost biti  $500_{16}$ , a uvjet za kraj petlje je kada brojač postane jednak  $1000_{16}$ , tj. veći od adrese zadnjeg broja u bloku.

U potprogramu **CTZ** treba prvo sa stoga dohvatiti parametar. Prebrajanje završnih ništica u podatku se izvodi petljom po svim bitovima podatka, korištenjem rotacije udesno za jedan bit i provjeravanjem izlaznog bita (zastavica **C**). Dok je izlazni bit ništica, brojač se povećava. Kada izlazni bit više nije ništica, potprogram vraća vrijednost brojača.

#### Rješenje:

	ORG	0	
GLAVNI	MOV	SP, #10<12	; inicijalizacija stoga
	LDR	R3, ZAMJ	; podatak F0F0F0F0 za zamjenu
	MOV	R1, #5<8	; adresa početka bloka - brojač za petlju
PETLJA	LDR	R2, [R1]	; dohvat podatka iz bloka
	STMFD	SP!, {R2}	; staviti podatak na stog
	BL	CTZ	; poziv potprograma
	ADD	SP, SP, #4	; uklanjanje parametra sa stoga
	CMP	R0, #11	; usporedba broja ništica sa 11
	STRHI	R3, [R1]	; ako je više od 11, upiši F0F0F0F0 u blok
DALJE	ADD	R1, R1, #4	; pomak na sljedeći
	CMP	R1, #1<12	; provjera kraja bloka
	BNE	PETLJA	; ako nije kraj bloka, ponavljanje petlje
	HALT		
ZAMJ	DW	0F0F0F0F0	
(nastavak na sljedećoj stranici)			
			; potprogram
CTZ	STMFD	SP!, {R1, R2}	; spremanje konteksta
	LDR	R1, [SP, #8]	; dohvat parametra (podatka)
	MOV	R0, #0	; inicijalizacija rezultata
	MOV	R2, #32	; inicijalizacija brojača za petlju
LOOP	MOVS	R1, R1, ROR #1	; rotacija desnog bita u zastavicu C
	BCS	VAN	; ispitivanje bita: ako je 1, postupak je gotov

	ADD	R0, R0, #1	; povećavanje rezultata
	SUBS	R2, R2, #1	; smanji brojač petlje
	BNE	LOOP	
VAN	LDMFD	SP!, {R1, R2}	; obnova konteksta
	MOV	PC, LR	; povratak

**Komentar rješenja:**

Brojač za petlju u glavnom programu čuva se u **R1**, a to je ujedno i adresa pojedinog podatka u bloku. Svaki podatak se pomoću adrese dohvaća iz memorije i stavlja na stog kako bi se kao parametar poslao u potprogram (**STMFD SP!, {R2}**). Potprogram dohvaća poslani parametar u **R1** naredbom **LDR R1, [SP, #8]**, pri čemu je odmak 8 jer su spremanjem konteksta na stog (naredbom **STMFD SP!, {R1, R2}**) stavljena 2 registra (8 bajtova). Za brojenje završnih ništica koriste se brojač ništica u **R0** (koji pohranjuje izlazni rezultat) i brojač ispitanih bitova u **R2** (koristi se kao brojač petlje koja se smije izvesti najviše 32 puta i to u slučaju kad u podatku nema jedinica). Operacija rotacije udesno za jedno mjesto (**MOVS R1, R1, ROR #1**) omogućuje ispitivanje izlaznog, odnosno najnižeg bita pomoću zastavice **C**. Ako je zastavica **C** postavljena, pronađena je jedinica, petlja se prekida te se izlazi iz potprograma. Ako zastavica **C** nije postavljena, povećava se brojač završnih ništica i petlja se ponavlja (ali najviše 32 puta).

Za vježbu možete pokušati prebrajati vodeće (najviše) ništice. Također možete varirati i širinu podatka (na primjer: sve se radi sa 16-bitnim podacima). Riješite zadatak ako se kao parametar prenosi adresa podatka. Možete varirati i način prenošenja parametra u potprogram, kao i način vraćanja rezultata.

**2.4.6. Računanje sume niza (ZI06)****Riješen: DA    Težina: ★**

Napisati potprogram koji računa sumu  $N$  članova sljedećeg niza:  $Y(X, N) = X^2 + X^4 + X^8 + X^{16} \dots + X^{2^N}$ . Parametri  $X$  i  $N$  prenose se u potprogram preko registara **R0=X** i **R3=N**, a suma  $Y$  se vraća preko registra **R0**. Napisati i glavni program koji poziva potprogram s parametrima  $X=4$  i  $N=5$ . Zanimariti slučajeve prekoračenja opsega.

**Prijedlog rješenja:**

U glavnom programu smo proizvoljno zadali brojeve 4 i 5 kao parametre, te ih prenosimo putem registara **R0** i **R3** u potprogram. Prilikom spremanja konteksta primijetite da nismo spremili registar **R0** (čak štoviše, bilo bi krivo da smo ga spremili jer bi se obnavljanjem konteksta rezultat u **R0** prepisao početnom vrijednošću - u ovom zadatku je to parametar  $X$ ).

Rješenje:			
	ORG	0	; glavni program
	MOV	SP, #10<12	; inicijalizacija stoga
	MOV	R0, #4	; zadavanje vrijednosti parametara
	MOV	R3, #5	
(nastavak na sljedećoj stranici)			
	BL	NIZ1	; poziv potprograma
	HALT		; završetak
; potprogram			
NIZ1	STMFD	R13!, {R1, R2}	; spremanje konteksta
	MOV	R1, #1	; inicijalizacija brojača
	MOV	R2, #0	; inicijalizacija sume

PETLJA	ADD	R2, R2, R0, LSL R1	; množi s potencijom (LSL brojač) i sumira
	ADD	R1, R1, #1	; povećanje brojača
	CMP	R1, R3	; je li brojač došao do N?
	BLS	PETLJA	
NIZ1_VAN	MOV	R0, R2	; prebaciti rezultat u R0
	LDMFD	R13!, {R1,R2}	; obnavljanje konteksta
	MOV	PC, LR	; povratak iz potprograma

**Komentar rješenja:**

Rješenje zadatka je značajno olakšano činjenicom navedenom u tekstu zadatka, a to je da se zanemaruju prekoračenja opsega. Razmislite što biste sve trebali napraviti u realnom slučaju kad je moguće prekoračenje opsega (\*\*). Što biste napravili s preljevom? Biste li sumu mogli vratiti preko jednog registra? Kako bi izgledalo spremanje konteksta?

**2.4.7. Izračunavanje svih djelitelja 32-bitnog broja****Riješen: DA    Težina: ★★**

Napisati program koji računa sve djelitelje 32-bitnog broja u zapisu NBC. Broj je pohranjen na lokaciji **BR0J**. Djelitelje treba rastućim redoslijedom pohraniti u memoriju počevši od adrese 1000<sub>16</sub>. Niz djelitelja zaključiti brojem -1 u zapisu 2<sup>k</sup>. U blok djelitelja nije potrebno upisivati sam broj.

Za dijeljenje 32-bitnih brojeva u formatu NBC koristiti potprogram **DIJELI** (nije ga potrebno pisati). Djeljenik i djelitelj prenose se u potprogram preko stoga. Rezultat se vraća preko registra **R0**, a ostatak preko **R1**.

Specijalne slučajeve, brojeve 0 i 1 moguće je zanemariti.

**Prijedlog rješenja:**

Za broj N će raspon brojeva za koje treba utvrditi djeljivost biti od 1 do N/2. Brojeve u rasponu od N/2 do N nije potrebno ispitivati, jer, osim samog broja N, sigurno ne mogu biti njegovi djelitelji. Zbog toga ćemo u potprogram redom slati brojeve iz raspona od 1 do N/2 i provjeravati ostatak dijeljenja. Ako je ostatak jednak ničiti, znači da je broj poslan u potprogram djelitelj od N i treba ga spremati u blok memorije.

<b>Rješenje:</b>			
	ORG	0	
	MOV	SP, #10<12	; inicijalizacija stoga
	MOV	R2, #1<12	; adresa na koju se spremaju djelitelji
	LDR	R3, BR0J	; učitavanje broja
	MOV	R4, #1	; ispitivanje počinje od broja 1 (R4 je brojač)
	MOV	R5, R3, LSR #1	; izračunati BR0J/2
PETLJA	CMP	R4, R5	; je li kraj petlje?
	BHI	KRAJ	; kraj
	STMFD	SP!, {R3,R4}	; stavljanje parametara na stog
(nastavak na sljedećoj stranici)	BL	DIJELI	; poziv potprograma
	ADD	SP, SP, #8	; uklanjanje parametara sa stoga
	CMP	R1, #0	; ako nema ostatka...
	STREQ	R4, [R2], #4	; ...znači da broj jeste djelitelj -> spremanje
	ADD	R4, R4, #1	; uvećavanje mogućeg djelitelja za 1
	B	PETLJA	; i ponovo ispitavanje



KRAJ	MVN	R4, #0	; niz se zaključuje brojem -1
	STR	R4, [R2]	
	HALT		

Komentar rješenja: U registar **R3** spremljen je broj **N** kojemu je potrebno utvrditi sve djelitelje, a u registar **R5** je spremljen broj **N/2**. Registar **R4** je brojač petlje koji će poslužiti za prolazak kroz sve brojeve uključivo od 1 do **N/2**. Zbog toga što želimo ispitati i djeljivost s brojem **N/2** korišten je uvjet kraja petlje **HI** (*unsigned higher*) u naredbi skoka **BHI** nakon naredbe usporedbe **CMP R4,R5**. Razmotrite slučaj da je naredba skoka bila **BEQ**. Nakon ispitivanja kraja petlje, slijedi spremanje parametara na stog i poziv potprograma **DIJELI**. Ako nema ostatka, djelitelj se sprema na lokacije od adrese 1000<sub>16</sub>. Na kraju programa, niz djelitelja zaključuje se brojem -1 (u dvojnem komplementu).

### 2.4.8. Provjera točke na paraboli

Riješen: DA    Težina: ★★

Napisati potprogram koji provjerava pripada li točka (x,y) zadanoj paraboli. Točka je na paraboli ako vrijedi  $y=ax^2 + bx + c$ . Parametri x, y, a, b, c se u potprogram prenose preko stoga. Ako točka pripada paraboli u **R0** se vraća vrijednost 1, a ako ne pripada u **R0** se vraća 0. U glavnom programu pozvati potprogram koji će provjeriti pripada li točka (2,3) paraboli s parametrima a=2, b=4, c=1. Ukoliko pripada, na memorijsku lokaciju PBL upisati jedinicu, inače upisati ničticu.

#### Prijedlog rješenja:

U ovom zadatku, kao i kod nekoliko prethodnih zadataka, prikazan je prijenos parametara u potprogram putem stoga. U usporedbi s procesorom FRISC, kod procesora ARM 7 je prijenos jednostavniji jer se povratna adresa iz potprograma ne sprema na stog, nego u registar **R14 (LR)**.

Shodno tome se ni odmak, za učitavanje parametara sa stoga, ne uvećava za dodatne četiri adrese na stogu, niti je potrebno da povratna adresa bude na vrhu stoga neposredno prije povratka iz potprograma kao što je slučaj za procesor FRISC.

#### Rješenje:

	ORG	0	
GLAVNI	MOV	R13,#10<12	; inicijalizacija stoga
	MOV	R0, #2	; x
	MOV	R1, #3	; y
	MOV	R2, #2	; a
	MOV	R3, #4	; b
	MOV	R4, #1	; c
	STMF	R13!, {R0, R1, R2, R3, R4}	; parametri na stog
	BL	PARAB	; poziv potprograma
(nastavak			
na sljedećoj stranici)	ADD	R13, R13, #14	; „čišćenje“ stoga
	STR	R0, PBL	; spremi rezultat
	HALT		; kraj
PARAB	STMF	R13!, {R1, R2, R3, R4, R5, R12}	; spremanje konteksta
	ADD	R12, R13, #18	; izračunavanje adrese parametara

	LDMFD	R12, {R0, R1, R2, R3, R4}	; čitanje parametara
	MUL	R5, R0, R0	; $R5 = x * x$
	MLA	R5, R2, R5, R4	; $R5 = a * R5 + c$
	MLA	R5, R3, R0, R5	; $R5 = R5 + bx$ , računanje funkcije
	TEQ	R5, R1	; usporedba izračunate funkcije sa y
ISTI	MOVEQ	R0, #1	; povratna vrijednost 1
RAZLICITI	MOVNE	R0, #0	; povratna vrijednost 0
	LDMFD	R13!, {R1, R2, R3, R4, R5, R12}	; obnavljanje konteksta
	MOV	PC, LR	; povratak iz potprograma
PBL	DW	0	

### Komentar rješenja:

Primijetite da se u potprogramu koristi dodatni registar R12 (koji je prethodno spremljen na stog) u koji se sprema izračunata adresa parametara. Pri tome je svejedno jesmo li napisali naredbu **LDMFD R12, {R0, R1, R2, R3, R4}** ili **LDMFD R12!, {R0, R1, R2, R3, R4}** jer **R12** više ne koristimo dalje u potprogramu i nebitno nam je na koju adresu pokazuje nakon izvođenja. Ako ne bismo htjeli koristiti dodatni registar **R12**, nego bismo koristili izravno **R13** za adresiranje parametara, postoji nekoliko mogućnosti. U prvom slučaju prije učitavanja parametara treba pomaknuti **R13** za  $14_{16}$  ( $20_{10}$ ), tako da pokazuje na parametre. Prilikom učitavanja parametara ne mijenjamo vrijednost **R13** (nema uskličnika iza **R13** u trećoj naredbi potprograma **PARAB**). Budući da smo **R13** prethodno uvećali, prije obnavljanja konteksta moramo ga smanjiti za isti iznos  $14_{16}$ .

PARAB	STMFD	R13!, {R1, R2, R3, R4, R5}	; spremanje konteksta
	ADD	R13, R13, #14	; pozicioniranje R13 na parametre
	LDMFD	R13, {R0, R1, R2, R3, R4}	; čitanje parametara
	SUB	R13, R13, #14	; vraćanje R13 na kontekst
	...		

U drugom slučaju se **R13** također prije učitavanja parametara povećava za  $14_{16}$ , ali se prilikom učitavanja parametara mijenja (**R13!**), tj. uvećava se za još  $14_{16}$ . Zato je potrebno za ispravan povratak konteksta vrijednost **R13** smanjiti za  $14_{16} + 14_{16}$ , a to je  $28_{16}$ .

PARAB	STMFD	R13!, {R1, R2, R3, R4, R5}	; spremanje konteksta
	ADD	R13, R13, #14	; pozicioniranje R13 na parametre
	LDMFD	R13!, {R0, R1, R2, R3, R4}	; čitanje parametara
	SUB	R13, R13, #28	; vraćanje R13 na kontekst
	...		

Najkraće rješenje je da se kontekst spremi bez mijenjanja **R13**, tako da on nastavlja pokazivati na položaj parametara. Zatim se izravno mogu čitati parametri, ali opet bez promjene registra **R13**. Prije obnove konteksta treba **R13** pomaknuti tako da pokazuje na kontekst, a iznos pomaka odgovara veličini konteksta koji je u ovom slučaju  $14_{16}$  ( $20_{10}$ ).

PARAB	STMFD	R13, {R1, R2, R3, R4, R5}	; spremanje konteksta
	LDMFD	R13, {R0, R1, R2, R3, R4}	; čitanje parametara
	SUB	R13, R13, #14	; postavljanje R13 na kontekst
	...		

### 2.4.9. Izbacivanje podataka iz bloka

Riješen: DA    Težina: ★★

Napisati potprogram **IZBACI** koji iz niza 16-bitnih brojeva u zapisu NBC izbacuje brojeve vrijednosti 0. Potprogram treba prepisati izvorni blok podataka u novi, izmijenjeni blok koji neće sadržavati ničtice. Potprogram preko stoga kao parametre prima: adresu početka izvornog bloka brojeva, adresu početka izmijenjenog bloka brojeva i broj brojeva u bloku. Također, potprogram treba prebrajati izbačene ničtice, te njihov broj vratiti preko registra **R0**.

U glavnom programu treba iz memorije učitavati adrese početka izvornog bloka brojeva, izmijenjenog bloka brojeva i broj brojeva u bloku. Postoji ukupno 5 takvih blokova. Adrese početaka izvornih blokova zapisane su od adrese  $1000_{16}$ , adrese početaka izmijenjenih blokova zapisane su od adrese  $2000_{16}$ , a brojevi brojeva u bloku zapisani su od adrese  $3000_{16}$ . Nakon učitavanja adresa svakog bloka, potrebno je za taj blok pozvati potprogram **IZBACI**. Primljene rezultate – brojeve izbačenih ničtica – treba spremati od adrese  $4000_{16}$ . Parametre sa stoga treba izbrisati pozivatelj.

#### Prijedlog rješenja:

U glavnom programu treba obraditi više blokova na isti način, pozivanjem potprograma **IZBACI**. Zbog toga je potrebno napraviti glavnu petlju koja će se izvoditi onoliko puta koliko imamo različitih blokova: 5 puta kako je zadano u tekstu. U svakom koraku petlje treba pripremiti parametre za potprogram te ga pozivati.

Potprogram **IZBACI** će nakon dohvaćanja parametara izvoditi petlju čiji je broj koraka zadan parametrom koji definira broj podataka u izvornom bloku. U svakom koraku petlje dohvaća se po jedan podatak iz izvornog bloka te se ispituje. Ako je podatak ničtica, preskače se njegovo kopiranje u odredišni blok i povećava brojač ničtica, a u suprotnom se podatak kopira u odredišni blok.

Rješenje:			
	ORG	0	
	MOV	SP, #10<12	; inicijalizacija stoga
	MOV	R1, #1<12	; adresa bloka s adresama izvorišnih blokova
	MOV	R2, #2<12	; adresa bloka s adresama izmijenjenih blokova
	MOV	R3, #3<12	; adresa bloka s brojevima podataka u blokovima
	MOV	R4, #4<12	; adresa bloka za upis brojeva izbačenih ničtica
	MOV	R5, #5	; broj blokova
PETLJA	LDR	R7, [R1], #4	; učitati adresu jednog izvorišnog bloka
	LDR	R8, [R2], #4	; učitati adresu jednog odredišnog bloka
	LDR	R9, [R3], #4	; učitati veličinu jednog izvorišnog bloka
	STMFD	SP!, {R7, R8, R9}	; spremanje na stog
	BL	IZBACI	; poziv potprograma
	ADD	SP, SP, #0C	; pomicanje SP da se izbrišu parametri
(nastavak na sljedećoj stranici)	STR	R0, [R4], #4	; spremanje broja izbačenih nula
	SUBS	R5, R5, #1	; smanjivanje brojača blokova
	BNE	PETLJA	; skok na početak petlje
KRAJ	HALT		; ili SWI 123456

```

; potprogram
IZBACI  STMFD    SP!, {R1, R2, R3, R4, R5} ; spremanje konteksta
        ADD     R4, SP, #14             ; izračunati adresu parametara i staviti je u R4

        LDMFD   R4, {R1, R2, R3}; učitavanje parametara
        MOV     R0, #0                  ; brojač izbačenih ništica

LOOP    LDRH     R5, [R1], #2           ; učitavanje 16-bitnog broja iz izvorišnog bloka

        CMP     R5, #0                  ; usporedba s ništicom
        ADDEQ   R0, R0, #1              ; povećavanje brojača ako je ništica
        STRNEH  R5, [R2], #2           ; spremanje u odredišni blok ako nije ništica

        SUBS    R3, R3, #1              ; smanjenje brojača petlje
        BNE     LOOP

OUT     LDMFD   SP!, {R1, R2, R3, R4, R5} ; obnova konteksta
        MOV     PC, LR                  ; povratak iz potprograma

```

### Komentar rješenja:

U glavnom programu u registre **R1**, **R2**, **R3** i **R4** prvo učitavamo adrese pomoćnih blokova koji čuvaju informacije o blokovima podataka koje obrađujemo. Redom učitavamo: adresu bloka u kojem se nalaze početne adrese blokova, adresu bloka u kojem se nalaze početne adrese odredišnih blokova, adresu bloka u kojem se nalaze veličine izvorišnih blokova te adresu na koju ćemo spremati brojeve izbačenih ništica. Nakon toga slijedi petlja koja se ponavlja 5 puta jer imamo 5 blokova koje treba obraditi. U svakom prolazu petlje iz pomoćnih blokova učitati ćemo informacije (adrese i veličinu) o pojedinom bloku s podacima. Informacije se učitavaju nizom od tri naredbe **LDR**, koje odmah uvećavaju adrese pokazivača na pomoćne blokove za sljedeći korak petlje. Informacije se učitavaju u registre **R7**, **R8** i **R9**, zatim se spremaju na stog, poziva se potprogram **IZBACI**, nakon čega se uklanjaju parametri sa stoga. Na kraju se sprema broj izbačenih ništica u blok za rezultate te se ispita uvjet za kraj petlje.

U potprogramu **IZBACI** prvo se spremaju svi korišteni registri na stog, a nakon toga se u registar **R4** upisuje potrebna adresa (**ADD R4, SP, #14**) tako da možemo dohvatiti parametre sa stoga pomoću naredbe **LDMFD R4, {R1, R2, R3}**. Slijedi petlja u kojoj prvo dohvaćamo sljedeći podatak naredbom **LDRH** te ispitujemo je li jednak ništici. Ako je, povećava se brojač nula i preskače spremanje podatka u odredišni blok. Ako je podatak različit od nule, sprema se u odredišni blok uvjetnom naredbom **STRNEH R5, [R2], #2**. Na kraju petlje ispitujemo jesmo li obradili sve podatke u bloku, a nakon toga, ako ima još podataka, ponavlja se izvođenje petlje.

### 2.4.10. Prebrajanje elemenata liste (ZI07)

Riješen: DA    Težina: ★★★

U memoriji procesora ARM nalazi se popis od  $512_{10}$  32-bitnih podataka o studentima koji su dolazili na predavanja. Zapisani podatak predstavlja identifikacijski broj studenta ID (broj između 0 i  $100_{16}$ ) koji je došao na predavanje. Pretpostavka je da se evidentira samo ulaz na predavanje, a izlaz se ne bilježi.

Napisati potprogram **BROJI** koji će prebrojati na koliko je predavanja student prisustvovao i napraviti listu prisutnosti za sve studente. Potprogram preko stoga prima dva ulazna

parametra: adresu popisa i adresu ulazne liste prisutnosti. Potprogram zapisuje u listu 8-bitne podatke o prisutnosti (dakle na početnoj adresi liste je podatak o broju prisutnosti predavanjima za studenta s ID brojem 0, na sljedećoj adresi podatak za studenta s ID brojem 1, itd.). Dodatno, potprogram na kraju treba pronaći identifikacijski broj studenta koji je bio na najviše predavanja (pretpostaviti da postoji samo jedan takav) i njegov ID vratiti preko registra R0 iz potprograma.

### Prijedlog rješenja:

Ideja rješavanja zadatka je da se prvo u petlji prođe kroz sve podatke u popisu prisutnosti i uveća broj prisutnosti u listi za svakog studenta koji se pojavi u popisu (primijetite da je moguće da se popisu pojedini studenti pojave više puta, ne radi se o dolasku na samo jedno predavanje, nego na više njih). Budući da u listi prisutnosti mora postojati onoliko zapisa koliko ima studenata ( $101_{16}$ ), a da se u popisu prisutnosti pojavljuju upravo ID brojevi studenata (od 0 do  $100_{16}$ ), onda se uvećavanje prisutnosti u listi jednostavno ostvaruje adresiranjem zapisa u listi pomoću bazne adrese liste i adresnog odmakta koji je jednak ID-u studenta.

