

ARM

PROGRAMIRANJE V
ZBIRNEM JEZIKU

1. del

ARM

PROGRAMIRANJE V
ZBIRNEM JEZIKU

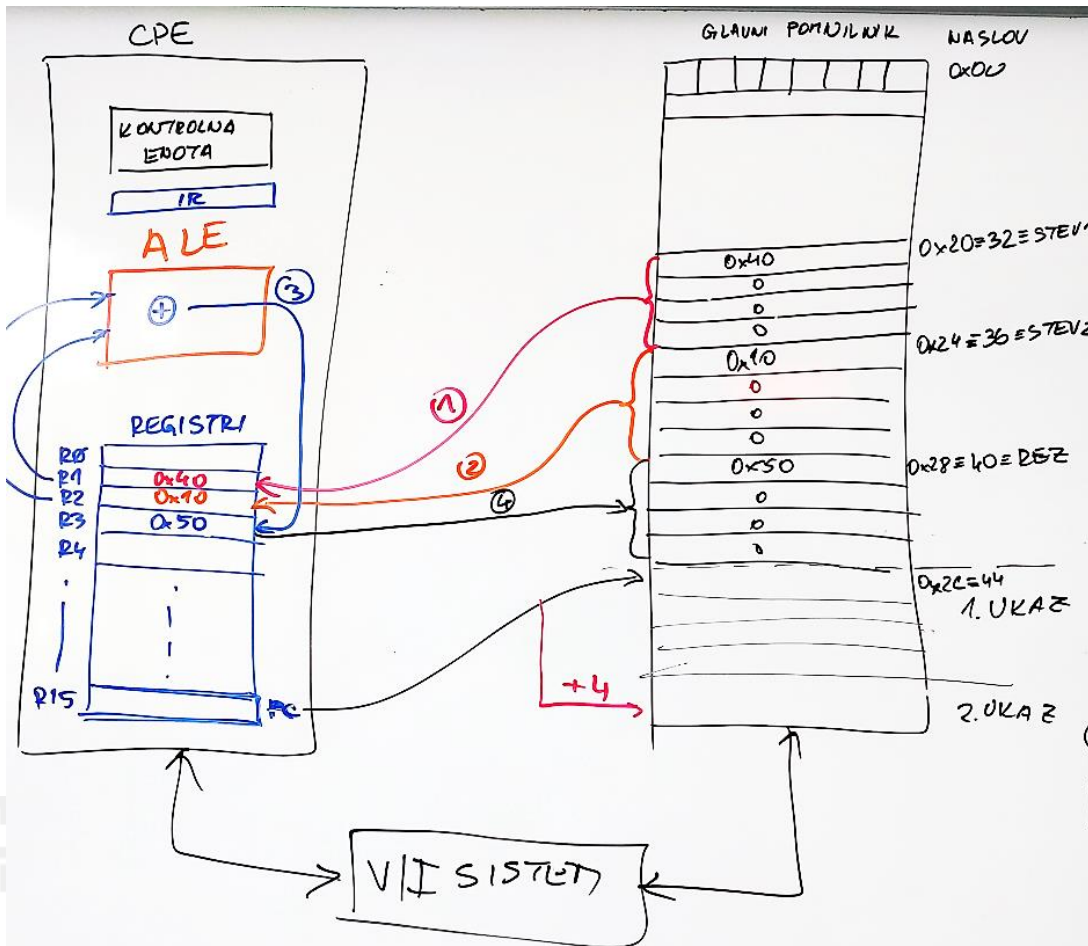
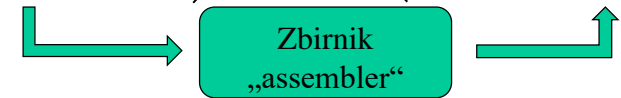
1. del

