Sklad, podprogrami II

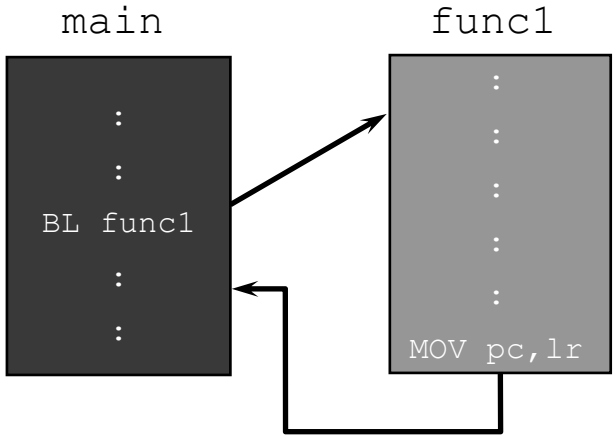
Pri klicanju podprogramov si je potrebno zapomniti povratni naslov.
ARM :

- **pri klicu podprograma povratni naslov shrani v register r14 (link register)**
- **pri vračanju iz podprograma je potrebno povratni naslov iz r14 (lr) prepisati v r15 (pc)**

Klic podprograma:

- **BL** : Branch with Link (L = 1) - shrani povratni naslov v r14.

```
    Zgled:
        bl   PODPROG
        ..
PODPROG: ..
        ..
        mov r15,r14 @ ali mov pc,lr
```



Problem gnezdenja klicev podprogramov – zahteva drugačno rešitev! -> SKLAD

Load/store multiple – beri/shrani več registrov

Z ukazom **ldm/stm (load multiple/store multiple)** je mogoče prebrati/shraniti več registrov:

- pomnilniški naslov za branje/shranjevanje mora biti **poravnan** (deljiv s 4)
- **registri z nižjimi indeksi** se vedno zapišejo na **nižji naslov**

Začetni naslov za shranjevanje/nalaganje je določen z baznim registrom in se pred ali po shranjevanju posameznega registra poveča ali zmanjša za 4. Pripona ukaza določa :

- ali se naj **naslov povečuje** ali **zmanjšuje**
- ali se to zgodi **pred ali po** branju/pisanju posameznega registra

Imamo štiri mogoče pripone:

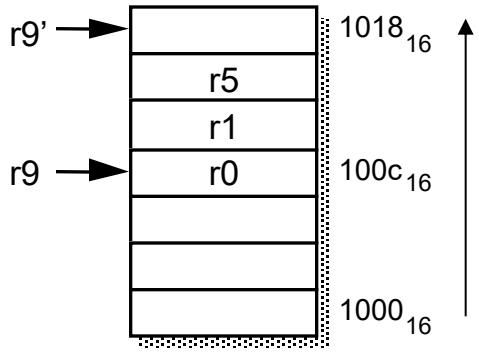
- **db** (decrement before), **da** (decrement after), **ib** (increment before), **ia** (increment after).

Če za baznim registrom stoji **!**, bo vrednost baznega registra enaka **naslovu po branju/shranjevanju zadnjega registra**. Sicer se vrednost baznega registra ne spremeni.

```
stmdb r13!, {r2-r9}      @ mem32[r13-4..r13-32] <- r9,…,r2
                         @ r13 <- r13-32   (8reg*4bajte=32bajtov)
stmdb r13, {r2-r9}       @ mem32[r13-4..r13-32] <- r9,…,r2
                         @ r13 ostane nespremenjen
ldmia r0!, {r2-r9}       @ r2,…,r9<-mem32[r0..r0+28]
                         @ r0 <- r0+32
stmda r1!, {r2-r9}       @ mem32[r1..r1-28] <- r9,…,r2
                         @ r1 <- r1-32
ldmib r13!, {r2-r9}      @ r2,…,r9 <- mem32[r13+4..r13+32]
                         @ r13 <- r13+32
```

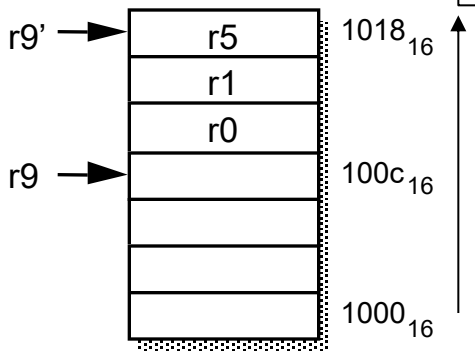
Load/store multiple – beri/shrani več registrov

ia (inc. after)



