# ARM

# PROGRAMIRANJE V ZBIRNEM JEZIKU

## 1. del

# *ARM (Advanced RISC Machine) = RISC?*

**+ load/store arhitektura**

**+ cevovodna zgradba**

**+ reduciran nabor ukazov, vsi ukazi 32-bitni**

**+ ortogonalen registrski niz. Vsi registri 32-bitni**

**- veliko načinov naslavljanja**

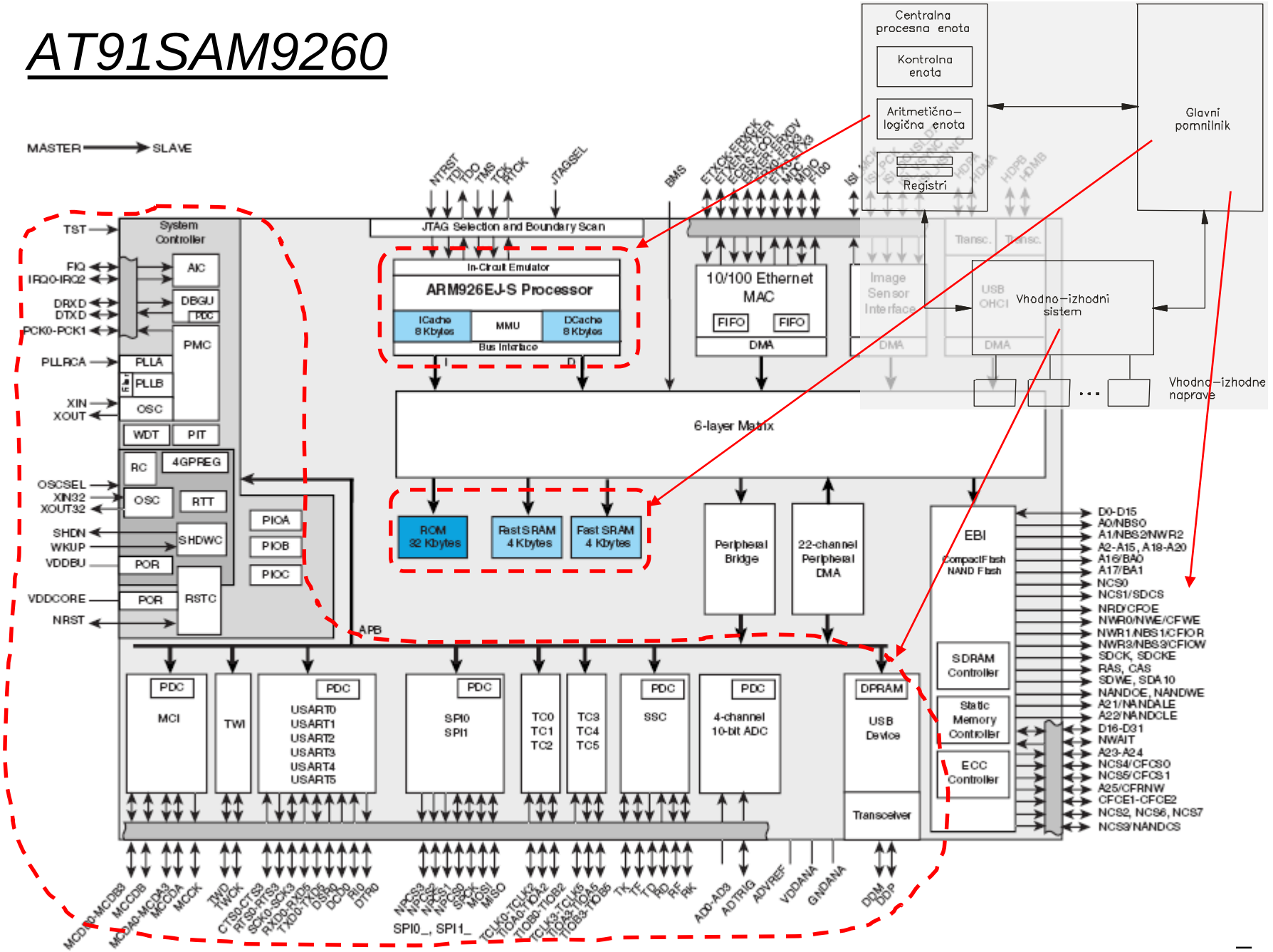