RA LAB 2.1 Osnove ARM mikrokrmilnika

Uvodna vaja: Programiranje v zbirniku

Zgled seštevanja dveh števil :
rez := stev1 + stev2

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

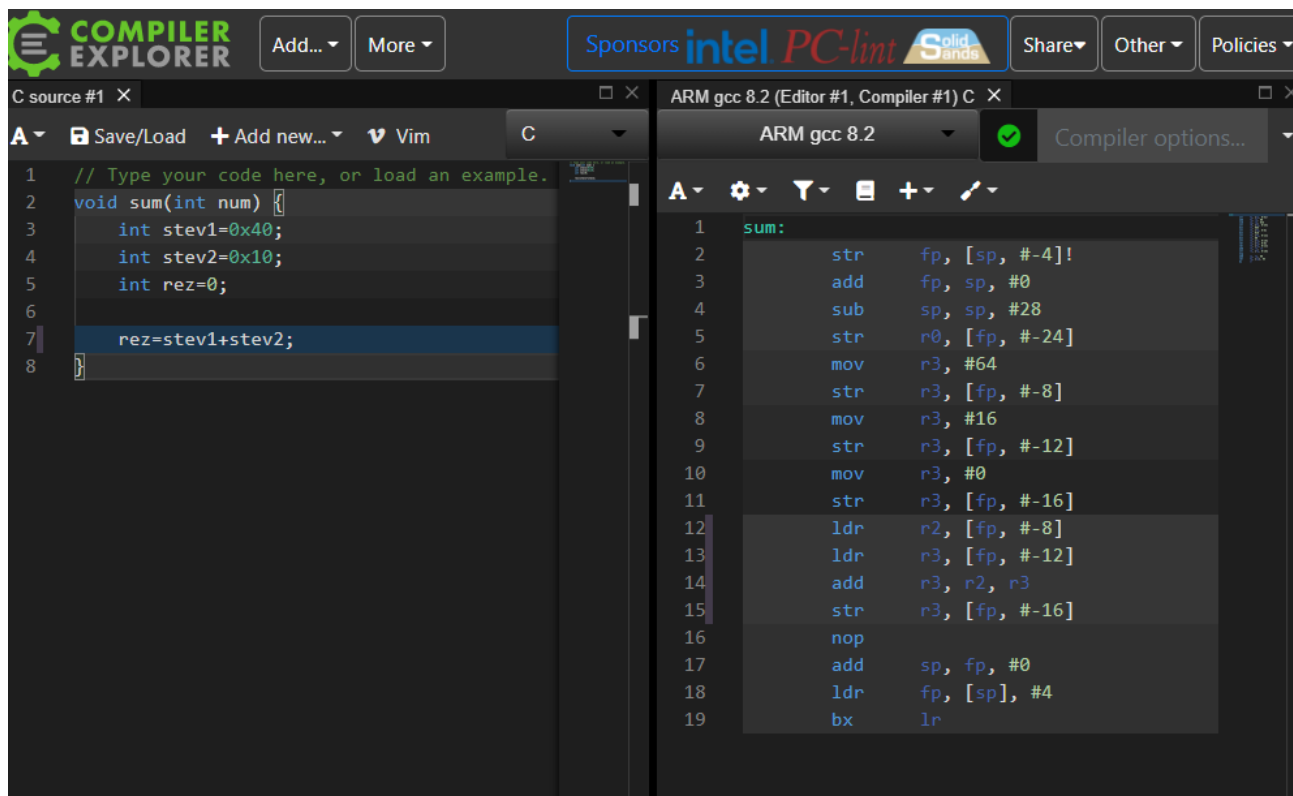
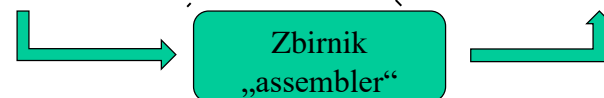