Nakon toga se u petlji prolazi kroz stvorenu listu i pronalazi se ID studenta koji je bio na najviše predavanja. Pronalaženje najveće vrijednosti u nizu podataka standardno se rješava tako da se kao trenutno najveći element zapamti prvi element niza. Zatim se prolazi preostalim elementima niza - od drugog elementa do zadnjeg elementa - te se uspoređuje svaki element s trenutno najvećim. Ako se pronađe veći element, onda njega pamtimo kao trenutno najvećeg i postupak se nastavlja usporedbom s preostalim elementima niza. Analogno se može tražiti i najmanji element u nizu.

### Rješenje:

BROJI	STMFD	R13!, {R1, R2, R3, R4, R5} ; spremanje konteksta
	ADD	R5, R13, #14 ; pomak da se učitaju podaci sa stoga
	LDMFD	R5, {R0, R1} ; učitavanje parametara: R0 popis, R1 lista
	MOV	R2, #2<8 ; brojač za 512 podataka u popisu prisutnosti
POD1	LDR	R3, [R0], #4 ; učitavanje prvog ID podatka
	LDRB	R4, [R1, R3] ; učitavanje dosadašnjeg broja prisustava za ID
	ADD	R4, R4, #1 ; povećavanje za 1
	STRB	R4, [R1, R3] ; spremanje uvećanog broja prisustava za ID
	SUBS	R2, R2, #1 ; petlja 512 puta
	BNE	POD1

(nastavak na sljedećoj stranici)

; pronalaženje tko je bio najviše puta na predavanjima

	MOV	R2, #1 ; brojač od 1 do 256 (100 heksadekadski)
	MOV	R5, #0 ; ID trenutno najvećeg je 0
	LDRB	R3, [R1] ; učitavanje prvog koji je trenutno najveći
P2	LDRB	R4, [R1, R2] ; učitavanje sljedećeg
	CMP	R4, R3 ; usporedba
	MOVHI	R3, R4 ; R3 = nova vrijednost najvećeg
	MOVHI	R5, R2 ; R5 = novi ID najvećeg
	ADD	R2, R2, #1 ; povećavanje brojača

CMP	R2, #1<8	; i to se ponavlja 256 puta
BLS	P2	; petlja
MOV	R0, R5	; povratna vrijednost
LDMFD	R13!, {R1, R2, R3, R4, R5}	; obnova konteksta
MOV	PC, R14	; povratak

### Komentar rješenja:

Povećavanje broja prisutnosti za studenta ostvareno je u petlji označenoj labelom **POD1**. Prva naredba učitava podatak iz liste prisutnosti (32-bitni zapis predstavlja ID studenta) u registar **R3**. Dobiveni ID ćemo iskoristiti za indeksiranje u tablici ID-ova čija početna (bazna) adresa se nalazi u registru **R1**. Tako će na primjer za studenta s ID-om 4, broj njegovih prisustvovanja predavanjima biti na adresi **R1+4**. Zbog toga se u nastavku petlje dohvaća broj pojavljivanja kao bajt s adrese **R1** uvećane za **R3** (naredba **LDRB R4, [R1, R3]**) u registar **R4**, koji se nakon toga uvećava za jedan te ponovno sprema na isto mjesto. Ova petlja ponavlja se sve dok se ne obrade svi podatci u popisu. Pronalaženje studenta s najvećim brojem prisustava na predavanju svodi se na pronalaženje najvećeg bajta u popisu studenata. Prilikom pretraživanja se u registru **R5** stalno pamti odmak trenutno najvećeg broja (tj. pamti se ID studenta s najvećim brojem prisustava predavanjima), a u **R3** se pamti trenutni najveći broj (tj. najveći broj prisustava). Na kraju pretraživanja, odmak najvećeg podatka od bazne adrese liste bit će traženi ID studenta, koji se naredbom **MOV R0, R5** sprema kao povratna vrijednost u registar **R0**.

### 2.4.11. Predznačno proširivanje 2'k brojeva (ZI08)

Riješen: DA    Težina: ★★

Napisati potprogram **PROSIRI** koji predznačno proširuje 16-bitni 2'k broj na 32 bita. 16-bitni broj zapisan je u memoriji, a njegova adresa se prenosi stogom kao parametar potprograma (parametar uklanja pozivatelj). Rezultat se vraća preko **R0**.

Dodatno, napisati potprogram **BLOK** koji proširuje deset 16-bitnih 2'k podataka iz bloka od adrese 1000<sub>16</sub>. Proširivanje se obavlja pozivom potprograma **PROSIRI**. Prošireni podaci se spremaju od adrese 2000<sub>16</sub>.

### Prijedlog rješenja:

Predznačno proširivanje 16-bitnog na 32-bitni 2'k broj jednostavno je ostvariti naredbom **LDRSH**. Ta naredba učitava 16-bitni podatak i odmah ga predznačno proširuje, pa nisu potrebne dodatne operacije. Kako se rezultat vraća preko registra **R0**, vrijednost toga registra nije potrebno čuvati, što znači da u potprogramu **PROSIRI** neće biti potrebno čuvati kontekst.

U potprogramu **BLOK** potrebno je, prilikom spremanja konteksta, spremiti i registar **R14 (LR)**, jer se u njemu nalazi povratna adresa za izlazak iz potprograma **BLOK**. Ovo radimo jer se koriste ugniježđeni potprogrami (potprogram **PROSIRI** se poziva iz potprograma **BLOK**). Za potprogram **PROSIRI** na stog se stavlja vrijednost **R1** (naredba **STMFD R13!, {R1}** na labeli **PETLJA**), što je adresa podatka. Tu je vrijednost potrebno ukloniti sa stoga nakon povratka iz potprograma **PROSIRI**.

U rješenju sa navedena samo dva potprograma, jer u zadatku nije zadano da treba napisati glavni program koji bi, na primjer mogao pozivati potprogram **BLOK** i koji bi trebao inicijalizirati pokazivač stoga.

PROSIRI	; potprogram PROSIRI		
	LDR	R0, [R13]	; čitanje adrese 16-bitnog podatka sa stoga
	LDRSH	R0, [R0]	; 2'k proširivanje 16->32
	MOV	PC, LR	; povratak
BLOK	; potprogram BLOK		
	STMFD	R13!, {R1, R2, R3, R14}	; spremanje konteksta
	MOV	R3, #0	; R5 - brojač 10
	MOV	R1, #1<12	; adresa izvorišnog bloka
	MOV	R2, #2<12	; adresa odredišnog bloka
PETLJA	STMFD	R13!, {R1}	; R1 na stog (adresa podatka)
	BL	PROSIRI	
	ADD	SP, SP, #4	; pozivatelj mora ukloniti parametar sa stoga
	STR	R0, [R2], #4	; spremanje podatka i povećanje odredišne adrese
	ADD	R1, R1, #2	; povećanje izvorišne adrese
	SUBS	R3, R3, #1	; smanjivanje brojača
	BNE	PETLJA	
	LDMFD	R13!, {R1, R2, R3, R14}	; obnova konteksta
	MOV	PC, LR	; povratak

#### Komentar rješenja:

U potprogramu **PROSIRI** je umjesto naredbe **LDRSH R0, [R0]** (kojom se 16-bitni 2'k broj predznačno proširivao na 32 bita) mogao stajati i sljedeći programski odsječak:

LDRH	R0, [R0]	
MOV	R0, R0, LSL #16	; logički pomak u lijevo za 16 <sub>10</sub> mjesta
MOV	R0, R0, ASR #16	; aritmetički pomak u desno za 16 <sub>10</sub> mjesta

Neposredne vrijednosti kojima se zadaje broj pomaka podrazumijevano su zadane u dekadskoj bazi (**LSL #16** i **ASR #16** u naredbama **MOV**).

#### 2.4.12. Provjera je li broj savršen (Z112)

Riješen: DA    Težina: ★★★

Napisati potprogram **SAVRSEN** koji provjerava je li broj savršen – jednak zbroju svojih pravih djelitelja (tj. zbroju svih svojih djelitelja, uključujući i 1, osim samoga sebe). Na primjer, 28 je savršen jer vrijedi  $28=1+2+4+7+14$ , a 8 nije savršen jer vrijedi  $8 \neq 1+2+4$ . Parametar se u potprogram šalje stogom (uklanja ga pozivatelj), a rezultat se vraća pomoću R0. Rezultat iznosi 1 ako je broj savršen, a 0 ako broj nije savršen.

U glavnom programu treba provjeriti 32-bitne NBC-brojeve iz bloka memorije na adresi 500<sub>16</sub> i zamijeniti nulom one koji nisu savršeni. U bloku je 10<sub>10</sub> podataka.

Potprogram **SAVRSEN** mora djeljivost brojeva provjeravati pomoću potprograma **DJELJIVO**. Napišite potprogram **DJELJIVO** koji provjerava je li prvi parametar (prenosi se registrom R1) djeljiv s drugim parametrom (prenosi se lokacijom iza naredbe poziva potprograma). Ako je djeljiv, preko R0 se vraća broj 1, a inače se vraća 0.

**Prijedlog rješenja:**

Potprogram **SAVRSEN** moguće je ostvariti na sljedeći način: u petlji treba provjeravati djeljivost broja N sa svim brojevima od 1 do broja N-1 (pri čemu želimo saznati je li N savršen broj). Broj kojeg provjeravamo u trenutnom koraku petlje spremamo u registar **R5**. Ako je broj N djeljiv sa **R5**, pribrajammo **R5** zbroju djelitelja (zbroj se čuva u registru **R4**). Nakon završetka petlje (kada vrijednost registra **R5** bude jednaka N), uspoređujemo konačni zbroj u **R4** s brojem N. Ako su brojevi jednaki, N je savršen pa se u **R0** upisuje povratna vrijednost 1. U suprotnom, N nije savršen pa se u **R0** upisuje povratna vrijednost 0. Napomenimo da bi za djeljivost zapravo bilo dovoljno provjeravati brojeve od 1 do N/2, jer broj N sigurno ne možemo podijeliti brojevima većima od N/2. Međutim, zbog nešto jednostavnije programske izvedbe, ovdje se provjeravaju brojevi od 1 do N-1.

Drugi parametar potprograma **DJELJIVO** nalazi se na prvoj memorijskoj lokaciji iza poziva potprograma. Za ostvarivanje takvog načina prijenosa potrebno je prije poziva potprograma spremati parametar na potrebnu memorijsku lokaciju, na primjer naredbom **STR R5,PARAMETAR**. Iza naredbe poziva potprograma, definira se jedna lokacija za parametar: **PARAMETAR DW 0**. Zbog lakšeg upisa vrijednosti parametra, definira se i labela (u ovom slučaju **PARAMETAR**). Sama provjera djeljivosti ostvaruje se dijeljenjem metodom uzastopnog oduzimanja. Prilikom dijeljenja zanemaruje se rezultat i provjerava se samo postoji li ostatak dijeljenja ili ne.

**Rješenje:**

	ORG	0	
GLAVNI	MOV	SP, #1<16	; inicijalizacija stoga
	MOV	R6, #5<8	; R6 - adresa početka bloka (500)
	MOV	R2, #0A	; R2 - broj brojeva u bloku (10)
POC	LDR	R0, [R6]	; učitavanje podatka
	STMFD	R13!, {R0}	; stavljanje parametra na stog
	BL	SAVRSEN	; poziv potprograma SAVRSEN
	ADD	R13, R13, #4	; brisanje parametra sa stoga
	CMP	R0, #0	; je li broj N savršen?
	STREQ	R0, [R6]	; upisivanje 0 umjesto nesavršenog broja
	ADD	R6, R6, #4	; sljedeći podatak
	SUBS	R2, R2, #1	; smanjivanje broja podataka
	BNE	POC	; natrag na početak petlje
KRAJ	HALT		
			; potprogram SAVRSEN
SAVRSEN	STMFD	R13!, {R1, R4, R5, R14}	; spremanje konteksta
	LDR	R1, [R13,#10]	; „preskakanje“ konteksta, parametar u R1
	MOV	R4, #0	; R4 - zbroj djelitelja
(nastavak na sljedećoj stranici)	MOV	R5, #0	; R5 - mogući djelitelj, od (1) do R1-1
TESTIRAJ	ADD	R5, R5, #1	; uvećavanje djelitelja
	CMP	R1, R5	; ako je djelitelj == broj ...
	BEQ	GOTOVO	; ... petlja je došla do kraja
	STR	R5, PARAMETAR	; parametar ide na adresu iza poziva potprograma
	BL	DJELJIVO	; poziv potprograma



```

PARAMETAR DW      0          ; parametar je odmah iza poziva potprograma

        CMP      R0, #1      ; R0 - rezultat. Djeljivost? 1 - da
        ADDEQ    R4, R4, R5  ; djeljivo - dodavanje djelitelja u zbroj
        B        TESTIRAJ    ; sljedeći djelitelj

GOTOVO  CMP      R1, R4      ; usporedba: početni broj == zbroj djelitelja?
        MOVNE    R0, #0      ; vraćanje 0 - NESavršen
        MOVEQ    R0, #1      ; vraćanje 1 - savršen
        LDMFD    R13!, {R1, R4, R5, R14} ; obnova konteksta
        MOV      PC, LR      ; povratak iz potprograma SAVRSEN

        ; potprogram DJELJIVO
DJELJIVO STMF    R13!, {R1, R2} ; spremanje konteksta
        ; R1 - prvi parametar
        LDR      R2, [LR], #4 ; R2 - učitavanje drugog parametra i istovremeno
        ; povećavanje LR
PONOVO  SUBS     R1, R1, R2    ; dijeljenje metodom uzastopnog oduzimanja
        BHI      PONOVO

        MOVEQ    R0, #1      ; vraćanje 1 ako je djeljiv
        MOVNE    R0, #0      ; vraćanje 0 ako nije djeljiv
        LDMFD    R13!, {R1, R2} ; obnova konteksta
        MOV      PC, LR      ; povratak iz potprograma

        ORG 500
        DW      1, 2, 8, %D28, 5, 6, 7, 8, 9, 0A

```

**Komentar rješenja:**

U rješenju treba obratiti pažnju na prijenos parametara u potprogram **DJELJIVO**, odnosno na smještanje parametra na memorijsku lokaciju iza naredbe poziva potprograma te na učitavanje parametra u potprogramu. Kada se poziva potprogram **DJELJIVO**, vrijednost u registru **LR** – adresa povratka iz potprograma – zapravo je adresa memorijske lokacije na kojoj se nalazi vrijednost drugog parametra. Zbog toga taj parametar učitavamo naredbom **LDR R2, [LR], #4**. Istodobno se registar **LR** povećava za 4 tako da pokazuje na prvu naredbu iza lokacije u kojoj je parametar, a upravo je to naredba na koju se treba vratiti prilikom izlaska iz potprograma.

Unutar petlje u potprogramu **SAVRSEN**, registar **R5** (mogući djelitelji) poprima vrijednost od 0 do N. No, odmah na početku prvog prolaska, povećavamo vrijednost na 1. Jednako tako, vrijednost N prekida petlju. Tako se mogući djelitelji kreću od 1 do N – 1.

U potprogramu **SAVRSEN** se parametar sa stoga dohvaća naredbom **LDR R1, [R13, #10]**. Ovakvo pojednostavljenje pogodno je kad se stogom prenosi jedan ili dva parametra. U slučaju dva parametra, drugi parametar mogao bi se dohvatiti dodatnom naredbom **LDR R1, [R13, #14]**. Već za tri parametra je kraće rješenje s izračunom adrese, jer se tada nakon izračunavanja adrese sva tri parametra mogu učitati samo jednom naredbom **LDMFD**.

**2.4.13. Težinska suma niza (ZI09)****Riješen: DA    Težina: ★**

Napisati potprogram **TEZ\_SUM** koji računa težinsku sumu niza. Potprogram prima tri parametra: adresu niza podataka (**R0**), adresu niza težina (**R1**), te duljinu niza koja je

zajednička za oba niza (**R2**). Potprogram treba pomnožiti vrijednosti na istim pozicijama u oba niza, te zbrojiti umnoške u jedan rezultat. Rezultat potprograma vraća se u registru **R0**. Svi podaci su u 8-bitnom 2'sk formatu, dok rezultat treba biti u 32-bitnom 2'sk formatu. Pretpostavka je da kod množenja i zbrajanja neće doći do prekoračenja opsega od 32 bita.

Primjer: niz podataka je 1, 2, 3, -5, 8

niz težina je 6, 3, -1, 4, 7

duljina nizova je 5

Rezultat potprograma je:  $1*6 + 2*3 + 3*(-1) + (-5)*4 + 8*7 = 45_{10}$ .

### Prijedlog rješenja:

Rješenje ovog zadatka za procesor ARM 7, u odnosu na procesor FRISC, pojednostavljeno je uporabom naredbe za istovremeno učitavanje i predznačno proširivanje (**LDRSB**), te naredbe za množenje s pribrajanjem (**MLA**). Težinska suma se računa u petlji u čijem svakom koraku se obrađuje po jedan podatak i jedna težina. Petlja za obradu podataka izvršava se onoliko puta kolika je duljina niza (**R2**). U petlji se učitavaju 8-bitni podatci koji se odmah predznačno proširuju, te se adresa lokacije za sljedeće učitavanje povećava za 1. Naredba **MLA R5, R3, R4, R5** množi podatak (**R3**) s težinom (**R4**) te umnožak odmah pribraja konačnoj sumi (**R5**).

### Rješenje:

TEZ_SUM	STMFD	SP!, {R1, R2, R3, R4, R5}	; spremanje konteksta ; R1 i R2 su parametri - ni njih se ne smije mijenjati
	MOV	R5, #0	; inicijalizacija težinske sume
PET	LDRSB	R3, [R0], #1	; učitavanje podatka i predznačno proširivanje
	LDRSB	R4, [R1], #1	; učitavanje težine i predznačno proširivanje
	MLA	R5, R3, R4, R5	; množenje i zbrajanje
	SUBS	R2, R2, #1	; provjera kraja petlje
	BNE	PET	
KRAJ	MOV	R0, R5	; upis rezultata u R0
	LDMFD	SP!, {R1, R2, R3, R4, R5}	; obnova konteksta
	MOV	PC, LR	; povratak iz potprograma

### Komentar rješenja:

Najjednostavnije rješenje za množenje dva člana i zbrajanje s prethodnom sumom pomoću samo jedne naredbe je **MLA R5, R3, R4, R5**. Problem se može riješiti na još nekoliko načina. Najslićnije prethodnom rješenju je uporaba dvije naredbe čiji je učinak jednak naredbi **MLA**, ali smo zauzeli dodatna 4 bajta za zapis jedne dodatne naredbe i usporili izvođenje programa (i upotrijebili dodatni registar **R6**):

MUL	R6, R3, R4
ADD	R5, R6, R5

Kada bismo morali paziti na prekoračenje opsega (realan slučaj kod množenja) onda bi ispravno rješenje bilo **SMLAL R5, R6, R3, R4**. Međutim, tada rezultat potprograma ne bismo vraćali pomoću jednog, već pomoću dva registra. Pogledajmo kako bi izgledao primjer kada bismo rezultat vraćali preko R0 (niža 32 bita) i R1 (viša 32 bita):

TEZ_SUM	STMFD	SP!, {R2, R3, R4, R5, R6}	; spremanje konteksta
	MOV	R5, #0	; inicijalizacija 64-bitnog...
	MOV	R6, #0	; ...rezultata u registrima R5 i R6
PET	LDRSB	R3, [R0], #1	; učitavanje i predznačno proširivanje
	LDRSB	R4, [R1], #1	
	SMLAL	R5, R6, R3, R4	; množenje i zbrajanje
	SUBS	R2, R2, #1	
	BNE	PET	
KRAJ	MOV	R0, R5	; R0 niža 32 bita rezultata
	MOV	R1, R6	; R1 viša 32 bita rezultata
	LDMFD	SP!, {R2, R3, R4, R5, R6}	; obnova konteksta
	MOV	PC, LR	; povratak iz potprograma

Naravno, množenje se može ostvariti i metodom uzastopnog zbrajanja, koje dodatno usporava izvođenje programa, mnogostruko je složenije i koristilo bi se samo ako bi tekstem zadatka bilo zadano da ne smijemo koristiti naredbe za množenje.

#### 2.4.14. Izračun funkcije $16 \cdot A + B/4$

Riješen: DA    Težina: ★★

Napisati potprogram koji prima dva ulazna parametra preko stoga (nazovimo ih A i B). Pozivatelj prvo na stog stavlja parametar B, a zatim parametar A. Parametre sa stoga uklanja pozivatelj. Parametri su 32-bitni brojevi u formatu 2'k. Povratne vrijednosti vraćaju se preko registara **R0** i **R1**. Potprogram treba izračunati  $16_{10} \cdot A + B/4_{10}$  i vratiti ga kao rezultat preko **R0**. Ako je prilikom računanja rezultata (ili međurezultata) došlo do prekoračenja 32 bitnog 2'k opsega, onda u **R1** treba vratiti broj 1 (sadržaj **R0** je proizvoljan), a ako je rezultat ostao u opsegu, onda u **R1** treba vratiti broj 0.

Napisati i glavni program koji treba pozvati potprogram za  $A = -100_{10}$  i  $B = +30_{10}$ , te zatim spremi rezultat na lokaciju s adresom  $1000_{16}$  i zaustaviti procesor. Ako je došlo do prekoračenja 32 bitnog 2'k opsega, glavni program na lokaciju  $1000_{16}$  treba spremi broj -1.

##### Prijedlog rješenja:

U potprogramu, koji smo nazvali **FUNC**, nakon spremanja konteksta, treba učitati parametre (i pritom „preskočiti“ kontekst spremljen na stog). Dijeljenje sa 4 obavlja se logičkim pomakom udesno za 2 bita. Naredba **MLAS R0, R4, R6, R5** obavlja množenje parametra **R4** sa **R6** ( $16_{10}$ ) te pribrajanje već podijeljenom iznosu  $B/4$  koji se nalazi u registru **R5**. Rezultat se sprema u **R0**, a nastavak **S** postavlja zastavice nakon množenja i zbrajanja, radi provjere prekoračenja opsega. Ako je postavljena zastavica **V** (*overflow*), došlo je do prekoračenja opsega, što se može ispitati ako u polju uvjeta zadamo nastavak **VS**. Ako ispituje uvjet da preljeva nije bilo, koristimo nastavak **VC** (*no overflow*).

U glavnom programu, parametri A i B se nakon učitavanja šalju u potprogram preko stoga (parametar **R0**=A bit će na nižoj adresi stoga). Po povratku iz potprograma, sa stoga se uklanjaju parametri uvećavanjem pokazivača stoga **R13** za 8. Vraćena vrijednost u **R1** označava je li došlo do prekoračenja opsega, što provjeravamo naredbom **TEQ** (koja ispituje jednakost dva podatka izvođenjem logičke operacije ekskluzivno-ili na temelju čega se postavljaju zastavice). Kako je broj naredaba koje treba izvesti u oba slučaja mali, koristimo uvjetno izvođenje naredaba (nastavci **EQ** i **NE**).

