ARM Adresiranja i potprogrami

Sugestija

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

Načini adresiranja

- Procesor ARM omogućuje različita adresiranja. Oblici adresiranja svrstani su u pet načina (engl. Addressing Modes) i u tablicama u Prilogu su označeni brojevima 1 do 5.
- Načini adresiranja koriste se u pojedinim naredbama prema sljedećoj tablici:

  - načini adresiranja 1: naredbe za obradu podataka
  - načini adresiranja 2: naredbe Load i Store word ili unsigned byte
  - načini adresiranja 3: naredbe Load i Store halfword ili signed byte
  - načini adresiranja 4: naredbe Load i Store Multiple
  - načini adresiranja 5: naredbe Load i Store Coprocessor

Načini adresiranja 1

- Načini adresiranja 1 su adresiranja u "širem smislu" jer se u njima ne dohvaća podatak iz memorije (ne koristi se adresa)
- Koriste se za **definiranje jednog od operanada** u **naredbama za obradu podataka**. Osnovni oblik naredaba koje koriste taj način adresiranja izgleda ovako:

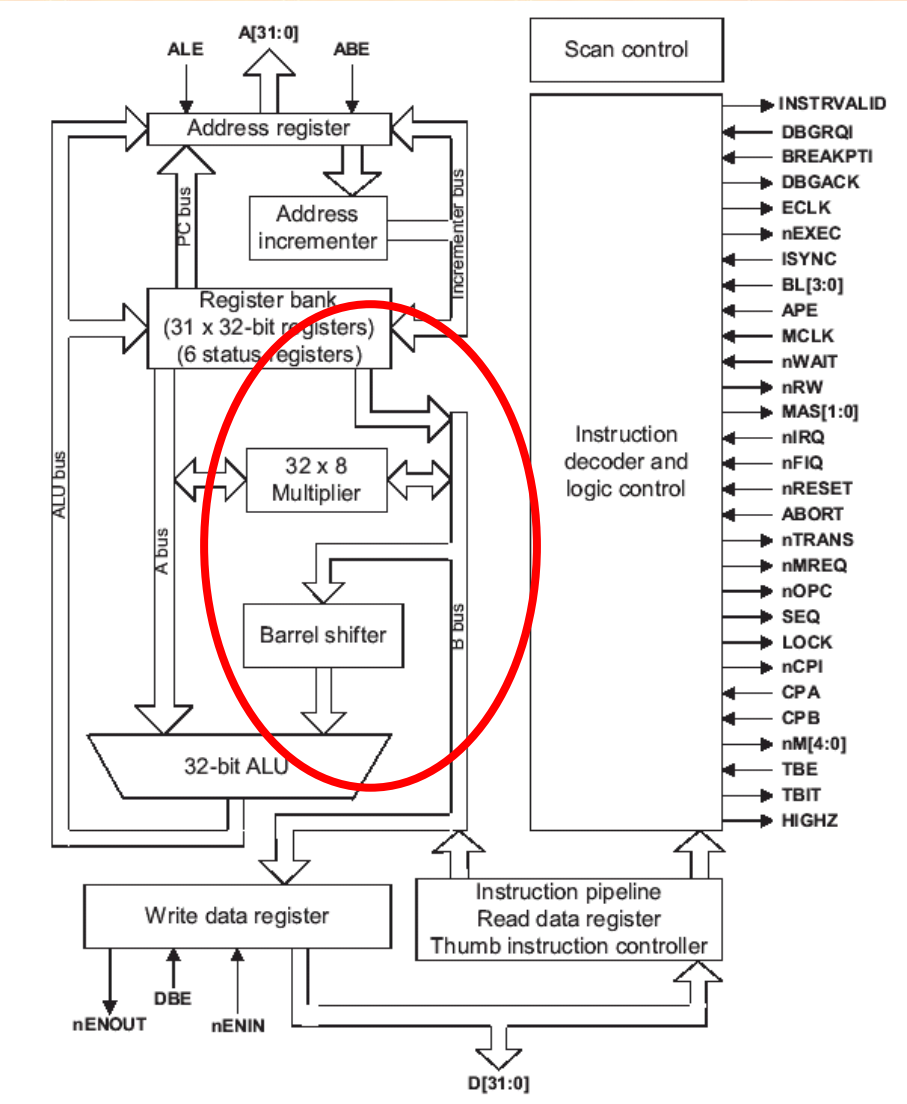
<opcode>{<cond>}{S} <Rd>, <Rn>, <shifter_operand>

- Drugi operand <shifter_operand> može se "adresirati" na 11 različitih načina koji tvore načine adresiranja 1

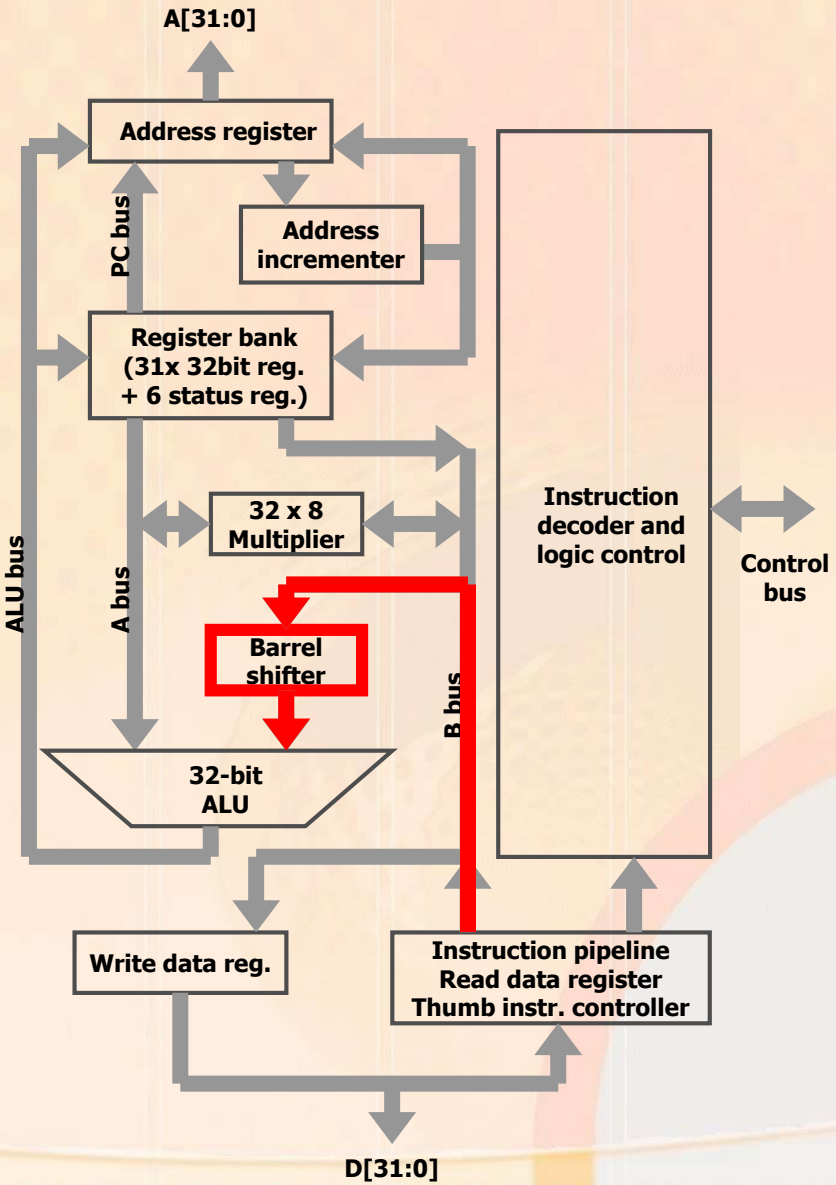
Načini adresiranja 1

- Već smo prije naučili da drugi operand može biti
  - Neposredna vrijednost
  - Vrijednost iz registra
  - Vrijednost iz registra s pomakom

