Organizacija računalnikov

Laboratorijske vaje

LAB – 03

Podprogrami, sklad

Podprogrami

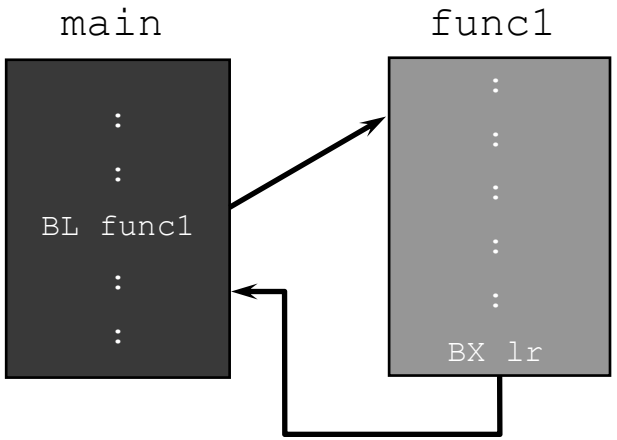
Pri klicanju podprogramov si je potrebno zapomniti povratni naslov.
ARM :

- **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 - shrani povratni naslov (pc) v r14 (lr).

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



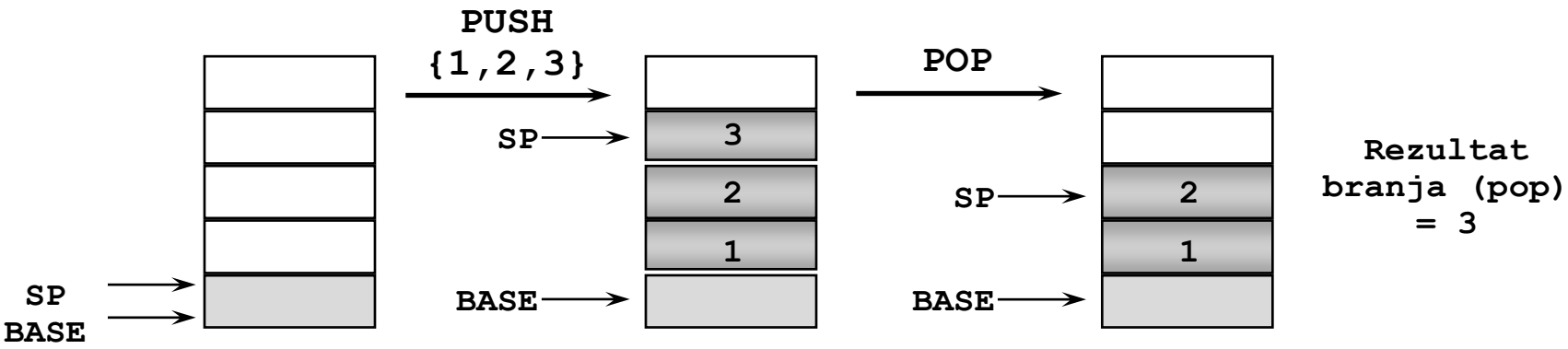
Sklad - splošno

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“



Sklad je ena najbolj pogosto uporabljanih konceptov:

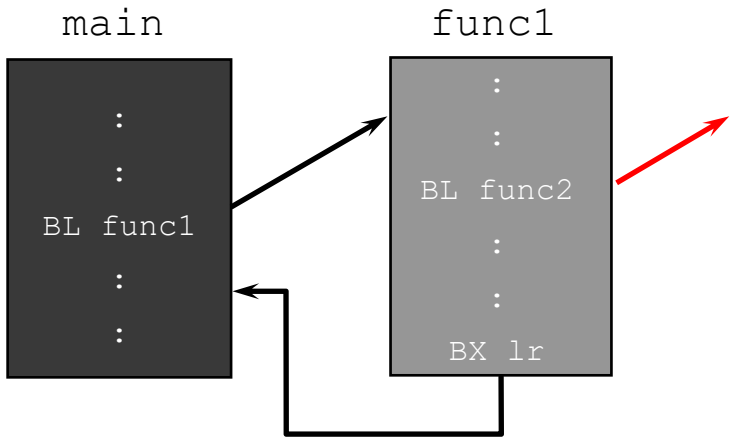
- gnezdenje klicev
- lokalni pomn. prostor (proces, lokalne spremenljivke, …)

Gnezdeni klici podprogramov

Pri klicanju podprogramov pogosto želimo gnezditi klice podprogramov
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)**

```
    Zgled:
          bl   func1
          ..
func1:    ..
          bl   func2    @ lr se prepiše?
          ..
          bx lr @ ali: mov r15,r14, 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, ne glede na vrstni red zapisa**
- **nista strogo gledano „RISC“ ukaz**

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 dve mogoči priponi:

- **db** (decrement 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.

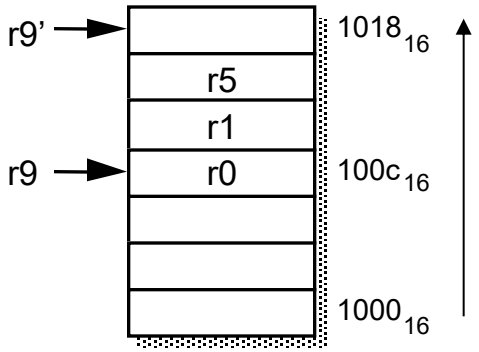
```
stmdb r13!, {r2-r9}        @ mem32[r13-4..r13-32] <- r9,…,r2
                           @ r13 <- r13-32   (8reg*4bajte=32bajtov)

