ZBIRKA NALOG Z REŠITVAMI IN ŠE KAJ 2012

1. Zapišite ukaz v zbirniku za procesor ARM, ki v register **R1** naloži konstanto **128**. Uporabite takojšnje naslavljanje (ukaz **MOV**). Izpišite ustrezen strojni ukaz (32 – bitno število v šestnajstiški obliki). Program izvajajte po korakih in opazujte vrednosti registrov **R15(PC)** in **R1**. Kakšni sta njuni vrednosti na koncu programa?

```
        .text
@Deklaracija spremenljivk

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

        mov r1,#128

@Konec programa
__end:    b __end

@Strojni ukaz mov: 8010A0E3
@PC=0x24
@R1=0x80
```

2. Zapišite ukaz(e) v zbirniku za procesor ARM, ki v register naloži vrednost spremenljivke:

1. naloži 32-bitno vrednost **0x12345678** v register **R1**
2. naloži 8-bitno vrednost **128** v register **R1**
3. naloži 16-bitno vrednost **0xF123** v register **R1**

Naloge rešite s posrednim naslavljanjem preko **PC** in posrednim naslavljanjem brez odmika preko registra **R0**.

```
        .text
@Deklaracija spremenljivk
spr_32:   .word 0x12345678
spr_8:    .byte 128
        .align
spr_16:   .hword 0xF123
@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@S posrednim naslavljanjem preko PC
        ldr r1,spr_32
        ldrb r1,spr_8
        ldrh r1,spr_16

@S posrednim brez odmika
        adr r0,spr_32
        ldr r1,[r0]

        adr r0,spr_8
        ldrb r1,[r0]

        adr r0,spr_16
        ldrh r1,[r0]

@Konec programa
__end:    b __end
```

3. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki vrednost 32-bitne spremenljivke **STEV1** prepiše v 32-bitno spremenljivko **STEV2**. Nalogo rešite še za primer, če sta spremenljivki **STEV1** in **STEV2** 8-bitni oziroma 16-bitni. Vse primere rešite s posrednim naslavljanjem preko **PC** in posrednim naslavljanjem brez odmika preko registra **R0**.

```
        .text
@Deklaracija spremenljivk
stev1_32: .word 0x12345678
stev2_32: .space 4

stev1_16: .hword 0xAABB
stev2_16: .space 2

stev1_8:  .byte 0x10
stev2_8:  .space 1

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@S posrednim naslavljanjem preko PC
        ldr r1,stev1_32
        str r1,stev2_32

        ldrh r1,stev1_16
        strh r1,stev2_16

        ldrb r1,stev1_8
        strb r1,stev2_8

@S posrednim brez odmika
        adr r0,stev1_32
        ldr r1,[r0]
        adr r0,stev2_32
        str r1,[r0]

        adr r0,stev1_16
        ldrh r1,[r0]
        adr r0,stev2_16
        strh r1,[r0]

        adr r0,stev1_8
        ldrb r1,[r0]
        adr r0,stev2_8
        strb r1,[r0]
```

4. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki zamenja vrednosti 32-bitnih spremenljivk **STEV1** in **STEV2**. Nalogo rešite s posrednim naslavljanjem preko **PC**.

```
        .text
@Deklaracija spremenljivk

stev1:   .word 0x12345678
stev2:   .word 0x87654321

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

        ldr r1,stev1
        ldr r2,stev2
        str r1,stev2
        str r2,stev1

@Konec programa
__end:   b __end
```

5. Rezervirajte prostor za tabelo z oznako **TABELA**, v kateri je 5 8-bitnih vrednosti. Napišite zaporedje ukazov v zbirniku za ARM, ki v vse bajte tabele zapiše vrednost 0xFF. Nalogo rešite s *posrednim naslavljanjem preko PC* in *posrednim naslavljanjem preko registra R0 in uporabo takojšnjega odmika*. Uporabite le ukaze za nalaganje konstant v registre in ukaze za shranjevanje vrednosti registrov v pomnilnik.

```
        .text
@Deklaracija spremenljivk

tabela:  .byte 1,2,3,4,5

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@S posrednim naslavljanjem preko PC
        mov r1,#0xFF
        strb r1,tabela
        strb r1,tabela+1
        strb r1,tabela+2
        strb r1,tabela+3
```

```
        strb r1,tabela+4

@S posrednim preko r0 in odmikom
        mov r1,#0xFF
        adr r0,tabela
        strb r1,[r0,#0]
        strb r1,[r0,#1]
        strb r1,[r0,#2]
        strb r1,[r0,#3]
        strb r1,[r0,#4]

@Konec programa
__end:   b __end
```

PORAVNANOST:

Rekli smo, da morajo biti pomnilniški operandi vedno poravnani, sicer pride do težav, saj v registre nalagamo napačne vrednosti.

Zapomnimo si: *Spremenljivke morajo biti poravnane na take naslove, da daje deljenje naslova z njihovo velikostjo v bajtih ostanek 0.*

Torej, 32-bitna spremenljivka (.word) se lahko nahaja na naslovih 0x10, 0x14, ... 16-bitna spremenljivka (.hword) pa se lahko nahaja na 0x10, 0x12, ...