- Sve ove mogućnosti zasnivaju se na arhitekturi puta podataka prema ALU

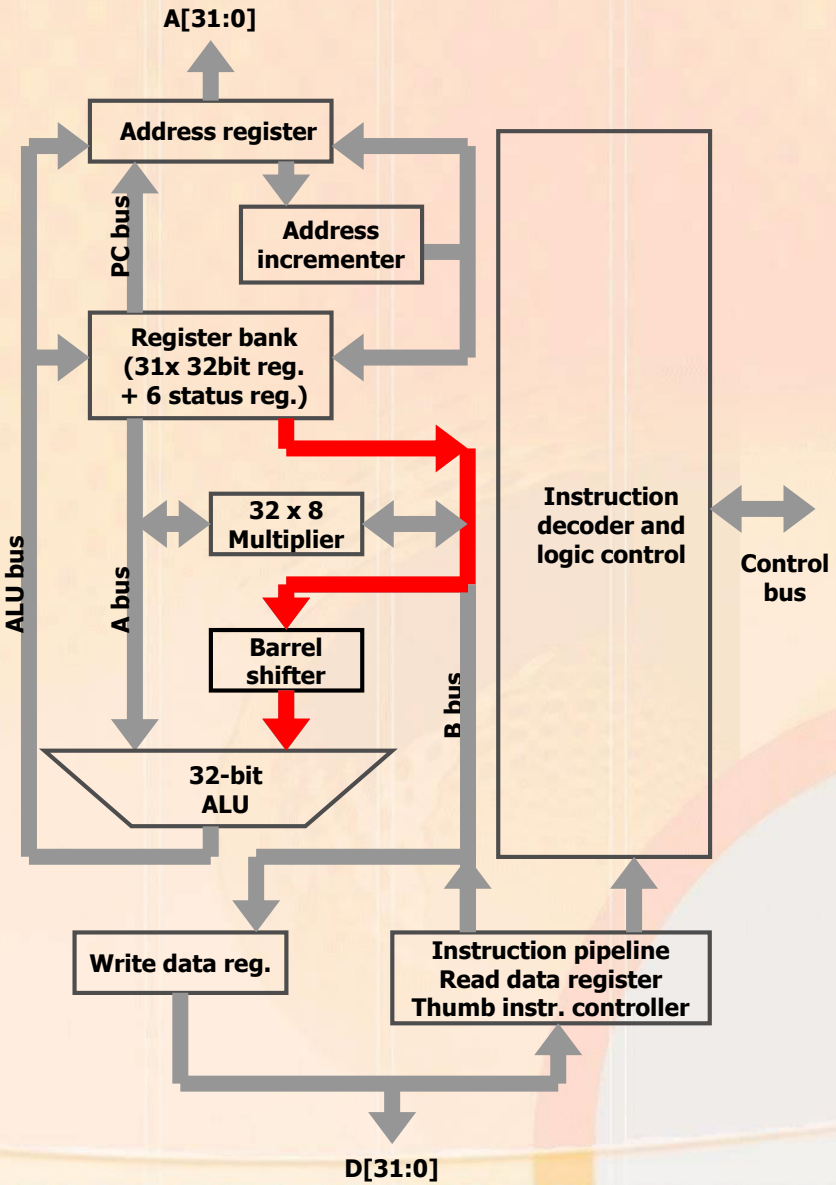


Podloga za način1 u arhitekturi procesora



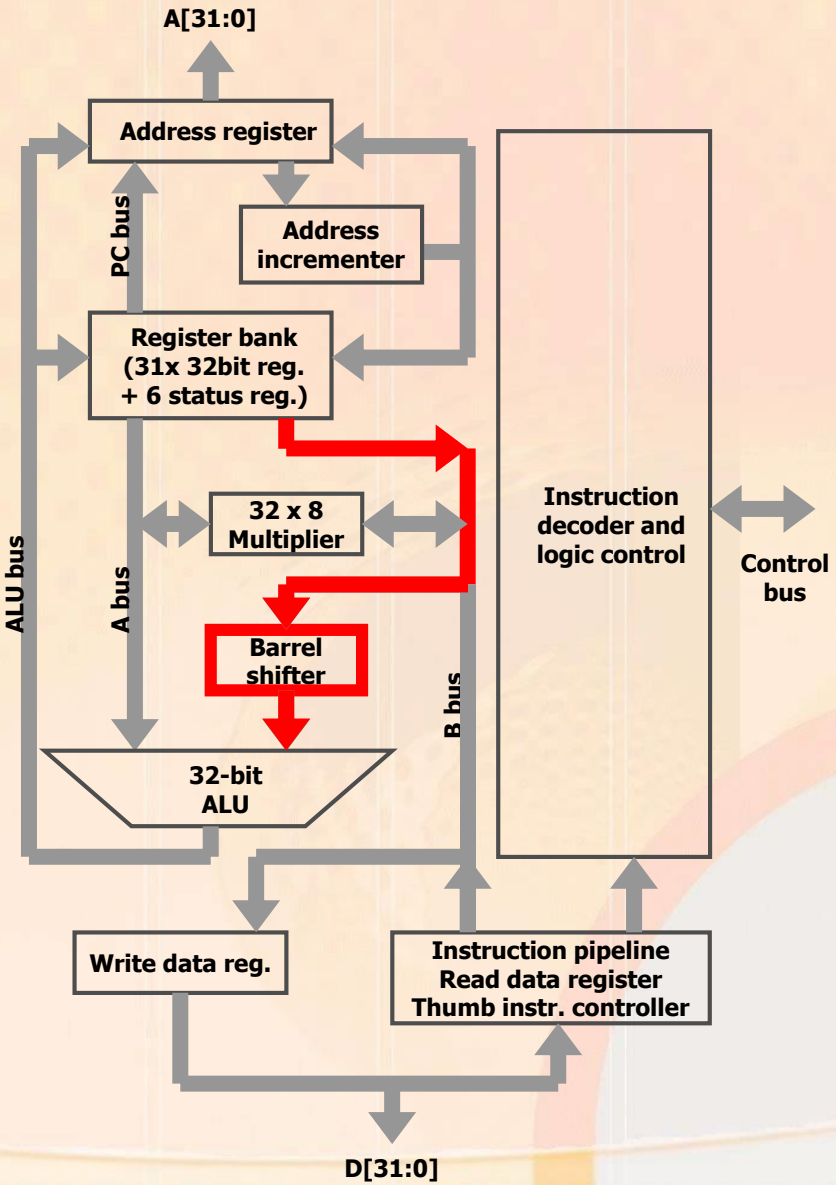
Neposredna vrijednost

Podloga za način1 u arhitekturi procesora



Vrijednost iz registra

Podloga za način1 u arhitekturi procesora



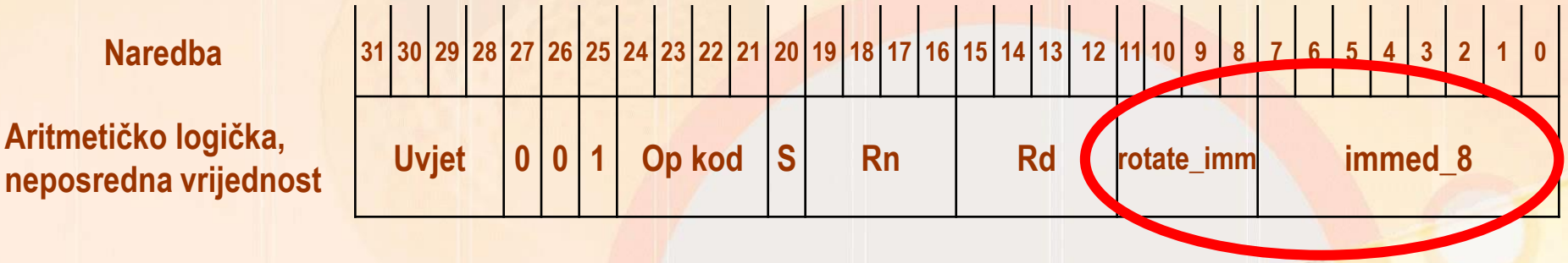
Vrijednost iz registra s pomakom

Načini adresiranja 1

Adresiranje	Sintaksa adresiranja
Neposredno	#<immediate>
Registarsko	<Rm>
Registarsko s neposrednim logičkim pomakom ulijevo (ASL je sinonim za LSL)	<Rm>,LSL #<rshift_imm> <Rm>,ASL #<shift_imm>
Registarsko s registarskim logičkim pomakom ulijevo (ASL je sinonim za LSL)	<Rm>,LSL <Rs> <Rm>,ASL <Rs>
Registarsko s neposrednim logičkim pomakom udesno	<Rm>,LSR #<shift_imm>
Registarsko s registarskim logičkim pomakom udesno	<Rm>,LSR <Rs>
Registarsko s neposrednim aritmetičkim pomakom udesno	<Rm>,ASR #<shift_imm>
Registarsko s registarskim aritmetičkim pomakom udesno	<Rm>,ASR <Rs>
Registarsko s neposrednim rotiranjem udesno	<Rm>,ROR #<shift_imm>
Registarsko s registarskim rotiranjem udesno	<Rm>,ROR <Rs>
Registarsko s proširenim rotiranjem udesno	<Rm>,RRX

Neposredno #<immediate>

- Neposredna 32-bitna konstanta
- Za zapis konstante u strojnom kôdu naredbe na raspolaganju je samo 12 bitova !!!
- poseban način računanja konstanti:
  - od 12 bitova koji su na raspolaganju, 8 ih se koristi za neposrednu vrijednost (immed_8), a preostala 4 bita (rotate_imm) definiraju broj rotacija udesno 8-bitne neposredne vrijednosti:



Definiranje neposredne vrijednosti

- Neposredna 32-bitna vrijednost računa se sljedećom formulom:
  - <immediate> = <immed_8> rotirano udesno za (2 *<rotate_imm>)
- Ovakvim postupkom opseg svih brojeva koji se mogu opisati neposredno, proširen je na sva 32 bita, odnosno od 0 do $2^{32}-1$
  - Naravno, na ovaj način **ne mogu se zapisati sve moguće 32-bitne konstante**, već samo one koje se mogu dobiti rotiranjem 8-bitnog broja (0-255) za **paran** broj mjesta udesno (0, 2, 4, ..., 30).
- U **ARM-ovim** alatima je dovoljno napisati željeni broj, a asemblerski prevoditelj će **sam izračunati** može li se taj broj prikazati na ovaj način. Nažalost u ATLAS-u to nije moguće.

Primjeri dozvoljenih i nedozvoljenih vrijednosti