**Rješenje:**

```

ORG      0
GLAVNI   MOV    R13, #1<16      ; inicijalizacija stoga
        LDR     R0, PARAM      ; parametar A=-100
        MOV     R1, %%D30      ; parametar B=30
        STMFD   R13!, {R0, R1} ; parametri se spremaju na stog
        BL      FUNC          ; poziv potprograma, rezultat je u R0 i R1
        ADD     R13, R13, #8    ; uklanjanje parametara sa stoga

        MOV     R2, #1<12      ; adresa 1000 za rezultat
        TEQ     R1, #1         ; je li došlo do prekoračenja opsega
        MVNEQ   R1, #0         ; ako je došlo do prekoračenja, spremi -1
        STREQ   R1, [R2]

        STRNE   R0, [R2]       ; ako nema prekoračenja spremi rezultat
        HALT

PARAM    DW      %D-100

        ; potprogram FUNC
FUNC     STMFD   R13!, {R4, R5, R6} ; spremanje konteksta
        ADD     R0, R13, #0C    ; sa R0 preskoči spremljene registre (0C16=1210)
        LDMFD   R0, {R4, R5}   ; učitavanje ulaznih parametara A, B

        MOV     R5, R5, LSR #2 ; dijeljenje R5 sa 4
        MOV     R6, #10        ; R6=16
        MLAS    R0, R4, R6, R5 ; R0= R4*R6+R5

        MOVVS   R1, #1         ; ako ima preljeva R1=1
        MOVVC   R1, #0         ; ako nema preljeva R1=0
        LDMFD   R13!, {R4, R5, R6} ; obnavljanje konteksta
        MOV     PC, LR

```

**Komentar rješenja:**

Prokomentirajmo učitavanje parametara A i B u registre. Budući da pišemo programe za simuliranje u ATLAS-u, vrijednost koju možemo izravno upotrijebiti u naredbi **MOV** je veličine 8 bita s rotacijom za parni broj bitova u lijevo. Zato se parametar A = -100<sub>10</sub> mora ili učitati s neke labele što je ovdje i napravljeno, ili upisati kao rotirana 8-bitna vrijednost pomoću naredbe **MVN**, što ne bi bilo tako razumljivo (izračunajte sami koju vrijednost bi trebalo navesti u naredbi **MVN** da bi se dobio broj -100). Naredbom **TEQ R1, #1** ispituje se da li je došlo do prekoračenja opsega, odnosno, da li je u registru **R1** jedinica. Naredba **MVNEQ R1, #0** upisuje u **R1** negiranu vrijednost od ničice (broj -1) ukoliko je uvjet **EQ** ispunjen, odnosno ukoliko je u prethodnoj naredbi utvrđeno prekoračenje opsega.

**2.4.15. Usporedba brojeva i računanje apsolutne vrijednosti (ZI11)****Riješen: DA    Težina: ★★**

U memoriji procesora ARM na lokaciji 1000<sub>16</sub> nalazi se niz od 20<sub>16</sub> parova 8-bitnih brojeva u formatu 2'k, gdje su brojevi u pojedinom paru zapisani neposredno jedan iza drugog. Napisati glavni program, koji pomoću potprograma **PAROVI** treba obraditi ovaj blok podataka.

Napisati potprogram **PAROVI** koji preko fiksne lokacije **POCETAK** prima adresu bloka memorije, a preko fiksne lokacije **BROJ** prima broj parova 8-bitnih brojeva u bloku, zapisanih u formatu 2'k. Potprogram treba za svaki par u bloku usporediti brojeve i veći od njih poslati potprogramu **ABS** kao 32-bitni broj. Rezultat potprograma **ABS** treba kao jedan 16-bitni broj pohraniti na ista dva bajta koja je zauzimao odgovarajući par brojeva.

Napisati potprogram **ABS** koji treba izračunati apsolutnu vrijednost 32-bitnog broja zapisanog u formatu 2'k. Potprogram prima parametar preko stoga, a rezultat vraća preko registra. Parametre sa stoga treba brisati pozivatelj.

### Prijedlog rješenja:

U ovom zadatku potrebno je obratiti pažnju na pozive potprograma iz potprograma. Zbog toga je prilikom spremanja konteksta u potprogramu **PAROVI** nužno spremi i registar **R14** na stog jer se u njemu nalazi povratna adresa za povratak u glavni program (adresa naredbe **HALT**), a prilikom poziva potprograma **ABS** u registar **R14** se zapisuje povratna adresa poziva **BL ABS** u potprogramu **PAROVI** (adresa naredbe **ADD SP, SP, #4**).

### Rješenje:

```

ORG      0
MOV      SP, #1<16      ; inicijalizacija stoga

MOV      R0, #1<12      ; upis vrijednost prvog parametra ...
STR      R0, POCETAK    ; ... u fiksnu lokaciju POCETAK

MOV      R0, #20         ; upis vrijednosti drugog parametra
STR      R0, BROJ        ; ... u fiksnu lokaciju BROJ
BL       PAROVI          ; poziv 1. potprograma
HALT

; potprogram PAROVI
PAROVI   STMFD    SP!, {R0, R1, R2, R3, R4, R14} ; spremanje konteksta
LDR      R1, POCETAK    ; R1 - adresa početka niza
LDR      R2, BROJ        ; R2 - broj parova - brojač za petlju

PETLJA   LDRSB    R3, [R1] ; učitavanje prvog broja i proširivanje
LDRSB    R4, [R1, #1]    ; učitavanje drugog broja i proširivanje

CMP      R3, R4          ; koji broj u paru je veći?
STMGEFD  SP!, {R3}       ; R3 je veći
STMLTFD  SP!, {R4}       ; R4 je veći

BL       ABS             ; rezultat je u R0
ADD      SP, SP, #4      ; brisanje parametra sa stoga
STRH     R0, [R1], #2    ; spremanje na isto mjesto i povećavanje adrese
SUBS     R2, R2, #1      ; smanjivanje brojača parova
BNE      PETLJA

(nastavak na sljedećoj stranici)
LDMFD    SP!, {R0, R1, R2, R3, R4, R14} ; obnova konteksta
MOV      PC, LR

POCETAK  DW       0      ; lokacije za parametre ...
BROJ     DW       0      ; ... potprograma PAROVI

; potprogram ABS
ABS      LDR      R0, [SP] ; čitanje parametra sa stoga

```

```

CMP      R0, #0          ; test za postavljanje zastavica
                        ; pozitivni brojevi se ne mijenjaju
MVNMI    R0, R0          ; negativne brojeve treba dvojno komplementirati
ADDMI    R0, R0, #1
MOV      PC, LR

```

**Komentar rješenja:**

U potprogramu **PAROVI** je potrebno obratiti pažnju na sljedeće četiri naredbe:

```

...
LDRSB    R4, [R1, #1]    ; učitavanje drugog broja i proširivanje
CMP      R3, R4          ; koji je veći?
STMGEFD  SP!, {R3}       ; R3 je veći
STMLTFD  SP!, {R4}       ; R4 je veći
...

```

Predznačno proširivanje sa 8 na 32 bita (odnosno sa 16 na 32 bita), za razliku od procesora FRISC, moguće je ostvariti samo jednom naredbom **LDRSB** (odnosno **LDRSH**). Naredba **STM** služi za istovremeno spremanje bloka podataka, međutim u bloku može biti i samo jedan podatak, kao u ovom slučaju. Sufiks **FD** označava način rada stoga (*full descending*). Sufiksi **GE** i **LT** označavaju uvjete koji moraju biti ispunjeni da bi se pojedine naredbe izvele. Ovaj programski odsječak mogao se zapisati i na sljedeći način:

```

...
LDRSB    R4, [R1, #1]
CMP      R3, R4
STRGE    R3, [SP, #-4]!
STRLT    R4, [SP, #-4]!
...

```

Kontekst potprograma **PAROVI** uključuje sve registre koji se mijenjaju tijekom izvođenja tog potprograma. To nisu samo registri **R1**, **R2**, **R3** i **R4** koje potprogram izravno mijenja. Osim toga treba spremati i registar **R14** u kojemu se čuva povratna adresa iz potprograma **PAROVI**, jer će **LR** biti promijenjen naredbom **BL ABS** koja poziva potprogram **ABS**. Također će potprogram **ABS** vratiti rezultat registrom **R0** pa se i taj registar mijenja i time spada u kontekst potprograma **PAROVI**.

U potprogramu **ABS**, pretvaranje u dvojni komplement (naredbe **MVNMI** i **ADDMI**) se moglo kraće napisati samo jednom naredbom **RSBMI R0, R0, #0**.

**2.4.16. Kvadrat 16-bitnog broja u formatu 2'k**

Riješen: NE    Težina: ★★

Napisati potprogram **SQR** koji računa kvadrat 16-bitnog broja u formatu 2'k. Ulazni broj prenosi se preko stoga u nižih 16 bitova 32-bitnog podatka na stogu, a rezultat (32-bitni broj u formatu 2'k) se zapisuje na memorijsku lokaciju (32-bitnu) čija adresa se nalazi također zapisana na stogu, odmah nakon ulaznog broja.

Pretpostavite da rezultat nikad nije veći od 32 bita. Parametre sa stoga treba ukloniti u potprogramu.

### 2.4.17. Dijeljenje uzastopnim oduzimanjem i rastavljanje na proste faktore

Riješen: NE    Težina: ★★

Napisati potprogram **DIV** za dijeljenje 32-bitnih brojeva u zapisu NBC. Dijeljenje ostvariti metodom uzastopnog oduzimanja. Djeljenik se prenosi preko registra **R0**, a djelitelj preko registra **R1**. Rezultat se vraća preko registra **R0**, a ostatak preko registra **R1**. Sadržaj svih preostalih registara treba ostati sačuvan.

Napisati potprogram **FACT** koji zadani broj u NBC-u, prenesen preko registra **R0**, rastavlja na proste faktore. Faktore je potrebno rastućim redoslijedom pohraniti u memoriju počevši od adrese koja je zadana registrom **R1**. Svaki faktor zauzima po jednu 32-bitnu lokaciju, a listu faktora treba zaključiti ničicom.

Primjer: Prosti faktori broja 52 su: 2, 2 i 13, jer je  $2 \cdot 2 \cdot 13 = 52$ .

### 2.4.18. Oduzimanje i dijeljenje brojeva u dvostrukoj preciznosti

Riješen: NE    Težina: ★★

Napisati potprogram **SUB\_DBL** za oduzimanje brojeva u dvostrukoj preciznosti (64 bita). Brojevi dvostruke preciznosti pohranjuju se u memoriju tako da je prvo zapisan niži pa zatim viši dio. U registru **R0** nalazi se adresa prvog operanda (niži dio), a u registru **R1** nalazi se adresa drugog operanda (nižeg dijela). Od prvog operanda treba oduzeti drugi operand. Registar **R2** sadrži adresu na koju treba spremiti rezultat.

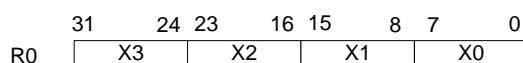
Napisati potprogram **DIV\_DBL** koji (koristeći potprogram **SUB\_DBL**) dijeli dva broja u dvostrukoj preciznosti metodom uzastopnog oduzimanja. U registru **R0** nalazi se adresa djeljenika (niži dio), u registru **R1** nalazi se adresa djelitelja (niži dio), a u registru **R2** nalazi se adresa na koju treba spremiti rezultat. Ostatak pri dijeljenju zanemariti.

Napisati glavni program koji poziva potprogram **DIV\_DBL** i dijeli neka dva broja.

### 2.4.19. Potprogram za izračun funkcije sa četiri parametra

Riješen: NE    Težina: ★★

Potrebno je napisati potprogram **FUNC** koji u registru **R0** prima 32-bitni podatak koji je sačinjen od četiri bajta u zapisu  $2^k$ , nazovimo ih **X0**, **X1**, **X2** i **X3**, raspoređenih na sljedeći način:



Potprogram **FUNC** treba izračunati sljedeću funkciju:  $y = X3 \cdot 8 + X2 \cdot 4 + X1 \cdot 2 + X0$ , te 32-bitni rezultat u obliku  $2^k$  vratiti u registru **R0**. Sadržaj ostalih registara treba ostati sačuvan. Množenje ostvariti uvažavanjem činjenice da se množi brojevima koji su potencija broja 2.

Napisati glavni program koji će, koristeći potprogram **FUNC**, sve brojeve od adrese  $1000_{16}$  do  $2000_{16}$  zamijeniti dobivenim rezultatima iz potprograma.

### 2.4.20. Zrcalni bitovi i provjera palindroma

Riješen: NE    Težina: ★★

Napisati potprogram **INVERZ** koji računa bitovni inverz (ili zrcalni podatak) od 32-bitnog podatka. Podatak se prenosi u potprogram preko **R0**, a rezultat se vraća također preko **R0**.

Primjer: bitovni inverz 16-bitnog broja  $1110\ 0010\ 1010\ 1111_2$  je broj  $1111\ 0101\ 0100\ 0111_2$

Napisati potprogram **PALIN** koji za svaki podatak zadanog bloka, korištenjem potprograma **INVERZ**, ispituje je li podatak bitovni palindrom, odnosno ispituje je li njegova zrcalna vrijednost jednaka originalnoj vrijednosti. Potprogram **PALIN** prima adresu i duljinu zadanog bloka preko stoga, a u **R0** vraća ukupni broj pronađenih palindroma.

Napisati glavni program koji poziva potprogram **PALIN** za zadani blok podataka koji počinje na adresi  $500_{16}$  i ima  $512_{10}$  podataka, te vraćeni ukupni broj palindroma pohranjuje na adresu  $450_{16}$ .

### 2.4.21. Potprogram za pretvorbu iz zapisa BCD u NBC

Riješen: NE    Težina: ★★★

Napisati potprogram **BCD2NBC** koji 32-bitni podatak zapisan nepakiranim BCD-om pretvara u 32-bitni broj u zapisu NBC. Broj u zapisu BCD se prima kao parametar u registru **R0** u kojemu se vraća i rezultat pretvorbe. Broj je zapisan tako da svaki bajt u 32-bitnoj riječi predstavlja jednu dekadsku znamenku.

Primjer: broj  $08090504_{16}$  odnosno  $00001000\ 00001001\ 00000101\ 00000100_2$  je prikaz broja  $8954_{10}$  u nepakiranom BCD-u.

Napisati glavni program koji svaki originalni podatak zapisan od adrese  $2000_{16}$  do  $2500_{16}$  zamjenjuje podatkom koji mu vrati potprogram **BCD2NBC**.

### 2.4.22. Brojenje pojavljivanja 2-bitnog uzorka u broju

Riješen: NE    Težina: ★★★

Napisati potprogram **CNT2BIT** koji broji pojavljivanje pojedinog 2-bitnog uzorka ( $00_2$ ,  $01_2$ ,  $10_2$  ili  $11_2$ ) u 32-bitnoj riječi počevši od najmanje značajnog bita prema najviše značajnom bitu. Ulazni 2-bitni uzorak prenosi se preko registra **R1** (u najniža 2 bita), a ulazna 32-bitna riječ preko registra **R0**. Broj pojavljivanja uzorka vraća se preko registra **R2**. Pronađeni uzorci se mogu preklapati.

Napisati glavni program koji za svaku 32-bitnu riječ od adrese  $1000_{16}$  do adrese  $1200_{16}$  broji pojavljivanje svih mogućih 2-bitnih uzoraka ( $00_2$ ,  $01_2$ ,  $10_2$  ili  $11_2$ ) korištenjem potprograma **CNT2BIT**. Rezultat brojenja se sprema od lokacije  $1500_{16}$  kao 32-bitni rezultat i to tako da se broj pojavljivanja uzorka  $00_2$  nalazi u prvom (najnižem) bajtu; broj pojavljivanja uzorka  $01_2$  nalazi se u drugom bajtu; broj pojavljivanja uzorka  $10_2$  nalazi se u trećem bajtu; te broj pojavljivanja uzorka  $11_2$  nalazi se u četvrtom (najvišem) bajtu rezultata.

Primjer:      ulazni podatak:      **00101001 01110110 10100101 01111101<sub>2</sub>**

                 u sebi ima:    7 uzoraka **11<sub>2</sub>**, 10 uzoraka **10<sub>2</sub>**, 11 uzoraka **01<sub>2</sub>** i 3 uzorka **00<sub>2</sub>**

                 izlazni podatak:    **00000111 00001010 00001011 00000011<sub>2</sub> = 070A0B03<sub>16</sub>**



### 2.4.23. Pronalazak podniza u nizu bitova

Riješen: NE    Težina: ★★★

Napisati potprogram **SUBS** koji pronalazi 8-bitni uzorak u 32-bitnom podatku i vraća u glavni program sva mjesta (pozicije) pojavljivanja uzorka, kao i ukupni broj pojavljivanja. Ulazni 32-bitni podatak se prenosi preko **R0**, a 8-bitni uzorak se prenosi preko donjih 8 bita u **R1**. Rezultat je 32-bitni podatak koji se vraća u **R0** i u sebi sadrži sve pozicije pojavljivanja uzorka (u donjih 26 bita) i ukupan broj pojavljivanja uzorka (u najviših 6 bita). Pozicije se vraćaju tako da se na mjestima pronalaska uzorka na mjestu najnižeg bita pronađene pozicije uzorka postavi jedinica.

Primjer: Za broj  $00010110\ 01011001\ 10010110\ 01011000_2$  i uzorak  $00101100_2$  pronaći će se četiri pojavljivanja uzorka pa će u gornjih 6 bitova rezultat biti upisan broj 4, a u nižih 26 bitova će na 4 mjesta biti postavljene jedinice. Uzorci će biti pronađeni na sljedeća 4 mjesta:

```
00010110 01011001 10010110 01011000
 0010110 0
    0 0101100
          0010110 0
              0 0101100
```

Podcrtani su najniži bitovi pronađenih pojavljivanja uzorka, a to su pozicije na kojima u 26 nižih bitova rezultata treba postaviti jedinice. Dakle, rezultat izgleda ovako (najviših 6 bitova, označenih sivom bojom, sadrže broj 4):

```
00010000 10000010 00000000 10000010
```

Glavni program treba u bloku 32-bitnih podataka obrisati sve podatke u kojima nema zadanog 8-bitnog uzorka. Blok se nalazi od adrese  $500_{16}$  pa do  $1000_{16}$ , a zadani uzorak je  $A1_{16}$ . Uočite da se pozicije pronađenog uzorka mogu preklapati.

### 2.4.24. Zbrajanje brojeva u zapisu nepomičnog zareza

Riješen: NE    Težina: ★★

U najnižem bajtu 32-bitnog podatka zapisan je pozitivni realni broj u sljedećem obliku **LLLLDDDD**, dok su u tri najviša bajta upisane ništice. **LLLL** predstavlja 4 bita lijevo od decimalnog zareza, a **DDDD** 4 bita desno od decimalnog zareza. Na primjer, broj  $01111100_2$  je zapravo zapis broja  $0111,1100_2$  što u dekadskoj bazi iznosi  $7,75_{10}$ . Ovakav zapis naziva se nepomičnim zarezom (*fixed point*).

Napisati potprogram **ZBR\_PLD** koji zbraja dva broja u gore opisanom obliku. Podatci se prenose preko stoga, a rezultat se vraća u registru **R0**. Pretpostaviti da neće doći do prekoračenja opsega pri zbrajanju.

Napisati glavni program koji pozivima potprograma **ZBR\_PLD** računa izraz **A+B+C**, gdje su **A**, **B** i **C** brojevi u gore opisanom obliku pohranjeni na lokacijama  $600_{16}$ ,  $601_{16}$  i  $602_{16}$ , te rezultat sprema na lokaciju  $500_{16}$ .

Dodatne inačice zadatka:

- Zamjena navedenog oblika **LLLLDDDD** sa **PLLLLLDD**, gdje **P** predstavlja predznak (1 za negativan, 0 za pozitivan), a **LLLLL** i **DD** imaju značenja analogna izvorom zadatku. Na primjer: 0111 1101 =  $31,25_{10}$  i 1001 1110 =  $-7,5_{10}$ .
- Zamjena izraza  $A+B+C$  sa  $A+B-C$

#### 2.4.25. Izračun cjelobrojnog dijela broja u zapisu IEEE

Riješen: NE    Težina: ★★★

Napisati potprogram **IEEE\_INT** koji za zadani 32-bitni broj u zapisu po normi IEEE 754 računa njegov cjelobrojni dio. Broj se kao parametar prima registrom **R0**, a rezultat se vraća registrom **R1**. Pretpostavka je da cjelobrojni dio broja stane u 32 bita, odnosno da eksponent nije veći od 32.

Primjeri: za  $36,47_{10}$  rezultat je  $36_{10}$ , za  $-3,89_{10}$  rezultat je  $-3_{10}$ , za  $0,345_{10}$  rezultat je 0.

## 3. Programiranje vanjskih jedinica

Na predavanjima se za procesor ARM 7 obrađuju samo dvije vanjske jedinice. To su ulazno-izlazni sklop (GPIO) i vremenski sklop (RTC). Obje su vanjske jedinice spojene na sabirnicu AMBA APB, a preko AHB-APB mosta spojene su sa sabirnicom AMBA-AHB te samim procesorom i memorijom. Same vanjske jedinice nisu objašnjene u ovoj zbirci jer je to dio gradiva koji se objašnjava na predavanjima.

Na sklop GPIO mogu biti spojeni jednostavni uređaji poput temperaturnog sklopa i LCD-prikaznika. Temperaturni sklop omogućuje mjerenje i očitavanje vanjske temperature. LCD-prikaznik je izlazni uređaj koji omogućuje ispisivanje do osam ASCII-znakova na zaslonu LCD-a. Programiranje sklopa GPIO kod procesora ARM 7 znatno je jednostavnije od programiranja usporedivog vanjskog sklopa PIO na procesoru FRISC. GPIO ne može generirati prekid, a nema ni sinkronizacijskih priključaka pa eventualno potrebnu sinkronizaciju s vanjskim uređajem treba ostvariti programski koristeći slobodne priključke sklopa GPIO.

Sklop RTC usporediv je sa sklopom CT kod procesora FRISC. Primarna zadaća sklopa RTC je mjerenje vremena (iako se njime mogu prebrajati i obični impulsi). Upravljanje sklopom RTC obavlja se pomoću pet registara od kojih su u zadacima najčešće korišteni **RTCMR**, **RTCCR**, **RTCLR** i **RTCEOI**. Registar **RTCMR** služi za upis konstante s kojom se uspoređuje brojilo, **RTCCR** za omogućavanje/onemogućavanje generiranja prekida i **RTCLR** za upisivanje broja od kojeg se broji (inicijalno „0“). Upisom bilo kojeg podatka u **RTCEOI** zapravo se briše prekidni signal **RTCINTR** i pripadni bistabil stanja (slično kao kod sklopa CT).

### 3.1. Vanjske jedinice GPIO i RTC

U ovom poglavlju se osim programiranja sklopova GPIO i RTC obrađuju i iznimke (prekidi) IRQ i FIQ. Potpoglavlje sadrži sljedeće gradivo:

- prekidi IRQ
- prekidi FIQ
- programiranje sklopa GPIO
- programiranje sklopa RTC
- uporaba temperaturnog sklopa
- uporaba LCD-prikaznika

### 3.1.1. Brzi prekid FIQ

Riješen: DA    Težina: ★

Napisati program koji inicijalizira brze prekide (FIQ) i nakon toga izvodi praznu petlju. Napisati prekidni potprogram za FIQ. Prekidni potprogram treba uvećavati za jedan sadržaj 8-bitne lokacije na adresi 1000<sub>16</sub>.

#### Prijedlog rješenja:

U glavnom programu potrebno je jedino dozvoliti brze prekide. Za razliku od procesora FRISC, procesor ARM podržava normalne (IRQ) i brze (FIQ) prekide. U ovom slučaju potrebno je u registru **CPSR** obrisati bit **F** (bit na položaju 6). Prekidni potprogram se nalazi od adrese 1C (kod IRQ-a se nalazi od adrese 18<sub>16</sub>). U načinu rada **FIQ** prilikom adresiranja registara **R8–R14** zapravo se pristupa posebnim registrima **R8\_fiq – R14\_fiq**, (kod IRQ-a ne postoje posebni registri). Registar stanja **CPSR** također nije potrebno spremati, jer procesor ARM to radi automatski tako da prilikom prihvaćanja FIQ prekida sadržaj registra **CPSR** spremi u registar **SPSR\_fiq**. Za povratak iz prekidnog potprograma (neovisno je li se obrađivao FIQ ili IRQ) koristimo naredbu **SUBS PC,LR,#4**, za razliku od povratka iz običnog potprograma gdje koristimo naredbu **MOV PC,LR**. Ova naredba je posebna po tome što će kopirati sadržaj registra **R14\_fiq** (povratna adresa iz FIQ prekida spremljena prilikom prihvata prekida) i staviti ga u registar **PC** te obnoviti sadržaj registra **CPSR** iz **SPSR\_fiq** gdje se prilikom prihvaćanja prekida sprema originalna vrijednost registra **CPSR**.