Spodnja koda kaže nekaj primerov pravilne in napačne poravnave. Znak % med komentarji pa pomeni operacijo modulo - ostanek pri deljenju.

```
                .text
@Deklaracija spremenljivk

@Primer 1 - napacno:
            .align
p1_1:       .byte 1     @Pravilno: 0x20 % 1 = 0
p1_2:       .word 0x10  @Napaka: 0x21 % 4 = 1

@Primer 1 - pravilno:
            .align
p1_1ok:     .byte 1     @Pravilno: 0x28 % 1 = 0
            .align
p1_2ok:     .word 0x10  @Pravilno: 0x2C % 4 = 0

@Primer 2 - napacno:
            .align
p2_1:       .byte 2     @Pravilno: 0x30 % 1 = 0
p2_2:       .hword 0x22 @Napaka: 0x31 % 2 = 1
p2_3:       .hword 0x23 @Napaka: 0x33 % 2 = 1

@Primer 2 - pravilno:
            .align
p2_1ok:     .byte 2     @Pravilno: 0x38 % 1 = 0
```

```
          .align
p2_2ok:   .hword 0x22 @Pravilno: 0x3C % 2 = 0
p2_3ok:   .hword 0x23 @Pravilno: 0x3E % 2 = 0

@Primer 3 - pravilno:
          .align
p3_1:     .word 0x12345678 @Pravilno: 0x40 % 4 = 0
p3_2:     .word 0x87654321 @Pravilno: 0x44 % 4 = 0
p3_3:     .hword 0xABCD    @Pravilno: 0x48 % 2 = 0
p3_4:     .hword 0xDCBA    @Pravilno: 0x4A % 2 = 0
p3_5:     .byte 1,2        @Pravilno: 0x4C % 1 = 0 in 0x4D % 1 = 0
p3_6:     .hword 0xFFFF    @Pravilno: 0x4E % 2 = 0

@Konec deklaracije spremenljivk
          .align
          .global __start
__start:
@Zacetek programa

@Primer 1
          ldrb r1,p1_1
          ldr  r2,p1_2

          ldrb r1,p1_1ok
          ldr  r2,p1_2ok
@Primer 2
          ldrb r1,p2_1
          ldrh r2,p2_2
          ldrh r3,p2_3

          ldrb r1,p2_1ok
          ldrh r2,p2_2ok
          ldrh r3,p2_3ok
@Primer 3
          ldr  r1,p3_1
          ldr  r2,p3_2
          ldrh r3,p3_3
          ldrh r4,p3_4
          ldrb r5,p3_5
          ldrb r6,p3_5+1
          ldrh r7,p3_6

@Konec programa
__end:    b __end
```

6. Rezervirajte prostor za tabelo z oznako **TABELA**, v kateri je 6 8-bitnih vrednosti. Nato napišite zaporedje ukazov v zbirniku za ARM, ki najprej v zaporedne bajte tabele zapiše števila od 1 do 6, nato pa zamenja vrstni red elementov tabele tako, da so po zamenjavi vrstnega reda števila razporejena v obratnem vrstnem redu kot na začetku.
Nalogo rešite s posrednim naslavljanjem preko PC.

```
        .text
@Deklaracija spremenljivk

tabela:   .space 6

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@Zapis stevil
        mov r1,#1
        strb r1,tabela
        mov r1,#2
        strb r1,tabela+1
        mov r1,#3
        strb r1,tabela+2
        mov r1,#4
        strb r1,tabela+3
        mov r1,#5
        strb r1,tabela+4
        mov r1,#6
        strb r1,tabela+5

@Zamenjava vrstnega reda
        ldrb r1,tabela
        ldrb r2,tabela+5
        strb r2,tabela
        strb r1,tabela+5

        ldrb r1,tabela+1
        ldrb r2,tabela+4
        strb r2,tabela+1
        strb r1,tabela+4

        ldrb r1,tabela+2
        ldrb r2,tabela+3
        strb r2,tabela+2
        strb r1,tabela+3
```

```
@Konec programa
__end:    b __end
```

7. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki 32-bitno spremenljivko **STEV1**, ki je v pomnilniku shranjena po pravilu **tankega konca**, prepiše v **STEV2**, pri čemer naj bo STEV2 v pomnilniku shranjena po pravilu **debelega konca**.
   Naredite dve različici, pri eni uporabite *posredno naslavljanje s takojšnjim odmikom preko registra **R0***, pri drugi pa *posredno naslavljanje preko **PC***.

```
        .text
@Deklaracija spremenljivk

stev1:    .word 0x12345678
stev2:    .space 4

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@S posrednim naslavljanjem preko r0 in odmikom
        adr r0,stev1

        ldrb r1,[r0,#0]
        strb r1,[r0,#7]

        ldrb r1,[r0,#1]
        strb r1,[r0,#6]

        ldrb r1,[r0,#2]
        strb r1,[r0,#5]

        ldrb r1,[r0,#3]
        strb r1,[r0,#4]

@S posrednim preko PC
        ldrb r1,stev1
        strb r1,stev1+7

        ldrb r1,stev1+1
        strb r1,stev1+6

        ldrb r1,stev1+2
        strb r1,stev1+5

        ldrb r1,stev1+3
```

```
        strb r1,stev1+4

@Konec programa
__end:    b __end
```

8. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki sešteje 32-bitni spremenljivki **STEV1** in **STEV2**, rezultat pa zapiše v 32-bitno spremenljivko **REZ**.
Nalogo ponovite še na primeru, če spremenljivko **STEV2** odštejemo odspremenljivke **STEV1**. Naredite tudi različice za seštevanje in odštevanje 16-bitnih in 8-bitnih spremenljivk.
STEV1 in **STEV2** naj imata začetni vrednosti 10_{10} in 5_{10} zaporedoma, za **REZ** pa samo rezervirajte prostor.

```
        .text
@Deklaracija spremenljivk

stev1_32: .word 10
stev2_32: .word 5
rez_32:   .space 4

stev1_16: .hword 10
stev2_16: .hword 5
rez_16:   .space 2

stev1_8:  .byte 10
stev2_8:  .byte 5
rez_8:    .space 1

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa

@32-bitno sestevanje
        ldr r1,stev1_32
        ldr r2,stev2_32
        add r3,r1,r2
        str r3,rez_32

@32-bitno odstevanje
        ldr r1,stev1_32
        ldr r2,stev2_32
        sub r3,r1,r2
        str r3,rez_32

@16-bitno sestevanje
        ldrh r1,stev1_16
        ldrh r2,stev2_16
```

```
        add r3,r1,r2
        strh r3,rez_16

@16-bitno odstevanje
        ldrh r1,stev1_16
        ldrh r2,stev2_16
        sub r3,r1,r2
        strh r3,rez_16

@8-bitno sestevanje
        ldrb r1,stev1_8
        ldrb r2,stev2_8
        add r3,r1,r2
        strb r3,rez_8

@8-bitno odstevanje
        ldrb r1,stev1_8
        ldrb r2,stev2_8
        sub r3,r1,r2
        strb r3,rez_8

@Konec programa
__end:  b __end
```

9. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
STEV1 = STEV2 + STEV3 – STEV1
STEV1, **STEV2** in **STEV3** 32-bitne spremenljivke z začetnimi vrednostmi (določite jih s psevdoukazi): STEV1 = 50_{16}, STEV2 = 100_{10}, STEV3 = $2F_{16}$.

```
        .text
@Deklaracija spremenljivk

STEV1:  .word 0x50
STEV2:  .word 100
STEV3:  .word 0x2F

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa
        ldr r1,STEV1
        ldr r2,STEV2
        ldr r3,STEV3
        add r4,r2,r3
        sub r1,r4,r1
```

```
        str r1,STEV1

@Konec programa
__end:   b __end
```

10. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
REZ = STEV2 + STEV3 – STEV1
STEV1 in **STEV2** sta 8-bitni spremenljivki, **STEV3** je 16-bitna spremenljivka, **REZ** pa je 32-bitna spremenljivka. Začetne vrednosti spremenljivk naj bodo (določite jih s psevdoukazi): STEV1 = 50_{16}, STEV2 = 100_{10}, STEV3 = $2F_{16}$.
Spremenljivke naj bodo na zaporednih naslovih tako, da zaradi poravnanosti spremenljivk ni neizkoriščenega pomnilnika.

```
        .text
@Deklaracija spremenljivk

STEV1:   .byte 0x50
STEV2:   .byte 100
STEV3:   .hword 0x2F
REZ:     .space 4

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa
        ldrb r1,STEV1
        ldrb r2,STEV2
        ldrh r3,STEV3
        add r4,r2,r3
        sub r1,r4,r1
        str r1,REZ

@Konec programa
__end:   b __end
```

11. Napišite program, ki sešteje bolj pomembni bajt 16-bitne spremenljivke **S1** in manj pomembni bajt 16-bitne spremenljivke **S2**. Vsoti nato odštejte DF_{16}. Rezultat shranite v 8-bitno spremenljivko **S3**. Spremenljivka **S1** naj ima začetno vrednost 43980_{10}, **S2** naj ima začetno vrednost 1234_{16}, za **S3** pa s psevdoukazom samo rezervirajte 1 bajt prostora.