Uvodna vaja: Programiranje v zbirniku

Zgled seštevanja dveh števil :
rez := stev1 + stev2

Primer C kode prevedene v zbirnik ARM

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



<https://godbolt.org/>

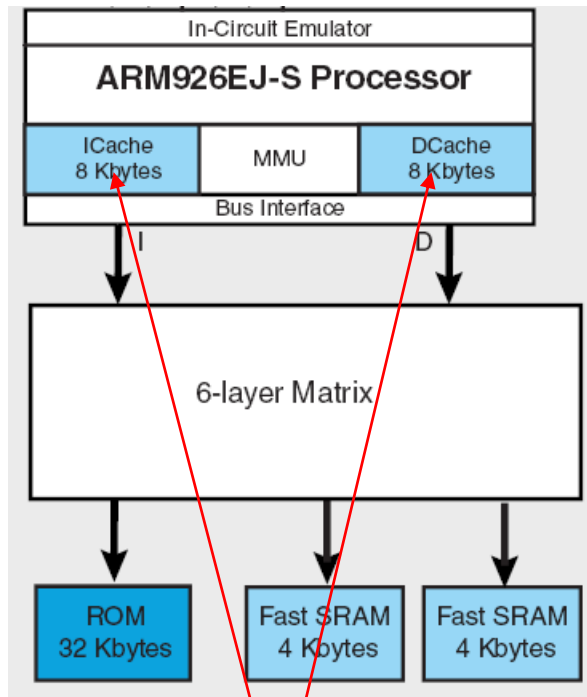
ARM (Advanced RISC Machine) = RISC?

- + 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

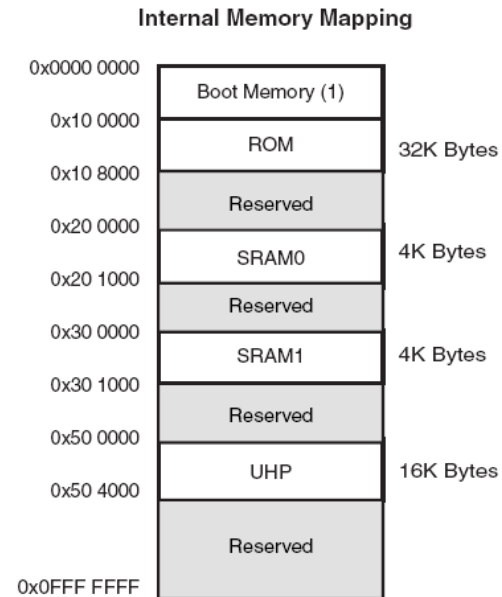
- nekateri ukazi se izvajajo več kot en cikel (npr. *load/store multiple*) – obstaja nekaj kompleksnejših ukazov, kar omogoča manjšo velikost programov
- dodaten 16-bitni nabor ukazov Thumb omogoča krajše programe
- pogojno izvajanje ukazov – ukaz se izvede le, če je stanje zastavic ustrezno.