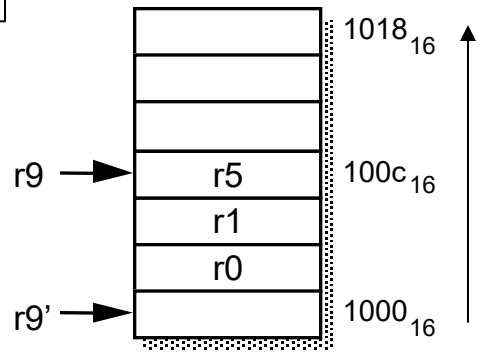

```
stmia r9!, {r0,r1,r5}
```

ib (inc. before)



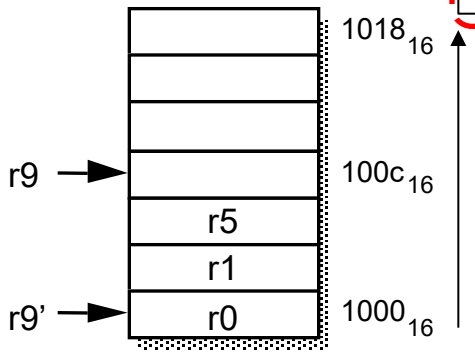

```
stmib r9!, {r0,r1,r5}
```

da (dec. after)




```
stmda r9!, {r0,r1,r5}
```

db (dec. before)




```
stmdb r9!, {r0,r1,r5}
```

Load/store multiple
Primer 1: bločno kopiranje vsebine

```
start
        LDR r0, =src            ; r0 = pointer to source block
        LDR r1, =dst            ; r1 = pointer to destination block

        MOV r2, #num            ; r2 = number of words to copy
        MOVS r3,r2, LSR #3      ; Number of eight word multiples
        …

octcopy LDM r0!, {r4-r11}       ; Load 8 words from the source
        STM r1!, {r4-r11}       ; and put them at the destination
        SUBS r3, r3, #1         ; Decrement the counter
        BNE octcopy             ; ... copy more
```

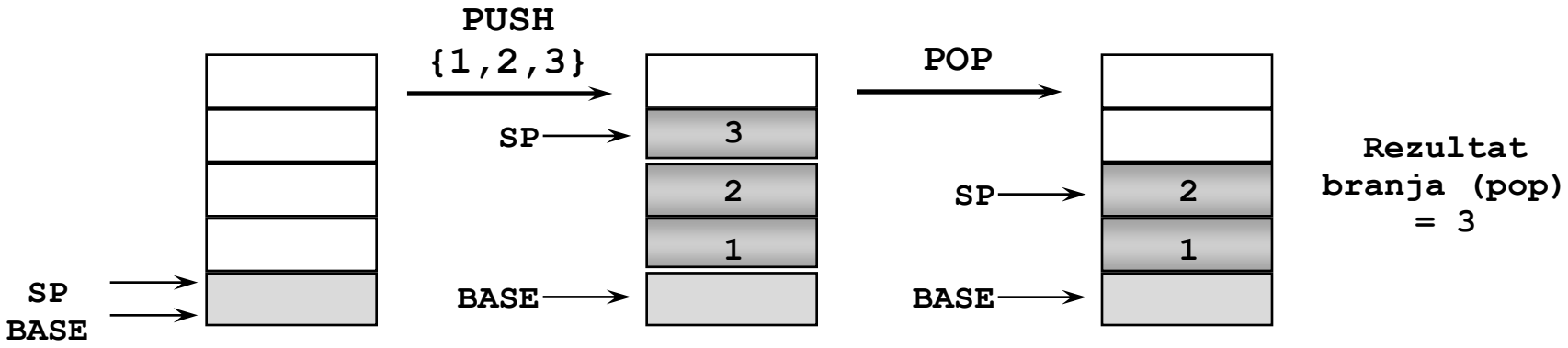
Sklad

Sklad je del pomnilnika, ki se:

- **poveča**, ko se operand shrani na **„vrh“** sklada - PUSH
- **zmanjša,** ko se podatek **prebere** iz vrha sklada - POP

Delovanje sklada zaznamujeta 2 kazalca :

- kazalec na začetni naslov („dno sklada“) - BASE
- **skladovni kazalec** („vrh sklada“) - SP - „Stack pointer“



Load/store – več registrov, sklad

Prenos **več registrov** se najpogosteje uporablja pri delu **s skladom** (shranjevanje na sklad, jemanje s sklada)

Podprte so vse različice skladov, od tod kratice:

- *ED (Empty Descending): širi se proti nižjim naslovom, SP kaže na prazen prostor*
- ***FD (Full Descending): širi se proti nižjim naslovom, SP kaže na zadnji element***
- *EA (Empty Ascending): širi se proti višjim naslovom, SP kaže na prazen prostor*
- *FA (Full Ascending): širi se proti višjim naslovom, SP kaže na zadnji element na skladu*

Uporabljamo FD sklad :

- ***vpis-DB: STMFD≡STMDB***
- ***branje-IA: LDMFD≡LDMIA***

		Ascending		Descending	
		Full	Empty	Full	Empty
Increment	Before	STMIB STMFA			LDMIB LDMED
	After		STMIA STMEA	LDMIA LDMFD	
Decrement	Before		LDMDB LDMEA	STMDB STMFD	
	After	LDMDA LDMFA			STMDA STMED

Podprogrami, sklad, uporaba/obnovitev registrov

Kazalec na sklad je običajno register r13 (sp). Pred uporabo sklada moramo v r13 vpisati naslov vrha sklada. Pri določitvi tega naslova upoštevamo, da se sklad širi proti nižjim naslovom.

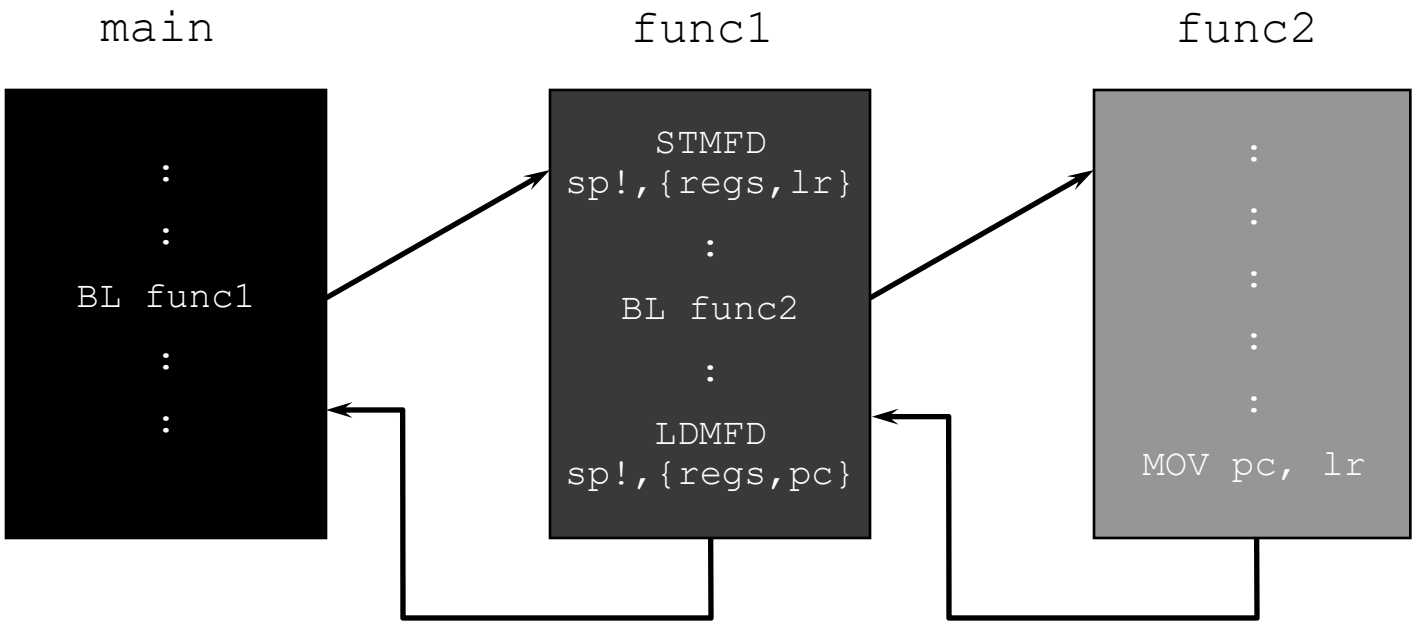
Podprogrami:

