Skup naredaba procesora ARM

Detaljan pregled svih naredbi

Sugestija

- Prilikom slušanja ovog predavanja ili ponavljanja obavezno pored sebe otvorite službeni šalabahter za naredbe ARM

Naredbe load i store

- Među najvažnijim naredbama za svaki procesor, pa tako i ARM, su naredbe za prijenos podataka između procesora i memorije (naredbe load/store).
- Naredbe Load čitaju podatak iz memorije i spremaju ga u registar dok naredbe Store spremaju podatak iz registra u memoriju.
- Ovo su jedine naredbe ARM-a kojima se može pristupiti podatku iz memorije !!!

Load Word	**LDR**{cond} Rd, <a_mode2>		Rd := [adresa]
Load branch (and exchange)	**LDR**{cond} R15, <a_mode2>		R15 := [adresa][31:1]
Load Byte	**LDR**{cond}B Rd, <a_mode2>		Rd := bajt s [adrese], proširen s 0
Load signed Byte	**LDR**{cond}SB Rd, <a_mode3>		Rd := bajt s [adrese], predznačno proširen
Load Halfword	**LDR**{cond}H Rd, <a_mode3>		Rd := poluriječ s [adrese], proširena s 0
Load signed Halfword	**LDR**{cond}SH Rd, <a_mode3>		Rd := poluriječ s [adrese], predznačno proširena
Load multiple Pop, or Block data load	**LDM**{cond}<a_mode4L> Rd{!}, <reglist-pc>		Učitaj listu registara sa [Rd]
Load multiple return (and exchange)	**LDM**{cond}<a_mode4L> Rd{!}, <reglist+pc>		Učitaj registre, R15 := [adresa][31:1]
Load multiple and restore CPSR	**LDM**{cond}<a_mode4L> Rd{!}, <reglist+pc>^		Učitaj registre, skoči, CPSR := SPSR
Store Word	**STR**{cond} Rd, <a_mode2>		[adresa] := Rd
Store Byte	**STR**{cond}B Rd, <a_mode2>		[adresa][7:0] := Rd[7:0]
Store Halfword	**STR**{cond}H Rd, <a_mode3>		[adresa][15:0] := Rd[15:0]
Store multiple Push, or Block data store	**STM**{cond}<a_mode4S> Rd{!}, <reglist>		Spremi listu registara na [Rd]
Store multiple User mode registers	**STM**{cond}<a_mode4S> Rd{!}, <reglist>^		Spremi listu registara korisničkog moda na [Rd]

Naredbe load i store

- Procesor ARM posjeduje dva osnovna tipa naredaba load/store:
  - **Prvi tip** naredaba može učitati ili upisati 32-bitnu riječ ili bajt bez predznaka.
  - Takve naredbe imaju općeniti format:

    LDR|STR{<cond>}{B} Rd, <addressing_mode>

  - gdje pojedine oznake znače:
    - {<cond>} polje uvjeta
    - {B} adresiranje bajta
    - Rd odredišni registar
    - <addressing_mode> opis načina adresiranja

Load / store

- **Drugi tip** može učitati ili upisati 16-bitnu poluriječ bez predznaka, a također se može učitati poluriječ ili bajt te ih predznačno proširiti do širine riječi.
- Ovakve naredbe imaju općeniti format:

  LDR|STR{<cond>}H|SH|SB Rd, <addressing_mode>

- gdje pojedine oznake znače:
  - {<cond>} polje uvjeta
  - H|SH|SB adresiranje poluriječi (H), predznačene poluriječi (SH) ili predznačenog bajta (SB)
  - Rd odredišni registar
  - <addressing_mode> opis načina adresiranja

Load / store

Ime naredbe	Engleski naziv
LDR	Load Word
LDRB	Load Unsigned Byte (zero extend)
LDRH	Load Unsigned Halfword (zero extend)
LDRSB	Load Signed Byte (sign extend)
LDRSH	Load Signed Halfword (sign extend)
STR	Store Word
STRB	Store Byte
STRH	Store Halfword

Load/store: Bazni registar

- Za sve naredbe load/store se adresa memorije izračunava korištenjem dva dijela:
  - vrijednost u nekom baznom registru
  - odmak (offset) u odnosu na vrijednost u baznom registru

- Bazni registar može biti bilo koji registar opće namjene*

* Ako se kao bazni registar izabere PC, tada se može postići relativno adresiranje u odnosu na trenutačnu poziciju u programu, te na taj način i izvedba programa koji su potpuno neovisni o položaju u memoriji. Kod direktnog kodiranja (bez pomoći asemblera i korištenja labela) programer mora paziti na vrijednost PC-a u trenutku izvođenja naredbe