#### Rješenje:

	ORG	0	
	B	GLAVNI	; skok na glavni
PREKIDNI	ORG	1C	; adresa potprograma za FIQ, slijedi potprogram
	MOV	R9, #1<12	; adresa 1000 u registar R9
	LDRB	R8, [R9]	; učitati bajt s adrese 1000
	ADD	R8, R8, #1	; povećati podatak za 1
	STRB	R8, [R9]	; spremiti bajt natrag na adresu 1000
	SUBS	PC, LR, #4	; povratak iz potprograma
GLAVNI	MOV	R13, #10<12	; inicijalizacija stoga
	MRS	R0, CPSR	; kopiranje registra stanja u opći registar R0
	BIC	R0, R0, #40	; dozvoljavanje FIQ, brisanje bita 6 u CPSR (F)
	MSR	CPSR_c, R0	; kopiranje R0 u registar stanja (CPSR)
PETLJA	B	PETLJA	; prazna petlja

#### Komentar rješenja:

U ovom jednostavnom zadatku vidi se što je sve potrebno napraviti da bi se prihvatio prekid, kako se obrađuje i kako se vraća iz prekidnog potprograma. Stoga u ovom primjeru nije bilo potrebno inicijalizirati jer se na stog ne spremaju nikakvi podaci (za razliku od FRISC-a gdje se prilikom poziva potprograma na stog sprema povratna adresa). S obzirom da se radi o FIQ-u moguće je koristiti posebne registre **R8\_fiq–R14\_fiq**, koji „maskiraju“ opće registre **R8–R14** pa ih nije potrebno spremati kao dio konteksta. Zbog toga se u prekidnom potprogramu koriste registri **R8** i **R9**. Naravno, ako se u prekidnom potprogramu koristi neki od registara **R0–R7**, potrebno ih je spremati kao dio konteksta. Inicijalni skok na glavni program (naredba **B GLAVNI**) preko prekidnog potprograma potreban je da se po pokretanju računala ne bi

odmah izveo prekidni potprogram. Zadatak nema uvjeta zaustavljanja, odnosno izvodi se beskonačno.

Odvojeno ćemo za usporedbu riješiti isti zadatak tako da se umjesto FIQ-a koristi obični prekid IRQ. U glavnom programu se prekid omogućuje brisanjem bita **I** u registru **CPSR** (bit na položaju 7)

Rješenje:			
	ORG	0	
	B	GLAVNI	; skok na glavni
	ORG	18	; adresa prekidnog potprograma za IRQ
	B	PREKIDNI	; preskakanje prekidnog potprograma za FIQ
GLAVNI	MOV	R13, #1<12	; inicijalizacija stoga
	MRS	R0, CPSR	; kopiranje registra stanja u opći registar R0
	BIC	R0, R0, #80	; 80=dozvoljavanje IRQ
	MSR	CPSR_c, R0	; kopiranje R0 u registar stanja
PETLJA	B	PETLJA	; prazna petlja
PREKIDNI	STMFD	R13!, {R8, R9}	; spremanje konteksta
	MOV	R9, #1<12	; adresa 1000 u registar R9
	LDRB	R8, [R9]	; učitati bajt s adrese 1000
	ADD	R8, R8, #1	; povećati podatak za 1
	STRB	R8, [R9]	; spremiti bajt natrag na adresu 1000
	LDMFD	R13!, {R8, R9}	; povratak konteksta
	SUBS	PC, LR, #4	; povratak iz potprograma

U ovom slučaju potrebno je na početku glavnog programa inicijalizirati stog jer na njega u prekidnom potprogramu spremamo registre **R8** i **R9** kao dio konteksta. Potrebna je još jedna napomena: na adresi  $18_{16}$  pišemo naredbu skoka **B PREKIDNI** da bismo preskočili prekidni potprogram za obradu brzog prekida koji je na sljedećoj adresi  $1C_{16}$  (ako postoji). Budući da ga u ovom primjeru nije bilo, mogli smo cijeli potprogram za obradu normalnog prekida napisati od adrese  $18_{16}$ . Da smo imali omogućene i FIQ i IRQ prekida, onda bi se na adresi  $18_{16}$  nalazila naredba skoka (npr. **B PREKID\_IRQ**), a na adresi označenoj labelom **PREKID\_IRQ** bi bio prekidni potprogram za IRQ (kao što je u zadacima 3.1.9. i 3.1.10.).

### 3.1.2. Prijenos podatka pomoću GPIO i RTC (ZI08)

Riješen: DA    Težina: ★★

Sklopom RTC generirati brzi prekid (FIQ) svake 2 sekunde. Na RTC je spojen signal frekvencije 10kHz. U prekidnom potprogramu za obradu prekida RTC-a pročitati 8-bitni podatak u NBC formatu sa vrata B sklopa GPIO. Pročitani podatak potrebno je podijeliti sa 4 korištenjem potprograma DIJELI i poslati ga na vrata A sklopa GPIO. Napisati potprogram DIJELI koji prima 8-bitni podatak u NBC formatu preko registra R0, dijeli ga sa 4 i rezultat (isto u NBC formatu) vraća u registru R0.

Sve potrebne inicijalizacije RTC i GPIO sklopova napraviti u glavnom programu. Adrese sklopova odabrati po volji. Nakon inicijalizacije glavni program beskonačno izvodi praznu petlju.

**Prijedlog rješenja:**

Ako adrese vanjskih jedinica nisu zadane, mogu se proizvoljno postaviti, poštujući dostupan adresni prostor za vanjske jedinice i međusobne odmake pojedinih registara vanjske jedinice u odnosu na baznu adresu. Vanjskim jedinicama se pristupa slično kao kod procesora FRISC: pomoću memorijskih naredaba. Pri tome treba poznavati bazne adrese vanjskih jedinica, a pojedini registri nalaze se na propisanim odmacima od bazne adrese (za točne odmake treba konzultirati tablice adresa dane na predavanjima i dostupne u prilogu ove zbirke). Najčešće ćemo u zadatcima koristiti najviše memorijske adrese, slično kao kod FRISC-a (npr. adresu FFFF0000). Budući da ARM nema apsolutno adresiranje u memorijskim naredbama nego samo registarsko, ovakve se adrese ne mogu izravno zadati u naredbama **STR**, pa ih valja prvo upisati u neki registar koji će se koristiti kao bazni registar prilikom pristupa dotičnoj vanjskoj jedinici. U ovom zadatku se u glavnom programu koristi registar **R0** za čuvanje bazne adrese GPIO-a i registar **R1** za baznu adresu RTC-a. Budući da izravno adresiranje u naredbi **MOV** ne može izravno zadati ovakve adrese (npr. nije moguće napisati **MOV R1, #FFFF0000**), najjednostavnije ih je upisati na unaprijed poznate memorijske lokacije i odatle ih pročitati naredbom **LDR**. U ovom zadatku su bazne adrese vanjskih jedinica zapisane u memoriju na adresu 100 (**ORG 100** na kraju rješenja). Tu se nalazi i konstanta 20000<sub>10</sub> koja će biti potrebna pri inicijalizaciji RTC-a, a također se ne može izravno zapisati u naredbi **MOV**.

U glavnom programu treba inicijalizirati sklop GPIO, tako da se zadaju smjerovi i na vratima A i na vratima B (jer su obrnuti od podrazumijevanih smjerova). Uočite da se za vrata A i B smjerovi ne zadaju na isti način: bit postavljen u 1 zadaje ulazni smjer na vratima B, ali na vratima A zadaje izlazni smjer. U RTC je u registar **RTCMR** potrebno upisati konstantu koja će ostvariti mjerenje 2 sekunde. Konstanta treba podijeliti ulaznu frekvenciju od 10000Hz na 0,5Hz pa iznosi 20000<sub>10</sub>. Osim toga, briše se brojilo RTC-a, iako to nije nužno učiniti na početku rada jer mu je ničica ionako podrazumijevana vrijednost. Da bi RTC mogao postavljati prekide, mora mu se u registar **RTCCR** upisati broj 1. Na kraju glavnog programa potrebno je dozvoliti brze prekide, te izvoditi beskonačnu petlju.

U prekidnom potprogramu potrebno je potvrditi prihvata prekida RTC-u, ponovno inicijalizirati RTC (kako bi se nastavilo s mjerenjem vremenskih intervala), pročitati 8-bitni podatak sa vrata B, pomoću potprograma **DIJELI** ga podijeliti sa 4 i zapisati na vrata A. RTC, za razliku od FRISC-ovog CT-a, ne ponavlja automatski prethodni ciklus brojenja pa ga je potrebno ponovno inicijalizirati. Registar usporedbe **RTCMR** se ne mijenja za vrijeme rada RTC-a pa u njega ne treba upisivati nikakvu vrijednost pri ponovnoj inicijalizaciji (naravno, izuzetak je ako se želi mjeriti drugačiji vremenski interval). Također nije potrebno ponoviti dozvoljavanje prekida. Međutim, brojilo (u registru **RTCLR**) se neće automatski vratiti na ničicu, pa ga treba programski obrisati. Isto tako treba upisati bilo koji podatak u **RTCSTAT/RTCEOI** da bi se deaktivirao prekidni signal **RTCINTR** i obrisao interni registar stanja (dojava prihvata prekida).

Potprogram **DIJELI** je trivijalan jer prima parametar i vraća rezultat preko istog registra, a samo dijeljenje sa 4 se svodi na jednu naredbu logičkog pomaka udesno za dva bita. Potprogram ne sprema kontekst jer ne mijenja druge registre osim **R0** preko kojega vraća rezultat.

**Rješenje:**

```

ORG      0
B        GLAVNI          ; preskočiti prekidni potprogram

PREKIDNI
ORG      1C              ; adresa brzog prekida
STMFD    R13!, {R0, R14} ; spremanje konteksta
MOV      R0, #1<8        ; ponovno učitati bazne adrese:
LDR      R9, [R0], #4     ; GPIO
LDR      R10, [R0]        ; RTC

MOV      R11, #0          ; za reset RTC-a u FIQ
STR      R11, [R10, #0C]  ; obrisati brojilo (obavezno)
STR      R11, [R10, #08]  ; obrisati status (IACK)

LDRB     R0, [R9, #4]     ; čitati sa vrata B (može i LDR)
BL       DIJELI           ; poziv potprograma za dijeljenje
STRB     R0, [R9]         ; pohraniti podatak na vrata A (može i STR)

LDMFD    R13!, {R0, R14} ; povratak konteksta
SUBS     PC, R14, #4      ; povratak iz prekidnog potporgama

DIJELI   MOV      R0, R0, LSR #2 ; potprogram za dijeljenje s 4
MOV      PC, LR

GLAVNI   MOV      R13, #1<16      ; inicijalizacija stoga
MOV      R2, #1<8              ; učitavanje baznih adresa s adrese 100:

LDR      R0, [R2], #4          ; R0 = bazna adresa GPIO-a
LDR      R1, [R2], #4          ; R1 = bazna adresa RTC-a; R2 na labelu BROJAC

; inicijalizacija sklopa GPIO
MOV      R3, #0FF             ; potrebno je promijeniti smjerove:
STR      R3, [R0, #8]         ; vrata A, registar smjera
STR      R3, [R0, #0C]        ; vrata B, registar smjera

; inicijalizacija sklopa RTC
LDR      R3, [R2]              ; dohvatiti konstantu 2000010 iz memorije...
STR      R3, [R1, #4]          ; ...i spremiti je u RTCMR
MOV      R2, #0
STR      R2, [R1, #0C]         ; obrisati brojilo (nije nužno)
MOV      R2, #1
STR      R2, [R1, #10]         ; omogućiti RTC da generira prekid

MRS      R0, CPSR
BIC      R0, R0, #40           ; dozvoliti prekid FIQ
MSR      CPSR_c, R0

LOOP     B        LOOP         ; beskonačna petlja

ORG      100                  ; adrese i konstante
GPIO_ADDR DW      0FFFFFF00    ; bazna adresa sklopa GPIO
RTC_ADDR  DW      0FFFFFFE00   ; bazna adresa sklopa RTC
BROJAC    DW      %D 20000     ; konstanta za inicijalizaciju sklopa GPIO

```

**Komentar rješenja:**

Ako je zadan zadatak sa brzim prekidom, dobro je koristiti činjenicu da su registri **R8–R14** višeznačno definirani, te da se adresiranjem ovih registara u načinu rada FIQ, zapravo pristupa registrima **R8\_fiq–R14\_fiq**, pa ih nije potrebno spremati na stog (jer su ti registri nevidljivi po povratku u glavni program, odnosno u korisnički način rada).

U prekidnom potprogramu potrebno je ponovno učitati bazne adrese iako su već učitane u glavnom programu. Vrijednosti u registrima koje koristi glavni program tretiraju se kao lokalne varijable. U ovom slučaju imamo vrlo kratak i jednostavan program: glavni program izvodi praznu petlju i ne mijenja vrijednosti registara te se točno zna na kojem mjestu će prekid biti prihvaćen (uvijek u praznoj petlji). Zato bi program radio ispravno čak i kada bi u prekidnom potprogramu za adresiranje sklopova GPIO i RTC koristili registre **R0** i **R1**. Bez obzira na to, takvo rješenje ne smijemo koristiti jer to predstavlja (vrlo) lošu programersku praksu i povećava mogućnost nastanka grešaka. Razlog je u tome što u bilo kojem stvarnom programu ne možemo znati na kojem mjestu glavnog programa dolazi do prekida i koje su vrijednosti u tom trenutku u pojedinim registrima. Zato prekidni program nikada ne smije polaziti od pretpostavke da pojedini registri imaju određene vrijednosti.

Primijetite da je kod spremanja konteksta u prekidnom potprogramu potrebno spremati i registar **R14**, jer se u prekidnom potprogramu zapravo koristi registar **R14\_fiq** u kojemu je spremljena povratna adresa iz prekidnog potprograma. Kad bi se, bez spremanja registra **R14**, iz prekidnog potprograma pozvao potprogram za dijeljenje, u **R14** (tj. u **R14\_fiq**) bi se upisala nova povratna adresa za povratak iz potprograma **DIJELI** čime bi se "prepisala" i izgubila povratna adresa za povratak iz prekidnog potprograma. Uočite da i za prekid IRQ treba spremati **R14**, jer i za način rada IRQ postoji jednoznačno definirani registar **R14\_irq** u koji se sprema povratna adresa i prekidnog potprograma.

Primijetite da se čitanje i pisanje na vrata sklopa GPIO zapravo izvodi bezuvjetno, jer je GPIO bezuvjetna jedinica koja nema bistabil stanja ni mogućnost postavljanja prekida. Kao što je već spomenuto, ako je potrebna sinkronizacija sa sklopovima spojenima na GPIO, ona će se morati izvesti programski, što će biti pokazano u nekima od kasnijih zadataka.

**3.1.3. Ispitivanje podatka na vratima sklopa GPIO (ZI09)****Riješen: DA    Težina: ★★**

Napisati program koji inicijalizira sklop RTC tako da generira prekide IRQ svakih 3,5 sekundi. Na RTC je spojen signal frekvencije 10kHz.

U prekidnom potprogramu za IRQ pročitajte stanje na vratima B sklopa GPIO i ako je bilo koja linija vrata postavljena (različita od 0) tada upišite vrijednost 1 na memorijsku lokaciju  $1000_{16}$ , a ako su sve linije vrata B jednake 0 tada upišite vrijednost 0 na memorijsku lokaciju  $1000_{16}$ . Sve potrebne inicijalizacije sklopova RTC i GPIO obavite u glavnom programu. Adrese sklopova odaberite po želji. Nakon inicijalizacije glavni program izvodi beskonačnu petlju.

**Prijedlog rješenja:**

U glavnom programu potrebno je inicijalizirati sklop GPIO tako da zadamo ulazni smjer na svim bitovima vrata B. Sklop RTC inicijaliziramo tako da generira prekid i u RTCMR upišemo  $35000_{10}$  ( $3,5s * 10000Hz$ ). U glavnom programu još treba dozvoliti prekid IRQ. U prekidnom potprogramu potrebno je ponovno učitati adrese sklopova, provjeriti stanje na vratima B i



upisati pripadnu vrijednost na memorijsku lokaciju 1000<sub>16</sub>. Osim toga, treba ponovno inicijalizirati RTC tako da nastavi s mjerenjem vremena.

**Rješenje:**

```

ORG      0
B        GLAVNI          ; skok na glavni program

IRQ      ORG      18          ; adresa prekidnog potprograma IRQ
        STMFD     SP!, {R0,R1,R2,R3} ; spremanje konteksta
        MOV       R0, #2<12    ; ponovno učitati adrese
        LDR       R1, [R0], #4  ; adresa RTC-a
        LDR       R2, [R0]      ; adresa GPIO-a

        LDR       R0, [R2, #4]  ; pročitati podatak s vrata B od sklopa GPIO
        CMP       R0, #0        ; provjeriti jesu li sve linije 0
        ; u slučaju da su sve linije 0, ne treba ponovno stavljati 0 u registar
        MOVNE     R0, #1        ; ako nisu sve 0, upisuje se podatak 1

DALJE    MOV       R3, #1<12    ; adresa 1000 u R3
        STR       R0, [R3]      ; spremanje podatka 0 ili 1 na adresu 1000

        ; reinicijalizacija RTC-a
        STR       R0, [R1, #8]  ; brisanje RTCINT
        MOV       R0, #0        ; brojač na 0
        STR       R0, [R1, #0C] ; spremanje na RTCLR

        MOV       R0, #2<12    ; u R0 upiši adresu za čitanje konstante
        LDR       R3, [R0, #0C] ; čitanje konstante 35000 za RTC
        STR       R3, [R1, #4]  ; ponovno napuniti RTCMR (ne treba)

        LDMFD     SP!, {R0,R1,R2,R3} ; obnova konteksta
        SUBS      PC, LR, #4    ; povratak iz prekidnog potprograma

GLAVNI   MOV       SP, #10<12   ; inicijalizacija stoga
        MOV       R0, #2<12    ; dohvat adresa iz memorije:
        LDR       R1, [R0], #4  ; R1 = RTC
        LDR       R2, [R0], #4  ; R2 = GPIO

        ; inicijalizacija RTC-a
        LDR       R3, [R0]      ; R3 = konstanta = 35000
        STR       R3, [R1, #4]  ; konstanta u RTCMR
        MOV       R3, #0
        STR       R3, [R1, #0C] ; brisanje brojila (ne treba)
        MOV       R3, #1
        STR       R3, [R1, #10] ; RTCCR = 1, prekidi se postavljaju

        ; inicijalizacija GPIO-a
        MOV       R3, #0FF      ; sve linije ulazne
        STR       R3, [R2, #0C] ; smjer GPIO B

        MRS       R0, CPSR      ; omogućavanje prekida
        BIC       R0, #80       ; 80: omogućavanje IRQ-a
        MSR       CPSR_c, R0

PET      B        PET          ; prazna petlja

```

(nastavak na sljedećoj stranici)

	ORG	1000	
	DW	0	; mjesto za upisivanje podatka
	ORG	2000	
RTC	DW	0FFFFFF2100	; adresa RTC-a
GPIO	DW	0FFFFFF2200	; adresa GPIO-a
KONST	DW	%D 35000	; konstanta za brojenje

### Komentar rješenja:

Primijetite da se u izvornom rješenju u prekidnom potprogramu radi cjelovita inicijalizacija RTC-a što nije nužno: suvišan je ponovni upis konstante u registar RTCMR, budući da se njegova vrijednost ne treba mijenjati.

Budući da se u rješenju ne koristi prekid FIQ, onda prekidni potprogram za IRQ možemo pisati odmah od adrese 18<sub>16</sub> i ne moramo na tom mjestu navoditi naredbu skoka **B IRQ**.

U ovom primjeru potrebno je komentirati rad RTC-a. Prilikom njegove inicijalizacije u glavnom programu treba definirati način rada sklopa upisom vrijednosti 1 ili 0 u registar RTCCR (čime se onemogućuje ili omogućuje generiranja prekida), te upisati vrijednost za usporedbu s brojiлом u registar usporedbe RTCMR. U predloženom se rješenju još inicijalno briše brojilo (registar RTCLR) što nije nužno jer taj registar već sadrži vrijednost 0.

Međutim, ako bismo na početku htjeli brojiti od neke vrijednosti različite od 0, to možemo ostvariti inicijalnim upisom željene vrijednosti u registar brojila. Primjer za to bi bio zadatak: „Napisati program koji inicijalizira sklop RTC tako da generira prekide IRQ svakih 3,5 sekundi, osim prilikom prvog prekida koji se mora generirati za 1 sekundu. Na RTC je spojen signal frekvencije 10kHz.“ Dakle, početno treba u RTCLR upisati 25000<sub>10</sub>, jer ćemo tako odbrojiti 10000<sub>10</sub> impulsa do 35000<sub>10</sub>, a potom ćemo u svakom sljedećem prolazu u prekidnom potprogramu resetirati brojilo na 0. Inicijalizacija RTC-a u glavnom programu bi u ovom slučaju izgledala ovako:

	...		
			; inicijalizacija RTC-a
	LDR	R3, [R0], #4	; R3 = konst = 35000
	STR	R3, [R1, #4]	; konstanta u RTCMR
	LDR	R3, [R0]	; R3 = početno brojilo = 25000
	STR	R3, [R1, #0C]	; prvo broji 10000 ciklusa
	MOV	R3, #1	
	STR	R3, [R1, #10]	; RTCCR = 1, prekidi
	...		
	ORG	2000	
RTC	DW	0FFFFFF2100	; adresa RTC-a
GPIO	DW	0FFFFFF2200	; adresa GPIO-a
KONST	DW	%D 35000	; konstanta za brojenje
KONST_1	DW	%D 25000	; početna vrijednost brojila za prvo brojenje

U odnosu na izvorno rješenje, razlika je u upisu početne vrijednosti u brojilo. Ponovna inicijalizacija u prekidnom potprogramu ne bi se mijenjala.

Zadatak s početnim prekidom nakon 1 sekunde mogao se riješiti i tako da je u glavnom programu brojilo RTC-a obrisano te da je upisan broj 10000 kao konstanta brojenja u RTCMR. Kako bi tada izgledala ponovna inicijalizacija u prekidnom potprogramu?

### 3.1.4. Kontinuirano mijenjanje bita na sklopu GPIO (ZI06)

Riješen: DA    Težina: ★★

Na procesor ARM spojeni su RTC (adresa 0FFFFFF00) i GPIO (adresa 0FFFFFFE00). Napisati program koji korištenjem prekida FIQ sa sklopa RTC, svake sekunde mijenja stanje na bitu 5 (početno stanje je 0) od vrata A sklopa GPIO. Na ulaz RTC-a doveden je signal frekvencije 10 Hz. Glavni program izvodi beskonačnu petlju.

#### Prijedlog rješenja:

U glavnom programu potrebno je inicijalizirati RTC (konstanta brojenja je 10), ali i GPIO jer je zadano da treba mijenjati stanje bita 5 od vrata A (mijenjanje bita znači pisanje u njega, a to znači da smjer bita moramo definirati kao izlazni). Primijetite da je potrebno promijeniti smjer samo bitu 5. Dodatno, potrebno je omogućiti prekide FIQ.

U prekidnom potprogramu treba promijeniti stanje na bitu 5. To je najlakše napraviti tako da se pročita trenutno stanje bita 5, zatim se pomoću naredbe za operaciju ekskluzivno ILI komplementira stanje bita (stanje „1“ prelazi u „0“ i obrnuto: stanje „0“ prelazi u „1“) te se konačno natrag na bit 5 upiše komplementirana vrijednost. U prekidnom potprogramu još treba ponovno inicijalizirati RTC.