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
```

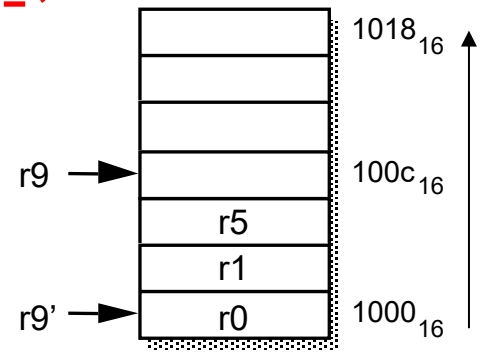
Load/store multiple – beri/shrani več registrov

ia (inc. after)




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

db (dec. before)




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

Load/store multiple
Primer 1: bločno kopiranje vsebine

```
main:
        ldr r0, =IZVOR          // r0 = naslov izvora
        ldr r1, =CILJ           // r1 = naslov cilja
        mov r2, #N_BESED        // r2 = stevilo besed
        bl KOPIRAJ
        …

k_blok:
        ldmia r0!, {r4-r11}     // preberi 8 besed, r0 += 32
        stmia r1!, {r4-r11}     // zapisi 8 besed, r1 += 32
        subs r3, r3, #1
        bne k_blok
```

Load/store – več registrov, sklad

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

Podprti sta dve različici skladov, od tod kratice:

- ***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*

Običajno uporabljamo FD sklad :

- ***vpis-DB:*** ***STMFD=STMDB=PUSH***
- ***branje-IA:*** ***LDMFD=LDMIA=POP***

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:

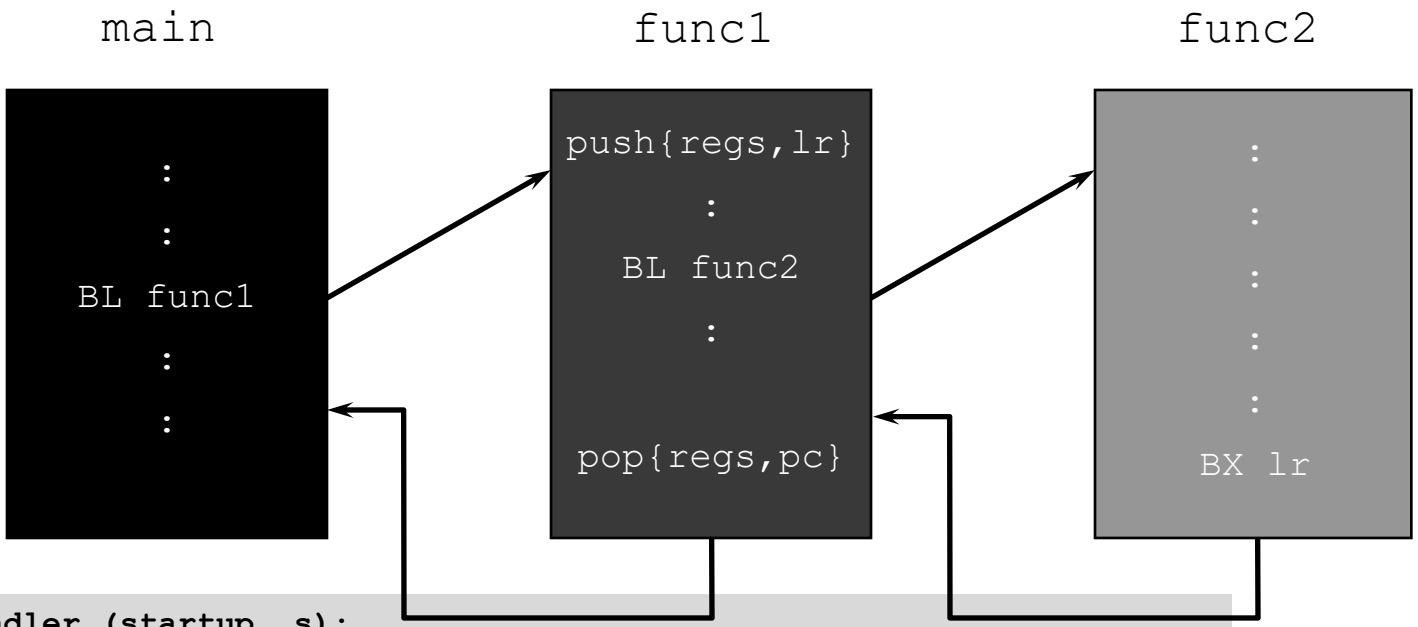
- ***klic podprograma:***
  - parametre v podprogram prenašamo v registrih od r0 do r3 (ostali na sklad)
  - na sklad se poleg "**delovnih registrov**" shrani tudi **r14** (lr - Link Register), v katerem je povratni naslov – s tem omogočimo **gnezdenje klicev podprogramov**
- ***vrnitev iz podprograma:***
  - "**delovni registri**" se obnovijo s sklada; povratni naslov se namesto v r14 zapiše v pc (hkrati tudi vrnitev iz podprograma)
- ***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**