AT91SAM9260



Harvardska arhitektura
predpomnilnikov

Shema pomnilniškega prostora

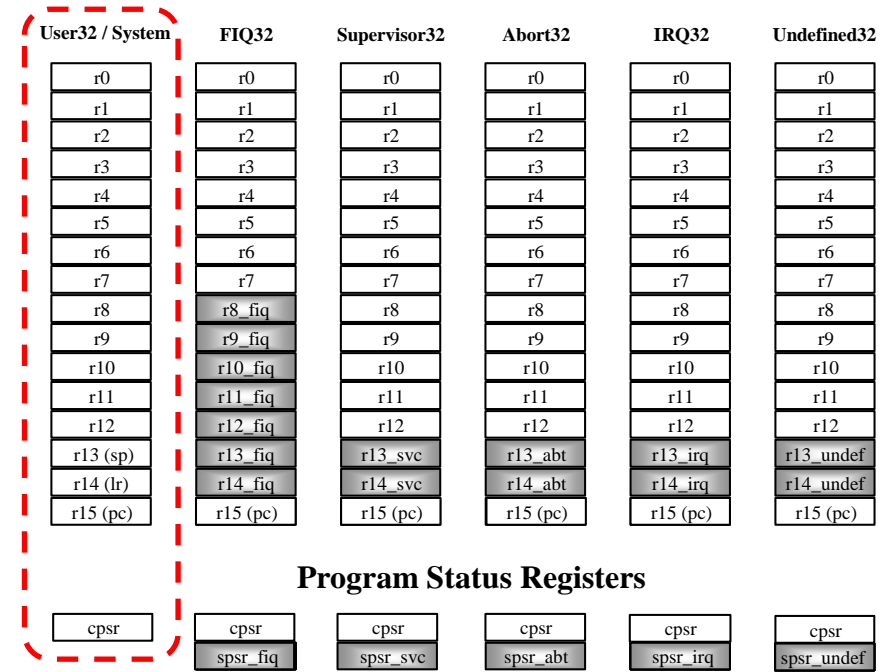


Princetonska arhitektura
glavnega pomnilnika

ARM programski model

General registers and Program Counter

- 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

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)

Programsko je vidnih 17 32-bitnih registrov:
r0 – r15 ter CPSR

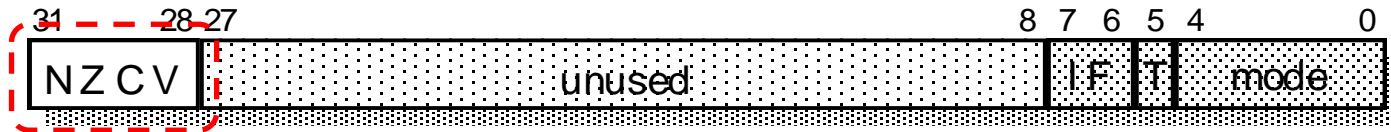
Vidni registri:

- 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*)

CPSR

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	(<i>Negative</i>)
Z = 1: rezultat je 0,	Z = 0: rezultat je različen od nič	(<i>Zero</i>)
C = 1: rezultat je povzročil prenos,	C = 0: rezultat ni povzr. Prenosa	(<i>Carry</i>)
V = 1: rezultat je povzročil preliv,	V = 0: rezultat ni povzr. Preliva	(<i>oVerflow</i>)

ARM

PROGRAMIRANJE V
ZBIRNEM JEZIKU

1. del

RA LAB 2.2 Osnove ARM zbirnika

Programiranje v zbirniku

- **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*) :

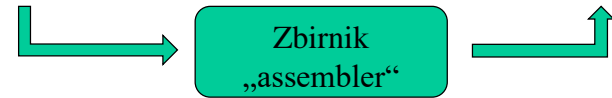
- prevede simbolično predstavitev ukazov v ustrezne strojne ukaze,
- izračuna dejanske naslove ter
- ustvari pomnilniško sliko programa