Load/store: Odmak (offset)

- Za pojedinu naredbu programer može izabrati jedan od tri formata odmaka:
  - U najjednostavnijem formatu odmak se definira kao **broj, odnosno neposredna vrijednost** (immediate) koji se izravno upisuje u kôd naredbe. Neposredna vrijednost zapisana je jednim bitom predznaka i iznosom odmaka od 12 ili 8 bitova (ovisno o naredbi). 12-bitnim odmakom se može adresirati memorijska lokacija odmaknuta za +/- 4095 mjesta, a 8-bitnim odmaknuta za +/- 255 mjesta u odnosu na vrijednost izabranog baznog registra
  - Odmak se može definirati i pomoću **vrijednosti iz registra opće namjene***
  - Treća mogućnost je definiranje odmaka pomoću **vrijednosti registra opće namjene* koja je još pomaknuta ulijevo ili udesno**

* osim registra PC

Load/store...

- Osim prethodno opisana tri načina za definiciju odmaka, procesor ARM omogućuje još tri različite inačice adresiranja memorije.
- Ove inačice zadaju treba li i kako mijenjati bazni registar tijekom izvođenja naredbe load/store. Ove tri inačice su:

- **Osnovni odmak**
  - Adresa se izračuna zbrajanjem ili oduzimanjem odmaka od baznog registra, a zatim se pristupi memoriji. Vrijednost baznog registra ostaje nepromijenjena:

    Reg $\Leftrightarrow$ mem[BazniReg + Odmak]

Load /store

- **Predindeksiranje**
  - Odmak se zbroji ili oduzme od baznog registra, a izračunata vrijednost se spremi natrag u bazni registar. Zatim se pristupi memoriji.

    BazniReg = BazniReg + Odmak

    Reg ⇔ mem[BazniReg]

- **Postindeksiranje**
  - Memoriji se pristupi samo na temelju vrijednosti baznog registra. Tek nakon obavljenog prijenosa, odmak se zbroji ili oduzme od baznog registra, a izračunata vrijednost se spremi natrag u bazni registar.

    Reg ⇔ mem[BazniReg]

    BazniReg = BazniReg + Odmak

Load/store - Rekapitulacija

- Osnovni odmak:

  Reg <=> mem[BazniReg + Odmak]

- Predindeksiranje:

  BazniReg = BazniReg + Odmak

  Reg <=> mem[BazniReg]

- Postindeksiranje:

  Reg <=> mem[BazniReg]

  BazniReg = BazniReg + Odmak

Pregled adresiranja u naredbama load/store

	Osnovni odmak	Predindeksiranje	Postindeksiranje
Neposredni	[R0, #4]	[R0, #4]!	[R0], #4
Registarski	[R7, -R3]	[R7, R3]!	[R7], R3
Registarski s pomakom	[R3, R5, LSL #2]	[R3,R5,LSL #2]!	[R3],R5,LSL #2

Primjer (8-bitno zbrajanje)

Treba napisati program za zbrajanje dvaju 8-bitnih brojeva bez predznaka koji su zapisani u memoriji odmah iza programa. Rezultat treba spremiti u memoriju iza operanada.

```
;Program za 8 bitno zbrajanje dva broja bez predznaka, ver.1
        LDRB R0, A              ; prvi operand se ucitava u R0 (za
                                ; signed se koristi LDRSB)
        LDRB R1, B              ; drugi operand se ucitava u R1 (za
                                ; signed se koristi LDRSB)
        ADDS R4, R0, R1     ; izvodi se zbrajanje i postavljaju se
                                ; zastavice (S)
        STRB R4, REZ        ; rezultat se sprema u memoriju
        SWI 123456

A       DB %D 100               ; prvi operand (oktet)
B       DB 20               ; drugi operand (oktet)
REZ     DS 1                ; rezultat
```

Naredbe Load Multiple i Store Multiple

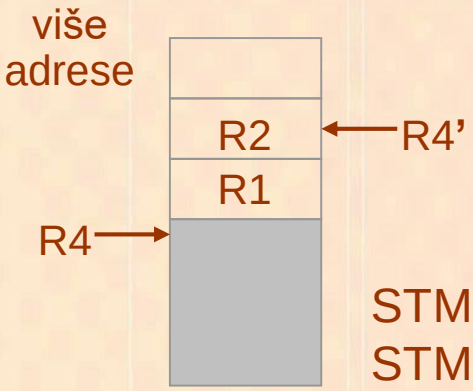
- Pored osnovnih naredaba load i store, koje obavljaju prijenos podataka samo iz jednog registra, ARM ima dvije naredbe (Load Multiple i Store Multiple) koje programeru omogućuju da jednom naredbom obavi prijenos podataka između memorije i bilo kojeg podskupa registara ili čak svih registara.