Začetni vrednosti **S1** in **S2** določite s psevdoukazi. Ob vsakem ukazu kot komentar zapišite način naslavljanja.

```
        .text
@Deklaracija spremenljivk
S1:     .hword 43980
S2:     .hword 0x1234
S3:     .space 1

@Konec deklaracije spremenljivk

        .align
        .global __start
__start:
@Zacetek programa
        ldrb  r1,S1+1    @posredno PC
        ldrb  r2,S2      @posredno PC
        add   r3,r1,r2   @registrsko
        @mov r4,#0xDF    @takojsnje
        @sub r3,r3,r4    @registrsko
        sub   r3,r3,#0xDF @takojsnje
        strb  r3,S3      @posredno PC

@Konec programa
__end:  b __end
```

12. Napišite program, ki vsoto 8-bitnih spremenljivk **S2**+**S3** zapiše v bolj pomembni bajt in razliko **S2**-**S3** v manj pomembni bajt 16-bitne spremenljivke **S1**. Spremenljivki**S2** naj nato prišteje konstanto 35_{16} Za spremenljivko **S1** s psevdoukazi samo rezervirajte prostor, **S2** in **S3** pa naj imata vrednost $A1_{16}$ in 115_{10} zaporedoma.
Ob vsakem ukazu kot komentar napišite način naslavljanja.

```
        .text
@Deklaracija spremenljivk

S1:     .space 2
S2:     .byte 0xA1
S3:     .byte 115

@Konec deklaracije spremenljivk
        .align
        .global __start
__start:
@Zacetek programa
        ldrb  r1,S2      @posredno PC
        ldrb  r2,S3      @posredno PC
        add   r3,r1,r2   @registrsko
        strb  r3,S1+1    @posredno PC
        sub   r4,r1,r2   @registrsko
        strb  r4,S1      @posredno PC
```

```
        add   r1,r1,#0x15   @takojsnje
        strb  r1,S2         @posredno PC

@Konec programa
__end:    b __end
```

13. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna spodnji izraz:
STEV1 = STEV1 + 1
STEV1 je 32-bitna spremenljivka z začetno vrednostjo $0F_{16}$. Pri prištevanju konstante 1 uporabite takojšnje naslavljanje.

```
        .text
@Spremenljivke
STEV1:    .word 0x0F

        .align
        .global __start
__start:
@Program
        ldr r1,STEV1
        add r1,#1
        str r1,STEV1

@Konec
__end:    b __end
```

14. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna spodnji izraz:
STEV1 = 128_{10} - STEV1
STEV1 je 32-bitna spremenljivka z začetno vrednostjo 80_{16}. Pri delu s konstanto uporabite takojšnje naslavljanje.

```
        .text
@Spremenljivke
STEV1:    .word 0x80

        .align
        .global __start
__start:
@Program
        ldr r1,STEV1
        mov r2,#128
        sub r1,r2,r1
        str r1,STEV1
@Konec
__end:    b __end
```

15. Zapišite ukaze v zbirniku, ki v register naložijo vrednosti spremenljivk:

1. Nepredznačeno naloži 32-bitno vrednost 0x12345678 v register R1.
2. Nepredznačeno naloži 8-bitno vrednost 128 v register R1.
3. Predznačeno naloži 8-bitno vrednost 128 v register R1.
4. Nepredznačeno naloži 16-bitno vrednost 0xF123 v register R1.
5. Predznačeno naloži 16-bitno vrednost 0xF123 v register R1.

Vrednosti spremenljivk deklarirajte s psevdoukazi. Vse primere rešite s posrednim naslavljanjem preko PC.

```
        .text
@Spremenljivke
S1:     .word 0x12345678
S2:     .byte 128
        .align
S3:     .hword 0xF123

        .align
        .global __start
__start:
@Program
        ldr r1,S1
        ldrb r1,S2
        ldrsb r1,S2
        ldrh r1,S3
        ldrsh r1,S3

@Konec
__end:  b __end
```

16. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki v register najprej naloži vrednost 0 – uporabite takojšnje naslavljanje. Kakšno je stanje zastavic Z, C, V in N po tem ukazu? Zakaj?
Nato naj program registru odšteje vrednost 1. Kakšna je vrednost registra, če predstavlja nepredznačeno / predznačeno število? Kakšno je stanje zastavic Z, C, V in N po tem ukazu? Zakaj?
Nato naj program registru prišteje vrednost 2. Kakšno je stanje zastavic Z, C, V in N po tem ukazu? Zakaj?
Kakšna je končna vrednost registra, če predstavlja nepredznačeno / predznačeno število?

```
        .text
        .align
        .global __start
__start:
        movs r1,#0  @Če uporabimo "navadni" ukaz mov, se
                @zastavice ne postavijo!
        @C=0, pri ukazu movs ne pride do prenosa
        @V=0, pri ukazu movs ne pride do preliva
        @Z=1, rezultat je enak 0
```

```
        @N=0, število 0 ni negativno število

        subs r1,r1,#1 @Če uporabimo "navadni" sub, se
                @zastavice ne bodo postavile
        @C=0, pri odštevanju je prišlo do "izposoje", ker
        @     se zastavica C pri odštevanju obnaša ravno
        @     obratno, kot pri seštevanu, je C enak 0
        @V=0, ker nismo prekoračili mej intervala
        @     predznačenih števil
        @Z=0, ker rezultat (0xFFFFFFFF) ni enak 0
        @N=1, ker je 31. bit rezultata enak 1
        @Če na rezultat gledamo kot na predznačeno število,
        @potem je r1 = -1
        @Če na rezultat gledamo kot na nepredznačeno število,
        @potem je r1 = 0xFFFFFFFF

        adds r1,r1,#2 @Če uporabimo "navaden" ukaz add, se
                @zastavice ne bodo postavile
        @C=1, ker je pri prištevanju prišlo do prenosa
        @     0xFFFFFFFF + 2!
        @V=0, ker nismo prekoračili intervala predznačenih
        @     števil
        @Z=0, ker rezultat (1) ni enak 0
        @N=0, ker je 31. bit rezltata enak 0
        @Če na rezultat gledamo kot nepredznačeno ali kot na
        @predznačeno število, je r1 = 1
__end:  b __end
```

17. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki s pomočjo ALE ukaza v register najprej naloži vrednost **1**. Kakšne vrednosti imajo zastavice Z, C, V in N po tem ukazu? Zakaj?
Nato naj program odšteje od registra vrednost **1**. Kakšne vrednosti imajo zastavice Z, C, V in N po tem ukazu? Zakaj?
Nato naj program še enkrat odšteje vrednost **1** od registra. Kakšne vrednosti imajo zastavice Z, C, V in N po tem ukazu? Zakaj?
Kakšna je končna vrednost registra, če predstavlja nepredznačeno / predznačeno število?

```
        .text
@Spremenljivke

        .align
        .global __start
__start:
@Program
@Napišite zaporedje ukazov v zbirniku za procesor ARM,
@ki s pomočjo ALE ukaza v register najprej naloži vrednost 1.
@Kakšne vrednosti imajo zastavice Z, C, V in N po tem ukazu?
@ Zakaj?
```

```
        adds r1,r1,#1
@Z=0, ker 1 ni enako 0
@C=0, ker pri 0+1 ne pride do prenosa
@V=0, ker ostanemo znotraj intervala predznačenih števil
@N=0, ker je najpomembnejši bit enak 0 (število je pozitivno)

        subs r1,r1,#1
@Z=1, ker je rezultat enak 0
@C=1, ker pri operaciji 1-1 ne pride do "izposoje" bitov,
@     inker se zastavica C obnaša ravno obratno kot pri seštevnaju
@V=0, ker ostanemo znotraj intervala predznačenih števil
@N=0, ker je najpomembnejši bit enak 0 (število je pozitivno)

        subs r1,r1,#1
@Z=0, ker je rezultat enak 0xffffffff, kar ni enako 0
@C=0, ker pri operaciji 0-1 pride do "izposoje"
@     in ker se zastavica C obnaša ravno obratno kot pri seštevanju
@V=0, ker ostanemo znotraj intervala predznačenih števil
@N=1, ker je najpomembnejši bit enak 1 (če na število gledamo
@     kot predznačeno, je rezultat enak -1)

@ r1 = -1, če gledamo nanj kot na predznačeno ptevilo
@ r1 = 0xffffffff, če gledamo nanj kot na nepredznačeno število

@Konec
__end:    b __end
```

18. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
STEV1 = STEV2 + STEV3 – STEV1
STEV1 je 16-bitna predznačena spremenljivka, **STEV2** in **STEV3** pa 8-bitni predznačeni spremenljivki z začetnimi vrednostmi (določite jih s psevdoukazi): **STEV1** = -10_{10}, **STEV2** = 64_{16}, **STEV3** = -2_{10}. Preverite pravilnost rezultata!

```
        .text
@Spremenljivke
@ STEV1 = STEV2 + STEV3 – STEV1
STEV1:    .hword -10
STEV2:    .byte 0x64
STEV3:    .byte -2

        .align
        .global __start
__start:
@Program
        ldrsh r1,STEV1
        ldrsb r2,STEV2
        ldrsb r3,STEV3
```

```
        add r4,r2,r3
        sub r1,r4,r1
        strh r1,STEV1

@STEV1=0x6c (0x62 + (-0xA) = 0x6c)

@Konec
__end:    b __end
```

19. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
STEV1 = STEV2 + STEV3
STEV1, **STEV2** in **STEV3** so 32-bitne nepredznačene spremenljivke z začetnimi vrednostmi (določite jih s psevdoukazi): **STEV2** = $7fffffff_{16}$, **STEV3** = 80000001_{16}. Za STEV1 le rezervirajte prostor.
Razložite stanje zastavic Z, C, V in N po izvedbi ukaza za seštevanje. Kakšna je končna vrednost **STEV1?** Zakaj?

```
        .text
@Spremenljivke
STEV1:    .space 4
STEV2:    .word 0x7fffffff
STEV3:    .word 0x80000001

        .align
        .global __start
__start:
@Program
        ldr r2,STEV2
        ldr r3,STEV3
        adds r1,r2,r3
        str r1,STEV1

@C=1, ker je pri seštevanju prišlo do prenosa (rezultat je daljši
@     od 32 bitov (glej vrednost STEV1)
@Z=1, ker je rezultat enak 0 (gledano iz strani registra)
@V=0, ker smo ostali znotraj intervala predznačenih števil
@     če bi delali s predznačenimi števili
@N=0, ker rezultat ni negativno število (najbolj pomemben bit je enak 0)
@STEV1=0, ker je 0x7fff ffff + 0x8000 0001 = 0x01 0000 0000.

@Konec
__end:    b __end
```

20. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
STEV1 = STEV2 + STEV3
STEV2 in **STEV3** sta 32-bitni nepredznačeni spremenljivki z začetnima vrednostma (določite ju s psevdoukazi): **STEV2** = 0, **STEV3** = $ffffffff_{16}$. Za spremenljivko **STEV1**le rezervirajte prostor. Razložite stanje zastavic Z, C, V in N po izvedbi ukaza za seštevanje. Kakšna je končna vrednost **STEV1**? Zakaj?

```
    .text
@Spremenljivke
STEV1:   .space 4
STEV2:   .word 0
STEV3:   .word 0xffffffff

    .align
    .global __start
__start:
@Program
    ldr r2,STEV2
    ldr r3,STEV3
    adds r1,r2,r3
    str r1,STEV1

@C=0, ker pri seštevanju ni prišlo do prenosa
@Z=0, ker rezultat 0xffffffff ni enak 0
@V=0, ker smo ostali znotraj intervala predznačenih števil
@    če bi na števila gledali predznačeno, potem bi
@    dejansko seštevali 0 + (-1) = -1
@N=1, ker je najbolj pomemben bit je enak 1 (kljub temu
@    da mi število obravnavamo kot nepredznačeno!)
@STEV1=0xffffffff, ker je 0x0 + 0xffffffff = 0xffffffff.

@Konec
__end:   b __end
```

21. Kateri od naslednjih pogojnih skokov se bo izvršil, če je stanje zastavic N=1, C=0, Z=0 in V=0?

1. bcc
2. bne
3. bge
4. bls
5. bmi
6. bcs
7. blt

rešitev:

1. bcc: Se izvrši.
2. bne: Se izvrši.
3. bge: Se NE izvrši.
4. bls: Se NE izvrši.
5. bmi: Se izvrši.
6. bcs: Se NE izvrši.
7. blt: Se izvrši.

22. Kateri od naslednjih programov se vedno vrtijo v zanki **zanka**?

```
@Primer A
zanka:    mov r1,#127
          bne zanka

@Primer B
zanka:    mov r1,#127
          beq zanka

@Primer C
zanka:    mov r1,#0
          beq zanka

@Primer D
zanka:    mov r1,#1
          cmp r0,#25
          bmi zanka
```

Rešitev:

```
@Začetek
        .text

@Spremenljivke

        .align
        .global __start
__start:
@Program
@Primer A
@zanka:    mov r1,#127  @Ukaz mov brez s ne nastavlja zastavic,
@          bne zanka    @Z=0, kar pomeni, da je zanka neskončna
```

```
@Primer B
@zanka:    mov r1,#127  @Ukaz mov brez s ne nastavlja zastavic,
@          beq zanka    @ker je Z=0, ta zanka NI neskončna



@Primer C
@zanka:    mov r1,#0    @Ukaz mov brez s ne nastavlja zastavic,
@          beq zanka    @ker je Z=0, ta zanka NI neskončna

@Primer C2
@zanka:    movs r1,#0   @Ukaz movs nastavi zastavice,
@          beq zanka    @ker je Z=1, je zanka neskončna

@Primer D
@zanka:    mov r1,#1
@          cmp r0,#25   @Ukaz cmp vedno postavi zastavice
@          bmi zanka    @Ker je 0-25 = -25, je zastavica N=1,
@                    @kar pomeni, da je zanka neskončna

@Konec
__end:     b __end
```

23. Napišite zaporedje ukazov v zbirniku za procesor ARM, ki izračuna izraz:
STEV1 = MAX(STEV2, STEV3)
Vse spremenljivke so 32-bitne in nepredznačene. Program preizkusite z različnimi vrednostmi **STEV2** in **STEV3**. Uporabite pogojni skok. Program naj bo naslednje oblike:

```
STEV1 = STEV2
IF STEV2 > STEV3 THEN GOTO DALJE
STEV1 = STEV3
DALJE: ...
```

Rešitev:

```
          .text


STEV1:    .space 4         @Rezultat (večje število)
STEV2:    .word 0x20       @Prvo število
STEV3:    .word 0x10       @Drugo število


          .align
          .global __start
__start:
          ldr r2,STEV2
```

```
          ldr r3,STEV3


          cmp r2,r3        @Primerjamo števili
          bhi vecji        @Če je število v r2 (STEV2) večje od števila
                           @v r3 (STEV3), skočimo na vrstico "vecji"
          str r3,STEV1     @Sicer pa r3 shranimo na STEV1 (rezultat)
          b __end          @in skočimo na konec programa

vecji:    str r2,STEV1     @r2 shanimo na STEV1 (rezultat) in končamo
                           @program
__end:    b __end
```