- **Program v strojnem jeziku ni prenosljiv:**

- namenjen je izvajanju le na določeni vrsti mikroprocesorja

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

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



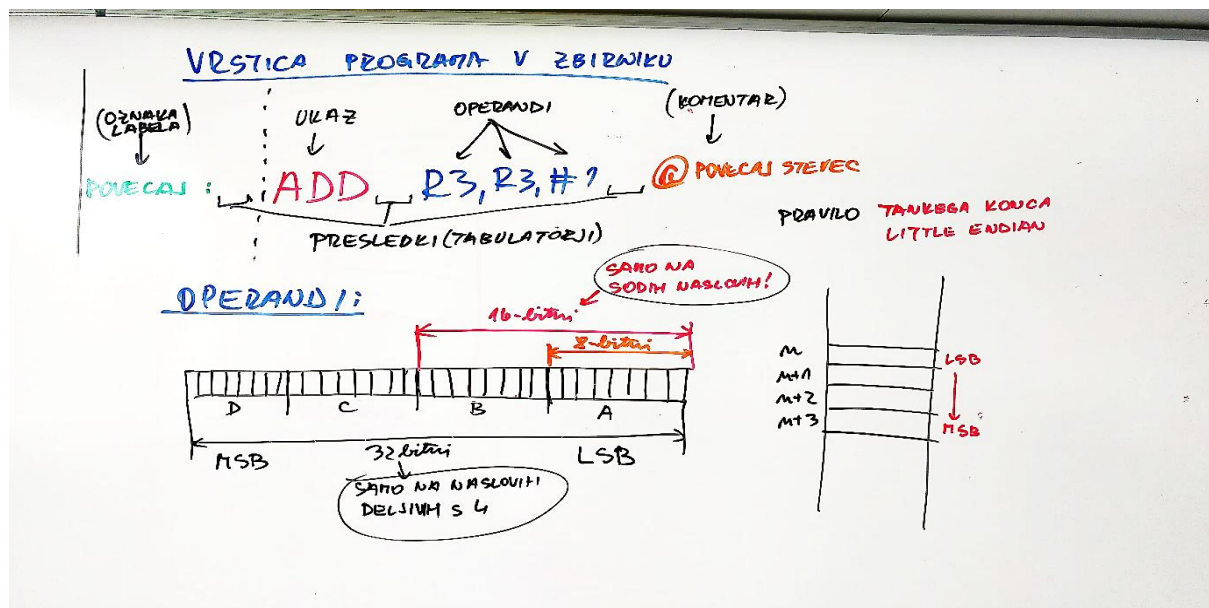
Programiranje v zbirniku – pripomočki

ARMv4T Partial Instruction Set Summary

- Spisek vseh ukazov
 - E-učilnica

Operation	Syntax	
Move	Move	<code>mov{cond}{s} Rd, shift_op</code>
	with NOT	<code>mvn{cond}{s} Rd, shift_op</code>
	CPSR to register	<code>mrs{cond} Rd, cpsr</code>
	SPSR to register	<code>mrs{cond} Rd, spsr</code>
	register to CPSR	<code>msr{cond} cpsr_fields, Rm</code>
	register to SPSR	<code>msr{cond} spsr_fields, Rm</code>
	immediate to CPSR	<code>msr{cond} cpsr_fields, #imm8r</code>
	immediate to SPSR	<code>msr{cond} spsr_fields, #imm8r</code>
Arithmetic	Add	<code>add{cond}{s} Rd, Rn, shift_op</code>
	with carry	<code>adc{cond}{s} Rd, Rn, shift_op</code>
	Subtract	<code>sub{cond}{s} Rd, Rn, shift_op</code>
	with carry	<code>sbc{cond}{s} Rd, Rn, shift_op</code>
	reverse subtract	<code>rsb{cond}{s} Rd, Rn, shift_op</code>
	reverse subtract with carry	<code>rsc{cond}{s} Rd, Rn, shift_op</code>
	Multiply	<code>mul{cond}{s} Rd, Rm, Rs</code>
	with accumulate	<code>mla{cond}{s} Rd, Rm, Rs, Rn</code>
	unsigned long	<code>umull{cond}{s} RdLo, RdHi, Rm, Rs</code>
	unsigned long with accumulate	<code>umlal{cond}{s} RdLo, RdHi, Rm, Rs</code>
	signed long	<code>smull{cond}{s} RdLo, RdHi, Rm, Rs</code>
	signed long with accumulate	<code>smlal{cond}{s} RdLo, RdHi, Rm, Rs</code>

- Lasten A4 list z zapiski – primer zapiskov na tablo



Ukazi

- **Vsi ukazi so 32-bitni**

```
add r3, r2, r1  $\implies$  0xE0823001=0b1110...0001
```

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

```
R1 + R2  $\implies$  R3
```

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

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

- **Load/store arhitektura (model delovanja)**

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

Programiranje v zbirniku

- Vsaka vrstica programa v zbirniku predstavlja običajno en ukaz v strojnem jeziku
- Vrstica je sestavljena iz štirih stolpcev:

oznaka:	ukaz	operandi			@ komentar
↓	↓	↙	↓	↓	↓
rutina1:	add	r3,	r3,	#1	@ povečaj števec
	ldr	r5,	[r0]		

