

Load/store – 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:

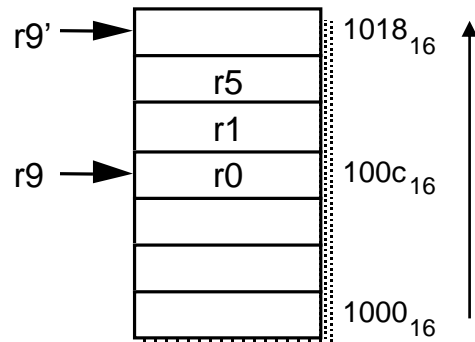
- **db** (decrement before), **da** (decrement after), **ib** (increment before), **ia** (increment after).

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
stmdb 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
```

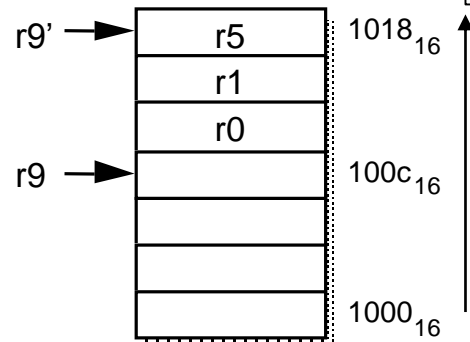
Load/store – več registrov

ia (inc. after)



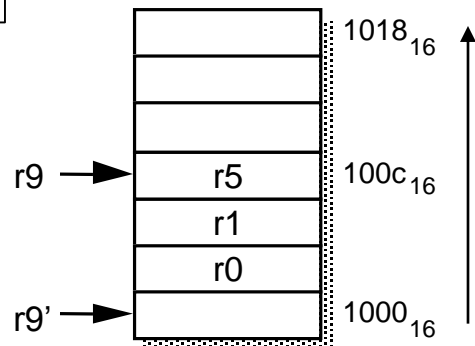
`stmia r9!, {r0,r1,r5}`

ib (inc. before)



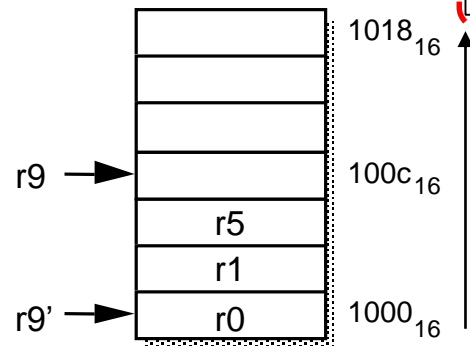
`stmib r9!, {r0,r1,r5}`

da (dec. after)



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

db (dec. before)



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

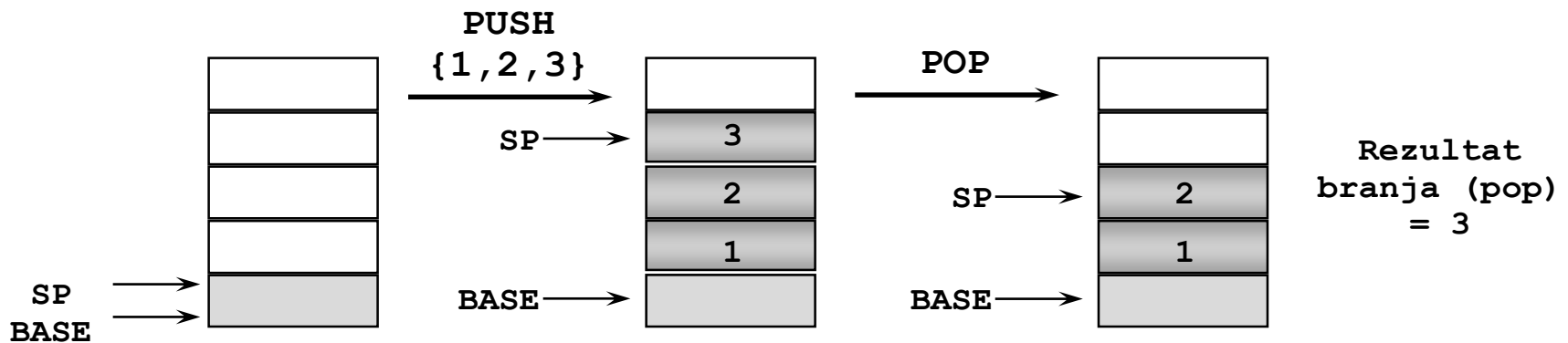
Sklad

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			LDMIB
	After	STMFA			LDMED
Decrement	Before		STMIA	LDMIA	
	After		STMEA	LDMFD	
Increment	Before		LDMDB	STMDB	