- Dozvoljene vrijednosti
  - FF, 104, FF0, FF00, FF000, FF000000, F000000F,…
- Nedozvoljene vrijednosti
  - 101, 102, FF1, FF04, FF003, FFFFFFFF, F000001F,…

- Napomena:
  - mnoge vrijednosti mogu se dobiti upotrebom naredbe MVN (Move NOT) koja radi komplement operanda
  - Npr: običnom naredbom ne možemo koristiti neposrednu vrijednost FFFFFF00. No, tu neposrednu vrijednost možemo učitati u registar koristeći naredbu MVN, SUB i slično:

NE!!! MOV R0, #0FFFFFFFF

Da!!! MVN R0, #0

Primjeri dozvoljenih i nedozvoljenih vrijednosti

NE!!! MOV R4, #0FFFFFF0

Da!!! MVN R4, #0F000000F

- Neke aritmetičke operacije možemo zamijeniti KOMPLEMENTARNIM OPERACIJAMA

  NE!!! ADDS R4, R1, #0FFFFFFFF

  Da!!! SUBS R4, R1, #1

- Primjer dozvoljene neposredne vrijednosti kodiran u binarnom obliku:

  [e29142ff] adds r4,r1,#0f000000f

Naredba	31 30 29 28	27	26	25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
Aritmetičko logička, neposredna vrijednost	Uvjet	0	0	1	Op kod	S	Rn	Rd	rotate_imm	immed_8
ADDS R4,R1,#0F000000F	1110	0	0	1	0100	1	0001	0100	0010	11111111

ATLAS - NAPOMENA

- U primjerima koje radite za laboratorijske vježbe (**u ATLAS-u**) neposredne vrijednosti **morate kodirati sami** (asembler ne podržava automatsku pretvorbu)

- Način pisanja:

  baza<rotacija (u ATLAS-u se koristi **LIJEVA** rotacija)

- Primjer:

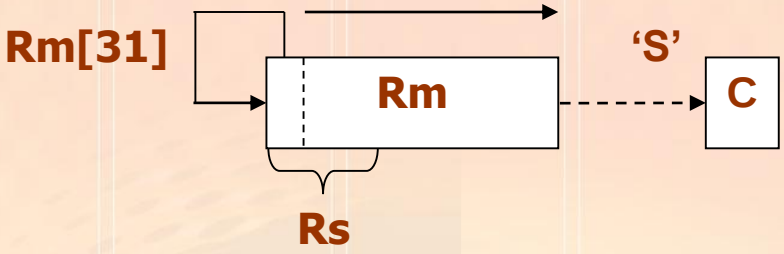
  broj 100_{16} piše se kao 1<8 $\underline{1}\ 0000\ 0000_2 = 1_2<8$

  broj 180_{16} piše se kao 18<4 $\underline{1\ 1000}\ 0000_2 = 11000_2<4$

  ili kao 6<6 $\underline{1\ 1}000\ 0000_2 = 110_2<6$

Registarsko i registarsko s pomakom

- Ako se nastavak 'S' doda naredbi koja inače ne mijenja zastavicu C, tada se C puni bitom označenim strelicom
  - Primjer: ... <Rm>, ASR <Rs>



- Registar Rm se ne mijenja, nego njegova pomaknuta vrijednost služi samo kao operand
- Iznos pomaka zadaje se registrom ili neposredno (brojem)
- Ostale pomake i rotiranja proučite iz tablice u Prilogu

Načini adresiranja 1 u Prilogu "Zbrike ARM"

- Pogledajte tablicu za npr. naredbu MOV:

  MOV{cond}{S} Rd, <Oprnd2>

- U tablici "Polje Operand2 <Oprnd2>" izaberete neki od načina adresiranja

- Napomena!!! **Ne postoji** naredba npr. LSL R0, #2 (dešava se na ispitima). Treba koristiti naredbu MOV (u ovom primjeru: MOV R0, R0, LSL #2).

