ARM zbirnik Quick Reference (v0.5)

(Pripomoček za izvedbo laboratorijskih vaj pri predmetu Organizacija računalnikov)

Načini naslavljanja:

Bazno naslavljanje: A = r0 + D (odmik)
D .. dolžina krajša od dolžine naslova
Indeksno naslavljanje:
Kadar je odmik D enak dolžini naslova
Lahko ga nadomestimo z D1 = r1 + D in dobimo :
A = r0 + r1 + D = r0 + D1

Povzetek načinov naslavljanja		
Način naslavljanja	Primer	ODM je lahko:
Posredno nasl.	ldr r1, [r0,ODM]	▪ brez » «
Posr. s pred-ind.	ldr r1,[r0,ODM]!	▪ #±odmik »#-4«
Posr. s po-ind.	ldr r0,[r1],ODM	▪ register »r1 «
		▪ reg. s pom. »*r2,LSL #2*«

1.Posredno (bazno) naslavljanje brez odmika

```
adr r0, stev1
ldr r1, [r0]      @ r1 <- mem32[r0]
```

adr ni pravi ukaz: adr r0,stev1 zamenja npr. sub r0,pc,#2c

2.Posredno (bazno) naslavljanje s takojšnj.odmikom

```
ldr r0,[r1, #n12]; r0<-mem32[r1+n12]
strh r0,[r1, #n8]; mem16[r1+n8]<-r0[b0..b15]
```

3.Takojšnje naslavljanje $\text{Takojšnji operand} = (0..255) * 2^{2*(0..12)}$

```
mov r1, #3              @ r1 <- 3
add r2, r7, #0x20       @ r2 <- r7 + 32
```

4.Neposredno registrsko naslavljanje

```
add r2, r7, r12         @ r2 <- r7 + r12
mov r1, r4              @ r1 <- r4
```

5.Posredno naslavljanje z registrskim odmikom

```
ldr r0,[r1,r2] @ r0 <-mem32[r1+r2]
```

6.Posredno naslavljanje s pomaknjenim registrskim odmikom

```
ldr r0,[r1,r2, lsl #2] @ r0 <-mem32[r1+r2*2^2]
```

7.Avtomatsko pred-indeksiranje s takojšnjim odmikom

```
ldr r0,[r1,#4]! @ r1<-r1+4; r0<-mem32[r1]
```

8.Avtomatsko pred-indeksiranje z registrskim odmikom:

```
ldr r0,[r1,r2]! @ r1<-r1+r2; r0<-mem32[r1];
```

9.Avtomatsko pred-indeksiranje s pomaknjenim registrskim odmikom:

```
ldr r0,[r1,r2, lsl #2]! @ r1<-r1+r2*2^2; r0<-mem32[r1];
```

10.Avtomatsko po-indeksiranje s takojšnjim odmikom:

```
ldr r0,[r1],#4  ; r0<-mem32[r1]; r1<-r1+4
```

11.Avtomatsko po-indeksiranje z registrskim odmikom:

```
ldr r0,[r1],r2  ; r0<-mem32[r1]; r1<-r1+r2
```

12.Avtomatsko po-indeksiranje s pomaknjenim registrskim odmikom:

```
ldr r0,[r1],r2,LSL #2  ; r0<-mem32[r1] ; r1<-r1+r2*4
```

Razširitev ničle / razširitev predznaka

Pri nalaganju 8 in 16 – bitnih pomnilniških operandov (load) je potrebno razširiti predznak ali ničlo (ker so registri in ALE operacije 32 bitni).

- pri nepredznačenih operandih se razširi z ničlo:
  ldrh, ldrb
- pri predznačenih operandih se razširi z predznakom:
  ldr**s**h, ldr**s**b

Primerjave nepredznačenih/predznačenih števil

cmp r0,r1		
Nepredznačena štev.	Pogoj	Predznačena štev.
HI	r0 > r1	**GT**
HS	r0 >= r1	**GE**
EQ	r0 = r1	**EQ**
NE	r0 ≠ r1	**NE**
LS	r0 <= r1	**LE**
LO	r0 < r1	**LT**

Logični ukazi (delo z biti):

1. **AND,BIC (brisanje določenih bitov)**
   and r1, r2, r3 @brisanje z ničlami v maski r3
   bic r1, r2, r3 @brisanje z enicami v maski r3
2. **ORR (postavljanje določenih bitov)**
   orr r1, r2, r3 @postavljanje z enicami v maski r3
3. **EOR (invertiranje določenih bitov)**
   eor r1, r2, r3 @invertiranje z enicami v maski r3

