

ARM

PROGRAMIRANJE V
ZBIRNEM JEZIKU

2. del

Povzetek – prevajanje (psevduokazi, ukazi)

0x20	
0x21	
0x22	
0x23	
0x24	
0x25	
0x26	
0x27	
0x28	
0x29	
0x2A	
0x2B	
0x2C	
0x2D	
0x2E	
0x2F	

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

BUF:    .word   0x01020304

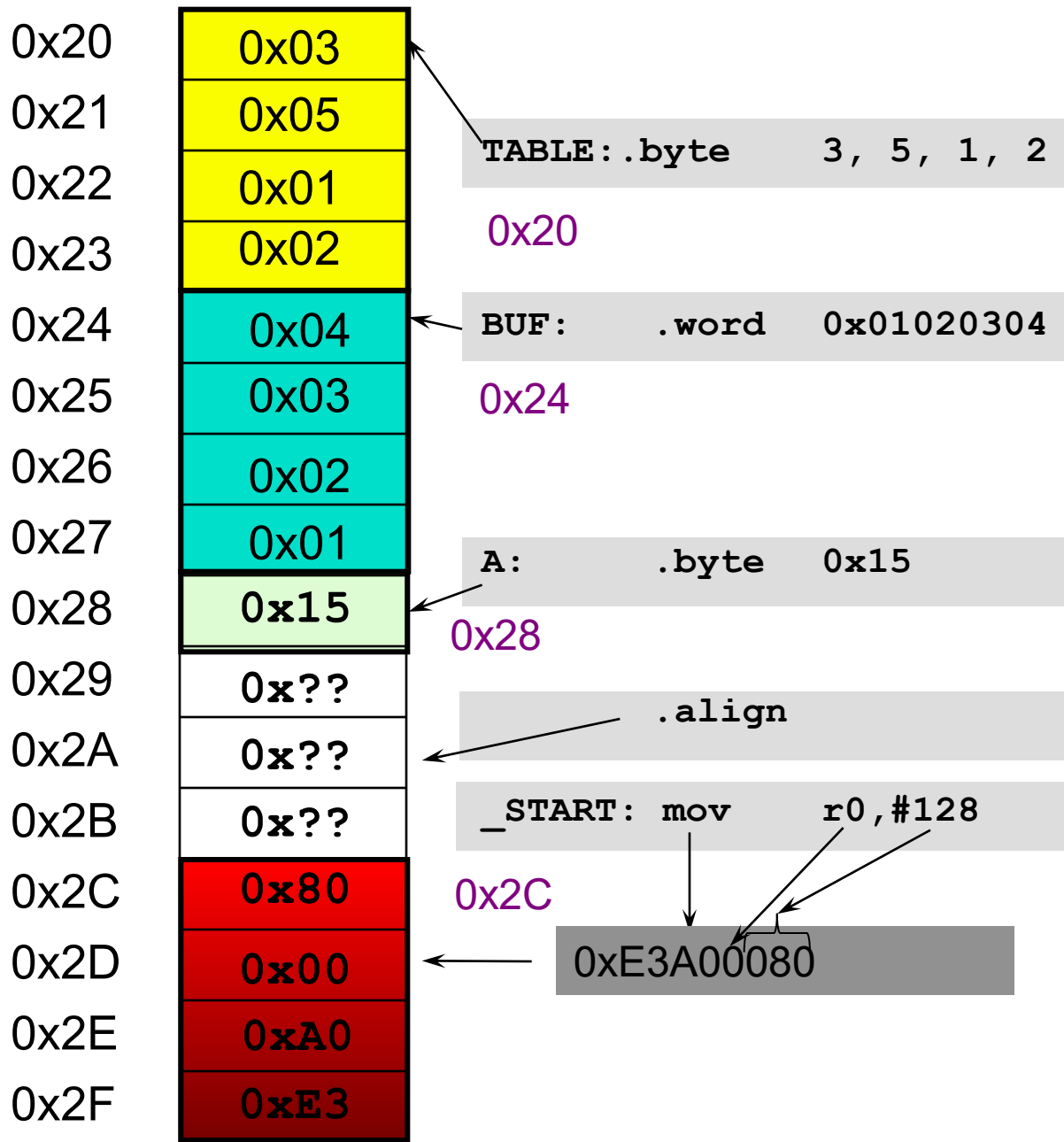
A:      .byte   0x15

        .align

_START: mov     r0, #128
```



Povzetek – prevajanje (psevduokazi, ukazi)



Ukazi load/store – načini naslavljanja

Zbirni jezik