Načini adresiranja 2

- Karakteristični su za naredbe load i store kojima se premješta riječ ili nepredznačeni bajt
- Postoji devet mogućih kombinacija adresiranja s obzirom na vrstu odmaka te vrstu indeksiranja
- Odmak se zadaje neposrednim podatkom, registrom ili pomaknutim registrom*. Ako je izabran pomak registra, tada se vrsta pomaka (LSL, LSR, ASR, ROR, RRX) zadaje kao u načinu adresiranja 1, no uz ograničenje da se iznos pomaka zadaje samo neposredno

---

* Odmak se ne mora napisati i tada se podrazumijeva da je 0 (zadan neposredno)

Načini adresiranja 2

Adresiranje	Sintaksa adresiranja
Neposredni odmak	[<Rn>, #+/-<offset_12>]
Registarski odmak	[<Rn>, +/-<Rm>]
Registarski odmak s pomakom	[<Rn>, +/-<Rm>, LSL #<shift_imm>] [<Rn>, +/-<Rm>, LSR #<shift_imm>] [<Rn>, +/-<Rm>, ASR #<shift_imm>] [<Rn>, +/-<Rm>, ROR #<shift_imm>] [<Rn>, +/-<Rm>, RRX]
Neposredni predindeksirani	[<Rn>, #+/-<offset_12>]!
Registarski predindeksirani	[<Rn>, +/-<Rm>]!
Registarski predindeksirani odmak s pomakom	[<Rn>, +/-<Rm>, LSL #<shift_imm>]! [<Rn>, +/-<Rm>, LSR #<shift_imm>]! [<Rn>, +/-<Rm>, ASR #<shift_imm>]! [<Rn>, +/-<Rm>, ROR #<shift_imm>]! [<Rn>, +/-<Rm>, RRX]!
Neposredni postindeksirani	[<Rn>], #+/-<offset_12>
Registarski postindeksirani	[<Rn>], +/-<Rm>
Registarski postindeksirani odmak s pomakom	[<Rn>], +/-<Rm>, LSL #<shift_imm> [<Rn>], +/-<Rm>, LSR #<shift_imm> [<Rn>], +/-<Rm>, ASR #<shift_imm> [<Rn>], +/-<Rm>, ROR #<shift_imm> [<Rn>], +/-<Rm>, RRX

Načini adresiranja 2 u Prilogu "Zbirke ARM"

- Pogledajte tablicu za npr. naredbu LDR:

  LDR{cond} Rd, <a_mode2>

- U tablici "Način adresiranja <a_mode2>" izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Na primjer:

  LDRHI R0, [R3, -R5, ROR #22]!

Načini adresiranja 3

- Šest formata adresa koje se koriste kod ostalih naredaba load i store (za poluriječ i predznačeni bajt)

- Ovih šest formata adresa slični su formatima iz načina adresiranja 2. Razlike su:
  - odmak se može zadati neposrednom vrijednošću ili registrom (ne može se zadati registar s pomakom)
  - neposredna vrijednost odmaka se zapisuje samo sa osam bitova

Načini adresiranja 3

Adresiranje	Sintaksa adresiranja
Neposredni odmak	[<Rn>, #+/-<offset_8>]
Registarski odmak	[<Rn>, +/-<Rm>]
Neposredni predindeksirani odmak	[<Rn>, #+/-<offset_8>]!
Registarski predindeksirani odmak	[<Rn>, +/-<Rm>]!
Neposredni postindeksirani odmak	[<Rn>], #+/-<offset_8>
Registarski postindeksirani odmak	[<Rn>], +/-<Rm>

Načini adresiranja 3

- Pogledajte tablicu za npr. naredbu LDR:

  LDR{cond}SB Rd, <a_mode3>

- u tablici "Način adresiranja <a_mode3>" izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Na primjer:

  LDRGTSB R0, [R3], #20

Načini adresiranja 4

- Naredbama load i store multiple moguće je pročitati ili upisati sadržaj jednog, sadržaj više ili čak sadržaje svih registara u memoriju, odnosno u niz uzastopnih memorijskih lokacija.
- Registar s najmanjim brojem je uvijek zapisan na najmanju memorijsku adresu, a registar s najvećim brojem na najveću.
- Na isti način, kod čitanja: u registar s najmanjim brojem učitava se podatak s najmanje adrese, a u registar s najvećim brojem učitava se podatak s najveće adrese.
- IA, IB, DA, DB (FD, FA, ED, EA)

Načini adresiranja 4

- Pogledajte tablicu za npr. naredbu LDM:

  LDM{cond} <a_mode4L> Rd{!}, <reglist-pc>

- U tablici "Način adresiranja <a_mode4L> izaberete neki od načina adresiranja i upotrijebite ga u naredbi. Npr.

  LDMGTIA R0!, {R4,R5,R6}

- Analogno vrijedi za naredbe STM i tablicu "Način adresiranja <a_mode4S>.

- Napomena: <reglist-pc> u opisu naredbe znači da se u popis registara ne smije staviti PC

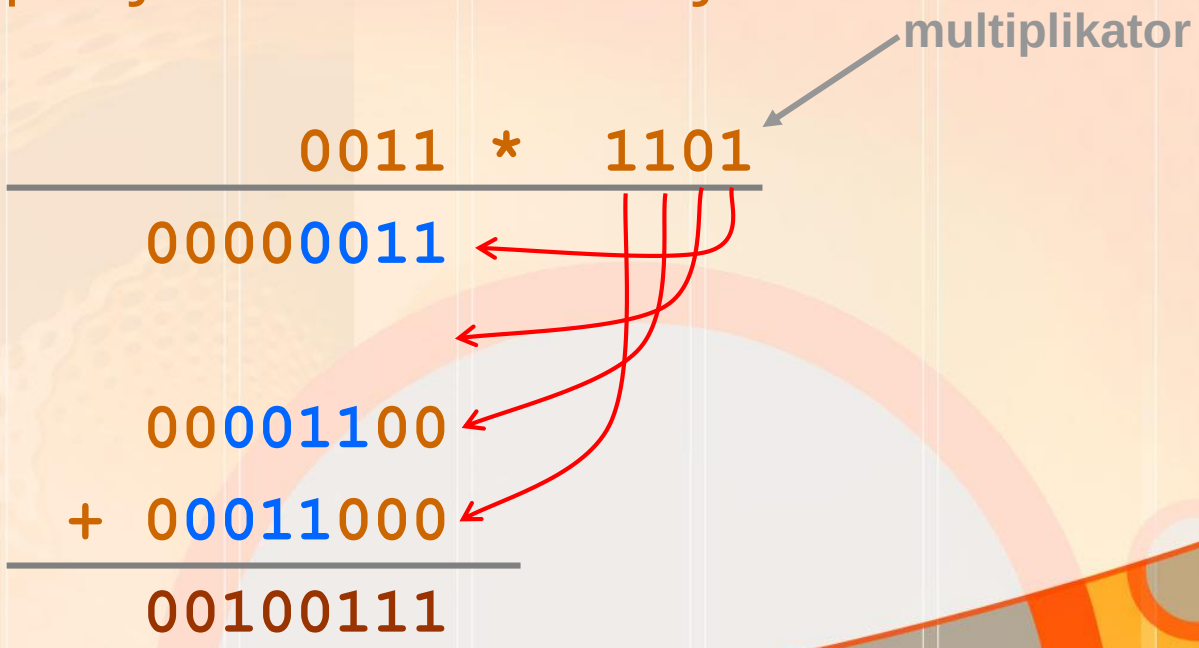
Načini adresiranja 5

- Koriste se kod naredaba za prijenos podataka prema koprocesoru
- Nećemo ih koristiti

Primjer: množenje dva 16-bitna broja

Treba pomnožiti dva 16-bitna broja metodom pomaka. Operandi su spremljeni od adrese 8100, a 32-bitni rezultat treba staviti iza operanada. Multiplikand može biti u zapisu NBC ili 2'k, a multiplikator može biti samo u zapisu NBC.