- Osnovno ime naredbe je LDM i STM nakon čega se stavlja neki od nastavaka kojima se opisuje kako se sprema niz podataka (tj. niz registara) u memoriju

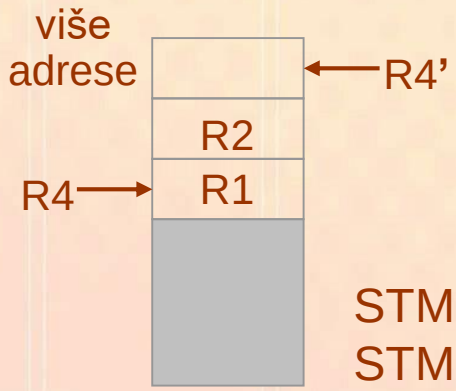
LDM/STM: načini adresiranja

- Pri pisanju (čitanju) više podataka u memoriju mora se definirati gdje će biti zapisan prvi te svaki sljedeći podatak.
- Da bi definirali gdje će se zapisati prvi podatak, moramo izabrati neki registar opće namjene koji pokazuje na početak memorijskog područja u koje će se upisivati podaci.
- Nakon toga moramo izabrati jednu od četiri kombinacije načina adresiranja niza: IB, IA, DB ili DA. Ove kratice znače:
  - IB - Increment Before (uvećaj prije)
  - IA - Increment After (uvećaj poslije)
  - DB - Decrement Before (smanji prije)
  - DA - Decrement After (smanji poslije)

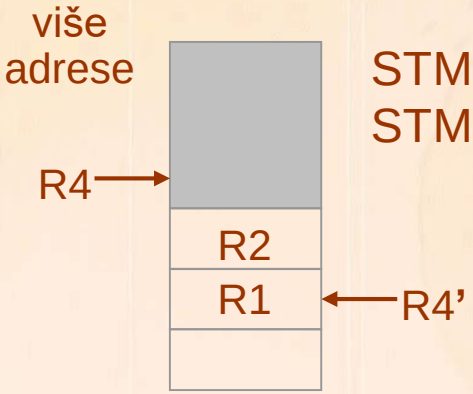
LDM/STM: načini adresiranja



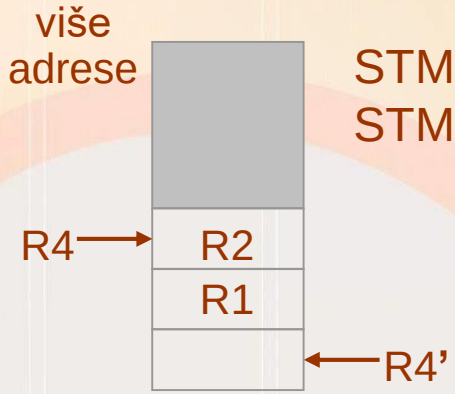
STMIB R4!, {R1,R2}
STMFA R4!, {R1,R2}



STMIA R4!, {R1,R2}
STMEA R4!, {R1,R2}



STMDB R4!, {R1,R2}
STMFD R4!, {R1,R2}



STMDA R4!, {R1,R2}
STMED R4!, {R1,R2}

R4 = stanje prije naredbe R4' = stanje poslije naredbe

LDM/STM

Naredbe s normalnim nastavcima	Ekvivalentne naredbe za rad sa stogom
LDMIA	LDMFD
LDMIB	LDMED
LDMDA	LDMFA
LDMDB	LDMEA
STMIA	STMEA
STMIB	STMFA
STMDA	STMED
STMDB	STMFD

ldm/stm - adresiranje

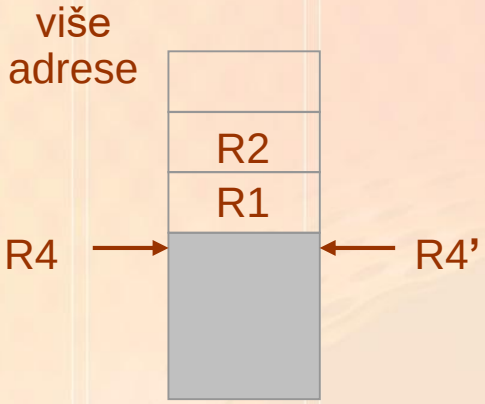
- Još jedna mogućnost koja se pruža programeru je automatsko pomicanje pokazivača, odnosno osvježavanje vrijednosti koja se nalazi u registru koji služi za adresiranje.