```
adr r0, stev1  
ldr r1, [r0]  
adr r0, stev2  
ldr r2, [r0]  
add r3, r2, r1  
adr r0, rez  
str r3, [r0]
```

1. Posredno registrsko naslavljanje – bazno naslavljanje brez odmika

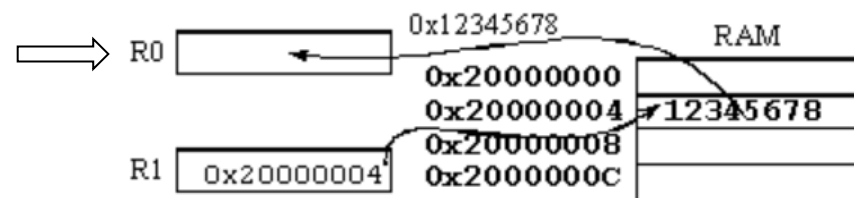
Dostop do operanda v dveh korakih :

a) naslov spremenljivke najprej naložimo v bazni register z:

```
adr r1, stev1
```

b) nato uporabimo ukaz load/store oblike za dostop do operanda:

```
ldr r0, [r1] @ r0 <- mem32[r1]  
str r5, [r0] @ mem32[r0] <- r5
```



Opomba:

adr je nepravilni (psevdo) ukaz. Prevajalnik ga nadomesti z ALE ukazom, ki izračuna naslov spremenljivke s pomočjo R15 (PC) in konstante.

Primer:

```
adr r0, stev1          prevajalnik nadomesti npr. s sub r0, pc, #2c  
                      (ALE ukaz, ki izračuna pravi naslov v r0)
```

Ukazi load/store – načini naslavljanja

Zgledi za bazno naslavljanje brez odmika

32-bitni operandi

```
adr r1, VAR1
ldr r0, [r1]
str r0, [r1]
```

```
@ r1 <- naslov sprem. VAR1
@ r0 <- mem32[r1]
@ mem32[r1] <- r0
```

16-bitni operandi

```
adr r1, VAR2
ldr(s)h r0, [r1]
strh ↑ r0, [r1]
```

```
@ r1 <- naslov sprem. VAR2
@ r0 <- mem16[r1]
@ mem16[r1] <- r0[b0..b15]
```

8-bitni operandi

```
adr r1, VAR3
ldr(s)b r0, [r1]
strb ↑ r0, [r1]
```

```
@ r1 <- naslov sprem. VAR3
@ r0 <- mem8[r1]
@ mem8[r1] <- r0[b0..b7]
```

S .. operand je predznačeno število

Load/store – načini naslavljanja

2. Posredno registrsko naslavljanje – bazno naslavljanje s takojšnjim odmikom (preindex with immediate offset):