Načelo množenja na primjeru dva 4-bitna broja:



Primjer programa za množenje

```
; Program za množenje 16-bitnih brojeva metodom pomaka

    MOV  R4, #81<8        ;postavlja u registar R4 adresu podataka 8100
    LDRH  R5, [R4], #2    ;multiplikator se učitava u R5
    LDRSH  R6, [R4], #2   ;multiplikand se učitava u R6 (predznak sačuvan)
    MOV  R7, #0           ;čisti se registar za spremanje rezultata (R7)

PETLJA
    MOVS  R5, R5, LSR #1  ;pomak multiplikatora za jedan bit udesno,
                          ;najniži bit otići će u C
    ADDCS  R7, R7, R6     ;ako je zastavica C=1, R6 se dodaje privremenom rezultatu
    MOV  R6, R6, LSL #1   ;pomak R6 za jedan bit ulijevo
                          ;(priprema za mogući sljedeći korak)
    CMP  R5, #0           ;provjerava je li multiplikator različit od nule
    BNE  PETLJA           ;ako je, onda se petlja ponavlja

    STR  R7, [R4]         ;spremanje rezultata
    . . .
```

Primjer dijeljenja metodom pomaka

- Dijeljenje A/B (djeljenik A, djelitelj B) odvija se u dva koraka:
- **Prvi korak**: poravnanje brojeva – traženje najvećeg višekratnika djelitelja sadržanog u djeljeniku
  - Djelitelj B pomičemo u lijevo sve dok vrijedi A>=2B i MSB(B)=0 te pamtimo **broj_pomaka**
  - Ponavlja se dok A>=2B zato da pronađemo **najveći** višekratnik
  - Prije ispitivanja A>=2B provjeravamo da li je MSB(B)=1
    - Ako MSB(B)=1 tada smo sigurni da je to najveći višekratnik jer bi množenjem s 2 dobili broj koji je veći od opsega brojeva koji se mogu prikazati u 32 bita što bi sigurno bilo veće od djeljenika
    - U slučaju MSB(B)=1 moramo preskočiti ispitivanje A>=2B jer bi usporedba mogla dovesti do krivog rezultata (B pomaknuto u lijevo za jedan bit prelazi 32 bita registra)
- **Drugi korak:** Nakon što od djeljenika oduzmemo najveći višekratnik djelitelja, postupak se ponavlja **broj_pomaka** puta (kao dijeljenje na papiru - vidi sljedeći slajd)

Postupak...

```
  1101101001 :  1010 = 1       max.6 pomaka djelitelja u lijevo,oduzmemo
- 1010000000                    od djeljenika i stavimo 1 u rez
  0011101001 :  1010 = 10      (5)Djelitelj za 1 u desno, rez 1 u lijevo
   101000000                   -Ne može se oduzeti,dodamo 0 u rez
  0011101001 :  1010 = 101     (4)Djelitelj za 1 u desno, rez 1 u lijevo
-   10100000                   -Može se oduzeti,dodamo 1 u rez
  0001001001 :  1010 = 1010    (3)Djelitelj za 1 u desno, rez 1 u lijevo
     1010000                   -Ne može se oduzeti,dodamo 0 u rez
  0001001001 :  1010 = 10101   (2)Djelitelj za 1 u desno, rez 1 u lijevo
-     101000                   -Može se oduzeti,dodamo 1 u rez
  0000100001 :  1010 = 101011  (1)Djelitelj za 1 u desno, rez 1 u lijevo
-      10100                   -Može se oduzeti,dodamo 1 u rez
  0000001101 :  1010 = 1010111 (0)Djelitelj za 1 u desno, rez 1 u lijevo
-       1010                   -Može se oduzeti,dodamo 1 u rez
  0000000011  (ostatak)

                               Broj_pomaka = 0 -> KRAJ POSTUPKA
```

Primjer programa za dijeljenje

```
; Program za 32-bitno dijeljenje metodom pomaka

    MOV  R4, #81<8          ; postavlja u registar R4 adresu podataka 8100
    LDR  R5, [R4], #4       ; djeljenik (A) se učitava u R5
    LDR  R6, [R4], #4       ; djelitelj (B) se učitava u R6
    MOV  R7, #0             ; čisti se registar za spremanje rezultata (R7)
    MOV  R8, #0             ; brojač inicijalnih pomaka
    CMP  R6, R5             ; ako je B>A, nema potrebe za dijeljenjem
    BHI  KRAJ

PORAVNAJ            ;inicijalni korak: pronalaženje najvećeg višekratnika djelitelja

    ANDS  R3, R6, #80<24    ; provjerava je li MSB djelitelja jednak 1
    BNE  DIV                ; ako jeste, poravnavanje je nepotrebno
    CMP  R5, R6, LSL #1     ; provjerava je li A >= 2*B
    MOVHS  R6, R6, LSL #1   ; ako je, pomiče B ulijevo
    ADDHS  R8, R8, #1       ; povećava brojač pomaka
    BHI  PORAVNAJ           ; samo ako je A>2*B, poravnavanje se nastavlja
```

Primjer programa za dijeljenje (2. dio)

```
DIV
    CMP  R5, R6              ; uspoređuje trenutačni ostatak i B
    SUBHS  R5, R5, R6        ; ako je ostatak >= B, onda ga umanji za B
    ADDHS  R7, R7, #1        ; i poveća rezultat za 1
    CMP  R8, #0              ; ako je brojač pomaka > 0, nastavi, a inače KRAJ
    MOVHI  R6, R6, LSR #1    ; pomakne B udesno,
    MOVHI  R7, R7, LSL #1    ; a rezultat ulijevo
    SUBHI  R8, R8, #1        ; umanji brojač pomaka
    BHI  DIV                 ; i ponovi petlju

KRAJ
    STR  R5, [R4], #4        ; spremanje ostatka
    STR  R7, [R4]            ; spremanje rezultata
    . . .
```

Još jedan primjer....

```
;*************************************************************
; Program koji niz podataka (terminiran nulom) koji se nalazi u memoriji
; na adresi 8200 ispisuje u obrnutom redoslijedu na istu adresu. (obrni_niz_stog)
;*************************************************************

        MOV     R13, #84<8      ; inicijalizacija stoga
        MOV     R1, #82<8       ; učitavanje početne adrese
        MOV     R0, #0          ; brojač duljine niza

PETLJA  LDR     R2, [R1], #4    ; petlja kojom čitamo niz
        CMP     R2, #0          ; ako je nula, niz je gotov
        BEQ     ISPIS
        STMFD   SP!, {R2}       ; spremanje na stog
        ADD     R0, R0, #1      ; povećaj brojač
        B       PETLJA          ; idi dalje
```

>>>>

... nastavak

<<<<

```
ISPIS   MOV     R1, #82<8       ; učitavanje početne adrese

POM     CMP     R0, #0
        BEQ     KRAJ
        LDMFD   SP!, {R2}       ; čitanje sa stoga
        STR     R2, [R1], #4
        SUB     R0, R0, #1      ; smanjujemo brojač
        B       POM

KRAJ    SWI     123456          ; Kraj
```

Ispitivanje specijalnog broja...

```
;*****************************************************************
;               Ispitivanje je li broj specijalan
;                        specNum.s
;*****************************************************************
; Napisati program koji ispituje je li troznamenkasti dekadski broj 'xyz' zapisan
; kao niz ASCII znamenaka (tzv. string) "xyz\0" specijalan broj za kojeg vrijedi:
; xyz = xy * xy - z * z, gdje su x y i z znamenke stotice, desetice i jedinice.
; Primjer takvog broja su brojevi: 100 = 10*10 - 0*0 i 147 = 14*14 - 7*7
; Broj zapisan kao niz ASCII znamenki sa završnim null-znakom nalazi se na
; adresi 1000. Ukoliko broj zadovoljava gornji uvjet, tada je potrebno u registar
; r1 staviti sve jedinice, a ako uvjet nije ispunjen, tada u r1 treba staviti sve
; nule.
;*****************************************************************
```

Specbroj...

```
; ovaj odsječak vadi znamenke iz ASCII zapisa: u r1 će se nalaziti znamenka
; stotica, u r2 će se nalaziti znamenka desetica, a u r3 će biti znamenka jedinica

main    MOV  r0, #10<8
        LDRB  r1, [r0], #1
        LDRB  r2, [r0], #1
        LDRB  r3, [r0]
        SUB  r1, r1, #%D48          ; znamenka stotica (48 je ASCII znak od 0)
        SUB  r2, r2, #%D48          ; desetice
        SUB  r3, r3, #%D48          ; jedinice

; množenje s konstantom 100 ostvareno je kombinacijom naredaba ADD i MOV
; s pomakom, što je efikasnije od korištenja naredbe MUL (multiply)

        ADD  r5, r1, r1, LSL #3     ; r5 = r1+8*r1 = 9*r1
        ADD  r5, r5, r1, LSL #4     ; r5 = r5+16*r1 = 9*r1+16*r1 = 25*r1
        MOV  r5, r5, LSL #2         ; konačno r5 = 4*r5 = 4*25*r1 = 100 * r1
```

>>>>

Specbroj...