#### Rješenje:

	ORG	0	
	B	GLAVNI	; skok u glavni program
PREKID	ORG	1C	
	STMFD	R13!, {R0, R2, R3}	; spremanje konteksta
	LDR	R2, GPIO_ADR	; učitavanje adrese RTC-a i GPIO-a
	LDR	R3, RTC_ADR	
	LDR	R0, [R2]	; promjena bita 5 na vratima A
	EOR	R0, R0, #20	
	STR	R0, [R2]	
	STR	R0, [R3, #8]	; obrađen prekid RTC-a
	MOV	R0, #0	
	STR	R0, [R3, #0C]	; vraćanje brojila na nulu
	LDMFD	R13!, {R0, R2, R3}	; obnavljanje konteksta
	SUBS	PC, R14, #4	; povratak iz prekida
GLAVNI	MOV	R13, #1<16	; inicijalizacija stoga
	LDR	R2, GPIO_ADR	; učitavanje adrese sklopa GPIO
	LDR	R3, RTC_ADR	; učitavanje adrese sklopa RTC
	MOV	R0, #0A	; 1 sekunda (10 dekadski)
	STR	R0, [R3, #4]	; upis u RTCMR
	MOV	R0, #1	; RTC generira prekid
	STR	R0, [R3, #10]	; upis u RTCCR
	MOV	R0, #20	; postavljanje bita 5 kao izlaznog
	STR	R0, [R2, #8]	; upis u registar smjera vrata A
	MOV	R0, #0	; početno stanje bita 5 je 0
	STR	R0, [R2]	; upis u vrata A

(nastavak na sljedećoj stranici)

	MRS	R0, CPSR	; omogućavanje prekida FIQ
	BIC	R0, R0, #40	
	MSR	CPSR_c, R0	
PETLJA	B	PETLJA	; beskonačna petlja
GPIO_ADR	DW	0FFFFFFE00	; adrese sklopova RTC i GPIO
RTC_ADR	DW	0FFFFFFF00	

### Komentar rješenja:

U glavnom programu načelno treba omogućiti prihvaćanje prekida tek nakon inicijalizacije svih vanjskih jedinica i ostalih parametara, jer bi se inače moglo dogoditi da neka od njih postavi prekid, a da još nisu inicijalizirane sve vanjske jedinice, brojači i ostali parametri programa.

Prilikom omogućavanja prihvaćanja prekida, potrebno je zapisati polje „c“ registra **CPSR**, ali nije greška ako se upišu sva polja (**MSR CPSR\_cesf, R0**). Primijetite da se, za razliku od FRISC-a, kod ARM-a prihvaćanje pojedinih prekida zadaje upisom „0“ u odgovarajući bit registra **CPSR** (bit 6 za FIQ, bit 7 za IRQ).

Prekidni potprogram koristi registre **R0, R2** i **R3** pa ih treba spremati kao kontekst. Bolje rješenje bi bilo korištenje registara **R8–R12**.

U ovom rješenju pokazan je drugačiji način upisivanja adresa vanjskih jedinica u registre (kojima će se vanjske jedinice kasnije adresirati). Umjesto da su adrese upisane na memorijsku lokaciju s poznatom adresom zadanom pseudonaredbom **ORG** (npr. **ORG 2000** u prethodnom zadatku), ovdje su adrese upisane na memorijske lokacije smještene neposredno iza glavnog programa. Dodatno su ove lokacije označene labelama (**GPIO\_ADR** i **RTC\_ADR**), jer sada ne znamo njihovu adresu. Budući da su sada lokacije s adresama sigurno dovoljno blizu glavnog programa i prekidnog potprograma (udaljene su sigurno manje od 4095 bajtova) može se upotrijebiti, na primjer, naredba **LDR R2, GPIO\_ADR** za dohvat adrese sklopa GPIO u registar **R2**. Podsjetite se da procesor ARM zapravo nema ovakav način adresiranja, nego ga assembleri prevoditelj pretvara u naredbu **LDR R2, [PC, #odmak]**, gdje prevoditelj automatski izračunava potreban iznos za **odmak**. Ovakvo učitavanje adresa je jednostavnije i kraće od do sada pokazanog, a nema ni korištenja dodatnog registra za dohvat adrese zadane pseudonaredbom **ORG**. Jedino treba paziti da lokacije s adresama budu dovoljno blizu mjestu njihovog korištenja, što ne bi trebao biti problem u slučaju kratkih programa kakvi su pokazani u ovoj zbirci.

### 3.1.5. Alarm ostvaren sklopovima GPIO i RTC

Riješen: DA    Težina: ★★

U računalnom sustavu nalazi se procesor ARM te sklopovi GPIO i RTC. Na vrata B sklopa GPIO spojeno je 8 senzora koji čine alarmni sustav. Napišite program koji svakih 10 sekundi obavlja provjeru stanja senzora. Ako je bilo koji od senzora postavljen u logičko 0, treba aktivirati alarm upisom 1 na bit 0 vrata A sklopa GPIO. Ako su svi senzori postavljeni u logičko 1, treba deaktivirati alarm upisom 0 na bit 0 vrata A sklopa GPIO.

Adresa sklopa GPIO je FFFFFFF0<sub>16</sub>, a sklopa RTC FFFFFE00<sub>16</sub>. Na ulaz sklopa RTC spojen je signal frekvencije 1kHz. Sklop RTC je spojen na IRQ.

**Prijedlog rješenja:**

U ovom primjeru vrata B se koriste za očitavanje stanja senzora pa umjesto podrazumijevanog izlaznog smjera treba zadati ulazni smjer na svim bitovima. Na vratima A se koristi samo najniži bit pa je samo njemu potrebno promijeniti smjer iz ulaznog u izlazni. U prekidnom potprogramu treba čitati vrata B i provjeravati stanje pojedinih bitova (odnosno senzora spojenih na njih).

**Rješenje:**

```

ORG      0
B        GLAVNI          ; skok u glavni program

ORG      18              ; adresa prekidnog potprograma
B        IRQ             ; skok ili direktno potprogram

IRQ      STMFD    SP!, {R0, R1, R2}    ; spremanje konteksta
LDR      R1, RTC          ; učitavanje adresa, R1 = RTC
LDR      R2, GPIO         ; R2 = GPIO
LDR      R0, [R2, #4]      ; pročitati stanje senzora sa GPIO B

CMP      R0, #0FF         ; provjeravanje jesu li svi bitovi 1
MOVNE    R0, #1           ; podatak 1 znači uključivanje alarma
STRNE    R0, [R2]         ; slanje na GPIO - aktiviranje alarma
MOVEQ    R0, #0           ; podatak 0 znači isključivanje alarma
STREQ    R0, [R2]         ; slanje na GPIO - deaktiviranje alarma

STR      R0, [R1, #8]      ; brisanje RTCINT
MOV      R0, #0           ; vratiti brojač RTC-a na 0
STR      R0, [R1, #0C]
LDMFD    SP!, {R0, R1, R2}    ; obnova konteksta
SUBS     PC, LR, #4        ; povratak iz prekidnog potprograma

GLAVNI   MOV      SP, #10<12    ; inicijalizacija stoga
LDR      R1, RTC          ; R1 = RTC
LDR      R2, GPIO         ; R2 = GPIO
LDR      R3, KONST        ; R3 = konstanta = 10000
STR      R3, [R1, #4]      ; konstanta u RTCMR
MOV      R3, #0
STR      R3, [R1, #0C]      ; brisanje brojila (ne treba)
MOV      R3, #1
STR      R3, [R1, #10]      ; RTCCR = 1, prekidi

MOV      R3, #0FF         ; sve linije ulazne
STR      R3, [R2, #0C]      ; smjer GPIO B
MOV      R3, #1           ; bit 0 - izlazni
STR      R3, [R2, #8]      ; smjer GPIO A

MRS      R0, CPSR          ; omogućavanje prekida
BIC      R0, R0, #80        ; 80 - IRQ
MSR      CPSR_c, R0

PETLJA   B        PETLJA      ; prazna petlja

RTC      DW      0FFFFFFE00    ; adresa RTC-a
GPIO     DW      0FFFFFFF00    ; adresa GPIO-a
KONST    DW      %D 10000      ; vremenska konstanta

```

**Komentar rješenja:**

U ovom primjeru potrebno je razumijevanje uvjeta za aktiviranje odnosno deaktiviranje alarma. Ako su svi senzori u 1 moramo isključiti alarm, a ako je pročitana bilo koja druga kombinacija (nije potrebno provjeravati 0 na pojedinim sensorima), potrebno je uključiti alarm. Kada bismo htjeli zakomplicirati zadatak, mogli bismo reći da se alarm uključuje tek kada su minimalno dva senzora postavljena u logičku 0. Prekidni potprogram tada bi izgledao ovako:

IRQ	STMFD	SP!, {R0, R1, R2, R3, R4}	; spremanje konteksta
	MOV	R3, #8	; brojač za petlju
	MOV	R4, #0	; brojač senzora u stanju 0
	LDR	R1, RTC	; učitavanje adresa, R1 = RTC
	LDR	R2, GPIO	; R2 = GPIO
	LDR	R0, [R2, #4]	; pročitati stanje senzora sa GPIO B
	CMP	R0, #0FF	; provjeravanje jesu li svi bitovi 1
	BNE	ALARM	
	MOV	R0, #0	; podatak 0
	STR	R0, [R2]	; spremanje na GPIO - deaktiviranje alarma
ALARM	B	VAN	
	MOVS	R0, R0, ROR #1	; rotacija za 1
	ADDCC	R4, R4, #1	; brojanje 0
	SUBS	R3, R3, #1	; za svih 8 bitova
	BNE	ALARM	
	CMP	R4, #2	; ako je 2 ili više senzora u 0
	MOVHS	R0, #1	; podatak 1 - uključivanje alarma
	MOVLO	R0, #0	; podatak 0 - isključivanje alarma
	STR	R0, [R2]	; slanje na GPIO - uključi ili isključi alarm
VAN	STR	R0, [R1, #8]	; brisanje RTCINT
	MOV	R0, #0	; vratiti brojač RTC-a na 0
	STR	R0, [R1, #0C]	
	LDMFD	SP!, {R0, R1, R2, R3, R4}	; obnova konteksta
	SUBS	PC, LR, #4	; povratak iz prekida

**3.1.6. Prijenos s vrata B na vrata A sklopa GPIO (ZI07)****Riješen: DA    Težina: ★★★**

U sustavu s procesorom ARM nalaze se RTC (na adresi FFFF1000) i GPIO (na adresi FFFF2000). RTC broji vanjske impulse i nakon svakih  $1000_{16}$  impulsa generira iznimku FIQ. U prekidnom potprogramu treba pročitati vrijednost postavljenu na vrata B sklopa GPIO, te izračunati paritet pročitano podataka. Ako pročitani podatak ima paran paritet tada se šalje na vrata A sklopa GPIO, a u slučaju neparnog pariteta ne radi se ništa. Funkciju računanja pariteta riješiti potprogramom **PAR**. Nakon što se primi  $300_{16}$  podataka, treba onemogućiti daljnje generiranje iznimaka od strane RTC-a.

**Prijedlog rješenja:**

U glavnom programu treba promijeniti smjer na vratima A i B GPIO sklopa, jer su potrebni smjerovi obrnuti od podrazumijevanih. Također treba inicijalizirati RTC da broji  $1000_{16}$  ulaznih impulsa, što je analogno dosadašnjim zadacima u kojima je RTC mjerio vrijeme. Ono što je specifično za ovaj zadatak je poziv potprograma iz prekidnog potprograma. Zbog toga je nužno na stog spremati povratnu adresu (prvotno spremljenu u **R14**) za povratak iz

prekidnog potprograma, jer bi inače bila prepisana povratnom adresom za povratak iz potprograma **PAR**.

Također, u prekidnom potprogramu valja prebrajati prekide, jer se nakon generiranih 300<sub>16</sub> prekida (i pročitanih 300<sub>16</sub> podataka) RTC-u mora onemogućiti generiranje daljnjih prekida. U rješenju se koristi memorijska lokacija **BROJAC** u koju je inicijalno upisana vrijednost 300. U prekidnom potprogramu se vrijednost lokacije **BROJAC** smanjuje za jedan i provjerava se je li **BROJAC** došao do ničice. Kad **BROJAC** postane ničica, RTC-u se zabrani generiranje daljnjih prekida. Ovdje je bitno razlikovati omogućavanje/onemogućavanje generiranja prekida na vanjskoj jedinici od omogućavanja/onemogućavanja prihvaćanja pojedinih prekida (iznimaka) na samom procesoru. U ovom slučaju, nećemo onemogućiti prihvrat daljnjih prekida, nego ćemo onemogućiti uzročnika prekida (RTC), što znači da bi procesor i dalje mogao prihvaćati prekide koje bi generirale druge vanjske jedinice (kada bi postojale).

Za potprogram **PAR** zadano je samo da mora računati paritet, ali ne i način primanja parametra i vraćanja rezultata. U rješenju je proizvoljno odabrano da se broj kojemu se želi izračunati paritet prenosi kao parametar pomoću registra **R9**. Rezultat se vraća registrom **R10** i može biti 0 (što označava da parametar ima parni paritet) ili 1 (što označava da parametar ima neparni paritet). Potprogram **PAR**, naravno, čuva registre koje mijenja. Podsjetite se da je paritet paran ako u broju postoji paran broj jedinica, a neparan je ako u broju postoji neparan broj jedinica.

#### Rješenje:

```

ORG      0
MOV      SP, #10<12      ; inicijalizacija stoga
B        GLAVNI          ; skok u glavni

ORG      1C              ; adresa potprograma za FIQ
STMFD    R13!,{R7,R14}   ; spremanje konteksta
LDR      R7, GPIO        ; učitavanje adresa...
LDR      R8, RTC          ; ...RTC-a i GPIO-a
LDR      R9, [R7,#GPIOBDR] ; čitanje podatka sa vrata B
BL       PAR              ; poziv potprograma za izračunavanje pariteta

CMP      R10, #0
STREQ    R9, [R7,#GPIOADDR] ; ako je paran paritet, poslati na vrata A

MOV      R9, #0
STR      R9, [R8,#RTCLR] ; resetiranje brojača
STR      R9, [R8,#RTCEOI]; potvrđivanje primitka prekida

LDR      R11, BROJAC      ; smanjivanje brojača prekida
SUBS     R11, R11, #1
STR      R11, BROJAC

BNE      VAN              ; ispitivanje je li bilo 300 prekida

STR      R9, [R8,#RTCCR] ; onemogućavanje daljnjih prekida
VAN      LDMFD    R13!,{R7,R14} ; obnova konteksta
SUBS     PC, LR, #4       ; povratak iz prekida

BROJAC   DW          300   ; inicijalizacija brojača

```

(nastavak na sljedećoj stranici)

```

GPIO      DW      0FFFF1000
GPIOPADR  EQU      0
GPIOPBDR  EQU      4
GPIOPADDR EQU      8
GPIOPBDDR EQU      0C

RTC        DW      0FFFF2000
RTCDR      EQU      0
RTCMR      EQU      4
RTCEOI     EQU      8
RTCLR      EQU      0C
RTCCR      EQU      10

; potprogram za računanje pariteta: parametar=R9, rezultat=R10
PAR        STMFD   R13!, {R1,R9}    ; spremanje konteksta
           MOV     R1, #8            ; brojač petlje (paritet je za 8-bitni podatak)
           MOV     R10, #0           ; brojač jedinica u podatku, tj. rezultat

PETLJA     MOVS    R9, R9, LSR #1    ; najniži bit u zastavicu C
           ADC     R10, R10, #0      ; dodavanje C rezultatu (ili ADDCS R2,R2,#1)
           SUBS    R1, R1, #1        ; smanjivanje brojača petlje
           BNE     PETLJA            ; skok ako nije obrađen cijeli bajt

           ; provjera parnosti prebrojenih jedinica pomoću najnižeg bita
           AND     R10, R10, #1      ; paritet paran=>R10=0, paritet neparan=>R10=1
           LDMFD   R13!, {R1,R9}    ; obnova konteksta
           MOV     PC, LR            ; povratak iz potprograma

GLAVNI     LDR     R3, GPIO          ; učitavanje adrese GPIO-a
           LDR     R4, RTC           ; učitavanje adrese RTC-a

INIT_GPIO  MVN     R0, #0            ; jedinice za promjenu smjera za vrata A i B
           STR     R0, [R3,#GPIOPADDR] ; vrata A: izlazna
           STR     R0, [R3,#GPIOPBDDR] ; vrata B: ulazna

INIT_RTC   MOV     R0, #0
           STR     R0, [R4,#RTCLR]   ; inicijalizacija brojača
           MOV     R0, #1<12
           STR     R0, [R4,#RTCMR]   ; konstanta MR=1000
           MOV     R0, #1
           STR     R0, [R4,#RTCCR]   ; omogućavanje prekida na RTC-u

           MRS     R0, CPSR
           BIC     R0, R0, #40       ; omogućavanje prihvata FIQ
           MSR     CPSR_c, R0

LOOP       B       LOOP             ; prazna petlja

```

**Komentar rješenja:**

Na ulaz RTC-a u ovom zadatku nije spojen generator takta već neki proizvoljni izvor impulsa. Neovisno o tome, način inicijalizacije i komunikacije s RTC-om je jednak.

U rješenju vidimo još jednu mogućnost adresiranja pojedinih lokacija u vanjskim jedinicama: pomoću labela koje imaju vrijednost odmak na kojima se nalaze pojedini registri vanjskih jedinica. Vrijednosti labela zadane su pomoću pseudonaredaba **EQU**, pri čemu su zadani



samo odmaci, a ne apsolutne adrese. Bazna adresa sklopa i dalje se mora zadavati jednim od općih registara. Ovakvo adresiranje se zapravo ne razlikuje od dosadašnjih, samo što je zbog korištenja labela tekst programa nešto čitljiviji, jer se odmah vidi kojemu registru u vanjskoj jedinici se pristupa (osim ako ne znate napamet sve odmake registara u vanjskim jedinicama).

U ovom primjeru je u glavnom programu konstanta OFF za zadavanje smjera vrata A i B zadana drugačije nego u ostalim primjerima. Umjesto do sada korištene naredbe **MOV R0, #0FF** upotrijebljena je naredba **MVN R0, #0**. Primijetite da smo naredbom **MVN R0, #0** zapravo upisali 32 jedinice u registar **R0**, iako su nam jedinice potrebne samo u najnižih 8 bitova. Međutim, u 8-bitni registar smjera za pojedina vrata ionako je moguće upisati samo najnižih 8 bitova, pa je nevažno što piše u gornjih 24 bita (kao što je svejedno upisujemo li podatak naredbom **STR** ili **STRB**).

Može li se optimirati spremanje konteksta u prekidnom potprogramu?

### 3.1.7. Termostat s dva sklopa GPIO

Riješen: DA    Težina: ★★

Procesor ARM upravlja sustavom grijanja na temelju željene temperature namještene na termostatu. Na ARM su spojeni sklopovi GPIO1 (adresa FFFF0000) i GPIO2 (adresa FFFF1000) te RTC (adresa FFFF2000, spojen na IRQ). Na vrata A sklopa GPIO1 spojen je termostat s kojeg se (sa nižih 6 bitova) bezuvjetno može učitati trenutno namještena željena temperatura.

Na vrata B sklopa GPIO2, na nižih 6 bitova, spojen je termometar kojim se očitava stvarna trenutna temperatura prostorije. Svake sekunde treba očitati temperaturu i na temelju toga uključiti ili isključiti grijanje. Grijanje se uključuje ako je trenutna temperatura strogo manja od željene temperature koja je trenutačno podešena na termostatu, a inače se isključuje. Grijanje se uključuje slanjem 1 na najviši bit vrata A sklopa GPIO2, a isključuje se slanjem 0. Kašnjenje od jedne sekunde ostvarite sklopom RTC na čiji ulaz je spojen signal frekvencije 10 kHz. Glavni program, nakon svih potrebnih inicijalizacija treba izvoditi beskonačnu petlju.

#### Prijedlog rješenja:

U glavnom programu potrebno je inicijalizirati vrata A i B sklopa GPIO2, a budući da vrata A sklopa GPIO1 koristimo kao ulazna, njih nije potrebno dodatno inicijalizirati. Prekidni potprogram uspoređuje očitane temperature sa željenom temperaturom i u ovisnosti o njihovom odnosu uključuje ili isključuje grijanje. Treba očitati obje temperature, jer se obje mogu promijeniti u bilo kojem trenutku. Zadano je da se temperatura s termometra čita bezuvjetno (dakle, ne radi se o termometru obrađenom na predavanjima koji zahtjeva sinkronizaciju).

Rješenje:			
ORG	0		
MOV	SP, #10<12		; inicijalizacija stoga
B	MAIN		; skok na glavni program
ORG	18		
STMFD	SP!, {R0, R1, R2, R3, R4, R5}		; spremanje konteksta
(nastavak na sljedećoj stranici)			
LDR	R0, GPIO1		; učitavanje adresa sklopova

	LDR	R1, GPIO2	
	LDR	R2, RTC	
	MOV	R5, #0	
	STR	R5, [R2, #8]	; potvrđivanje obrade prekida
	STR	R5, [R2, #0C]	; resetiranje brojača
	LDRB	R3, [R0]	; učitavanje željene temper. s vrata A od GPIO1
	AND	R3, R3, #3F	; nižih 6 bitova
	LDRB	R4, [R1, #4]	; učitavanje trenutne temper. s vrata B od GPIO2
	AND	R4, R4, #3F	; nižih 6 bitova
	CMP	R3, R4	; ako je željena temperatura veća od trenutne...
	MOVGE	R5, #80	; ... onda uključi grijanje...
	MOVLT	R5, #0	; ... a ako je manja, onda isključi grijanje
	STR	R5, [R1]	
	LDMFD	SP!, {R0, R1, R2, R3, R4, R5}	; obnavljanje konteksta
	SUBS	PC, LR, #4	; povratak iz potprograma
MAIN	LDR	R1, GPIO2	
	LDR	R2, RTC	
	MOV	R3, #80	; bit 7 je izlazni
	STR	R3, [R1, #8]	; GPIO2 vrata A
	MOV	R3, #3F	; bitovi 0-6 su ulazni
	STR	R3, [R1, #0C]	; GPIO2 vrata B
	LDR	R3, KONST	
	STR	R3, [R2, #4]	; RTC konstanta
	MOV	R3, #1	
	STR	R3, [R2, #10]	; omogućavanje prekida na RTC-u
	MRS	R0, CPSR	
	BIC	R0, R0, #80	; omogućavanje prihvata IRQ
	MSR	CPSR_C, R0	
LOOP	B	LOOP	; prazna petlja
GPIO1	DW	0FFFF0000	
GPIO2	DW	0FFFF1000	
RTC	DW	0FFFF2000	
KONST	DW	%D 10000	

### Komentar rješenja:

Način inicijalizacije i rukovanja većim brojem GPIO sklopova identičan je načinu inicijalizacije i rukovanja samo jednim GPIO sklopom. Specifičnost zadatka je u načinu upisa 1, odnosno 0 u najviši bit registra **R5** u prekidnom potprogramu:

...			
CMP	R3, R4		; ako je željena temperatura veća od trenutne...
MOVGE	R5, #80		; ... onda uključi grijanje...
MOVLT	R5, #0		; ... a ako je manja, onda isključi grijanje
STR	R5, [R1]		
...			

Uvjetima **GE** i **LT** smo obuhvatili sve moguće ishode usporedbe i omogućili rješenje bez skokova i nepotrebnih usporavanja programa.

Budući da se u rješenju ne koristi FIQ, već samo IRQ, onda na adresi  $18_{16}$  nije navedena naredba skoka u prekidni potprogram za obradu prekida IRQ, već je tu smješten sam prekidni potprogram.

### 3.1.8. Ispis teksta na LCD

Riješen: DA    Težina: ★★★

Napisati program koji stvara efekt rotirajućeg teksta na LCD prikazniku opisanom na predavanjima. LCD je spojen na vrata A sklopa GPIO (na adresi FFFFFFF80). Tekst koji treba prikazati spremljen je u memoriji od lokacije  $200_{16}$  i ima 8 ASCII znakova (jedan znak je jedan bajt). Glavni program treba prikazati cijeli tekst, te potom izvoditi beskonačnu petlju, a rotiranje teksta svakih 500ms treba ostvariti korištenjem sklopa RTC u prekidnom načinu (na adresi FFFFFFF20, spojenom na IRQ). Na ulaz CLK1HZ spojen je signal takta 10kHz. Za slanje znaka na LCD koristiti potprogram **LCDWR** kojemu se znak koji se želi ispisati šalje preko registra R4, a adresa vrata se šalje preko R2. Poslani tekst se prikazuje nakon slanja znaka LF (ASCII =  $A_{16}$ ) na LCD.