```
ldr r0,[r1, #n12]      @ r0<-mem32[r1+n12]
str r0,[r1, #n12]     @ mem32[r1+n12]<-r0
strb r0,[r1, #n12]    @ mem8[r1+n12]<-r0[b0..b7]
```

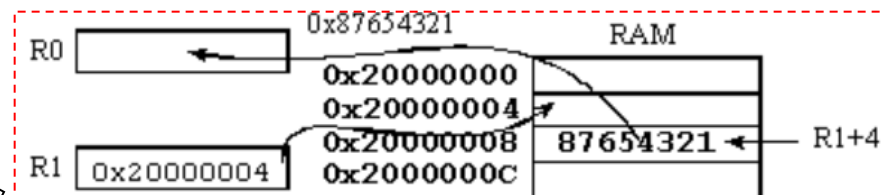
```
ldr(s)b r0,[r1, #n8]   @ r0<-mem8[r1+n8]
ldr(s)h r0,[r1, #n8]  @ r0<-mem16[r1+n8]
strh r0,[r1, #n8]     @ mem16[r1+n8]<-r0[b0..b15]
```

n12 – 12-bitni predznačen odmik

n8 – 8-bitni predznačen odmik

Zgledi:

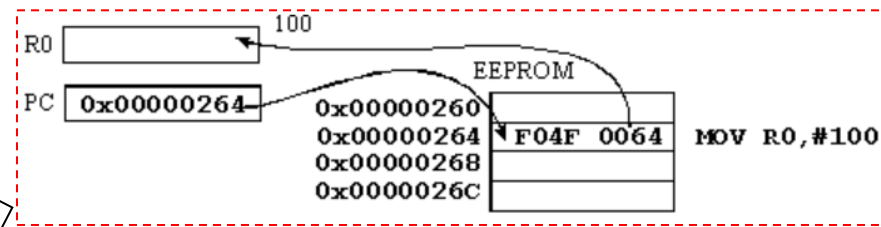
```
ldr r0, [r1, #4]      @ r0 <- mem32[r1 + 4]
ldr r5, [r0, #-20]    @ r5 <- mem32[r0 - 20]
                    @ v r0 mora biti ustrezen naslov!!!
strb r7, [r2,#10]     @ mem8[r2 + 10] <- r7[b0..b7]
                    @ v r2 mora biti ustrezen naslov!!!
```



Naslov operanda je vsota **baznega registra** in **predznačenega odmika**

Aritmetično-logični ukazi

3. Takojšnje naslavljanje



```
mov r0, #100           @ r0 <- 100
add r2, r7, #0x20     @ r2 <- r7 + 32
sub r5, r5, #1        @ r5 <- r5 - 1
```

Takojšnji operand = $(0..255) * 2^{2*(0..12)}$

32-bitni takojšnji operand je 8-bitno število, ki ga lahko še krožno pomaknemo za sodo število mest znotraj 32-bitne vsebine. Takojšnji operand torej ni poljuben!

Tvori ga prevajalnik - če ga ne more, nas opozori.

Takojšnji operand je del ukaza, torej mora biti v času prevajanja iz zbirnega v strojni jezik že znan. Takojšnji operandi so **konstante**.

Aritmetično-logični ukazi

4. Neposredno registrsko naslavljanje

- za računanje z registri in prepisovanje vrednosti iz enega registra v drugega.

```
add r2, r7, r12  
sub r4, r5, r1  
mov r1, r4
```

Aritmetično-logični ukazi, seznam

• Aritmetični ukazi:

```
add r0, r1, r2      @ r0 <- r1 + r2
adc r0, r1, r2      @ r0 <- r1 + r2 + C      (add with C)
sub r0, r1, r2      @ r0 <- r1 - r2
sbc r0, r1, r2      @ r0 <- r1 - r2 + C - 1  (-not(C)=- (1-C)= C-1
rsb r0, r1, r2      @ r0 <- r2 - r1          (reverse subtract)
rsc r0, r1, r2      @ r0 <- r2 - r1 + C - 1  (rev. sub -not(C) )
```

• Logični ukazi:

```
and r0, r1, r2      @ r0 <- r1 AND r2
orr r0, r1, r2      @ r0 <- r1 OR r2
eor r0, r1, r2      @ r0 <- r1 XOR r2
bic r0, r1, r2      @ r0 <- r1 AND NOT r2
```

• Prenos med registri:

```
mov r0, r2          @ r0 <- r2
mvn r0, r2          @ r0 <- NOT r2
```

• Primerjave:

```
cmp r1, r2          @ set CPSR flags on r1 - r2
cmn r1, r2          @ set CPSR flags on r1 + r2
tst r1, r2          @ set CPSR flags on r1 AND r2
teq r1, r2          @ set CPSR flags on r1 XOR r2      (equivalence test)
```