<<<<

```
; množenje s konstantom 10 pomoću kombinacije naredaba ADD i MOV s pomakom
        MOV  r6, r2
        ADD  r6, r6, r6, LSL #2
        MOV  r6, r6, LSL #1                 ; r6 = 10 * r2

        ADD  r7, r5, r6
        ADD  r7, r7, r3     ; konačan broj u binarnom zapisu, potreban za ispitivanje
                            ; uvjeta zadatka xyz = xy*xy - z*z

; generiranje broja xy a nakon toga i broja xy*xy te z*z
        ADD  r1, r1, r1, LSL #2
        MOV  r1, r1, LSL #1
        ADD  r1, r1, r2                     ; u r1 se nalazi broj xy
        MUL  r4, r1, r1                     ; u r4 se nalazi broj xy*xy
        MUL  r5, r3, r3                     ; u r5 se nalazi broj z*z
```

>>>>

Specbroj...

<<<<

```
        SUB  r4, r4, r5          ; r1 = xy*xy - z*z
        CMP  r4, r7
        MVNEQ  r1, #0            ; sve jedinice u R1
        MOVNE  r1, #0            ; sve nule u R1

KRAJ    SWI  123456              ; Kraj programa

        ORG 1000
        ; neki ASCII podaci …
```

Specbroj...

Komentari:

Množenje kombinacijom ALU-operacija može se ostvariti kad množimo s konstantom. Obično se može ostvariti na više načina. Na primjer, množenje sa 100:

100 = ((1+8)+16)*4	(iz primjera: 3 naredbe)
100 = 128-32+4	(3 naredbe)
100 = (1+32)*3+1	(3 naredbe)
100 = 64+32+4	(3 naredbe)
100 = (1+16)*3*2-2	(4 naredbe)

Potprogrami

Potprogrami

- Poziv potprograma: BL
- Povratak iz potprograma: MOV PC, LR
- Povratna adresa sprema se u registar R14 (LR)

Primjer atoh_v1.s

Zadatak:

Napisati program koji 32-bitni heksadekadski broj zapisan u ASCII kodu (niz od 8 ASCII znakova) pretvara u broj. Znakovi se nalaze u memoriji od adrese 8100_{16}, a rezultat treba spremiti u memoriju iza znakova.

Za pretvorbu ASCII heksadekadske znamenke u njenu brojevnu vrijednost treba koristiti potprogram. Potprogram prima ASCII znamenku preko R0 i vraća rezultat preko R0. Pretpostavka je da je heksadekadska ASCII znamenka ispravna, tj. da može sadržati dekadske znamenke od "0" do "9", mala i velika slova od "a" do "f".

Podsjetnik za ASCII kodove:

'0'	48	'A'	65	'a'	97
'1'	49	'B'	66	'b'	98
...	...	...	...	...	...
'9'	57	'F'	70	'f'	102

Primjer atoh_v1.s

```
GLAVNI  MOV  R4, #81<8          ; postavlja R4 na početak podataka
        MOV  R7, #8             ; brojač hex znamenki = 8
        MOV  R6, #0             ; resetiranje rezultata

PETLJA
        LDRB  R0, [R4], #1      ; učitavanje ASCII-znaka
        BL   FUNC_ATOH          ; poziv potprograma za pretvorbu
        ADD  R6, R0, R6, LSL #4 ; spremanje rezultata iz R0 u registar R6 uz
                                ; pomak trenutačnog rezultata za 4 bita

        SUBS  R7, R7, #1        ; brojač prolaza petlje
        BHI   PETLJA

        STR  R6, [R4]           ; spremanje rezultata

KRAJ    SWI  123456
```

>>>>

Primjer atoh_v1.s

```
<<<<
FUNC_ATOH               ; potprogram za pretvorbu ASCII-znaka u HEX
                        ; ulazni ASCII-znak je u R0, a rezultat je također u R0

    SUB  R0, R0, #%D48  ; ASCII:0 = DEC: 48
    CMP  R0, #%D10      ; ako je broj 10 ili veći, znači da se radi o slovu
    MOVLO  PC,LR        ; povratak iz potprograma ako je R0<10

    SUB  R0, R0, #%D7   ; ASCII:A = DEC: 65
                        ; treba oduzeti 65-55=10
                        ; ranije smo već oduzeli 48, pa trebamo još 7 (48+7=55)
    CMP  R0, #%D16      ; ako je broj 16 ili veći, znači da se radi o malom slovu
    MOVLO  PC,LR        ; povratak iz potprograma ako je R0<16

    SUB  R0, R0, #%D32  ; ASCII:a = DEC: 97
                        ; treba oduzeti 97-87=10
                        ; ranije smo već oduzeli 55, pa trebamo još 32 (55+32=87)
    MOV  PC, LR         ; povratak iz potprograma
```

Napomena: potprogram ne mijenja registre pa nema spremanja konteksta

Prijenos parametara

- preko registara*
  - problem sa brojem registara koji su na raspolaganju
  - isto kao kod FRISC-a
- preko stoga
  - često najpraktičnije rješenje
- parametri iza naredbe poziva
- preko fiksnih memorijskih lokacija
  - isto kao kod FRISC-a

* pokazano u prethodnom primjeru

Poziv potprograma s parametrima na stogu

Potprogram računa f(X,A,B) = (X<<A) + B, parametri i rezultat se prenose stogom. Parametre uklanja potprogram. Potprogram mijenja registre.

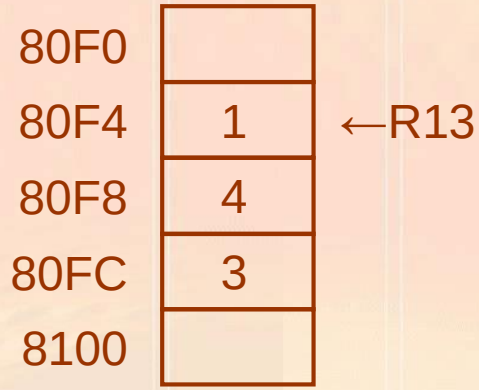
```
GLAVNI  MOV  R13, #81<8          ; definicija vrha stoga                (1)
        MOV  R0, #1              ; parametar X=1
        MOV  R1, #4              ; parametar A=4
        MOV  R2, #3              ; parametar B=3
        STMFD  R13!, {R0, R1, R2}  ; parametri se spremaju na stog      (2)
        BL   FUNC_PARAM          ; poziv potprograma
        LDMFD  R13!, {R0}        ; rezultat se sa stoga učitava u R0    (5)
DALJE   . . .                    ; nastavak glavnog programa


FUNC_PARAM                       ; potprogram
        LDMFD  R13!, {R4, R5, R6}  ; učitavanje parametara X,A,B       (3)
        ADD  R4, R6, R4, LSL R5  ; računanje funkcije
        STMFD  R13!, {R4}        ; spremanje rezultata                  (4)
        MOV  PC, LR              ; povratak iz potprograma
```