- ***klic podprograma:***
  - parametre v podprogram prenašamo v registrih od r0 naprej
  - na sklad se poleg "**delovnih registrov**" shrani tudi **r14** (lr - Link Register), v katerem je povratni naslov – s tem omogočimo **gnezdenje klicev podprogramov**
- ***vrnitev iz podprograma:***
  - "**delovni registri**" se obnovijo s sklada; povratni naslov se namesto v r14 zapiše v pc
- ***dogovor o rabi registrov:***
  - v podprogramu shranimo in obnovimo samo registre, ki so bili uporabljeni in niso služili za prenos parametrov – t.i. **delovni registri**

```
main:     ldr r13, =0x1000          @ initialize stack (stack pointer)
          mov r0, #10               @ put parameter in r0
          bl func1                  @ call subroutine func1
          …
-------------------------------------------------------------------
func1:    stmfd r13!, {r1-r3,r14}   @ save work & link regs
          …                         @ inside sub1 we use regs r1,r2,r3
          bl func2                  @ call subroutine func2
          …
          …
          ldmfd r13!, {r1-r3,pc}    @ restore work regs & return
```

```
Krajša različica
(r13,fd):
-----------------
  push {r1-r3,r14}
  …
  bl func2
  …
  …
  pop  {r1-r3,pc}
```

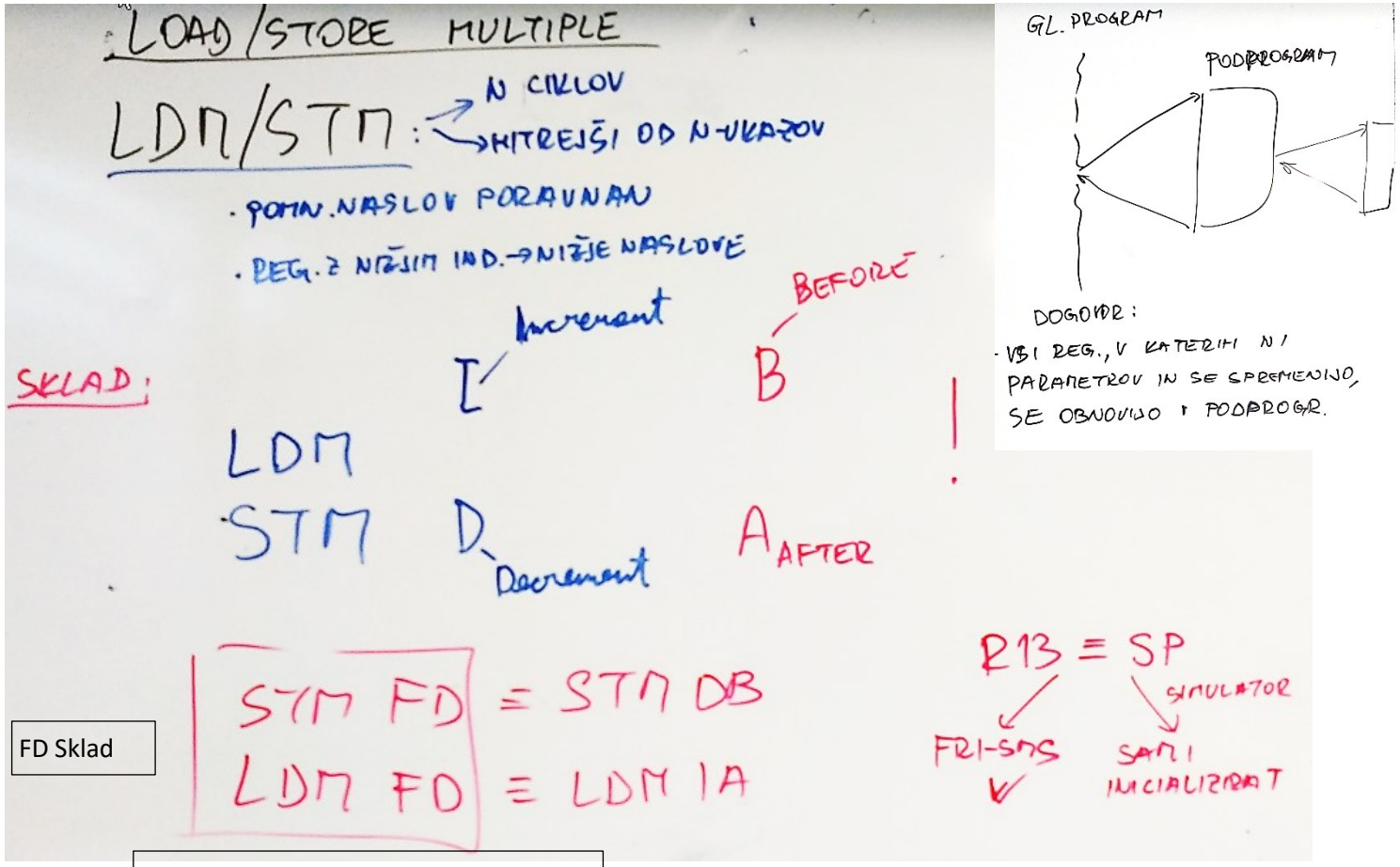
Podprogrami, sklad, uporaba/obnovitev registrov




```
main:     ldr r13, =0x1000          @ initialize stack (stack pointer)
          mov r0, #10               @ put parameter in r0
          bl func1                  @ call subroutine func1
          …
-------------------------------------------------------------------------
func1:    stmfd r13!, {r1-r3,r14}   @ save work & link regs
          …                         @ inside sub1 we use regs r1,r2,r3
          bl func2                  @ call subroutine func2
          …
          …
          ldmfd r13!, {r1-r3,pc}    @ restore work regs & return
```

```
Krajša različica
(r13,fd):
------------------
  push {r1-r3,r14}
  …
  bl func2
  …
  …
  pop  {r1-r3,pc}
```