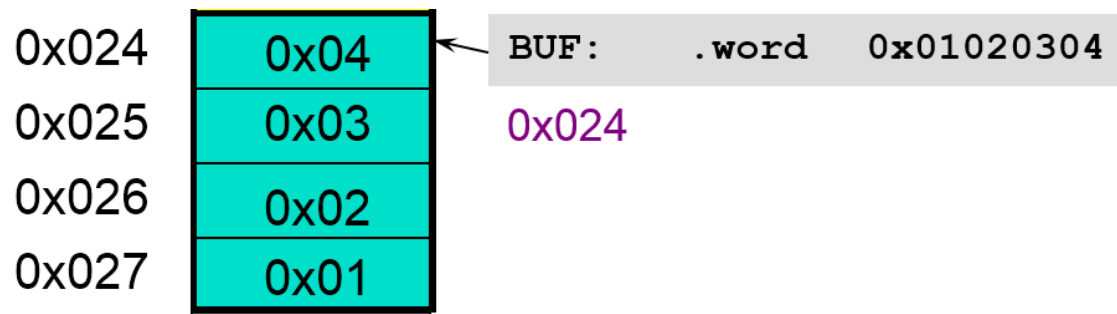
- Stolpce ločimo s tabulatorji, dovoljeni so tudi presledki

Operandi

- 8, 16, 32-bitni ter predznačeni ali nepredznačeni pomnilniški operandi
- Obvezna poravnanosť ukazov in operandov (16,32bitnih):
 - 16-bitni poravnani na sodih naslovih
 - 32-bitni poravnani na naslovih, deljivih s 4
- V CPE se vse izvaja 32-bitno (razširitev ničle ali predznaka)

0xFF \implies 0x000000FF

- Daljši operandi: 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
```

```
                .text
stev1:          .word  64
stev2:          .word  0x10
rez:           .space 4
                .align
                .global __start
__start:
                ldr   r1, stev1
                ldr   r2, stev2
                add  r3, r2, r1
                str  r3, rez
                b    __end
__end:
```

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

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

ARM

PROGRAMIRANJE V
ZBIRNEM JEZIKU

1. del

RA LAB 2.3 Psevdoukazi in direktive

Psevdoukazi in direktive - ukazi prevajalniku

Psevdoukazi :

- niso dejanski strojni ukazi za CPE, temveč jih prevajalnik vanje prevede

Primer:

```
adr r0, stevl    prevajalnik nadomesti npr. s  sub r0, pc, #2c  
                (ALE ukaz, ki izračuna pravi naslov v r0)
```

Direktive uporabljamo za:

- direktive so označene s piko pred ukazom
- 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**

Obojih v končnem strojnem programu (izvaja CPE) ni !!!

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.

```
.text  
.align @ obvezna poravnost!  
.space 4 @ rezerviraj 4 bajte za RADIUS
```

Poravna na naslov deljiv s 4

RADIUS:

Oznaka - ime
„spremenljivke“

Potrebujemo 4 bajte

```
.align @ ukazi morajo biti poravnani!  
ldr r7, RADIUS @ v r7 nalozi RADIUS
```

Prevajalnik bo 'RADIUS' nadomestil z
ustreznim naslovom lokacije – „spremenljivke“

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
```