- Tako se na primjer naredbe

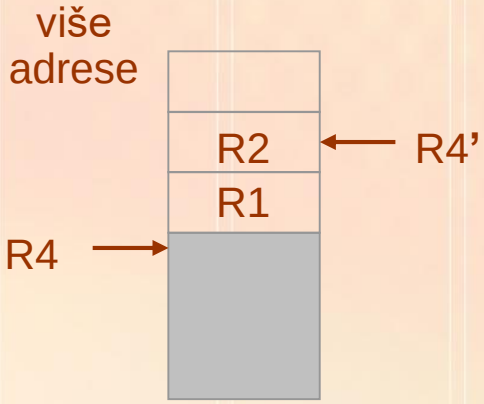
```
LDMIB R2, {R4,R8,R9}
LDMIB R2!, {R4,R8,R9}        ; iza R2 piše se uskličnik !
```

razlikuju po tome što će nakon izvođenja prve naredbe vrijednost registra R2 ostati **nepromijenjena**, dok će nakon druge naredbe registar R2 biti **promijenjen** (uvećan za 12_{10}, jer se R2 uvećava za 4 prije čitanja podatka za svaki registar iz niza)

Ldm/stm korištenje



STMIB R4, {R1,R2}



STMIB R4**!**, {R1,R2}

R4 = stanje prije naredbe R4**'** = stanje poslije naredbe

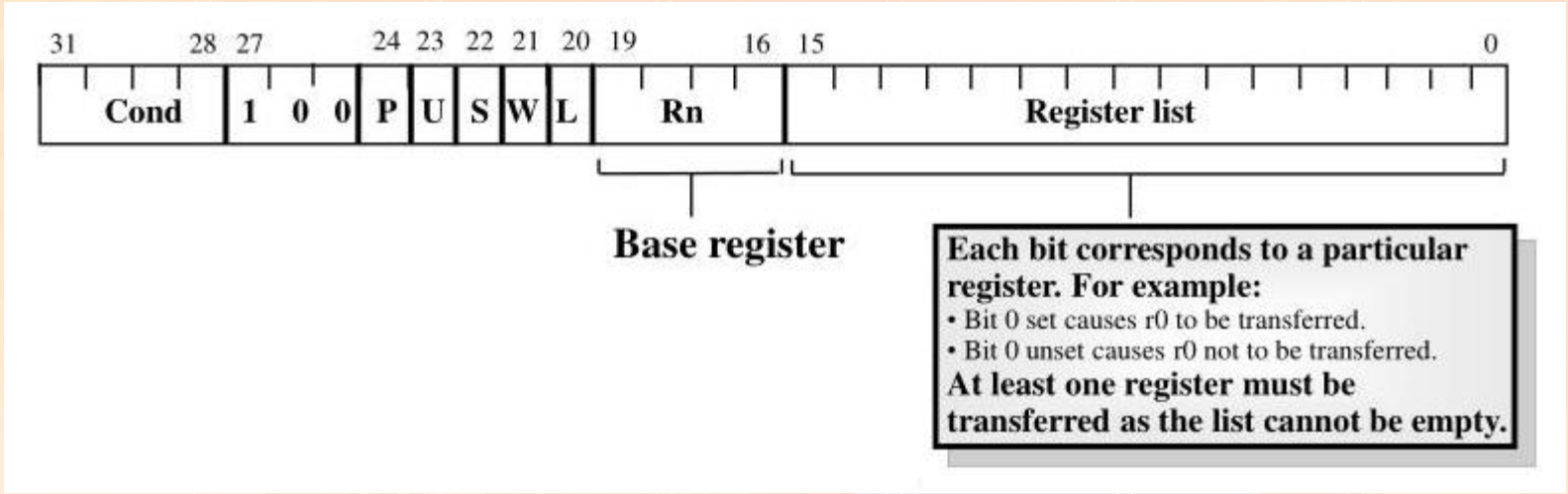
Ldm/stm PRAVILA!!!

- Bez obzira na izbor nekog od četiri adresiranja i bez obzira na redoslijed kojim su registri napisani u naredbi, **uvijek je na najnižoj adresi zapisan registar sa najnižim brojem**

  - **LDMIA R4, {R1,R8,R3,R12}**
  - **LDMIA R4, {R12,R3,R8,R1}**
  - **LDMIA R4, {R1,R3,R8,R12}**

- **Bilo koja od ovih gore naredaba (ili neki drugi redoslijed) DATI ĆE ISTI KOD NAREDBE I ISTI REZULTAT**

LDM/STM … popis registara



Ldm/stm PRAVILA!!!