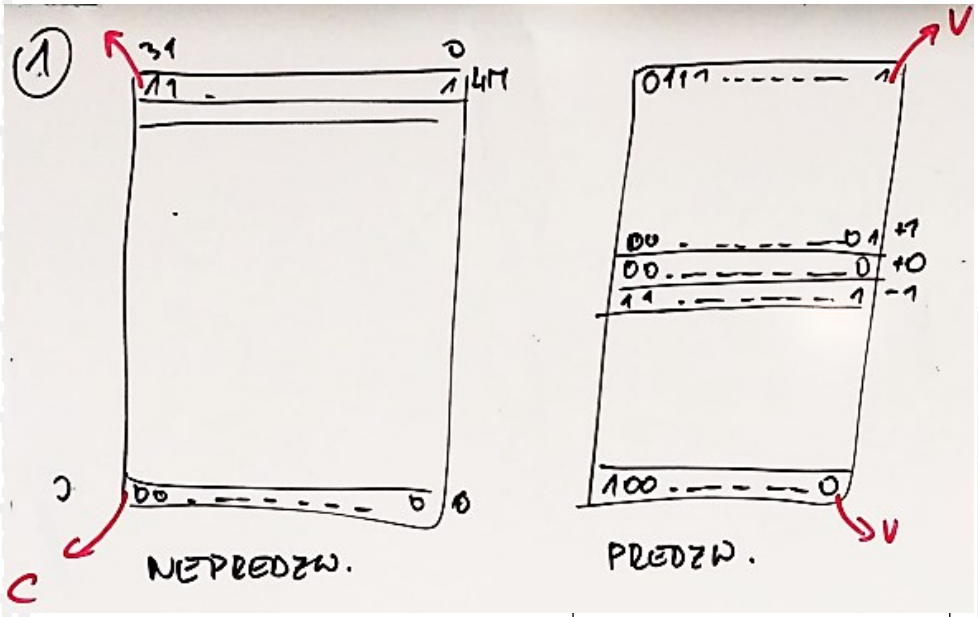
OR LAB - 3 : Tabla



FD Sklad

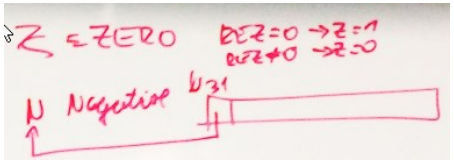
Standardni, pogosto uporabljen sklad.

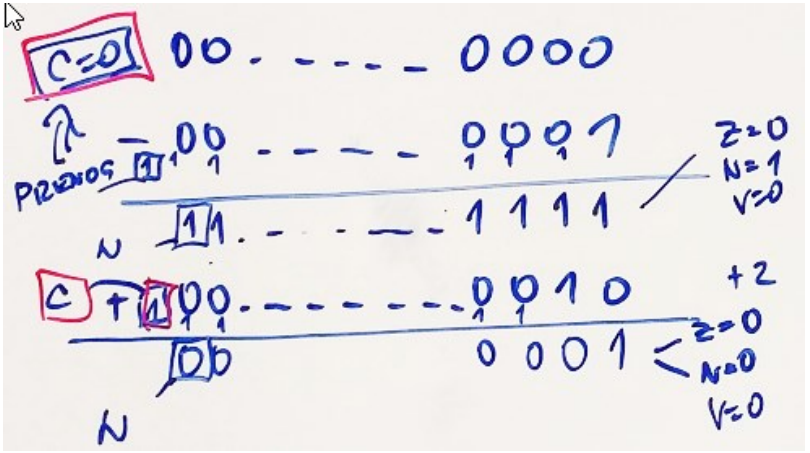
OR LAB - 3 : Tabla



Nepredznačena (0 .. +4milj.)

Predznačena (-2milj ..+2milj.)





CMP VEDNO VPLIVA NA ZAST

MOVS

ADD -

SUB -