**- veliko formatov ukazov**

**- nekateri ukazi se izvajajo več kot en cikel (npr. *load/store multiple*) – obstaja nekaj kompleksnejših ukazov, kar omogoča manjšo velikost programov**

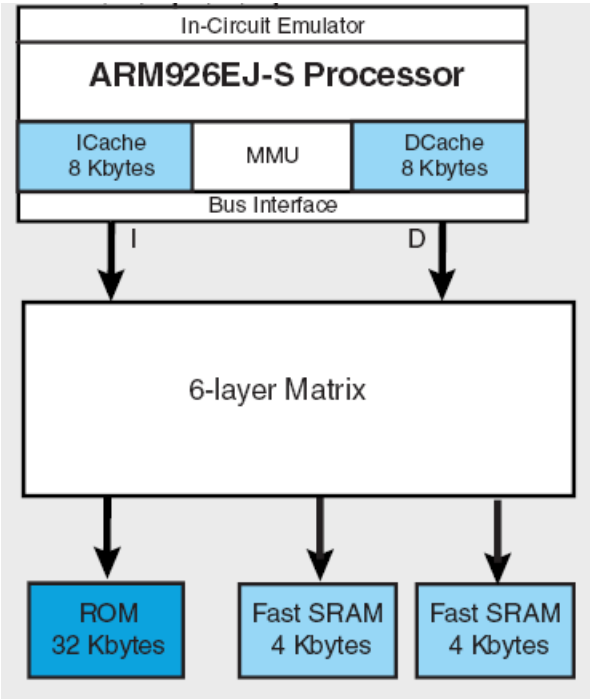
**- dodaten 16-bitni nabor ukazov Thumb omogoča krajše programe**