- Ako se podaci žele zapisati u memoriju naredbom STM, a zatim pročitati u istom redoslijedu naredbom LDM te ako se koriste nastavci za OBIČNO ADRESIRANJE (ne ekvivalenti za stog!!!), tada način adresiranja u naredbi LDM mora biti **INVERZAN** načinu adresiranja u naredbi STM
- Primjer:
  - Zapisivanje STMIA
  - Čitanje LDMDB

U naredbama se koristi inverzno adresiranje IA ↔DB

Ldm/stm PRAVILA!!!

- Ako se podaci žele zapisati u memoriju naredbom STM, a zatim pročitati u istom redoslijedu naredbom LDM te ako se koriste nastavci za RAD SA STOGOM, tada način adresiranja u naredbi LDM mora biti **ISTI** načinu adresiranja u naredbi STM

- Primjer:
  - Zapisivanje STMFD
  - Čitanje LDMFD

U obje naredbe koristi se jednako adresiranje FD

Primjeri korištenja LDM i STM ...

U donjim primjerima pretpostavljene vrijednosti prije izvođenja naredaba LDM/STM su:
r0=0 r1=1 r2=2 r3=3 r13=9000

a) STMIB r13, {r0,r1,r2,r3} ; vrijednosti se spreme, r13 ostane nepromijenjen

Nakon izvođenja gornje naredbe, stanje u memoriji je (heksadekadski, little-endian):
0x00009000: 10 00 FF E7 00 00 00 00 01 00 00 00 02 00 00 00
0x00009010: 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8
r13=9000

b) STMIB r13, {r3,r1,r0,r2} ; redoslijed pisanja registara ne utječe na redoslijed spremanja

0x00009000: 10 00 FF E7 00 00 00 00 01 00 00 00 02 00 00 00
0x00009010: 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8
r13=9000

c) STMIB r13!, {r0,r3} ; osvjezava se r13

0x00009000: 10 00 FF E7 00 00 00 00 03 00 00 00 E8 00 E8 00
r13=9008

… Primjeri korištenja LDM i STM …

d) STMDB r13!, {r0,r1,r2,r3} ; primjer koristenja DB

0x00008FF0: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
0x00009000: 10 00 FF E7 00 E8 00 E8 10 00 FF E7 00 E8 00 E8
r13 = 8FF0

e) STMDA r13!, {r0,r1,r2,r3} ; primjer koristenja DA i kasnijeg obnavljanja registara

;0x00008FF0 00 E8 00 E8 00 00 00 00 01 00 00 00 02 00 00 00
;0x00009000 03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8
;r13 = 8FF0

Da bi u registre r0, r1, r2 i r3 učitali iste podatke koje smo iz njih spremili u memoriju pomoću naredbe STMDA, moramo upotrijebiti naredbu LDM s inverznim nastavkom IB:

LDMIB r13!, {r0,r1,r2,r3}

Primjeri korištenja LDM i STM

```
f)   STMFA r13!, {r0,r1,r2,r3}        ; primjer spremanja registara pomoću STM i kasnijeg
                                      ; obnavljanja registara naredbom LDM uz korištenje
                                      ; nastavaka za STOG
```

```
0x00009000:  00 E8 00 E8 00 00 00 00 01 00 00 00 02 00 00 00
0x00009010:  03 00 00 00 00 E8 00 E8 10 00 FF E7 00 E8 00 E8
r13 = 9010
```

Da bi u registre r0, r1, r2 i r3 učitali iste podatke koje smo iz njih spremili u memoriju pomoću naredbe STMFA, moramo upotrijebiti naredbu LDM s istim nastavkom FA:

```
     LDMFA r13!, {r0,r1,r2,r3}        ; za adresiranje stoga nastavci moraju biti isti
```

Naredbe za obradu podataka

- ARM-ove naredbe za obradu podataka obrađuju podatke iz registara
- Postoje tri grupe naredaba za obradu podataka:
  - aritmetičko-logičke naredbe i naredbe za usporedbu
  - naredbe za množenje
  - naredba za brojenje vodećih nula
- Za razliku od FRISC-a kod kojeg se odredišni registar piše na kraju - iza operanada, kod ARM-a se odredišni registar zadaje na početku - prije operanada, kao kod normalnog pisanja jednadžbi

```
ADD R1, R2, R3   R1=R2+R3
```