Primjer izvođenja: na početku se ispisuje „PROCESOR“, nakon 500ms „ROCESORP“, nakon 500ms „OCESORPR“ itd.

#### Prijedlog rješenja:

Prilikom rješavanja ovog zadatka treba se podsjetiti kako radi LCD opisan na predavanjima. Znak poslan LCD-u se zapravo zapisuje u njegov interni registar na desnu poziciju, a svi prethodni znakovi se pomiču ulijevo (dotadašnji znak s lijeve pozicije se gubi). To je upravo ono što je zadano zadatkom i što treba ponavljati svakih 500ms.

Potprogram **LCDWR** sličan je onom objašnjenom na predavanjima. U njemu se ostvaruje potrebna sinkronizacija s LCD-om i to pomoću najvišeg bita. Da bi se znak poslao LCD-u, potrebno je prvo ASCII kôd znaka postaviti na najnižih 7 bitova, a zatim na najvišem bitu generirati pozitivni impuls (promijeniti bit iz 0 u 1 i zatim natrag iz 1 u 0). Ovaj impuls signalizira LCD-u da treba zapamtiti nižih 7 bita u svojem internom registru. Ovo je primjer sinkronizacije između vanjskog uređaja (LCD-a) i sklopa GPIO. Kao što je ranije bilo spomenuto, ova sinkronizacija mora se ostvariti programski, jer GPIO nema linije za sinkronizaciju (*handshake*, rukovanje). Budući da za vrijeme generiranja impulsa donjih 7 bitova moraju ostati nepromijenjeni, impuls se ostvaruje pomoću naredaba **AND** i **ORR** kojima mijenjamo samo najviši bit na vratima A.

U glavnom programu se prvo inicijaliziraju sklopovi GPIO i RTC. Nije potrebno napraviti nikakvu posebnu inicijalizaciju LCD-a, jer na početku rada LCD ionako ne prikazuje nikakav tekst. Bez obzira na to, na LCD je poslan znak 0D, koji će obrisati interni registar LCD-a. Zatim treba poslati cijeli tekst od adrese  $200_{16}$  i naposljetku ga prikazati (paziti da se znak za prikaz teksta 0A pošalje tek nakon što je svih 8 znakova teksta pohranjeno u interni registar LCD-a). Tekst se šalje u petlji **ISPIS**. U registru **R3** pamti se adresa znaka koji se šalje u pojedinom koraku petlje. Kad **R3** poprimi vrijednost  $208_{16}$ , znači da je poslano svih 8 znakova i petlja završava s radom. Za slanje svakog znaka LCD-u koristi se potprogram **LCDWR**.

Prekidni potprogram treba ponovno inicijalizirati RTC, te rotirati tekst ulijevo i prikazati ga. Kao što je malo prije rečeno, za rotaciju teksta dovoljno je poslati samo jedan znak, a jedino na što treba obratiti pažnju je da se znakovi šalju odgovarajućim redoslijedom. U prvom

izvođenju prekidnog potprograma treba poslati znak s adrese 200, u sljedećem izvođenju znak s adrese 201 itd. Nakon slanja znaka s adrese 207, u sljedećem izvođenju prekidnog potprograma treba ponovno slati znak s adrese 200 i to zatim kružno ponavlja (beskonačno).

Adresu znaka, kojega prekidni potprogram treba sljedećeg poslati na LCD, pamti se u posebnoj memorijskoj lokaciji **ADR\_ZNAKA**. Iako u ovako kratkom i jednostavnom zadatku možemo odvojiti jedan registar kojega bi se koristilo isključivo u prekidnom potprogramu, to bi bila loša programerska praksa. U općem slučaju, glavni program i njegovi potprogrami mogu koristiti bilo koje registre pa bi se stanje tog registra izgubilo nakon povratka iz prekidnog potprograma.

#### Rješenje:

```

ORG      0
B        GLAVNI          ; skok na glavni program

PREKID   ORG      18          ; adresa za obradu iznimke
        STMFD    R13!, {R0, R1, R2, R3, R4, LR} ; pohranjivanje konteksta
        MOV      R4, #3<8    ; učitavanje adresa sklopova...
        LDMIA    R4!, {R0, R1} ; ... GPIO (R0) i RTC (R1)

        MOV      R2, #0
        STR      R2, [R1, #0C] ; resetiranje brojača u RTC-u
        STR      R2, [R1, #8]  ; potvrđivanje primitka prekida RTC-u

        MOV      R2, R0        ; u R2 staviti adresu vrata A GPIO-a (za LCDWR)
        LDR      R3, ADR_ZNAKA ; dohvatiti adresu znaka kojeg treba poslati
        CMP      R3, #82<2    ; je li ispisan cijeli tekst? (adresa == 208?)
        MOVEQ    R3, #2<8    ; ako je ispisan cijeli tekst, krenuti ispočetka

        LDRB     R4, [R3], #1  ; učitavanje znaka i pomak adrese na sljedeći
        BL       LCDWR        ; slanje znaka na LCD
        MOV      R4, #0A      ; prikaz trenutnog teksta na LCD-u
        BL       LCDWR

        STR      R3, ADR_ZNAKA ; spremi adresu znaka za ispis za sljedeći...
                                ; ... poziv prekidnog potprograma
        LDMFD    R13!, {R0, R1, R2, R3, R4, LR} ; obnavljanje konteksta
        SUBS     PC, LR, #4    ; povratak iz prekidnog potprograma

ADR_ZNAKA DW      200          ; adresa znaka kojega treba slati na LCD

LCDWR    STMFD    R13!, {R4}    ; pohranjivanje konteksta

        AND      R4, R4, #7F    ; postaviti bit 7 u nulu (za svaki slučaj)
        STRB     R4, [R2]
        ORR      R4, R4, #80    ; postaviti bit 7 u jedan (podigni impuls)
        STRB     R4, [R2]
        AND      R4, R4, #7F    ; postaviti bit 7 u nulu (spusti impuls)
        STRB     R4, [R2]

        LDMFD    R13!, {R4}    ; obnavljanje konteksta
        MOV      PC, LR        ; povratak iz potpograma

(nastavak na sljedećoj stranici)
GLAVNI   MOV      R13, #1<16    ; inicijalizacija stoga
        MOV      R3, #3<8      ; na adresi 300 se nalaze adrese RTC i GPIO
        LDMIA    R3!, {R0, R1, R2} ; R0 = GPIO, R1 = RTC, R2 = RTC konstanta

```

	MOV	R4, #0FF	; svih 8 bitova su izlazni
	STR	R4, [R0, #8]	; smjer vrata A
	STR	R2, [R1, #4]	; upis konstante u RTCMR
	MOV	R3, #1	
	STR	R3, [R1, #10]	; omogućavanje prekida u RTCCR
	MOV	R4, #0D	; brisanje internog registra LCD-a (ne treba)
	BL	LCDWR	
	; inicijalni ispis u glavnom programu		
	MOV	R2, R0	; adresu vrata A upisati u R2 (za potprogram)
	MOV	R3, #2<8	; adresa prvog znaka za ispis
ISPIS	LDRB	R4, [R3], #1	; podatci se u potprogram šalju preko R4
	BL	LCDWR	
	CMP	R3, #82<2	; je li ispisan cijeli tekst? (adresa == 208?)
	BNE	ISPIS	
	MOV	R4, #0A	; inicijalni prikaz cijelog teksta
	BL	LCDWR	
	MRS	R3, CPSR	
	BIC	R3, R3, #80	; omogućavanje prihvata IRQ
	MSR	CPSR_C, R3	
PETLJA	B	PETLJA	; prazna petlja
	ORG	200	
	DW	50,52,4F,43,45,53,4F,52	; 8 znakova za ispis („PROCESOR“)
	ORG	300	
GPIO	DW	0FFFFFFF80	; adresa GPIO
RTC	DW	0FFFFFFF20	; adresa RTC-a
CONST	DW	%D 5000	; vremenska konstanta

**Komentar rješenja:**

Primijetite da u glavnom programu šaljemo znak za prikaz teksta (0A<sub>16</sub>) tek kad smo cijeli tekst pohranili u interni registar, a u prekidnom potprogramu pohranjujemo samo jedan znak na krajnje desno mjesto internog registra (ostali se znakovi automatski pomiču za jedno mjesto ulijevo) i odmah nakon toga prikažemo test slanjem znaka 0A<sub>16</sub>.

**3.1.9. Upravljanje pružnim prijelazom pomoću sklopovima GPIO i RTC****Riješen: DA    Težina: ★★★**

Napisati program koji pokreće svjetlosnu i zvučnu signalizaciju na prijelazu željezničke pruge. Pinovi vrata B sklopa GPIO proizvoljne adrese spojeni su kako slijedi:

- na pin 2 spojen je senzor dolaska vlaka (0 ako vlaka još nema, 1 signalizira dolazak)
- na pin 3 spojen je senzor odlaska vlaka (0 ako vlak još nije prošao, 1 signalizira da je vlak prošao)
- na pinove 4 i 5 su spojena dva crvena svjetla na znaku (upaljena kad je 1, ugašena kad je 0)
- na pin 6 spojeno je zvono (aktivira se impulsom – poslati 1 i odmah nakon toga 0)

Glavni program kontinuirano ispituje senzor dolaska vlaka i ako uoči da je postavljen u 1, počinje sa signalizacijom kojom upravlja sklop RTC1 spojen na IRQ. Na RTC1 proizvoljne adrese spojen je signal frekvencije 10kHz. Svakih 500ms se naizmjenično pale i gase dva crvena svjetla na znaku (jedno se pali, a drugo gasi) te se oglašava zvono. Tada počinje i kontinuirano ispitivanje senzora odlaska vlaka i ako se uoči da je postavljen u 1 sva signalizacija se gasi. Pretpostavite da se vlak ne zaustavlja na ili u blizini prijelaza te da se senzori sami vrate u inicijalno stanje 0 po prolasku vlaka.

Dodatno pomoću sklopa RTC2 spojenog na FIQ treba svakih 24 sata (86400 sekundi) zapisati na vrata A sklopa GPIO broj prolazaka vlakova (pretpostavite da ih neće biti više od 256<sub>10</sub>). Na RTC2 proizvoljne adrese spojen je signal frekvencije 1Hz.

### Prijedlog rješenja:

U glavnom programu je, osim inicijalizacije sklopa GPIO, potrebno omogućiti prekide FIQ i IRQ (ovo je moguće napraviti odjedanput). Inicijalizaciju sklopa RTC2, kao i inicijalizaciju sklopa RTC1 potrebno je napraviti u glavnom programu. Reinicijalizacija sklopa RTC1 se obavlja u potprogramu za obradu IRQ, a reinicijalizacija sklopa RTC2 se obavlja u potprogramu za obradu FIQ. Ovdje je potrebno napomenuti da je upisivanje konstante u RTCMR sklopova RTC moguće napraviti i samo jednom, na početku glavnog programa, jer se ona kasnije ne mijenja. Ono što je nužno, a spada u inicijalizaciju RTC1, je omogućavanje prekida u RTCCR, nakon svake pojave dolaska vlaka.

Stanja senzora ne mogu se procesoru dojaviti niti prekidom (jer GPIO ne može generirati prekid) niti uvjetno (jer GPIO nema bistabil stanja). Zato se mora izravno, tj. programski očitavati stanje senzora spojenog na GPIO i ispitivati njegovo stanje kako bi se prepoznala promjena stanja na senzoru. Ova ispitivanja izvodit ćemo u glavnom programu, nakon početnih inicijalizacija. Ispitivanja se naizmjenično izvode u dva odsječka koji počinju labelama **DOLAZAK** i **ODLAZAK**.

Odsječak **DOLAZAK** počinje petljom čekanja u kojoj se kontinuirano očitavaju stanja senzora, ispituje se samo senzor dolaska i ako nema vlaka (senzor je u ništici), nastavlja se s čekanjem u petlji. Kad se ustanovi da vlak nailazi, prekida se izvođenje petlje čekanja i uključi se jedno crveno svjetlo. Odmah nakon toga se inicijalizira RTC1 koji će generirati prekide IRQ svakih 500ms. U prekidnom potprogramu za IRQ izmjenjuju se svjetla i aktivira se zvono. Nakon toga se prelazi na odsječak **ODLAZAK**.

Odsječak **ODLAZAK** počinje petljom čekanja u kojoj se kontinuirano očitavaju stanja senzora, ispituje se samo senzor odlaska i ako vlak još nije otišao (senzor je u ništici), nastavlja se s čekanjem u petlji. Za vrijeme čekanja na odlazak vlaka, cijelo vrijeme se primaju prekidi IRQ i izmjenjuju su crvena svjetla te se aktivira zvono. Kad se ustanovi da je vlak otišao, prekida se izvođenje petlje čekanja i odmah nakon toga se sklopu RTC1 zabranjuje generiranje daljnjih prekida (čime se zaustavlja izmjena svjetala i aktiviranje zvona). Zatim se isključuju oba crvena svjetla. Na kraju se povećava brojač prolazaka vlakova, koji se čuva u memorijskoj lokaciji označenoj labelom **BROJAC**. Naposljetku se izvođenje vraća na odsječak **DOLAZAK** i cijeli postupak se ponavlja.

Nakon isteka jednog dana sklop RTC2 će generirati prekid FIQ. Prekidni potprogram za FIQ je jednostavan jer se u njemu mora samo ponovno inicijalizirati RTC2 i poslati na vrata A

vrijednost lokacije **BROJAC**. Također treba obrisati lokaciju **BROJAC** kako bi sljedećeg dana brojenje vlakova započelo od početka.

Primijetite da postoje dva prekidna potprograma, jedan za obradu prekida IRQ i drugi za obradu prekida FIQ. Potprogram za obradu prekida FIQ je moguće napisati direktno na adresi  $1C_{16}$ , dok se za obradu prekida IRQ mora s adrese  $18_{16}$  skočiti na potprogram koji je negdje u memoriji, jer su sljedeće memorijske lokacije rezervirane za prekidni potprogram za obradu prekida FIQ.

#### Rješenje:

```

ORG      0
B        GLAVNI

ORG      18
B        IRQ          ; skok u prekidni potprogram za IRQ (500 ms)

ORG      1C          ; prekidni potprogram za FIQ (1 dan)
LDR      R8, GPIO
LDR      R9, BROJAC
LDR      R10, RTC2

STR      R9, [R8]      ; slanje broja prolazaka na vrata A sklopa GPIO

MOV      R9, #0        ; reinicijaliziranje RTC2
STRB     R9, [R10, #0C] ; resetiranje brojača
STRB     R9, [R10, #8]  ; potvrda obrade prekida

STR      R9, BROJAC    ; resetiranje brojača prolazaka vlakova
SUBS     PC, LR, #4

GLAVNI   MOV      R13, #1<16 ; inicijalizacija stoga
        LDR      R0, GPIO    ; adrese sklopova
        LDR      R1, RTC1
        LDR      R5, RTC2

        MOV      R2, #%B 11111111 ; svih 8 bitova izlazni
        STR      R2, [R0, #08] ; upis u registar smjera od vrata A,
        MOV      R2, #%B 00001100 ; koriste se bitovi 2-6: 2-3 ulazni, 4-6 izlazni
        STR      R2, [R0, #0C] ; upis u registar smjera od vrata B

        LDR      R2, KONST2   ; učitavanje konstante za RTCMR2 (1 dan)
        STR      R2, [R5, #4]
        MOV      R2, #1       ; omogućavanje prekida u RTCCR2
        STR      R2, [R5, #10]

EINT     MRS      R2, CPSR
        BIC      R2, R2, #0C0 ; omogućavanje prekida FIQ i IRQ
        MSR      CPSR_c, R2

DOLAZAK  LDR      R2, [R0, #4] ; ispitivanje senzora dolaska vlaka
        TST      R2, #4       ; bit 2 je senzor dolaska vlaka
        BEQ      DOLAZAK     ; čekanje dolaska vlaka
        MOV      R2, #%B00100000 ; postavljanje različitih svjetala na...
        STR      R2, [R0, #4] ; ...bitovima 4 i 5 vratiju B (može 10 ili 01)

(nastavak na sljedećoj stranici)
        MOV      R2, #0       ; resetiranje brojila RTC1
        STR      R2, [R1, #0C]
        LDR      R2, KONST1   ; učitavanje konstante za RTCMR1

```

	STR	R2, [R1, #4]	
	MOV	R2, #1	; omogućavanje prekida u RTCCR1
	STR	R2, [R1, #10]	
ODLAZAK	LDR	R2, [R0, #4]	; ispitivanje senzora odlaska vlaka
	TST	R2, #8	; bit 3 je senzor odlaska vlaka
	BEQ	ODLAZAK	; čekanje odlaska vlaka
	MOV	R2, #0	; onemogućavanje daljnjih prekida u RTCCR1
	STR	R2, [R1, #10]	
	MOV	R2, #B00000000	; gašenje svjetala
	STR	R2, [R0, #4]	
	LDR	R2, BROJAC	
	ADD	R2, R2, #1	; povećavanje broja prolazaka vlakova
	STR	R2, BROJAC	
	B	DOLAZAK	
KONST1	DW	%D 5000	; mjerenje 0,5 sek (za RTC1)
KONST2	DW	%D 86400	; mjerenje 1 dana (za RTC2)
GPIO	DW	0FFFF0000	
RTC1	DW	0FFFF1000	
RTC2	DW	0FFFF2000	
BROJAC	DW	0	; brojač vlakova koji su prošli prijelazom
IRQ	STMFD	R13!, {R0, R1}	; spremanje konteksta
	LDR	R0, GPIO	
	LDRB	R1, [R0, #4]	; učitavanje trenutnog stanja svjetala
	EOR	R1, R1, #B 00110000	; zamjena svjetala
	ORR	R1, R1, #B 01000000	; dizanje impulsa za zvono
	STRB	R1, [R0, #4]	; slanje novih svjetala i impulsa na GPIO
	BIC	R1, R1, #B 01000000	; spuštanje impulsa za zvono
	STRB	R1, [R0, #4]	; slanje spuštenog impulsa na GPIO
	LDR	R0, RTC	
	MOV	R1, #0	
	STRB	R1, [R0, #8]	; potvrđivanje primitka prekida
	STRB	R1, [R0, #0C]	; resetiranje brojača
	LDMFD	R13!, {R0, R1}	; obnova konteksta
	SUBS	PC, LR, #4	

### Komentar rješenja:

U ovom zadatku je potrebno više puta zapisivati vrijednosti samo u bitove 4-6 na vratima B GPIO sklopa, a bitovi 2-3 su ulazni. Ostali bitovi se ne koriste i prilikom inicijalizacije im možemo zadati bilo kakav smjer. Prilikom zapisivanja vrijednosti, na nekoristene bitove možemo poslati bilo koju vrijednost (0 ili 1). Naredbom **EOR R1,R1,#B 00110000** komplementiraju se bitovi 4 i 5 koji označavaju trenutno stanje svjetla da bi se svjetlo koje je bilo upaljeno ugasio, a ono koje je bilo ugašeno upalilo kako je zadano u zadatku.

Za omogućavanje prekida IRQ i FIQ potrebno je izbrisati bitove 6 i 7 u registru CPSR (naredbom **BIC R0,R0,#0C0**).



### 3.1.10. Pješački semafor (ZI10)

Riješen: DA    Težina: ★★★

Za procesor ARM napisati program, potprograme i prekidne potprograme koji rješavaju problem semafora pješačkog prijelaza uporabom sklopova RTC i GPIO. Adrese sklopova odaberite proizvoljno. Na RTC je spojen signal frekvencije 10Hz. Glavni program izvodi praznu petlju.

Uporabom sklopa GPIO napisati 3 potprograma za upravljanje svjetlima vozačkog i pješačkog semafora na pješačkom prijelazu. Na vrata B sklopa GPIO spojena je redom signalizacija svjetla semafora kao u tablici. Slanje 0 na pin označava gašenje svjetla, a 1 paljenje.

Vrata B sklopa GPIO	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
Spojena signalizacija	Zeleno za pješake	Crveno za pješake	Nije spojeno	Nije spojeno	Nije spojeno	Zeleno za vozače	Žuto za vozače	Crveno za vozače

Treba napisati sljedeće potprograme:

- Potprogram ZELENO – zeleno svjetlo vozačima, crveno svjetlo pješacima
- Potprogram ZUTO – žuto svjetlo vozačima, crveno svjetlo pješacima
- Potprogram CRVENO – crveno svjetlo vozačima, zeleno svjetlo pješacima

Sklop RTC, spojen na IRQ, generira prekid nakon proteka perioda vremena za određeno svjetlo. U prekidnom potprogramu treba provjeravati u kojem stanju se semafor nalazio, prijeći u sljedeće stanje i postaviti novu kombinaciju svjetala te novi period čekanja. Trenutno stanje treba svaki put nakon promjene pohraniti na memorijsku lokaciju STANJE. Postoje 4 stanja koja se mijenjaju slijedno, a nakon što semafor dođe u stanje 3, sljedeće stanje je ponovno 0.

STANJE	SLJEDEĆE STANJE	AKCIJA	KORIŠTENI POTPROGRAM ZA SVJETLA
0	1	3 minute gori zeleno svjetlo za vozače, a crveno za pješake	potprogram ZELENO
1	2	5 sekundi gori žuto svjetlo za vozače, a crveno za pješake	potprogram ZUTO
2	3	30 sekundi gori crveno svjetlo za vozače, a zeleno za pješake	potprogram CRVENO
3	0	3 sekunde gori žuto svjetlo za vozače, a crveno za pješake	potprogram ZUTO

Na liniju brzog prekida (FIQ) spojen je gumb za pješake na semaforu, kojim pješaci „traže brži prelazak ulice“. Potrebno je napisati i prekidni potprogram kojim se trenutno prekida izvođenje ciklusa rada semafora i prelazi u STANJE = 1 (žuto svjetlo za vozače, a crveno za pješake) koje traje 5 sekundi. Ciklus se dalje normalno nastavlja kao u prethodnom zadatku (crveno vozači, zeleno pješaci). Pretpostavite da gumb radi samo kad je stanje semafora 0 (zeleno vozači, crveno pješaci).

**Prijedlog rješenja:**

Tri potprograma, ZELENO, ZUTO i CRVENO su vrlo jednostavni i međusobno slični. U njima se na GPIO šalje odgovarajuća kombinacija ništica i jedinica koja pali i gasi zadana svjetla na semaforima i to prema rasporedu spajanja tih svjetala na bitove GPIO-a (što je zadano prvom tablicom). Na primjer, za potprogram ZUTO je zadano da upali žuto svjetlo za vozače i crveno za pješake. Prema prvoj tablici, ova dva svjetla spojena su na bitove 1 (žuto za vozače) i 6 (crveno za pješake). Znači da na ta dva bita na sklopu GPIO treba postaviti jedinice, a na sve ostale bitove ništice (tj. ostala svjetla se gase) pa se GPIO-u šalje broj 01000010<sub>2</sub>.

U tekstu zadatka je već dana ideja i naputak kako se zadatak najlakše rješava. Trenutno stanje pamti se u memorijskoj lokaciji **STANJE** koja će ciklički poprimati vrijednosti 0, 1, 2, 3, pa onda opet 0, 1, 2, 3, itd. Svako stanje ima zadano trajanje. Čekanje na istek trajanja ostvaren je samo jednim sklopom RTC, a različita trajanja su ostvarena upisom različitih konstanti u **RTCMR**. RTC se inicijalizira tako da generira prekid kad trajanje trenutnog stanja istekne. U prekidnom potprogramu se prelazi u novo stanje tako da se ispituje lokacija **STANJE** i ovisno o njenoj vrijednosti se određuje novo stanje (zapisuje se u registar **R2**) i vremenska konstanta za novo stanje (zapisuje se u registar **R3**). Također se, ovisno o novom stanju, poziva jedan od potprograma ZELENO, ZUTO ili CRVENO koji će upaliti svjetla prema novom stanju. Na kraju prekidnog potprograma (na labeli **VAN**) upisuje se novo stanje iz **R2** u lokaciju **STANJE** i ponovno se inicijalizira RTC s upisom nove konstante iz **R3** u **RTCMR**.