**- pogojno izvajanje ukazov – ukaz se izvede le, če je stanje zastavic ustrezno.**

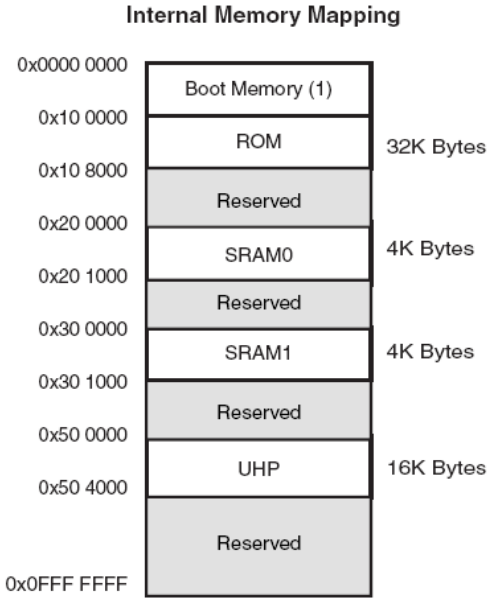
# AT91SAM9260



# AT91SAM9260



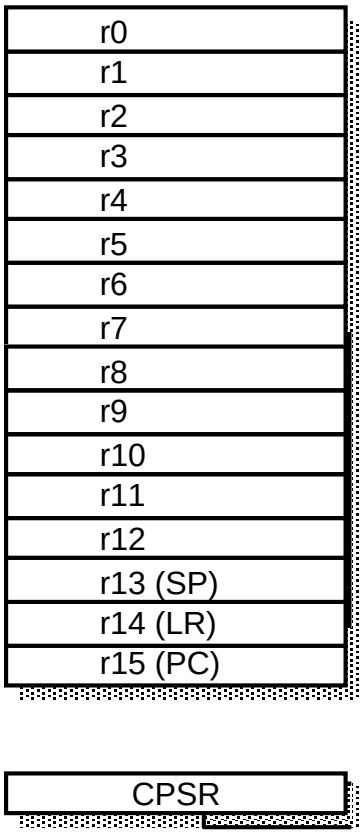

Shema pomnilniškega prostora



# *ARM programski model*

- **Programski model sestavlja 16 registrov ter statusni register CPSR (Current Program Status Register)**
- **Več** *načinov delovanja*, **vsak ima nekaj svojih registrov. Vseh registrov je v resnici 36**
- **Kateri registri so vidni je odvisno od načina delovanja procesorja (*processor mode*)**
- **Načine delovanja delimo v dve skupini:**
  - Privilegirani (dovoljena bralni in pisalni dostop do CPSR)
  - Neprivilegirani (dovoljen le bralni dostop do CPSR)

# *Programski model – uporabniški način*




Uporabniški način (*user mode*):

- edini neprivilegirani način
- v tem načinu se izvajajo uporabniški programi

Programsko je vidnih 17 32-bitnih registrov:

r0 – r15 ter CPSR

Vidni registri:

- r0-r12: splošnonamenski (ortogonalni) registri
- r13(sp): skladovni kazalec (*Stack Pointer*)
- r14(lr): povratni naslov (*Link Register*)
- r15(pc): programski števec (*Program Counter*)
- CPSR: statusni register (*Current Program Status Register*)

# Statusni register – CPSR

*CPSR - Current Program Status Register*

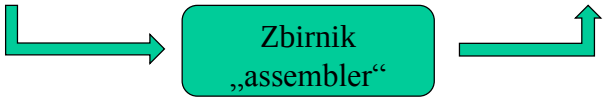

- zastavice (N,Z,V,C)
- maskirna bita za prekinitve (I, F)
- bit T določa nabor ukazov:
  T=0 : ARM arhitektura, procesor izvaja 32-bitni ARM nabor ukazov
  T=1: Thumb arhitektura, procesor izvaja 16-bitni Thumb nabor ukazov
- spodnjih 5 bitov določa način delovanja procesorja
- v uporabniškem (neprivilegiranem) načinu lahko CPSR beremo; ukazi lahko spreminjajo le zastavice.

**Zastavice (lahko) ukazi spreminjajo glede na rezultat ALE:**

N = 0: bit 31 rezultata je 0, N=1: bit 31 rezultata je 1 (*Negative*)
Z = 1: rezultat je 0, Z=0: rezultat je različen od nič (*Zero*)
C = 1: rezultat je povzročil prenos, C = 0: rezultat ni povzročil prenosa (*Carry*)
V = 1: rezultat je povzročil preliv, V = 0: rezultat ni povzročil preliva (*oVerflow*)

# Programiranje v zbirniku

| Zbirni jezik | Opis ukaza | Strojni jezik |
|---|---|---|
| ldr r1, stev1 | R1 ← M[0x20] | 0xE51F1014 |
| ldr r2, stev2 | R2 ← M[0x24] | 0xE51F2014 |
| add r3, r2, r1 | R3 ← R1 + R2 | 0xE0823001 |
| str r3, rez | M[0x28] ← R3 | 0xE50F3018 |




- **V zbirniku simbolično opisujemo:**
  - ukaze (z mnemoniki),
  - registre,
  - naslove
  - konstante

- **Programerju tako ni treba:**
  - poznati strojnih ukazov in njihove tvorbe
  - računati odmikov ter naslovov

**Prevajalnik za zbirnik (*assembler*) :**
- prevede simbolično predstavitev ukazov v ustrezne strojne ukaze,
- izračuna dejanske naslove ter
- ustvari pomnilniško sliko programa