Aritmetičko-logičke naredbe i naredbe za usporedbu

Ime	Engleski naziv	Opis naredbe
AND	Logical AND	Rd := Rn AND drugi_operand (logičko I)
EOR	Logical Exclusive OR	Rd := Rn EOR drugi_operand (logičko ekskl. ILI)
ORR	Logical (inclusive) OR	Rd := Rn OR drugi_operand
BIC	Bit Clear	Rd := Rn AND NOT(drugi_operand)
ADD	Add	Rd := Rn + drugi_operand
ADC	Add with Carry	Rd := Rn + drugi_operand + C zastavica
SUB	Subtract	Rd := Rn - drugi_operand
SBC	Subtract with Carry	Rd := Rn - drugi_operand - NOT(C zastavica)
RSB	Reverse Subtract	Rd := drugi_operand - Rn
RSC	Reverse Subtract with Carry	Rd := drugi_operand - Rn - NOT(C zastavica)
CMP	Compare	Osvježi zastavice nakon Rn - drugi_operand
CMN	Compare Negated	Osvježi zastavice nakon Rn + drugi_operand
TST	Test	Osvježi zastavice nakon Rn AND drugi_operand
TEQ	Test Equivalence	Osvježi zastavice nakon Rn EOR drugi_operand
MOV	Move	Rd := drugi_operand (prvog operanda nema)
MVN	Move Not	Rd := NOT drugi_operand (prvog operanda nema)

Aritmetičko-logičke naredbe

- Aritmetičko-logičke naredbe imaju dva operanda (osim naredaba MOV i MVN koje imaju samo jedan operand) i spremaju rezultat u zadani registar* (osim naredaba za usporedbu koje zanemaruju rezultat)
- Od dva operanda koji se mogu koristiti u operaciji jedan uvijek mora biti registar, a drugi može biti
  - neposredna vrijednost
  - registar
  - vrijednost registra nad kojom se obavlja određen pomak

Aritmetičko-logičke naredbe

- Ako se aritmetičko-logičkoj naredbi doda nastavak 'S' (Save condition codes) tada će se nakon izvršene naredbe osvježiti zastavice stanja
- Bez nastavka 'S', naredba **ne mijenja** zastavice*
- Naredbe za usporedbu uvijek osvježavaju zastavice stanja (i nigdje ne spremaju rezultat)

* Različito od FRISC-a, koji uvijek osvježava zastavice

Primjer programa za aritm. naredbe

64-bitno oduzimanje

Treba napisati program za oduzimanje dvaju 64-bitnih brojeva. Operandi neka su zapisani u memoriji s početkom na adresi 8100_{16}. Rezultat treba spremiti u memoriju iza operanada. Za dohvat operanada i spremanje rezultata koristiti naredbu Load/Store Multiple.

Rješenje:

```
MOV R4, #81<8                    ; postavlja registar R4 na 8100(16)
        ; ovaj način zadavanja konstante biti će detaljno opisan kasnije
LDMIA R4!,{R5,R6,R7,R8}          ; ucitavanje oba operanda
                                 ; koristenjem Load Multiple naredbe
SUBS R5, R5, R7                  ; oduzimanje niza 32 bita
                                 ; s postavljanjem zastavica
SBCS R6, R6, R8                  ; oduzimanje visa 32 bita i
                                 ; posudbe iz prethodne operacije
STMIA R4!, {R5,R6}       ; spremanje rezultata
SWI 123456
```

Naredbe za množenje

- Naredbe za množenje nisu smještene u grupu osnovnih aritmetičko-logičkih naredbi, već se definiraju zasebno
- Postoji šest naredbi za množenje: MUL, MLA, SMULL, UMULL, SMLAL i UMLAL
  - **U ATLAS-u su implementirane samo naredbe MUL i MLA!!!**
- Obično množenje (MUL) se obavlja nad dva 32-bitna podatka iz registara opće namjene, a 32-bitni rezultat (nižih 32 bita umnoška) se ponovo sprema u registar
- Naredbe za dugo množenje (SMULL i UMULL) množe dva 32-bitna podatka iz registara i spremaju svih 64 bita rezultata u dva odabrana registra za rezultat. Dugo množenje može množiti brojeve s predznakom (SMULL) ili bez predznaka (UMULL)

Naredbe za množenje

- Obično 32-bitno množenje koristimo kad znamo da su operandi "dovoljno mali" da rezultat stane u 32-bita, a dugo 64-bitno množenje koristimo u općenitim slučajevima

- Kao dodatna opcija gore navedenim naredbama nudi se mogućnost da se umnožak pribraja sadržaju nekog registra (ili paru registara za dugo množenje). Takve naredbe općenito su poznate kao Multiply Accumulate, a kod procesora ARM nazivaju se: MLA, SMLAL i UMLAL

- Ako se naredbama za množenje doda nastavak S, tada će se nakon izvođenja osvježiti zastavice N i Z.