Poziv potprograma s parametrima na stogu

Izgled stoga:

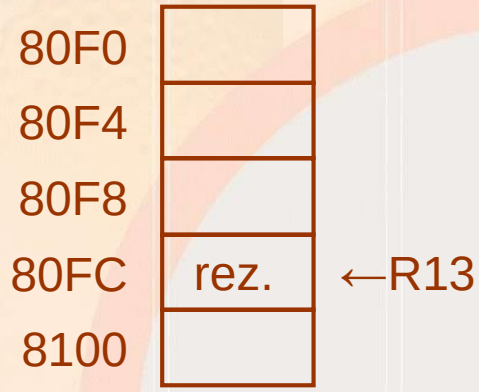
nakon (1):

nakon (2):

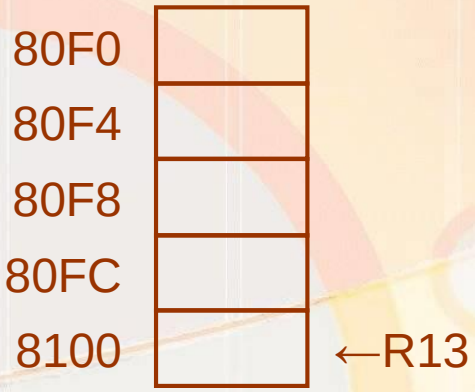


nakon (3):

nakon (4):



nakon (5):



Poziv potprograma s parametrima na stogu

Potprogram računa f(X,A,B) = (X<<A) + B kao i prije, ali uz čuvanje registara (DOBRO RJEŠENJE). Parametri se prenose stogom i uklanja ih pozivatelj. Rezultat se vraća u R0.

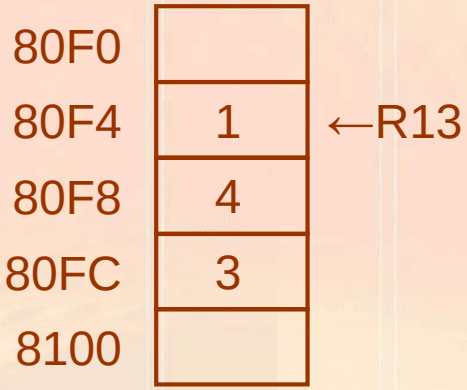
```
GLAVNI MOV  R13, #81<8              ; definicija vrha stoga                (1)
       MOV  R0, #1                  ; parametar X=1
       MOV  R1, #4                  ; parametar A=4
       MOV  R2, #3                  ; parametar B=3
       STMFD  R13!, {R0, R1, R2}    ; parametri se spremaju na stog        (2)
       BL   FUNC_PARAM              ; poziv potprograma
       ADD R13, R13, #%D 12         ; ukloni parametre sa stoga            (6)
DALJE  STR  R0, REZ                 ; spremi rezultat i nastavi
       . . .

FUNC_PARAM                          ; potprogram
       STMFD R13!, {R4, R5, R6}     ; spremi registre                      (3)
       ADD R0, R13, #%D 12          ; sa R0 "preskoči" spremljene registre (4)
       LDMFD  R0, {R4, R5, R6}      ; učitavanje ulaznih parametara X,A,B
       ADD  R0, R6, R4, LSL R5      ; računanje funkcije
       LDMFD  R13!, {R4, R5, R6}    ; obnovi registre                      (5)
       MOV  PC, LR                  ; povratak iz potprograma
```

Poziv potprograma s parametrima na stogu

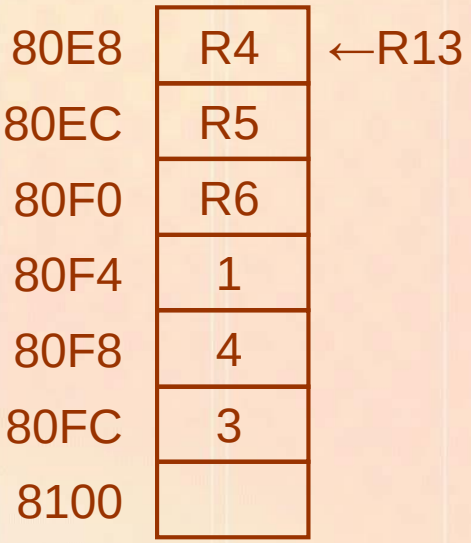
Izgled stoga:

nakon (1):

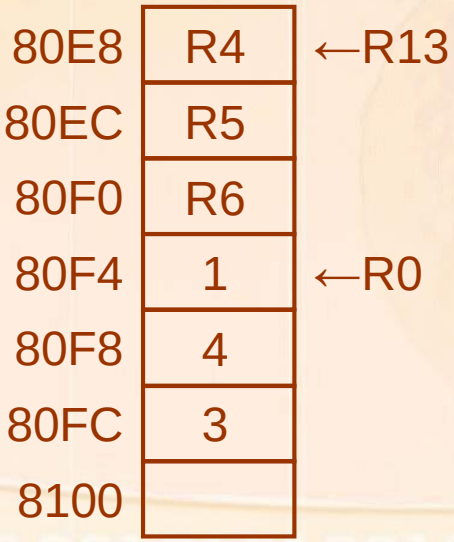
nakon (2):



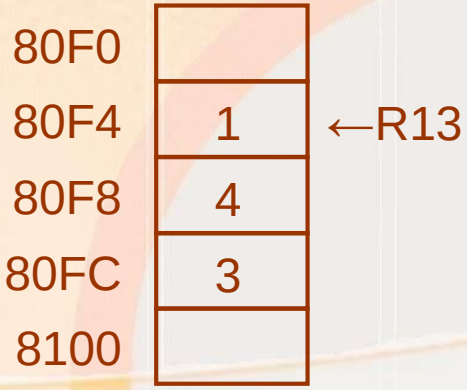
nakon (3):



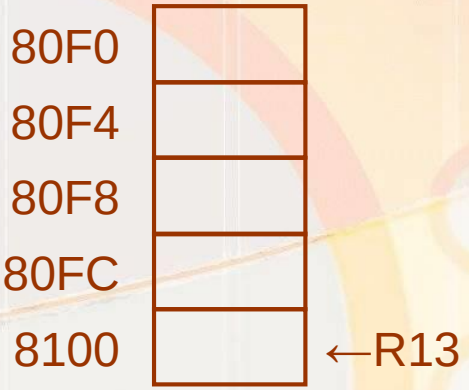
nakon (4):



nakon (5):



nakon (6):



Poziv potprograma s parametrima iza poziva

Potprogram računa f(X)=(X<<A) + B, parametri A i B su iza naredbe, a parametar X je u registru R0. Rezultat se vraća preko R0. Potprogram mijenja registre!!!

```
GLAVNI MOV  R0, #1              ; parametar X=1
       BL  FUNC_PARAM           ; poziv potprograma
       DW  4                    ; parametar A=4
       DW  3                    ; parametar B=3
DALJE  . . .                    ; nastavak programa, rezultat je u R0


FUNC_PARAM                      ; potprogram, adresa prvog parametra je u R14
       LDR  R1, [R14], #4       ; prvi parametar (A) se učitava u R1, a R14 se
                                ; poveća tako da pokazuje na drugi parametar
       LDR  R2, [R14], #4       ; drugi parametar (B) se učitava u R2, a R14 se
                                ; poveća tako da je u njemu povratna adresa
       ADD  R0, R2, R0, LSL R1  ; računanje funkcije
       MOV  PC, LR              ; povratak iz potprograma
```