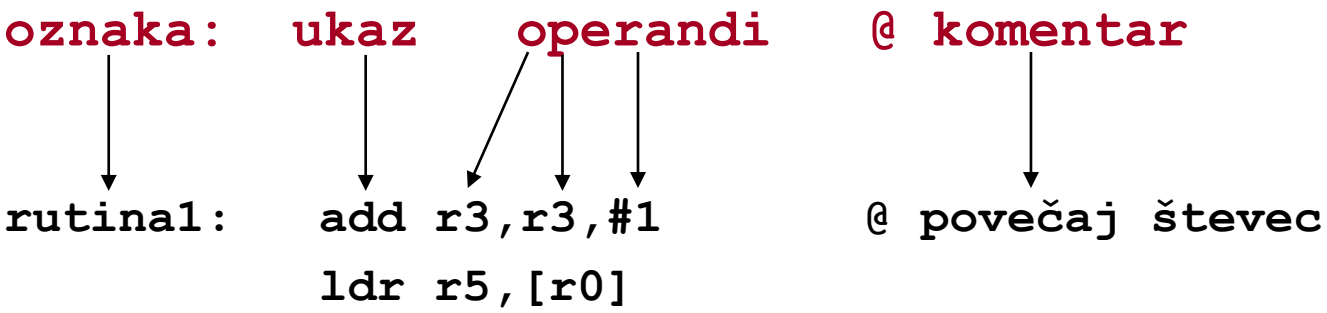
- **Program v strojnem jeziku ni prenosljiv:**
  - namenjen je izvajanju le na določeni vrsti mikroprocesorja

- **Zbirnik (*assembly language*) je „nizkonivojski“ programski jezik**

# *Programiranje v zbirniku*

- **Vsaka vrstica programa v zbirniku predstavlja običajno en ukaz v strojnem jeziku**

- **Vrstica je sestavljena iz štirih stolpcev:**




- **Stolpce ločimo s tabulatorji, dovoljeni so tudi presledki**

# *Ukazi*

- **Vsi ukazi so 32-bitni**

```
add r3, r2, r1 ⟹ 0xE0823001=0b1110…0001
```

- **Vsi operandi se v registrih razširijo na 32-bitov**

```
0xFF           ⟹ 0x000000FF
```

- **Rezultat je 32-biten. Izjema je le množenje**

```
R1 + R2        ⟹ R3
```

- **Aritmetično-logični ukazi so 3-operandni**

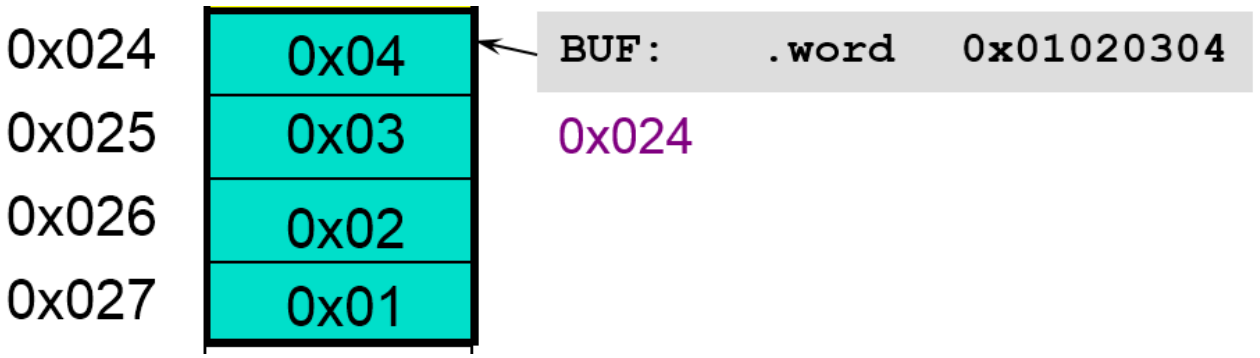
```
add r3,r3,#1
```

- **Load/store arhitektura**

```
ldr r1, stev1       @ prenos v registre
ldr r2, stev2       @ prenos v registre
add r3, r2, r1      @ vsota registrov
```