Naredbe za množenje

Ime	Engleski naziv	Opis naredbe
MUL	Multiply	Rd = (Rm * Rs)[31:0]
MLA	Multiply Accumulate	Rd = (Rm * Rs + Rn)[31:0]
SMULL	Signed Multiply Long	RdHi = (Rm * Rs)[63:32] /*Sa predznakom*/ RdLo = (Rm * Rs)[31:0]
UMULL	Unsigned Multiply Long	RdHi = (Rm * Rs)[63:32] /*Bez predznaka*/ RdLo = (Rm * Rs)[31:0]
SMLAL	Signed Multiply Accumulate Long	RdLo=(Rm * Rs)[31:0]+RdLo /*Sa predznakom*/ RdHi = (Rm * Rs)[63:32] + RdHi + prijenos iz ((Rm * Rs)[31:0] + RdLo)
UMLAL	Unsigned Multiply Accumulate Long	RdLo=(Rm * Rs)[31:0]+RdLo /*Bez predznaka*/ RdHi = (Rm * Rs)[63:32] + RdHi + prijenos iz ((Rm * Rs)[31:0] + RdLo)

Naredbe za množenje

Multiply	**MUL**{cond}{S} Rd, Rm, Rs	N	Z			Rd := (Rm * Rs)[31:0]
Multiply accumulate	**MLA**{cond}{S} Rd, Rm, Rs, Rn	N	Z			Rd := ((Rm * Rs) + Rn)[31:0]
Multiply unsigned long	**UMULL**{cond}{S} RdLo, RdHi, Rm, Rs	N	Z			RdHi,RdLo := nepredznačno(Rm*Rs)
Multiply unsigned accumulate long	**UMLAL**{cond}{S} RdLo, RdHi, Rm, Rs	N	Z			RdHi,RdLo := nepredznačno(RdHi,RdLo+Rm*Rs)
Multiply signed long	**SMULL**{cond}{S} RdLo, RdHi, Rm, Rs	N	Z			RdHi,RdLo := predznačno(Rm*Rs)
Multiply signed accumulate long	**SMLAL**{cond}{S} RdLo, RdHi, Rm, Rs	N	Z			RdHi,RdLo := predznačno(RdHi,RdLo+Rm*Rs)
Compare	**CMP**{cond} Rn, <Oprnd2>	N	Z	C	V	Rn - Oprnd2; osvježi CPSR zastavice

Primjer programa za množenje (bez korištenja naredaba za množenje)

Program za množenje dvaju 16-bitnih brojeva treba napisati metodom zbrajanja. Neka su multiplikator i multiplikand zapisani u memoriji s početkom na adresi 8100_{16}. Pretpostavite da je multiplikator broj bez predznaka dok multiplikand može biti s predznakom. Rezultat treba spremiti u memoriju iza operanada.

```
; Uvjet zadatka: multiplikator je broj bez predznaka
        MOV R4, #81<8                ; postavlja R4 adresu podataka
        LDRH R5, [R4], #2            ; multiplikator se ucitava u R5
        LDRSH R6, [R4], #2     ; multiplikand se ucitava u R6 (predznak sacuvan)
        MOV R7, #0             ; cisti se registar za  spremanje rezultata (R7)

PETLJA  CMP R5, #0             ; usporedba multiplikatora i nule
        BEQ KRAJ               ; ako je nula, mnozenje je gotovo
        ADD R7, R7, R6         ; pribroji multiplikand rezultatu.
        SUB R5, R5, #1         ; umanji multiplikator za jedan
        B PETLJA
KRAJ    STR R7, [R4]           ; spremi rezultat
        SWI 123456
```

Naredba za brojenje vodećih nula

- Naredba za brojenje vodećih nula je specifična naredba koja je vrlo korisna kod izvedbe matematičkih algoritama (normiranje brojeva) i algoritama za kompresiju (npr. metode duljine niza).
  - **U ATLAS-u nije implementirana naredba CLZ!!!**
- Ova naredba uzima jedan registar kao operand i vraća rezultat u drugom registru.
- Rezultat predstavlja broj nula koje prethode najvišoj jedinici u ulaznom operandu (promatranom kao niz binarnih znamenaka).