24. Napišite program za iskanje največjega skupnega delitelja s pomočjo Evklidovega algoritma. Uporabite pogojne skoke. Program naj bo take oblike:

```
ZANKA:  IF r1<r2 THEN GOTO L1
        IF r1=r2 THEN GOTO L3
        r1=r1-r2
        GOTO ZANKA
        L1: r2=r2-r1
        L2: GOTO ZANKA
        L3: ... @Največji skupni delitelj je v r1 in v r2
```

Rešitev:

```
@Začetek
          .text

@Spremenljivke
ST1:      .word 144
ST2:      .word 24

          .align
          .global __start
__start:
@Program
          ldr r1,ST1
          ldr r2,ST2

ZANKA:    cmp r1,r2
          blo L1
          beq L3
          sub r1,r1,r2
          b   ZANKA
```

```
L1:       sub r1,r2,r1
L2:       b ZANKA
L3:


@Konec
__end:    b __end
```

25. Z uporabo pogojnih skokov napišite program, ki ustreza naslednji psevdokodi:

IF A < 10 THEN B = B-1
IF A = 10 THEN B = B+1
IF A > 10 THEN B = B+2
A in B sta 16-bitni predznačeni spremenljivki.
Program rešite še s pogojnim izvajanjem ukazov.

```
@Začetek
        .text

@Spremenljivke
A:      .hword -10
B:      .hword 0x20

        .align
        .global __start
__start:
@Program rešen s pogojnimi skoki
        ldrsh r1,A
        ldrsh r2,B

        cmp r1,#10    @r1 ? 10
        blt manjsi    @r1 < 10
        beq enak      @r1 = 10

        add r2,r2,#2  @r1 > 10 => r2 = r2 + 2
        b konec

manjsi:   sub r2,r2,#1  @r1 < 10 => r2 = r2 - 1
        b konec

enak:     add r2,r2,#1  @r1 = 10 => r2 = r2 + 1
```

```
konec:   strh r2,B

@Program rešen s pogojnim izvajanjem ukazov
        ldrsh r1,A
        ldrsh r2,B

        cmp r1,#10      @r1 ? 10
        sublt r2,r2,#1  @r1 < 10 => r2 = r2 - 1
        addeq r2,r2,#1  @r1 = 10 => r2 = r2 + 1
        addgt r2,r2,#2  @r1 > 10 => r2 = r2 + 2
        strh r2,B

@Konec
__end:   b __end
```

26. Z uporabo pogojnih skokov napišite program, ki ustreza naslednji psevdokodi:
IF A = 10 THEN B = B+1
ELSE B = B+2
A in B sta 16-bitni predznačeni spremenljivki.
Program rešite še s pogojnim izvajanjem ukazov.

```
@Začetek
          .text

@Spremenljivke
A:        .hword 10
B:        .hword 0x20

          .align
          .global __start
__start:
@Program rešen s pogojnimi skoki
          ldrsh r1,A
          ldrsh r2,B

          cmp r1,#10    @r1 ? 10
          bne else      @r1 != 10
          add r2,r2,#1  @r1 = 10 => r2 = r2+1
          b konec

else:     add r2,r2,#2  @r1 != 10 => r2 = r2+2
```

```
konec:    strh r2,B


@Program rešen s pogojnim izvajanjem ukazov
          ldrsh r1,A
          ldrsh r2,B

          cmp r1,#10      @r1 ? 10
          addeq r2,r2,#1  @r1 = 10 => r2 = r2 + 1
          addne r2,r2,#2  @r1 != 10 => r2 = r2 + 2
          strh r2,B


@Konec
__end:    b __end
```

27. Rezervirajte prostor za tabelo osmih 8-bitnih elementov. Z zanko vpišite v tabelo vrednosti 8, 7, 6, ..., 1.
Napišite še eno zanko, v kateri seštejete vse elemente v tabeli.
Obakrat uporabite posredno naslavljanje brez odmika. Bazni register usmerite na začetek tabele in ga v zanki povečujte.

```
@Začetek
          .text

@Spremenljivke
TABELA:   .space 8

          .align
          .global __start
__start:
@Program
          adr r0,TABELA
          mov r1,#8
zanka1:   strb r1,[r0]
          add r0,r0,#1
          subs r1,r1,#1
          bne zanka1

          adr r0,TABELA
          mov r1,#8
          mov r2,#0
zanka2:   ldrb r3,[r0]
          add r2,r2,r3
```

```
          add r0,r0,#1
          subs r1,r1,#1
          bne zanka2

@Konec
__end:    b __end
```

28. Podana je tabela:

`tabela:  .byte  1,100,255,24,88,31,56,192,155,224,48,0,128,99,147,177`

Napišite program, ki v tabeli prešteje:

1. Vsa števila večja od 100 predznačeno.
2. Vsa števila večja od 100 nepredznačeno.
3. Vsa števila večja ali enaka 48 in manjša ali enaka 57.
4. Števila, ki so manjša od 50 ali večja od 100 nepredznačeno.
5. Vsa negativna števila.
6. Število ničel v tabeli.