# *Operandi*

- **8, 16, 32-bitni predznačeni ali nepredznačeni pomnilniški operandi**
- **Obvezna poravnanost ukazov in operandov (16,32bitnih)**
- **V CPE se vse izvaja 32-bitno (razširitev ničle ali predznaka)**
- **Uporablja se pravilo tankega konca**



# *Oznake (labele)*

Oznaka je nam razumljivo **simbolično poimenovanje :**

- pomnilniških lokacij ali
- vrstic v programu

Oznake običajno uporabljamo na dva načina:

- s poimenovanjem pomnilniških lokacij dobimo „spremenljivke“

```
STEV1:      .word   0x12345678
STEV2:      .byte   1,2,3,4
REZ:        .space  4
```

- za poimenovanje ukazov (vrstic), na katere se sklicujemo pri skokih.

```
            mov r4,#10
LOOP:       subs r4, r4, #1
            ...
            bne LOOP
```

# *Psevdoukazi - ukazi prevajalniku*

**Psevdoukazi:**

- so navodila prevajalniku
- običajno so označeni s piko pred ukazom
- niso strojni ukazi za CPE, temveč ukazi prevajalniku
- v končnem strojnem programu (izvaja CPE) jih ni

**Psevdoukaze uporabljamo za:**

| | |
|---|---|
| • določanje vrste pomnilniških odsekov | `.text .data` |
| • poravnavo vsebine | `.align` |
| • rezervacijo pomnilnika za „spremenljivke“ | `.space` |
| • rezervacijo prostora v pomnilniku | `.space` |
| • določanje začetne vsebine pomnilnika | `.(h)word, .byte,…` |
| • ustavljanje prevajanja | `.end` |

# *Določanje pomnilniških odsekov*

**Psevdoukaza za določanje pomnilniške slike sta:**

```
.data
.text
```

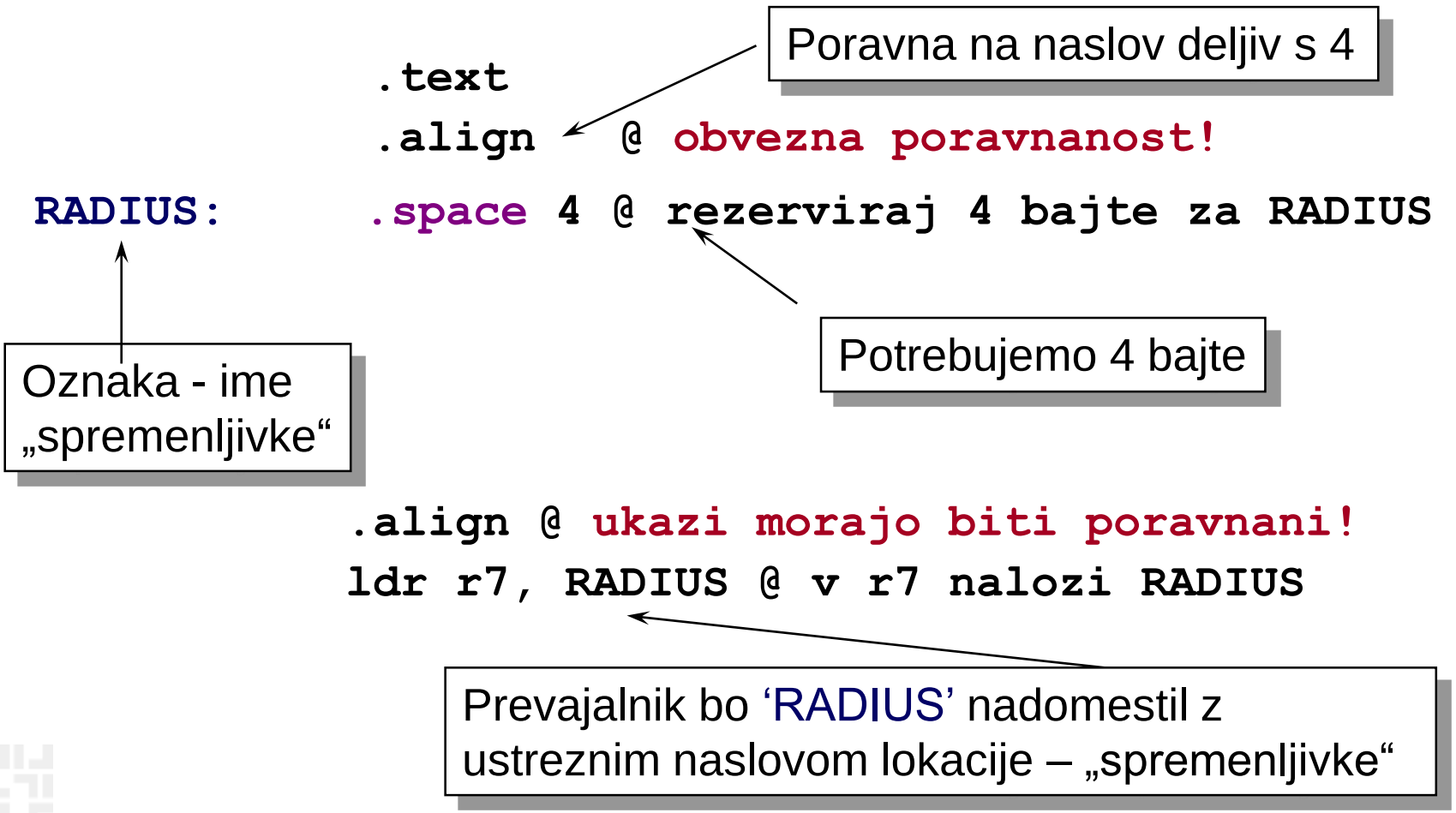
**S tema psevdoukazoma določimo, kje v pomnilniku bodo program(i) in kje podatki.**