CLZ

- Primjer korištenja naredbe za brojenje vodećih nula:

  Neka je u registru R1=**00000**11110001111011111111100011

  Nakon izvođenja naredbe:

```
CLZ  R2, R1
```

  u registru R2 bit će rezultat **5**

Naredbe grananja

- Naredba B (Branch): bezuvjetno ili uvjetno grananje na memorijsku adresu koja se nalazi 32 MB ispred ili iza trenutačne naredbe* (relativan skok u odnosu na sadržaj PC)

- Drugi način grananja je da se u registar PC izravno stavi neka vrijednost, čime se skok ne ograničava na udaljenost od 32 MB, već se može skočiti na bilo koju adresu u adresnom području (tj. može se skočiti bilo gdje unutar 4 GB) **

---

* Slično FRISC-ovoj naredbi `JR`

** Sličan učinak kod FRISC-a možemo dobiti naredbom `JP (Rx)`

Naredbe grananja - uvjeti

Mnemonički naziv	Puni naziv (engleski)	Ispitivano stanje 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	Minus/negative	N
PL	Plus/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 (unconditional)	-
(NV)	See Condition code 0b1111	-

Grananje u potprogram

- Naredba BL (Branch and Link) poziva potprogram:
  - Sprema adresu sljedeće naredbe (povratnu adresu) **u registar R14** (koji se tada naziva Link Registar, LR)
  - nakon toga izvodi grananje na početak potprograma.
- Povratak iz potprograma izvodi se jednostavnim kopiranjem sadržaja registra LR u PC (npr. **naredbom MOV PC,LR**)
- Koji problemi se mogu pojaviti ?

- Ovakvim pozivom i povratkom iz potprograma nije moguće ugniježđeno pozivanje potprograma!
- Za ugniježđene pozive, na početku svakog potprograma mora se spremiti vrijednost iz R14 na stog, a prije povratka iz potprograma treba vratiti tu vrijednost sa stoga u R14.

Primjeri naredaba za grananje

```
B      labela      ; bezuvjetni skok
BCC    labela      ; uvjetni skok (carry clear)
BEQ    labela      ; uvjetni skok (equal)

MOV    PC, #0      ; R15 = 0, tj. skoči na adresu 0
MOV    PC, R3      ; R15 = R3, skok na bilo koju 32-bitnu adresu
                   ; koja je zadana registrom R3

BL   func          ; poziv potprograma
MOV  PC, LR        ; R15=R14, tj. povratak iz potprograma
```

Primjer uvjetnog izvođenja niza naredaba

1. način - korištenjem naredbe uvjetnog grananja:

```
    CMP R0, #0
    BNE DALJE
    MOV R1, #1      ; uvjetni dio koda
    MOV R2, #2      ; uvjetni dio koda
    MOV R3, #3      ; uvjetni dio koda
DALJE
    . . .
```

Ovisno o protočnoj strukturi i ovisno koliko često je ispitivani uvjet istinit, može se odrediti koji način pisanja je efikasniji u pojedinoj situaciji.

2. način - korištenjem uvjetnog izvođenja naredaba:

```
    CMP R0, #0
    MOVEQ R1, #1     ; uvjetni dio koda
    MOVEQ R2, #2     ; uvjetni dio koda
    MOVEQ R3, #3     ; uvjetni dio koda
```

Naredbe za pristup registrima stanja

- Omogućuju prijenos podataka iz registara stanja (CPSR, SPSR) procesora ARM u neki registar opće namjene i obratno.
- Pisanjem u CPSR, na primjer, programer može postaviti stanja zastavica, stanja bitova za omogućavanje prekida kao i procesorski način.
- Naredba **MRS** (Move to Register from Status register) kopira sadržaj registra stanja CPSR ili SPSR iz načina rada u kojem se procesor trenutano nalazi u jedan od registara opće namjene koji se može dalje ispitivati ili mijenjati.
- Naredbom **MSR** Move to Status register from Register može se upisati neposredna vrijednost ili sadržaj registra opće namjene u registre stanja CPSR ili SPSR iz načina rada u kojem se procesor trenutno nalazi.

Naredbe za pristup registrima stanja

Oznaka	Bitovi	Naziv polja u registru
f	[31:24]	Polje zastavica (flags)
s	[23:16]	Polje stanja (status)
e	[15:8]	Polje proširenja (extension)
c	[7:0]	Polje upravljanja (control)

- Svaki registar stanja podijeljen je u 4 polja kojima se može neovisno pristupati. Prilikom upisa u registar stanja naredbom MSR, programer mora definirati u koje polje želi upisati podatak
- Primjer naredaba:
  - MRS R0, CPSR
  - MSR CPSR_cesf, R0
  - MSR CPSR_f, R0
  - MSR CPSR_fsc, R0

Primjer korištenja

- primjer omogućavanja prekida (brisanje bita I u registru CPSR)

```
MRS  R0, CPSR
BIC  R0, R0, #0x80
MSR  CPSR_C, R0
```