	After		LDMEA	STMFD	
Decrement	Before				
	After	LDMDA			STMDA
		LDMFA			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.

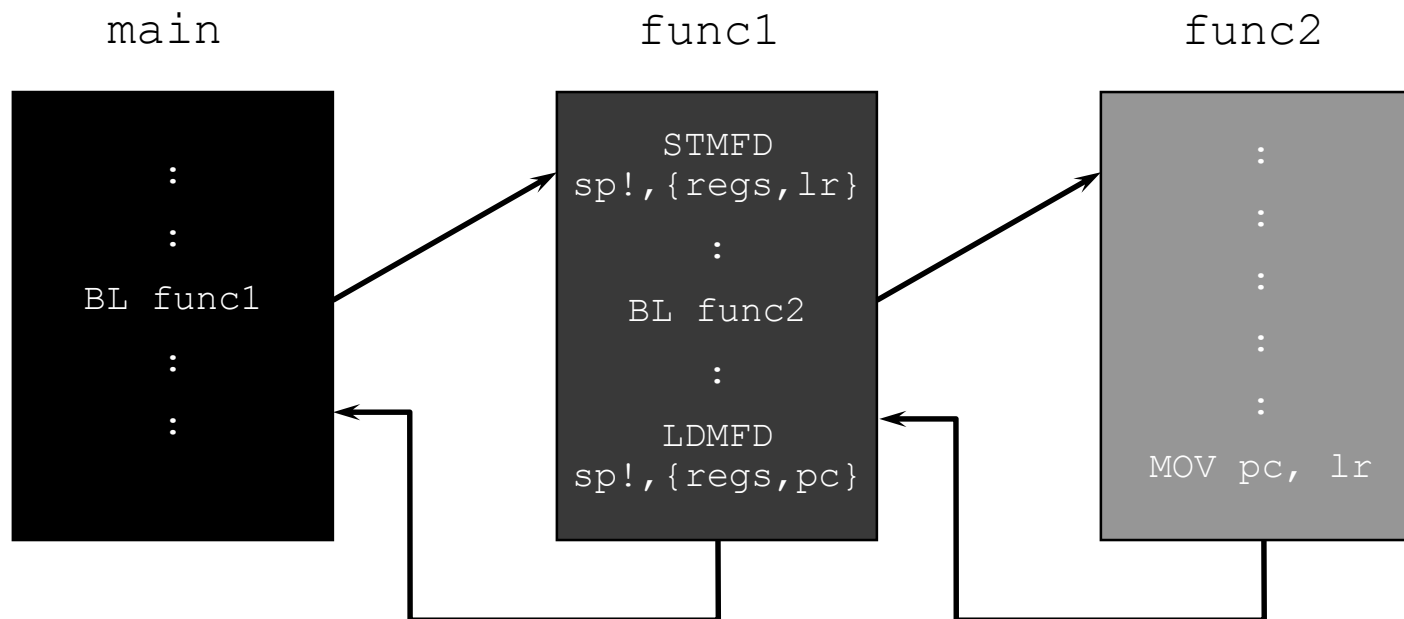
Podprogrami:

- **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
```

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
        ...
        ...
        ldmsd r13!, {r1-r3,pc} @ restore work regs & return
```

Delo na FRI-SMS razvojnem sistemu

Priključitev :

- **USB** prikllop na **daljši stranici**, sveti **zelena LED** dioda

Poseben projekt za FRI-SMS (e-učilnica) :

- **dodatne nastavitve** (informativno) :
 - frekvenca urinega signala (višja poveča porabo!)
 - vklop predpomnilnikov
 - inicializacija sklada oz. SP – kazalca na sklad
- **dodajanje vsebine (start.s):**
 - podatki/operandi:
 - dodamo v `/*constants*/` ,končamo z `.align`
 - program :
 - dodamo v `/* enter your code here */`
 - na koncu programa je mrtva zanka
 - podprograme dodamo za mrtvo zanko