```
Reset_Handler (startup….s):
          ldr sp, =_estack @ initialize stack (stack pointer)

main:     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}
```

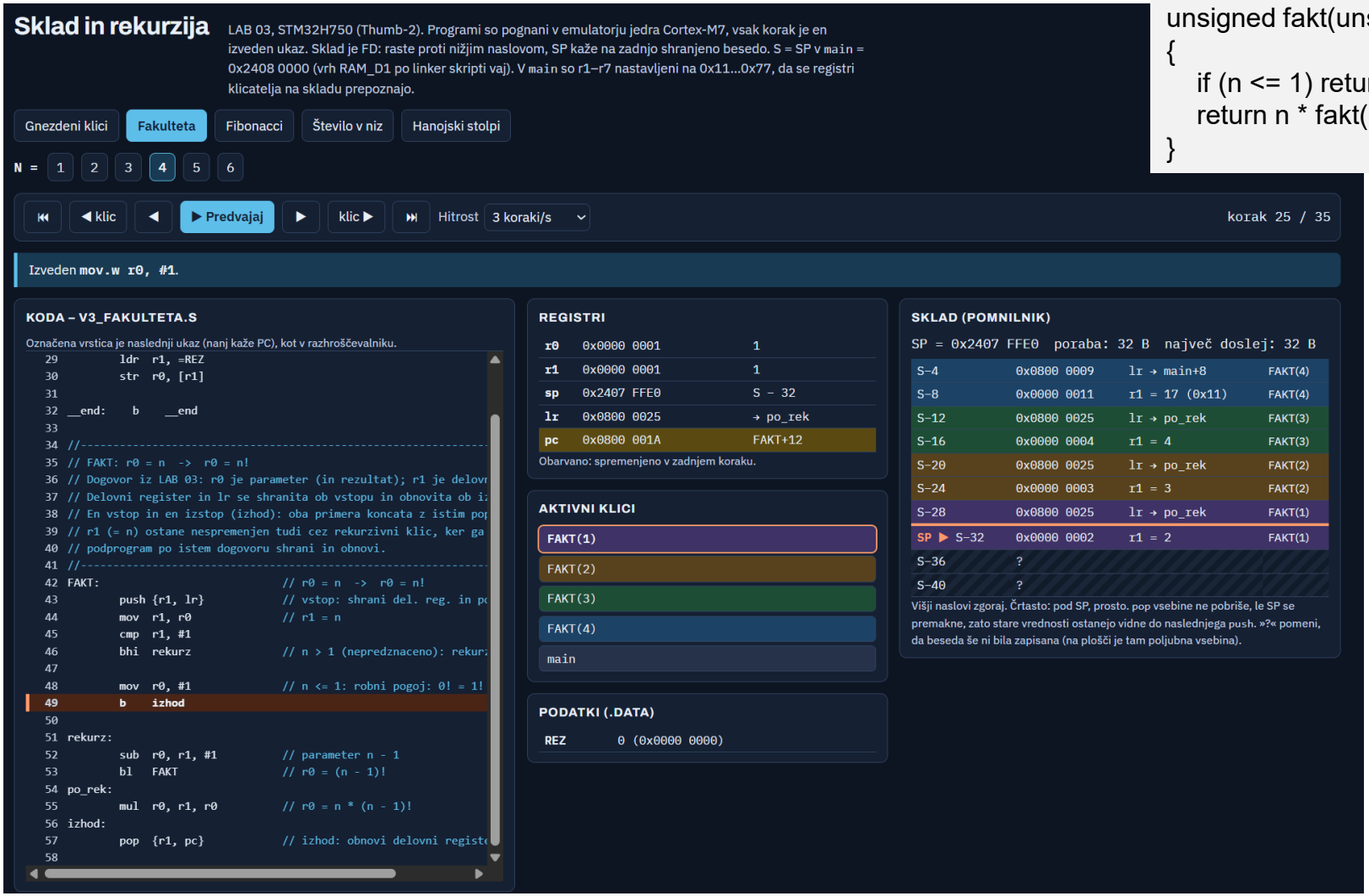
Podprogrami, sklad, uporaba/obnovitev registrov




```
Reset_Handler (startup….s):
          ldr sp, =_estack @ initialize stack (stack pointer)
          main:     mov r0, #10                @ put parameter in r0
          bl func1                    @ call subroutine func1
          …
-------------------------------------------------------------------------
func1:    stmfd r13!, {r1-r3,r14}     @ save work & link regs
          …                           @ inside func1 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: Primer rekurzije – izračun fakultete




```
unsigned fakt(unsigned n)
{
    if (n <= 1) return 1;
    return n * fakt(n - 1);
}
```

https://lapsylab.github.io/ORLAB/Sklad%20in%20rekurzija.html