Logični ukazi (preverjanje stanja bitov):

1. **TST (preverjanje stanja enega bita, ali več ničelnih bitov) – v bistvu AND s vplivom na zastavice**
   tst r1, r2 @zastavice postavi glede na r1 AND r2
2. **Maskiranje AND in CMP – preverjanje stanja večih bitov:**
   - nezanimive bite damo na 0 (operacija AND z masko)
   - zanimive bite primerjamo z ustreznimi vrednostmi

```
@preveri, da je bit7 v r1 enak 0 in bit 2 v r1 enak 1
and r2, r1, #0x84 @0x84 = ..010000100 => r2=..0?0000?00
cmp r2, #0x04     @0x04 = ..000000100; ustreza, če Z=1
```

ARM zbirnik Quick Reference (v0.5)

(Pripomoček za izvedbo laboratorijskih vaj pri predmetu Organizacija računalnikov)

Pomiki:

ARM ima v podatkovni poti **hitri pomikalnik** za pomikanje vsebine drugega operanda. S tem dobimo vrsto uporabnih operacij.

Možni pomiki drugega operanda:

- **LSL:** logični pomik v levo za 0-31 mest
- **LSR:** log. pomik v desno za 0-31 mest
- **ASL:** enako kot LSL
- **ASR:** aritmetični pomik v desno **(širi se predznak!)**
- **ROR:** rotacija v desno za 0-31 mest
- **RRX:** rotate right extended
  - Pri rotaciji se na najvišje mesto vpiše C bit, v C gre bit 0, ostali biti se pomaknejo v desno za eno mesto.

Load/store multiple:

- pomnilniški naslov za branje/shranjevanje mora biti **poravnan** (deljiv s 4)
- **registri z nižjimi indeksi** se vedno zapišejo na **nižji naslov**
- Če za baznim registrom stoji **!**, bo vrednost baznega registra enaka **naslovu po branju/shranjevanju zadnjega registra**. Sicer se vrednost baznega registra **ne spremeni**.

Pripone:

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

```
stmdb r13!, {r2-r9} @ mem32[r13-4..r13-32] <- r9,…,r2
                     @ r13 <- r13-32
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
stmda 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
```

Sklad:

- **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

		Ascending		Descending	
		Full	Empty	Full	Empty
Increment	Before	**STMIB STMFA**			**LDMIB LDMED**
	After		**STMIA STMEA**	**LDMIA LDMFD**	
Decrement	Before		**LDMDB LDMEA**	**STMDB STMFD**	
	After	**LDMDA LDMFA**			**STMDA STMED**

Uporabljamo FD sklad:

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

Podprogrami:

- **BL : Branch with Link** (L = 1) - shrani povratni naslov v r14.
- **brez sklada:**

```
            bl PODPROG
            ..
PODPROG:    ..
            ..
            mov pc, lr @vračanje iz podprograma
```

- **s skladom:**

```
main:       ldr r13, =0x1000       @ initialize stack pointer
            mov r0, #10            @ put parameter in r0
            bl sub1                @ call subroutine sub1
            …
sub1:       stmfd r13!, {r1-r3,r14} @ save work & link regs
            …              @ inside sub1 we use regs r1,r2,r3
            bl sub2         @ call subroutine sub2
            …
            ldmfd r13!, {r1-r3,pc} @ restore regs & return
```

Delo z registri V/I naprav (prisotni na posebnih naslovih)

```
.equ  RCC_AHB4ENR,0x580244E0 //RCCAHB4 periph. clock reg
.equ  GPIOI_BASE, 0x58022000  // GPIOI base address)
.equ  GPIOx_MODER,0x00        //GPIOx port mode register
```

- **• podan absolutni naslov registra:**

```
ldr r6, = RCC_AHB4ENR //Load periph. clock reg address to r6
ldr r5, [r6]          // Read its content to r5
orr r5, #0x00000100   // Set bit 8 to enable GPIOI clock
str r5, [r6]          // Store result back to I/O register
```

- **podan bazni naslov naprave z registri kot odmiki:**

```
ldr r6, =GPIOI_BASE       // Load GPIOD BASE address to r6
ldr r5, [r6,#GPIOx_MODER]// Read GPIOD_MODER content to r5
and r5, #0xF3FFFFFF       // Clear bits 27-26 for P13
orr r5, #0x04000000       // Write 01 to bits 27-26 for P13
str r5, [r6]              // Store result in GPIO MODER reg.
```