Glavni program postavlja početno stanje 0 upisom te vrijednosti u lokaciju **STANJE**, paljenjem semafora za stanje 0 (pozivom potprograma ZELENO). Zatim se inicijalizira RTC i to tako da generira prekid nakon 3 minute, jer je to zadano trajanje stanja 0. Nakon toga se dozvoljavaju svi prekidi i na kraju se izvodi beskonačna prazna petlja. Sve promjene stanja i upravljanja semaforima izvodit će se u prekidnom potprogramu, kako je prethodno opisano. Uočite da GPIO ne treba inicijalizirati jer podrazumijevani smjer vrata B već odgovara potrebnom izlaznom smjeru za upravljanje svjetlima.

**Rješenje:**

```

ORG    0
B      GLAVNI          ; skok na glavni program

ORG    18
B      PREKID          ; skok na obradu prekida FIQ

ORG    1C
FIQ    STMFD R13!, {R0, R1, R14} ; pohranjivanje konteksta

LDR    R0, RTC          ; adresa RTC-a
BL     ZUTO             ; poziv potprograma ZUTO
MOV    R1, #D 50        ; spremanje konstante
STR    R1, [R0, #4]     ; u RTCMR
MOV    R1, #1           ; STANJE = 1
STR    R1, STANJE       ; spremanje na lokaciju STANJE
MOV    R1, #0           ; brisanje RTCLR
STR    R1, [R0, #0C]

LDMFD  R13!, {R0, R1, R14} ; obnova konteksta
SUBS   PC, LR, #4       ; povratak

```

(nastavak na sljedećoj stranici)

```

PREKID    STMFD R13!, {R0, R1, R2, R3, R14} ; pohranjivanje konteksta
          LDR     R0, RTC                    ; adresa RTC
          LDR     R1, STANJE                 ; učitavanje stanja

IZ_0_U_1  CMP     R1, #0                    ; ako je stanje bilo 0 (zeleno)...
          MOVEQ   R2, #1                    ; ... onda sad prelazimo u stanje 1 (prvo žuto)
          BLEQ    ZUTO                      ; pozivanje potprograma ZUTO
          LDREQ   R3, KONST1                ; učitavanje konstante 50
          BEQ     VAN

IZ_1_U_2  CMP     R1, #1                    ; ako je stanje bilo 1 (prvo žuto)
          MOVEQ   R2, #2                    ; ... onda sad prelazimo u stanje 2 (crveno)
          BLEQ    CRVENO                    ; pozivanje potprograma CRVENO
          LDREQ   R3, KONST2                ; učitavanje konstante 300
          BEQ     VAN

IZ_2_U_3  CMP     R1, #2                    ; ako je stanje bilo 2 (crveno)
          MOVEQ   R2, #3                    ; ... onda sad prelazimo u stanje 3 (drugo žuto)
          BLEQ    ZUTO                      ; pozivanje potprograma ZUTO
          LDREQ   R3, KONST3                ; učitavanje konstante 30
          BEQ     VAN

IZ_3_U_0  CMP     R1, #3                    ; ako je stanje bilo 3 (drugo žuto)
          MOVEQ   R2, #0                    ; ... onda se vraćamo u stanje 0 (zeleno)
          BLEQ    ZELEN0                    ; pozivanje potprograma ZELEN0
          LDREQ   R3, KONST3                ; učitavanje konstante 1800

VAN        STR     R2, STANJE                ; pohranjivanje novog stanja
          STR     R3, [R0, #4]               ; konstanta u RTCMR
          STR     R3, [R0, #8]               ; brisanje RTCINTR
          MOV     R3, #0
          STR     R3, [R0, #0C]              ; brisanje RTCLR
          LDMFD   R13!, {R0, R1, R2, R3, R14}; obnova konteksta
          SUBS    PC, LR, #4                 ; povratak

ZELEN0     STMFD   R13!, {R0, R1}           ; pohranjivanje konteksta
          LDR     R0, GPIO                   ; R0 = GPIO
          MOV     R1, %%B 01000100          ; zeleno vozači, crveno pješaci
          STR     R1, [R0, #4]               ; slanje na GPIO B
          LDMFD   R13!, {R0, R1}           ; obnova konteksta
          MOV     PC, LR                     ; povratak

ZUTO       STMFD   R13!, {R0, R1}           ; pohranjivanje konteksta
          LDR     R0, GPIO                   ; R0 = GPIO
          MOV     R1, %%B 01000010          ; žuto vozači, crveno pješaci
          STR     R1, [R0, #4]               ; slanje na GPIO B
          LDMFD   R13!, {R0, R1}           ; obnova konteksta
          MOV     PC, LR                     ; povratak

CRVENO     STMFD   R13!, {R0, R1}           ; pohranjivanje konteksta
          LDR     R0, GPIO                   ; R0 = GPIO
          MOV     R1, %%B 10000001          ; crveno vozači, zeleno pješaci
          STR     R1, [R0, #4]               ; slanje na GPIO B
          LDMFD   R13!, {R0, R1}           ; obnova konteksta
          MOV     PC, LR                     ; povratak

```

(nastavak na sljedećoj stranici)

GLAVNI	MOV	SP, #1<16	; inicijalizacija stoga
	MOV	R1, #0	; inicijalno STANJE = 0
	STR	R1, STANJE	; pohranjivanje u memoriju
	BL	ZELENO	; početno pozivamo potprogram ZELENO
	LDR	R0, RTC	; adresa RTC
	LDR	R1, KONST0	; inicijalno čekanje KONST0 = 1800
	STR	R1, [R0, #4]	; konstanta u RTCMR
	MOV	R1, #1	
	STR	R1, [R0, #10]	; RTCCR = 1, prekid
	MRS	R0, CPSR	
	BIC	R0, R0, #0C0	; omogućavanje IRQ i FIQ (40 i 80)
	MSR	CPSR_c, R0	
PET	B	PET	; prazna petlja
RTC	DW	0FFFFFF00	; adresa RTC
GPIO	DW	0FFFFFFE00	; adresa GPIO
STANJE	DW	0	; stanje semafora
KONST0	DW	%D 1800	; konstanta za čekanje od 3 minute
KONST1	DW	%D 50	; konstanta za čekanje od 5 sekundi
KONST2	DW	%D 300	; konstanta za čekanje od 30 sekundi
KONST3	DW	%D 30	; konstanta za čekanje od 3 sekundi

### Komentar rješenja:

Potprogrami **ZELENO**, **ZUTO** i **CRVENO** su vrlo slični (pale različita svjetla), što znači da bi bolja praksa bilo napisati jedan potprogram za paljenje svjetla kojem bi se boja svjetla prenosila kao parametar. Prekidni potprogram za FIQ poziva potprogram **ZUTO** i prelazi u stanje 1.

U rješenju zadatka je pretpostavljeno da vanjskoj jedinici na koju je spojen gumb za pješake na semaforu nije potrebno dojavljivati prihvrat prekida.

### 3.1.11. Mikrovalna pećnica upravljana sklopovima GPIO i RTC (ZI11)

Riješen: DA    Težina: ★★★

Procesor ARM pomoću sklopa GPIO upravlja pametnom mikrovalnom pećnicom. Ovisno o vrsti i masi hrane, pećnica sama određuje trajanje pečenja. Pinovi ulaznih vrata A sklopa GPIO označavaju sljedeće:

- bitovi [3:0] – masa hrane zaokružena na kvante od 100 g (npr. broj 4 = 400 g)
- bitovi [7:4] – vrsta hrane (od 0 do 15).

Pinovi vrata B sklopa GPIO označavaju sljedeće:

- bit 0 – ulazni signal READY, kojim pećnica dojavljuje procesoru da su podaci o masi i vrsti hrane uneseni i da je pećnica spremna (slanjem vrijednosti 1)
- bit 1 – izlazni signal STROBE, kojim procesor dojavljuje pećnici da je primio podatke (slanjem impulsa 1). Tada će pećnica automatski deaktivirati signal READY

- bit 2 – izlazni signal za uključivanje pećnice (vrijednost 1) ili isključivanje pećnice (vrijednost 0)
- ostali bitovi se ne koriste, ali ih je potrebno postaviti ulaznima.

Svaka od 16 vrsta hrane ima svoje specifično trajanje pečenja za količinu od 100 grama, a trajanje je zadano u minutama. Ukupno trajanje pečenja dobiva se množenjem mase hrane sa specifičnim trajanjem rada za tu vrstu hrane. Specifična trajanja rada dana su u tablici velikoj 16 bajtova i zapisanoj u memoriji na adresi  $1000_{16}$ . Svaki bajt u tablici označava specifično trajanje rada za pojedinu vrstu hrane, pri čemu je redoslijed hrane u tablici od vrste 0 na adresi  $1000_{16}$  do vrste 15 na adresi  $100F_{16}$ .

U glavnom programu, procesor čeka da pećnica dojaví pomoću READY da je spremna za pečenje. Tada procesor pročíta podatke i impulsom na STROBE dojavljuje da je primio podatke i nakon toga uključuje pećnicu. Također pokreće sklop RTC, koji mjeri vrijeme rada mikrovalne pećnice. Po isteku potrebnog vremena rada za navedenu vrstu i masu hrane, RTC koji je spojen na signal FIQ, postavlja zahtjev za prekid. U prekidnom programu treba isključiti pećnicu, te se vratiti u glavni program i ponovno čekati da pećnica bude spremna. Adrese sklopova odaberite proizvoljno. Na RTC je spojen signal frekvencije 10Hz.

Memorijska lokacija	Vrijednost	Opis
$1000_{16}$	$1_{16}$	100 g vrste hrane 0000 kuha se 1 min
$1001_{16}$	$5_{16}$	100 g vrste hrane 0001 kuha se 5 min
...	...	...
$100F_{16}$	$3_{16}$	100 g vrste hrane 000F kuha se 3 min

### Prijedlog rješenja:

Iako na prvi pogled ovaj zadatak može djelovati komplicirano, zapravo je njegovo rješenje relativno jednostavno, a ideja rješavanja je već ponuđena u samom tekstu zadatka.

Glavni program treba u petlji čekati postavljanje signala READY. Nakon što se utvrdi da je READY postavljen, petlja čekanja završava s radom i s vrata A se čítaju masa i vrsta hrane. Iz ova dva podatka se množenjem odredi trajanje pečenja u minutama. Nakon toga inicijalizira se RTC da generira prekid nakon završetka pečenja, a glavni program uključi pećnicu i aktivira signal STROBE. Nakon toga, glavni program opet se vraća na početak i čeka postavljanje signala READY, ali do toga neće doći sve dok se trenutno pečenje ne završi i dok netko u budućnosti ne pokrene pečenje iznova. Program zanemaruje razne mogućnosti neispravnog rukovanja pećnicom (npr. kad bi za vrijeme pečenja osoba ponovno pokušala pokrenuti pečenje). Prekidni program jednostavno isključuje pećnicu i onemogućuje daljnje generiranje prekida RTC-u (generiranje prekida će se opet omogućiti tek kad započne novo pečenje).

Trajanje pečenja izračunava se množenjem mase hrane očitane iz niža 4 bita vrata A i specifičnog trajanja koje se pročíta iz tablice. Iz tablice u kojoj se u bajtovima pamte specifična trajanja za 16 vrsta hrane, najlakše je očitati vrijednost tako da se početna adresa tablice (zadano je da je to adresa  $1000_{16}$ ) stavi u jedan registar, a vrsta hrane (očitana s viša 4 bita vrata A) se stavi u drugi registar. U naredbi **LDRB**, kojom čítamo potrebno specifično trajanje iz tablice, adresa se zada tako da se prvi registar upotrijebi kao bazni, a drugi kao

odmak. Nakon množenja mase i specifičnog trajanja dobiveno trajanje pečenja izraženo je u minutama pa ga još treba pomnožiti sa 600 da bi se dobila vremenska konstanta za RTC u sekundama (u 1 minuti ima 600 impulsa signala frekvencije 10 Hz).

**Rješenje:**

```

ORG      0
MOV      SP, #10<12      ; inicijalizacija stoga
B        GLAVNI

FIQ
ORG      1C              ; FIQ
LDR      R8, GPIO        ; učitavanje adresa
LDR      R9, RTC

MOV      R10, #0
STR      R10, [R9, #8]    ; prihvaćanje prekida RTC
STR      R10, [R9, #10]   ; onemogućiti generiranje prekida RTC-u
STR      R10, [R8, #4]    ; isključivanje pećnice
SUBS     PC, LR, #4       ; povratak iz prekidnog potprograma

GLAVNI
MRS      R0, CPSR
BIC      R0, R0, #40      ; omogućavanje FIQ
MSR      CPSR_c, R0

LDR      R6, RTC
LDR      R0, GPIO

; GPIO inicijalizacija (vrata A već imaju dobar smjer - ulazni)
MOV      R1, %%B 11111001; 0. bit ulazni, 1. i 2. izlazni, ostali ulazni
STR      R1, [R0, #0C]    ; registar smjera vrata B

PETLJA
LDR      R1, [R0, #4]     ; čitanje s vrata B
ANDS     R1, R1, #1       ; maskiranje zadnjeg bita
BEQ      PETLJA          ; petlja čekanja na signal READY

LDR      R2, [R0]         ; čitanje s vrata A mase i vrste hrane
AND      R3, R2, #0F      ; R3 - masa hrane (niža 4 bita)
AND      R4, R2, #0F0     ; R4 - vrsta hrane (viša 4 bita)
MOV      R4, R4, LSR #4   ; gornja 4 bita s viših na niže pozicije

MOV      R2, #10<8       ; adresa 1000 je početak tablice
LDRB     R5, [R2, R4]     ; čitaj specifično trajanje iz točnog retka
LDR      R7, SESTO       ; 600 = broj signala od 10 Hz u 1 minuti
MUL      R5, R5, R3       ; specifično trajanje * masa hrane
MUL      R5, R5, R7       ; trajanje pečenja * 600
STR      R5, [R6, #4]     ; spremanje konstante u RTCMR

; inicijalizacija RTC
MOV      R5, #1
STR      R5, [R6, #10]    ; omogućavanje prekida
MOV      R5, #0
STR      R5, [R6, #0C]    ; resetiranje brojila

; pomoću GPIO-a postaviti impuls na STROBE i uključiti pećnicu
MOV      R1, #2           ; impuls na STROBE, prvo STROBE = 1 ...
STR      R1, [R0, #4]

```

(nastavak na sljedećoj stranici)

	MOV	R1, #0	; ... zatim STROBE = 0
	STR	R1, [R0, #4]	
	MOV	R1, #B 100	; uključuje se pećnica
	STR	R1, [R0, #4]	
	B	PETLJA	
SESTO	DW	%D 600	; konstanta za RTC za čekanje od 1 minute
GPIO	DW	0FFFF1000	; adresa GPIO
RTC	DW	0FFFF2000	; adresa RTC

**Komentar rješenja:**

Primijetite da je prije novog ciklusa uključivanja pećnice potrebno signal STROBE vratiti u „0“ jer bi inače signal READY ostao neaktivan i program bi se zablokirao u petlji čekanja na READY. Potrebno je primijetiti i da se ulaznim signalom READY ne može upravljati, već se njega može samo čitati (odnosno provjeravati njegovu aktivnost). Upis bilo koje vrijednosti na ulazni priključak sklopa GPIO se zanemaruje jer GPIO ne utječe na stanje ulaznog priključka. Dodatno, par naredaba za vraćanje signala STROBE u 0 (**MOV R1, #0** i **STR R1, [R0, #4]**) nije nužan jer se signal vraća u 0 pri izvođenju naredaba za uključivanje pećnice (**MOV R1, #B100** i **STR R1, [R0, #4]**).

**3.1.12. Upravljanje inkubatorom pomoću sklopova GPIO i RTC (ZI12)****Riješen: DA    Težina: ★★★**

Računalni sustav inkubatora za patkice sastoji se od procesora ARM te sklopova RTC (spojen na signal FIQ) i GPIO (na koji je spojen termometar opisan na predavanjima):

- vrata B, bitovi [5:0]: iznos temperature, 6-bitni NBC
- vrata B, bit 6: signal (aktivan visoko) kojim termometar dojavljuje GPIO-u da je postavljena nova temperatura vrata B
- vrata B, bit 7: signal (aktivan visoko) kojim GPIO dojavljuje termometru da je pročitao temperaturu
- vrata A, bit 0: izlazni bit kojim se uključuje grijanje (1 uključuje, a 0 isključuje)
- vrata A, bit 1: izlazni bit kojim se uključuje hlađenje (1 uključuje, a 0 isključuje)

Napisati program koji upravlja radom inkubatora i treba održavati željenu temperaturu. Temperaturu treba regulirati svakih 60 sekundi (na ulaz RTC-a spojen je signal od 1 Hz). Na početku željena temperatura treba biti 40 stupnjeva. Svaki treći dan (3 dana = 4320 minuta) treba smanjivati željenu temperaturu za 1 stupanj. Nakon 30 dana, treba zaustaviti rad programa.

**Prijedlog rješenja:**

Glavni program je jednostavan jer se u njemu samo inicijaliziraju zadani smjerovi na sklopu GPIO i inicijalizira se RTC tako da svake minute generira prekid FIQ. Učestalost prekida neće se mijenjati tijekom 30-dnevnog razdoblja, jer je temperaturu potrebno regulirati svake minute. U glavnom programu još treba dozvoliti prihvaćanje prekida i pokrenuti beskonačnu petlju za vrijeme čijeg izvođenja se svake minute dešavaju prekidi.

Sve aktivnosti zadane zadatkom odvijaju se u prekidnom potprogramu. Na početku se ponovno inicijalizira RTC, jer on stalno mora mjeriti razdoblja od jedne minute. Prekidni potprogram koristi tri memorijske lokacije koje imaju ulogu „globalnih“ varijabli koje čuvaju određene vrijednosti između dva izvođenja prekidnog potprograma. Prva od ovih varijabli je **ZELJENA** koja pamti trenutnu željenu temperaturu u inkubatoru, a početna vrijednost joj je 40. Varijabla se smanjuje za jedan svaki treći dan. Druge dvije varijable služe za utvrđivanje trenutka kad su prošla tri dana i trenutka kad je prošlo svih 30 dana. Varijabla **MINUTEU3** povećava se u svakom prekidu, tj. svake minute i ona prebraja broj minuta u jednom trodnevnom razdoblju. Kad varijabla dosegne vrijednost 4320 (3 dana \* 24 sata \* 60 minuta), zna se da su istekla točno tri dana u sekundama. Tada treba varijablu **ZELJENA** smanjiti za jedan (čime se željena temperatura smanjuje za jedan stupanj) i vratiti varijablu **MINUTEU3** na ništicu (tj. na početnu vrijednost) kako bi se započelo sa sljedećim trodnevnim razdobljem. Također treba povećati varijablu **TRIDANA** u kojoj se prebraja broj do tada proteklih trodnevlja. Kad se poveća varijable **TRIDANA**, provjeri se je li dosegla vrijednost 10. Ako je, zna se da je proteklo 10 trodnevlja (odnosno 30 dana) i tada treba zaustaviti program i generiranje prekida na sklopu RTC.

Bez obzira na to koliko dana je prošlo (osim kad prođe 30 dana), u prekidnom potprogramu treba očitati termometar. Prije čitanja, treba provjeravati i čekati na signal na bitu 6 na vratima B. Kad signal postane aktivan (visoko stanje), treba dizanjem i spuštanjem signala na bitu 7 vrata B dojaviti termometru da može početi s novim mjerenjem.

Temperatura inkubatora regulira se usporedbom trenutno očitane temperature (sa nižih 6 bitova vrata B) i trenutno željene temperature (iz varijable **ZELJENA**). Ako su temperature jednake, treba isključiti grijanje i hlađenje. Ako je očitana temperatura premalena, treba uključiti grijanje. Ako je očitana temperatura prevelika, treba uključiti hlađenje. Time regulacija završava i izlazi se iz prekidnog potprograma.

Rješenje:			
	ORG	0	
	B	GLAVNI	; skok na glavni program
PREKID	ORG	1C	; prekidni potprogram - FIQ
	STMFD	R13!, {R0,R1,R4,R5,R6,R7}	; spremanje konteksta
	LDR	R0, GPIO	; učitavanje adresa RTC i GPIO
	LDR	R1, RTC	
			; reinicijalizacija RTC-a:
	MOV	R9, #0	
	STR	R9, [R1,#0C]	; resetiranje brojača
	STR	R9, [R1,#8]	; dojava prihvata prekida
VRIJEME			; provjera jesu li prošla 3 dana i je li prošlo 30 dana
	LDR	R5, MINUTEU3	; povećati brojač trenutno proteklih minuta
	ADD	R5, R5, #1	
	STR	R5, MINUTEU3	
	LDR	R7, MIN4320	; učitavanje broja minuta u 3 dana
	CMP	R5, R7	; jesu li prošla 3 dana?
	BNE	CITAJ	; ako nisu -> preskočiti na čitanje i regulaciju
(nastavak na sljedećoj stranici)			



```

; prošla su tri dana
TRI      LDR      R4, ZELJENA      ; smanjiti željenu temp. za 1
        SUB      R4, R4, #1
        STR      R4, ZELJENA

        MOV      R5, #0            ; vratiti brojač minuta na 0
        STR      R5, MINUTEU3

MJESEC   LDR      R6, TRIDANA      ; povećati brojač prošlih 3-dnevnih perioda
        ADD      R6, R6, #1
        STR      R6, TRIDANA
        CMP      R6, #%D 10       ; je li prošlo 30 dana, tj. 10 perioda od 3 dana
        BEQ      KRAJ             ; ako je prošlo 30 dana -> skok na kraj

; čitanje podatka s vrata B
CITAJ    LDR      R10, [R0,#4]     ; čekanje na novu temperaturu
        ANDS     R12, R10, #40     ; 40 = %B 100 0000; bit 6 je signal nove temp.
        BEQ      CITAJ            ; čekanje dok je signal u niskom stanju

        MOV      R12, #80         ; 80 = %B 1000 0000; bit 7 u visoko
        STR      R12, [R0,#4]     ; slanje na vrata B
        MOV      R12, #00         ; 00 = %B 0000 0000; bit 7 natrag u nisko
        STR      R12, [R0,#4]     ; slanje na vrata B

        AND      R10, R10, #3F    ; izdvajanje iznosa temperature (nižih 6 bita)

; uključivanje grijanja/hlađenja na temelju željene i trenutne temper.
REGULIRAJ LDR      R4, ZELJENA      ; učitavanje željene temperature iz memorije
        CMP      R10, R4          ; usporedba trenutne sa željenom temperaturom:

        MOVHI    R10, #2          ; ako je pretoplo -> hlađenje = 10
        MOVMI    R10, #1          ; ako je prehladno -> grijanje = 01
        MOVEQ    R10, #0          ; ako je ista temperatura -> isključiti sve = 00
        STR      R10, [R0]        ; spremanje na vrata A

VAN      LDMFD    R13!, {R0,R1,R4,R5,R6,R7} ; obnova konteksta
        SUBS     PC, LR, #4       ; povratak iz prekidnog potprograma

KRAJ     LDMFD    R13!, {R0,R1,R4,R5,R6,R7} ; obnova konteksta
        MOV      R9, #0           ; zaustaviti generiranje prekida na RTC-u
        STR      R9, [R1, #10]
        HALT                    ; zaustaviti procesor

GLAVNI   MOV      R13,#10<12      ; inicijalizacija stoga
        LDR      R0, GPIO         ; bazna adresa sklopa GPIO
        LDR      R1, RTC          ; bazna adresa sklopa RTC

        MOV      R2, #3           ; 3 = %B11, bitovi 0 i 1 su izlazni
        STR      R2, [R0,#8]      ; registar smjera vrata A
        MOV      R2, #7F          ; 7F = %B01111111, bit 7 izlazni, ostali ulazni
        STR      R2, [R0,#0C]     ; registar smjera vrata B
        MOV      R2, #1           ; omogućavanje prekida
        STR      R2, [R1,#10]     ; upis u upravljački registar (RTCCR)
        MOV      R2, #%D60        ; konstanta = 60 sekundi
        STR      R2, [R1,#4]      ; upis u RTCMR

```

(nastavak na sljedećoj stranici)

```

        MRS      R2, CPSR
        BIC      R2, R2, #40      ; omogućavanje prekida FIQ
        MSR      CPSR_c, R2

PETLJA  B        PETLJA          ; prazna petlja

        ORG      300
GPIO    DW      0FFFFFF000      ; adresa GPIO
RTC     DW      0FFFFFF2000      ; adresa RTC

MIN4320 DW      %D 4320          ; konstanta: broj minuta u 3 dana

        ; varijable
ZELJENA DW      %D 40            ; željena temperatura, na početku 40
MINUTEU3 DW      0              ; trenutno prošle minute u 3-dnevnom periodu
TRIDANA DW      0              ; trenutno protekli 3-dnevni periodi

```

### Komentar rješenja:

Kraj razdoblja od 30 dana mogao se provjeriti i drugačije. Umjesto brojenja trodnevnih razdoblja, mogao se dodatnim brojačem prebrajati ukupan broj minuta i onda bi se kraj 30-dnevnog razdoblja prepoznao kad taj brojač dosegne vrijednost 43200. Kako bi riješili zadatak da je na raspolaganju bio dodatni RTC spojen na IRQ? Kako bi riješili zadatak da su postojala dva RTC-a, ali da su oba spojena na FIQ?