**Tako za ukaze programa kot operande bomo uporabljali segment**

```
.text
```

# *Rezervacija pomnilnika za „spremenljivke“*

Za spremenljivke moramo v pomnilniku rezervirati določen prostor.



# *Rezervacija prostora v pomnilniku*

Oznake omogočajo boljši pregled nad pomnilnikom:

– pomnilniškim lokacijam dajemo imena in ne uporabljamo absolutnih naslovov (preglednost programa)

```
BUFFER:     .space 40     @rezerviraj 40 bajtov
BUFFER2:    .space 10     @rezerviraj 10 bajtov
BUFFER3:    .space 20     @rezerviraj 20 bajtov

;poravnanost? Če so v rezerviranih blokih bajti, ni
težav, sicer je (morda) potrebno uporabiti .align
```

- **oznaka BUFFER ustreza naslovu, od katerega naprej se rezervira 40B prostora**
- **oznaka BUFFER2 ustreza naslovu, od katerega naprej se rezervira 10B prostora. Ta naslov ja za 40 večji kot BUFFER**
- **oznaka BUFFER3 ustreza naslovu, od katerega naprej se rezervira 20B prostora. Ta naslov ja za 10 večji kot BUFFER2**

# *Rezervacija prostora z zač. vrednostmi*

Večkrat želimo, da ima spremenljivka neko začetno vrednost.

```
niz1:   .asciz          "Dober dan"
niz2:   .ascii          "Lep dan"
        .align
stev1:  .word           512,1,65537,123456789
stev2:  .hword          1,512,65534
stev3:  .hword          0x7fe
Stev4:  .byte           1, 2, 3
        .align
naslov: .word           niz1
```

- **„spremenljivke“, inicializirane na ta način, lahko kasneje v programu spremenimo (ker so le naslovi pomnilniških lokacij)**
- **če želimo, da je oznaka vidna tudi v drugih datotekah projekta, uporabimo psevdoukaz .global, npr:**

```
.global niz1, niz2
```

# Povzetek – psevdoukazi