Nalogo rešite s pomočjo zanke, ki se ponovi 16-krat. Rezultat shranite v 8-bitno spremenljivko REZ.

```
@Začetek
          .text

@Spremenljivke
tabela:  .byte  1,100,255,24,88,31,56,192,155,224,48,0,128,99,147,177
rez1:    .space 1
rez2:    .space 1
rez3:    .space 1
rez4:    .space 1
rez5:    .space 1
rez6:    .space 1

          .align
          .global __start
__start:
@Program
@Števila večja od 100 predznačeno
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka1:   ldrsb r2,[r0]
          cmp r2,#100
          addgt r3,r3,#1
```

```
          add r0,r0,#1
          subs r1,r1,#1
          bne zanka1
          strb r3,rez1

@Števila večja od 100 nepredznačeno
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka2:   ldrb r2,[r0]
          cmp r2,#100
          addhi r3,r3,#1

          add r0,r0,#1
          subs r1,r1,#1
          bne zanka2
          strb r3,rez2

@Števila večja ali enaka 48 in manjša ali enaka 57.
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka3:   ldrb r2,[r0]
          cmp r2,#48
          blo naprej3
          cmp r2,#57
          bhi naprej3
          add r3,r3,#1

naprej3:  add r0,r0,#1
          subs r1,r1,#1
          bne zanka3
          strb r3,rez3

@Števila, ki so manjša od 50 ali večja od 100 nepredznačeno
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka4:   ldrb r2,[r0]
          cmp r2,#50
          addlo r3,r3,#1
```

```
          cmp r2,#100
          addhi r3,r3,#1

          add r0,r0,#1
          subs r1,r1,#1
          bne zanka4
          strb r3,rez4

@Negativna števila
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka5:   ldrsb r2,[r0]
          cmp r2,#0
          addmi r3,r3,#1

          add r0,r0,#1
          subs r1,r1,#1
          bne zanka5
          strb r3,rez5

@Število ničel
          adr r0,tabela
          mov r1,#16
          mov r3,#0
zanka6:   ldrsb r2,[r0]
          cmp r2,#0
          addeq r3,r3,#1

          add r0,r0,#1
          subs r1,r1,#1
          bne zanka6
          strb r3,rez6

@Konec
__end:    b __end
```

29. V tabeli so zbrane višine študentov. Preštejte študente, ki so manjši od 170 cm, visoki med 170 cm in 190 cm in večji od 190 cm. Rezultate shranite v 8-bitne spremenljivke SK1, SK2 in SK3.

```
TABELA:   .byte   161,178,192,200,156,175,182,189,190,177
```

Štetje mora potekati v zanki (zanka ima 10 ponovitev).

```
@Začetek
          .text

@Spremenljivke
TABELA:  .byte  161,178,192,200,156,175,182,189,190,177
SK1:     .space 1
SK2:     .space 1
SK3:     .space 1

          .align
          .global __start
__start:
@Program
          adr r0,TABELA
          mov r1,#10
          mov r3,#0  @<170
          mov r4,#0  @>=170 && <= 190
          mov r5,#0  @>190

zanka:    ldrb r2,[r0]
          cmp r2,#170
          blo manjsi
          cmp r2,#190
          bhi vecji
          add r4,r4,#1
          b naprej
manjsi:   add r3,r3,#1
          b naprej
vecji:    add r5,r5,#1
naprej:   add r0,r0,#1
          subs r1,r1,#1
          bne zanka

          strb r3,SK1
          strb r4,SK2
          strb r5,SK3

@Konec
__end:    b __end
```

30. Napišite program, ki izračuna vsoto števil v tabeli in prešteje pozitivna števila, negativna števila in ničle v tabeli. Definirajte 8-bitne spremenljivke STPOZ, STNEG, STNIC, VSOTA v katere zapišite ustrezne rezultate.

```
TABELA:  .byte  -1,2,-3,1,-4,0,5,-3,0,10
```

Štetje in računanje vsote mora potekati v zanki (zanka ima 10 ponovitev).

```
@Začetek
          .text
@Spremenljivke
TABELA:  .byte  -1,2,-3,1,-4,0,5,-3,0,10
STPOZ:   .space 1
STNEG:   .space 1
STNIC:   .space 1
VSOTA:   .space 1
          .align
          .global __start
__start:
@Program
          adr r0,TABELA
          mov r1,#10
          mov r3,#0  @pozitivne
          mov r4,#0  @negativne
          mov r5,#0  @ničle
          mov r6,#0  @vsota

zanka:    ldrsb r2,[r0]
          cmp r2,#0
          addgt r3,r3,#1
          addlt r4,r4,#1
          addeq r5,r5,#1
          add r6,r6,r2

          add r0,r0,#1
          subs r1,r1,#1
          bne zanka

          strb r3,STPOZ
          strb r4,STNEG
          strb r5,STNIC
          strb r6,VSOTA

@Konec
__end:    b __end
```