### 3.1.13. Očitavanje i ispisivanje temperature

Riješen: NE    Težina: ★★★

Napisati program koji kontinuirano očitava temperaturu ispitujući temperaturni sklop spojen na vrata B sklopa GPIO. U ovisnosti o vrijednosti pročitane temperature potrebno je ispisati poruku na 8-znakovni LCD prikaznik spojen na vrata A istog sklopa GPIO. Temperaturni sklop i LCD-prikaznik opisani su na predavanjima.

Priključci temperaturnog sklopa spojeni na ulazne bitove 0 do 5 služe za očitavanje temperature u rasponu od -20°C do 43°C (očitanja vrijednost 0 odgovara temperaturi -20°C, a vrijednost 63 odgovara temperaturi 43°C).

Ako pročitana temperatura iznosi više od 25°C potrebno je na LCD-prikazniku prikazati poruku "V", ako iznosi manje od 15°C prikazati poruku "H", a ako je između te dvije vrijednosti na prikazniku ne smije biti prikazana poruka. Prilikom prikaza znaka, "V" ili "H" se mora prikazati na desnoj poziciji, a preostalih sedam lijevih pozicija na prikazniku je prazno. ASCII kôd znaka "V" je 56<sub>16</sub>, a znaka "H" je 48<sub>16</sub>.

Primjer prikaza:

Temperatura iznad 25°C: " \_ \_ \_ \_ \_ V",    temperatura ispod 15°C: " \_ \_ \_ \_ \_ H",  
inače: " \_ \_ \_ \_ \_ ".

### 3.1.14. Rampa za parkiralište

Riješen: NE    Težina: ★★★

U računalnom sustavu nalazi se procesor ARM te sklopovi GPIO i RTC (koji treba postavljati prekid). Napisati program koji upravlja radom rampe na parkiralištu za automobile. Na sklop

GPIO spojena su dva senzora (jedan za dolazak i drugi za odlazak automobila) i sustav za podizanje i spuštanje rampe. Sustav mora zadovoljavati sljedeće kriterije:

- ako automobil nije pred rampom, rampa mora biti spuštena,
- automobil koji dođe pred rampu aktivirat će prvi senzor; tada treba podići rampu,
- rampa treba biti podignuta barem 5 sekundi,
- nakon što prođe tih 5 sekundi, potrebno je uzastopno provjeravati stanje drugog senzora sve dok ne postane neaktivan. Taj je senzor aktivan sve dok se automobil nalazi pod rampom,
- nakon što drugi senzor postane neaktivan, treba spustiti rampu.

Načine spajanja senzora i rampe na GPIO te sve ostalo što nije zadano u zadatku odaberite sami.

Zanemarite rubne slučajeve, kao na primjer mogućnost dolaska novog automobila dok je rampa još podignuta itd.

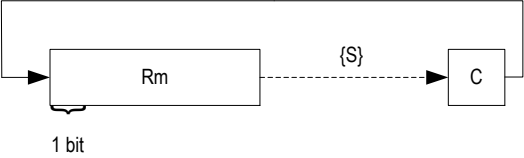
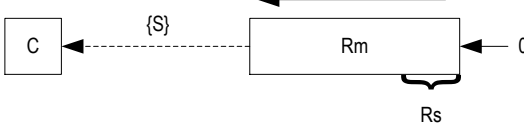
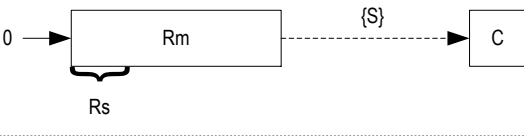
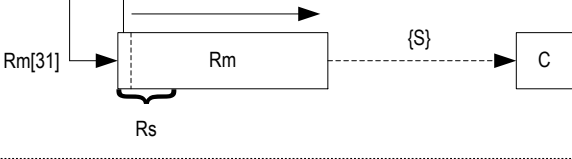
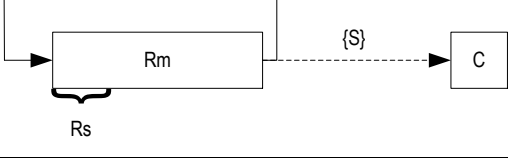
## 4. Prilog - Tablice programiranja procesora ARM 7

### Polje uvjeta {cond}

Uvjet	Značenje (engl.)	Način ispitivanja zastavica
EQ	Equal	Z
NE	Not equal	!Z
CS/HS	Carry Set / Unsigned higher or same	C
CC/LO	Carry Clear / Unsigned lower	!C
MI	Negative	N
PL	Positive or zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	C AND !Z
LS	Unsigned lower or same	!C OR Z
GE	Signed greater than or equal	N == V
LT	Signed less than	N != V
GT	Signed greater than	!Z AND N == V
LE	Signed less than or equal	Z OR N != V
AL	Always (normally omitted)	-

### Polje operand2 <Oprnd2>

Operand	Adresiranje	Objašnjenje
Rm	Registarsko	
#<immed_8r>	Neposredna vrijednost (sastoji se od dva polja <immed_8> i <rotate_imm> od kojih se dobije 32-bitna konstanta)	8 bitova se koristi za neposrednu vrijednost <immed_8>, 4 bita <rotate_imm> definiraju broj rotacija udesno 8-bitne neposredne vrijednosti: <immed_8> rotirano udesno za (2*<rotate_imm>)
Rm, LSL #<immed_5>	Registarsko s neposrednim logičkim pomakom u lijevo (LSL=ASL)	<p>Dozvoljeni pomaci 0-31</p>
Rm, LSR #<immed_5>	Registarsko s neposrednim logičkim pomakom u desno	<p>Dozvoljeni pomaci 1-32</p>
Rm, ASR #<immed_5>	Registarsko s neposrednim aritmetičkim pomakom u desno	<p>Dozvoljeni pomaci 1-32</p>
Rm, ROR #<immed_5>	Registarsko s neposrednim rotiranjem u desno	<p>Dozvoljeni pomaci 1-31</p>

<b>Rm, RRX</b>	Registarsko s proširenim rotiranjem u desno	
<b>Rm, LSL Rs</b>	Registarsko s registarskim logičkim pomakom u lijevo (LSL=ASL)	
<b>Rm, LSR Rs</b>	Registarsko s registarskim logičkim pomakom u desno	
<b>Rm, ASR Rs</b>	Registarsko s registarskim aritmetičkim pomakom u desno	
<b>Rm, ROR Rs</b>	Registarsko s registarskim rotiranjem u desno	

**Način adresiranja <a\_mode2>**

Predindeksiranje/ Adres. s odmakom	Neposredni odmak	[Rn, #+/-<immed_12>]{!}	
	Nulti odmak	[Rn]	Ekvivalentno sa [Rn,#0]
	Registarski odmak	[Rn, +/-Rm]{!}	
	Skalirani registarski odmak	[Rn, +/-Rm, LSL #<immed_5>]{!}	Dozvoljeni pomaci 0-31
		[Rn, +/-Rm, LSR #<immed_5>]{!}	Dozvoljeni pomaci 1-32
		[Rn, +/-Rm, ASR #<immed_5>]{!}	Dozvoljeni pomaci 1-32
[Rn, +/-Rm, ROR #<immed_5>]{!}		Dozvoljeni pomaci 1-31	
		[Rn, +/-Rm, RRX]{!}	
Postindeksiranje	Neposredni odmak	[Rn], #+/-<immed_12>	
	Registarski odmak	[Rn], +/-Rm	
	Skalirani registarski odmak	[Rn], +/-Rm, LSL #<immed_5>	Dozvoljeni pomaci 0-31
		[Rn], +/-Rm, LSR #<immed_5>	Dozvoljeni pomaci 1-32
		[Rn], +/-Rm, ASR #<immed_5>	Dozvoljeni pomaci 1-32
		[Rn], +/-Rm, ROR #<immed_5>	Dozvoljeni pomaci 1-31
		[Rn], +/-Rm, RRX	

**Način adresiranja <a\_mode3>**

<b>Predindeksiranje/ Adres. s odmakom</b>	Neposredni odmak	[Rn, #+/-<immed_8>]{!}	
	Nulti odmak	[Rn]	Ekvivalentno sa [Rn,#0]
	Registarski odmak	[Rn, +/-Rm]{!}	
<b>Postindeksiranje</b>	Neposredni odmak	[Rn], #+/-<immed_8>	
	Registarski odmak	[Rn], +/-Rm	

**Način adresiranja <a\_mode4L> (učitavanje bloka)**

Nastavci za rad s blokom podataka	Nastavci za rad sa stogom
IA Increment After	FD Full Descending
IB Increment Before	ED Empty Descending
DA Decrement After	FA Full Ascending
DB Decrement Before	EA Empty Ascending

**Način adresiranja <a\_mode4S> (spremanje bloka)**

Nastavci za rad s blokom podataka	Nastavci za rad sa stogom
IA Increment After	EA Empty Ascending
IB Increment Before	FA Full Ascending
DA Decrement After	ED Empty Descending
DB Decrement Before	FD Full Descending

**Registar stanja CPSR**

bit	31	30	29	28	...	24	23	...	16	15	...	8	7	6	...	0
značenje	N	Z	C	V	...								I	F		M (mode)
<fields>	Flags field mask byte (f)						Status field mask byte (s)				Extension field mask byte (e)				Control field mask byte (c)	

Postavljanje bita I ili F u ničitsu omogućit će prekid (I) ili brzi prekid (F).

**Posebnosti pisanja programa za procesor ARM 7 u programskom paketu ATLAS**

Pretpostavljena (default) baza za pisanje brojeva	heksadekadska (osim za definiranje iznosa pomaka i rotacija) MOV R0, #16 – ovdje je 16 u heksadekadskoj bazi MOV R0, #16<8 – ovdje je 16 u heksadekadskoj bazi, a 8 u dekadskoj bazi MOV R0, R0, LSL # 16 – ovdje je 16 u dekadskoj bazi
Pisanje heksadekadskog broja	%H
Pisanje dekadskog broja	%D
Pisanje binarnog broja	%B
Pseudonaredbe za definiranje sadržaja memorije	Bajt: DW, DB Poluriječ: DH Riječ: DW Prostor: DS (napomena: DB, DH i DW poravnavaju se na adresu djeljivu sa 4)
Pisanje neposredne vrijednosti u naredbama za obradu podataka (operand <immed_8>)	baza(hex)<rotacija_ulijevo(dekadski) npr. 3A<8 je zapis heksadekadskog broja 3A00
Naredba za čitanje/upis podatka sa memorijske lokacije označene labelom	LDR{cond}{B SB H SH} Rd, labela STR{cond}{B H} Rd, labela (napomena: naredba se prevodi tako da se koristi PC kao bazni registar, a odmak se računa automatski)
Definiranje niza registara u naredbama LDM i STM	U popisu registri moraju biti razdvojeni zarezima i navedeni u rastućem redoslijedu, npr. {R2,R4,R15}. Oblik {Rx-Ry} nije dozvoljen, izuzetak je navođenje {R0-R15}
Zaustavljanje procesora	HALT ili SWI 123456

	Opis (engl.)	Mnemonička naredba	Zastavice	Operacija
Registarske naredbe	Move	<b>MOV</b> {cond}{S} Rd, <Oprnd2>	N Z C	Rd := Oprnd2
	Move NOT	<b>MVN</b> {cond}{S} Rd, <Oprnd2>	N Z C	Rd := 0xFFFFFFFF EOR Oprnd2
	Move SPSR to register	<b>MRS</b> {cond} Rd, SPSR		Rd := SPSR
	Move CPSR to register	<b>MRS</b> {cond} Rd, CPSR		Rd := CPSR
	Move register to SPSR	<b>MSR</b> {cond} SPSR <fields>, Rm		SPSR := Rm (samo označena polja)
	Move register to CPSR	<b>MSR</b> {cond} CPSR <fields>, Rm		CPSR := Rm (samo označena polja)
	Move immediate to SPSR	<b>MSR</b> {cond} SPSR <fields>, #<immed_8r>		SPSR := immed_8r (samo označena polja)
	Move immediate to CPSR	<b>MSR</b> {cond} CPSR <fields>, #<immed_8r>		CPSR := immed_8r (samo označena polja)
Aritmetičko logičke naredbe	Add	<b>ADD</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2
	Add with carry	<b>ADC</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn + Oprnd2 + Carry
	Subtract	<b>SUB</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2
	Subtract with carry	<b>SBC</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Rn - Oprnd2 - NOT(Carry)
	Reverse subtract	<b>RSB</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn
	Reverse subtract with carry	<b>RSC</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C V	Rd := Oprnd2 - Rn - NOT(Carry)
	Multiply	<b>MUL</b> {cond}{S} Rd, Rm, Rs	N Z	Rd := (Rm * Rs)[31:0]
	Multiply accumulate	<b>MLA</b> {cond}{S} Rd, Rm, Rs, Rn	N Z	Rd := ((Rm * Rs) + Rn)[31:0]
	Multiply unsigned long	<b>UMULL</b> {cond}{S} RdLo, RdHi, Rm, Rs	N Z	RdHi, RdLo := nepredznačno(Rm*Rs)
	Multiply unsigned accumulate long	<b>UMLAL</b> {cond}{S} RdLo, RdHi, Rm, Rs	N Z	RdHi, RdLo := nepredznačno(RdHi, RdLo+Rm*Rs)
	Multiply signed long	<b>SMULL</b> {cond}{S} RdLo, RdHi, Rm, Rs	N Z	RdHi, RdLo := predznačno(Rm*Rs)
	Multiply signed accumulate long	<b>SMLAL</b> {cond}{S} RdLo, RdHi, Rm, Rs	N Z	RdHi, RdLo := predznačno(RdHi, RdLo+Rm*Rs)
	Count leading zeroes	<b>CLZ</b> {cond} Rd, Rm		Rd := broj vodećih 0 u Rm
	Test	<b>TST</b> {cond} Rn, <Oprnd2>	N Z C	Rn AND Oprnd2; osvježi CPSR zastavice
	Test equivalence	<b>TEQ</b> {cond} Rn, <Oprnd2>	N Z C	Rn EOR Oprnd2; osvježi CPSR zastavice
	AND	<b>AND</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND Oprnd2
	Exclusive OR	<b>EOR</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn EOR Oprnd2
	OR	<b>ORR</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn OR Oprnd2
	Bit Clear	<b>BIC</b> {cond}{S} Rd, Rn, <Oprnd2>	N Z C	Rd := Rn AND NOT Oprnd2
	No operation	<b>NOP</b>		R0 := R0
Memorijske naredbe	Compare	<b>CMP</b> {cond} Rn, <Oprnd2>	N Z C V	Rn - Oprnd2; osvježi CPSR zastavice
	Compare negative	<b>CMN</b> {cond} Rn, <Oprnd2>	N Z C V	Rn + Oprnd2; osvježi CPSR zastavice
	Load Word	<b>LDR</b> {cond} Rd, <a_mode2>		Rd := [adresa]
	Load branch (and exchange)	<b>LDR</b> {cond} R15, <a_mode2>		R15 := [adresa][31:1]
	Load Byte	<b>LDR</b> {cond}B Rd, <a_mode2>		Rd := bajt s [adrese], proširen s 0
	Load signed Byte	<b>LDR</b> {cond}SB Rd, <a_mode3>		Rd := bajt s [adrese], predznačno proširen
	Load Halfword	<b>LDR</b> {cond}H Rd, <a_mode3>		Rd := poluriječ s [adrese], proširena s 0
	Load signed Halfword	<b>LDR</b> {cond}SH Rd, <a_mode3>		Rd := poluriječ s [adrese], predznačno proširena
	Load multiple Pop, or Block data load	<b>LDM</b> {cond}<a_mode4L> Rd{!}, <reglist-pc>		Učitaj listu registara sa [Rd]
	Load multiple return (and exchange)	<b>LDM</b> {cond}<a_mode4L> Rd{!}, <reglist+pc>		Učitaj registre, R15 := [adresa][31:1]
	Load multiple and restore CPSR	<b>LDM</b> {cond}<a_mode4L> Rd{!}, <reglist+pc>^		Učitaj registre, skoči, CPSR := SPSR
	Store Word	<b>STR</b> {cond} Rd, <a_mode2>		[adresa] := Rd
Uprav	Store Byte	<b>STR</b> {cond}B Rd, <a_mode2>		[adresa][7:0] := Rd[7:0]
	Store Halfword	<b>STR</b> {cond}H Rd, <a_mode3>		[adresa][15:0] := Rd[15:0]
	Store multiple Push, or Block data store	<b>STM</b> {cond}<a_mode4S> Rd{!}, <reglist>		Spremi listu registara na [Rd]
	Store multiple User mode registers	<b>STM</b> {cond}<a_mode4S> Rd{!}, <reglist>^		Spremi listu registara korisničkog načina rada na [Rd]
	Branch	<b>B</b> {cond} label		R15 := labela
	Branch with link	<b>BL</b> {cond} label		R14 := R15-4, R15 := labela
	Software interrupt	<b>SWI</b> {cond} <immed_24>		Programska iznimka

- {S} ako je napisan S, osvježavaju se pripadne zastavice;
- {!} ako je napisan !, tada se bazni registar osvježava nakon prijenosa podataka;
- <immed\_x> x-bitna konstanta;
- <reglist> lista registara odvojenih zarezom, napisana unutar vitičastih zagrada;
- label labela mora biti unutar ±32MB od trenutne naredbe

## Sklop GPIO

Adresa	Naziv registra	Opis
PA	GPIOADR	8-bitni registar podataka, vrata A
PA + 0x4	GPIOBDR	8-bitni registar podataka, vrata B
PA + 0x8	GIOPADDR	8-bitni registar smjera podataka za vrata A
PA + 0xC	GPIOBDDR	8-bitni registar smjera podataka za vrata B

### Opis registara

GPIOADR (GPIO Port A Data Register) je 8-bitni registar podataka za vrata A. Podatci upisani u ovaj registar postavljeni su na izlazne priključke ako je pripadni bit u registru smjera podataka (GIOPADDR) postavljen u logičku jedinicu. Čitanje ovog registra vraća trenutno stanje priključaka koji su programirani kao ulazni. Za priključke programirane kao izlazne, vraća se vrijednost zadnjeg podatka upisanog u ovaj registar.

GPIOBDR (GPIO Port B Data Register) je 8-bitni registar podataka za vrata B. Podatci upisani u ovaj registar postavljeni su na izlazne priključke ako je pripadni bit u registru smjera podataka (GPIOBDDR) postavljen u logičku nulu (ovo je suprotno od vrata A). Čitanje ovog registra vraća trenutno stanje priključaka koji su programirani kao ulazni. Za priključke programirane kao izlazne, vraća se vrijednost zadnjeg podatka upisanog u ovaj registar.

GIOPADDR (GPIO Port A Data Direction Register) je 8-bitni registar smjera podataka za vrata A. Bit postavljen u logičku jedinicu u ovom registru konfigurira odgovarajući priključak od vrata A kao izlazni. Postavljanje bita u nulu konfigurira odgovarajući priključak vrata A kao ulazni.

GPIOBDDR (GPIO Port B Data Direction Register) je 8-bitni registar smjera podataka za vrata B. Bit postavljen u logičku nulu u ovom registru konfigurira odgovarajući priključak od vrata B kao izlazni. Postavljanje bita u jedinicu konfigurira odgovarajući priključak vrata B kao ulazni.

### Početna vrijednost registara

Svi registri unutar GPIO nakon inicijalizacije (resetiranja) postavljaju se u logičku nulu. Ovime se inicijalno vrata A postavljaju kao ulazna, a vrata B kao izlazna. Sadržaji obaju registara podataka su nula.

## Sklop RTC

Adresa	Naziv registra	Opis
PA	RTCDR	32-bitni registar podataka (može se samo čitati)
PA + 0x4	RTCMR	32-bitni registar usporedbe
PA + 0x8	RTCSTAT/RTCEOI	1-bitni registar stanja prekida (ako se čita) / 0-bitni registar za brisanje prekida (ako se piše)
PA + 0xC	RTCLR	32-bitni registar za punjenje brojila
PA + 0x10	RTCCR	1-bitni upravljački registar

### Opis registara

RTCDR (Real Time Clock Data Register) je 32-bitni registar podataka. Čitanjem ovog registra dobiva se trenutna vrijednost brojila. Pisanje u ovaj registar nije dozvoljeno.

RTCMR (Real Time Clock Match Register) je 32-bitni registar usporedbe. Upisivanjem podatka u ovaj registar postavlja se nova vrijednost koja služi za usporedbu s brojilom. Čitanjem ovog registra dobiva se zadnja vrijednost upisana u registar usporedbe.

RTCSTAT/RTCEOI (Real Time Clock Interrupt **STAT**us Register / Real Time Clock Interrupt Clear Register) je virtualni registar bez fizičkog sklopovlja za pohranjivanje podataka. Pisanjem bilo kojeg podatka na ovu adresu čisti se prekidni signal RTCINTR i pripadni registar. Čitanjem s ove adrese dobiva se podatak koji na bitu 0 (najniži bit) ima trenutnu vrijednost RTCINTR. Ako je bit 0 postavljen na jedinicu, to znači da je prekidni signal aktivan.

RTCLR (Real Time Clock Load Register) je 32-bitni registar koji služi za upis vrijednosti brojila ili čitanje zadnje vrijednosti koja je upisana. Pisanjem u ovaj registar započinje proces pisanja nove vrijednosti u brojilo. Pisanje se ne izvodi trenutno nego na prvi sljedeći rastući brid na ulazu CLK1HZ.

RTCCR (Real Time Clock Control Register) je 1-bitni upravljački registar kojim programer može omogućiti ili onemogućiti generiranje prekida. Ako se na bit 0 (najniži bit) ovog registra upiše logička nula, tada se RTC-u onemogućuje generiranje prekida. Ako se upiše jedinica, tada se omogućuje generiranje prekida. Čitanjem ovog registra na bitu 0 dobiva se zadnja upisana vrijednost prekidnog bita. Ostali bitovi u ovom registru ne postoje.

### Početna vrijednost registara

Svi registri unutar sklopa RTC nakon inicijalizacije (resetiranja) se postavljaju u logičku nulu.