Poziv potprograma s parametrima iza poziva

Potprogram računa f(X)=(X<<A) + B, parametri A i B su u lokacijama iza naredbe, a parametar X je u registru R0. Rezultat se vraća preko R0. Potprogram čuva registre (DOBRO RJEŠENJE).

```
GLAVNI  MOV  R13, #81<8          ; definicija vrha stoga
        MOV  R0, #1              ; parametar X=1
        BL  FUNC_PARAM           ; poziv potprograma
        DW   4                   ; parametar A=4
        DW   3                   ; parametar B=3
DALJE   . . .                    ; nastavak programa, rezultat je u R0


FUNC_PARAM                       ; potprogram, adresa prvog parametra je u R14
        STMFD R13!, {R1, R2}     ; spremi registre
        LDR  R1, [R14], #4       ; prvi parametar (A) se učitava u R1
        LDR  R2, [R14], #4       ; drugi parametar (B) se učitava u R2
                                 ; sada je u R14 povratna adresa
        ADD  R0, R2, R0, LSL R1  ; računanje funkcije
        LDMFD R13!, {R1, R2}     ; obnovi registre
        MOV  PC, LR              ; povratak iz potprograma
```

Ugniježđeni pozivi

- Problem: R14 služi za spremanje povratne adrese
  - već prvi ugniježđeni poziv uništava početni R14

- Jedino dobro rješenje je spremanje R14 na stog na početku potprograma te čitanje povratne adrese sa stoga prije povratka iz potprograma

- Primjeri:
  - Poziv potprograma iz potprograma
  - Rekurzivna funkcija

Primjer poziva potprograma iz potprograma

Treba napisati funkciju c(x)=(x/2) + 3, ali tako da se koriste pomoćne funkcije a(x)=x/2 i b(x)=x+3. Potprogrami trebaju čuvati registre.

```
glavni    MOV  R13, #80<8        ; inicijalizacija SP
          MOV  R0, #7            ; parametar x=7
          BL  FUNC_C             ; poziv potprograma
          . . .                  ; nastavak glavnog programa, rezultat je u R1

FUNC_A    MOV  R0, R0, ASR #1    ; R0= R0/2
          MOV  PC, LR            ; povratak iz potprograma

FUNC_B    ADD  R0, R0, #3        ; R0 = R0+3
          MOV  PC, LR            ; povratak iz potprograma

FUNC_C    STMFD  r13!, {R0,R14}  ; R1 = R0/2 + 3
          BL  FUNC_A             ; x=x/2
          BL  FUNC_B             ; x=x/2 +3
          MOV  R1, R0            ; rezultat stavi u R1
          LDMFD  r13!, {R0,R14}
          MOV  PC, LR            ; povratak iz potprograma
```

Primjer računanja rekurzivne funkcije (faktorijela)

Napisati potprogram za računanje faktorijele korištenjem rekurzivnog poziva. Parametar funkcije neka se prenosi preko registra R0, a rezultat funkcije neka se vraća u registru R1.

Glavni program treba izračunati faktorijela(6). Potprogram treba čuvati registre.

```
int faktorijela (int x) {
        if( x == 1)
                return (1);
        return ( faktorijela(x-1) * x );
}
```

Primjer računanja rekurzivne funkcije (faktorijela)

```
GLAVNI      MOV  R13, #81<8        ; vrh stoga
            MOV  R0, #6            ; ulazni parametar=6
            BL  FUNC_FACT          ; rezultat je u R1
KRAJ        SWI  123456            ; Kraj

FUNC_FACT   STMFD  R13!, {R0, LR}  ; spremanje registara na stog
            CMP  R0, #1
            BNE  X_NIJE_1          ; ako x != 1 onda rekurzija

X_JE_1      MOV  R1, #1            ; inicijalna vrijednost faktorijele
            LDMFD  R13!, {R0, LR}  ; obnovi registre sa stoga
            MOV  PC, LR            ; povratak iz potprograma za x==1

X_NIJE_1    SUB  R0, R0, #1        ; R0 = x-1
            BL  FUNC_FACT          ; R1 = f(x-1)
            ADD  R0, R0, #1        ; R0 = x
            MUL  R1, R0, R1        ; izračunaj R1 = x*f(x-1)
            LDMFD  R13!, {R0, LR}  ; obnovi registre sa stoga
            MOV  PC, LR            ; povratak iz potprograma
```

Primjer Manhattan

- Poziv potprograma iz potprograma
- Prijenos parametara preko stoga
- Čuvaju se registri

- Zadatak:

Napisati potprogram MANH koji računa Manhattan-udaljenost dvije točke: (x1, y1) i (x2, y2). Manhattan-udaljenost definirana je formulom:

$|x1 - x2| + |y1 - y2|$

Potprogram MANH preko stoga prima koordinate točaka x1,y1,x2,y2 kao parametre, a rezultat vraća u registru R7. Apsolutna vrijednost razlike dva broja računa se posebnim potprogramom ADIFF koji parametre prima preko stoga i rezultat vraća u registru R8.

...glavni program

```
        ORG     0
GLAVNI  MOV     R13, #1<16          ; inicijalizacija pokazivača stoga
        MOV     R0, #9              ; Prva točka je (9, 3)
        MOV     R1, #3
        STMFD   R13!, {R0, R1}      ; koordinate prve točke se stavljaju
                                    ; kao parametri na stog

        MOV     R0, #5              ; Druga točka je  (5, 8)
        MOV     R1, #8
        STMFD   R13!, {R0, R1}      ; koordinate druge točke se stavljaju
                                    ; kao parametri na stog
        BL      MANH
        ADD     R13, R13, #%D16     ; uklanjaju se parametri sa stoga
                                    ; rezultat je u R7

        SWI     123456              ; kraj
```

… potprogram MANH

```
MANH
    STMFD  R13!, {R0,R1,R2,R3,R4,R8,R14}  ; registri koji se koriste moraju se sačuvati
    ADD   R0, R13, #%D28                  ; R0 preskače nove podatke na stogu
    LDMFD  R0, {R1, R2, R3, R4}           ; učitavaju se pohranjeni parametri
                                          ; R1=x2,R2=y2,R3=x1,R4=y1
    STMFD  R13!, {R1, R3}
    BL  ADIFF                             ; poziva se ADIFF(R1,R3)
    ADD  R13, R13, #8                     ; uklanjaju se parametri sa stoga
    MOV  R1, R8                           ; rezultat iz r8 sprema se u r1

    STMFD  R13!, {R2, R4}
    BL  ADIFF                             ; poziva se ADIFF(R2,R4)
    ADD  R13, R13, #8                     ; uklanjaju se parametri sa stoga
    MOV  R2, R8                           ; rezultat iz r8 sprema se u r2

    ADD  R7, R1, R2                       ; zbrajaju se dvije apsolutne razlike

    LDMFD  R13!, {R0,R1,R2,R3,R4,R8, R14} ; obnavljaju se registri
    MOV  PC, LR
```

… potprogram ADIFF

```
ADIFF
    STMFD  R13!, {R0, R1, R2}      ; registri koji se koriste moraju se sačuvati

    ADD  R0, R13, #%D12            ; R0 preskače nove podatke na stogu
    LDMFD  R0, {R1,R2}             ; učitavaju se pohranjeni parametri
    SUBS  R8, R1, R2               ; izračunavanje razlike i postavljanje zastavica

    RSBLT  R8, R8, #0              ; ako je R8 negativan, onda R8 = 0-R8 = -R8

    LDMFD  R13!, {R0, R1, R2}      ; obnavljaju se pohranjeni registri
    MOV  PC, LR
```