;poravnano? 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

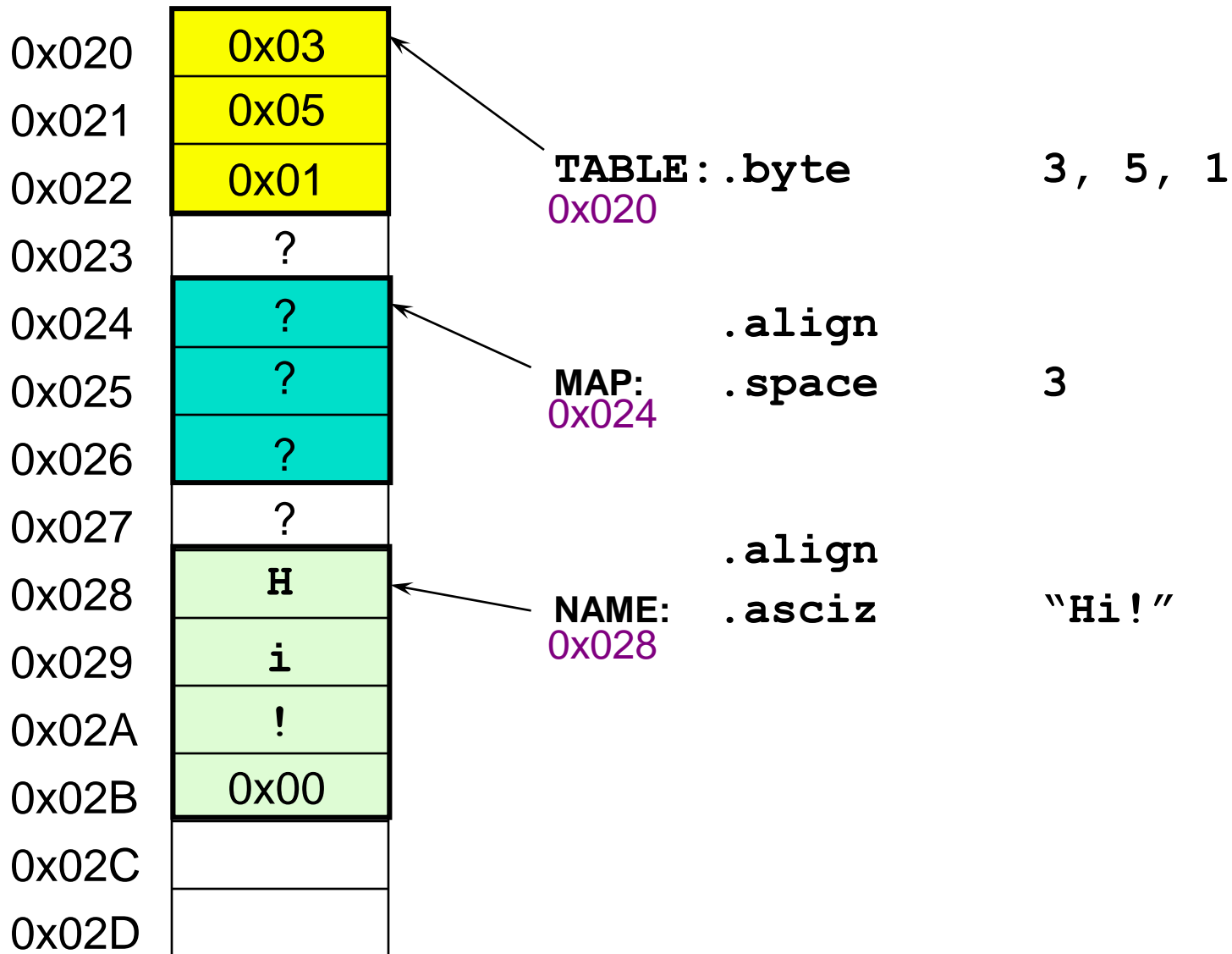
Večkrat želimo, da ima spremenljivka neko začetno vrednost.

```
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 & direktive



Povzetek – prevajanje (psevduukazi, ukazi)

0x020	
0x021	
0x022	
0x023	
0x024	
0x025	
0x026	
0x027	
0x028	
0x029	
0x02A	
0x02B	
0x02C	
0x02D	
0x02E	
0x02F	

```
TABLE: .byte    3, 5, 1, 2

BUF:   .word    0x01020304

A:     .byte    0x15

      .align

_START: mov     r0, #128
```

