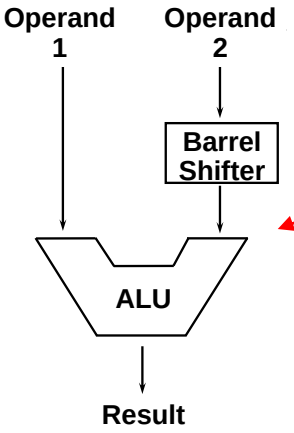
Aritmetično-logični ukazi (pomiki drugega operanda)

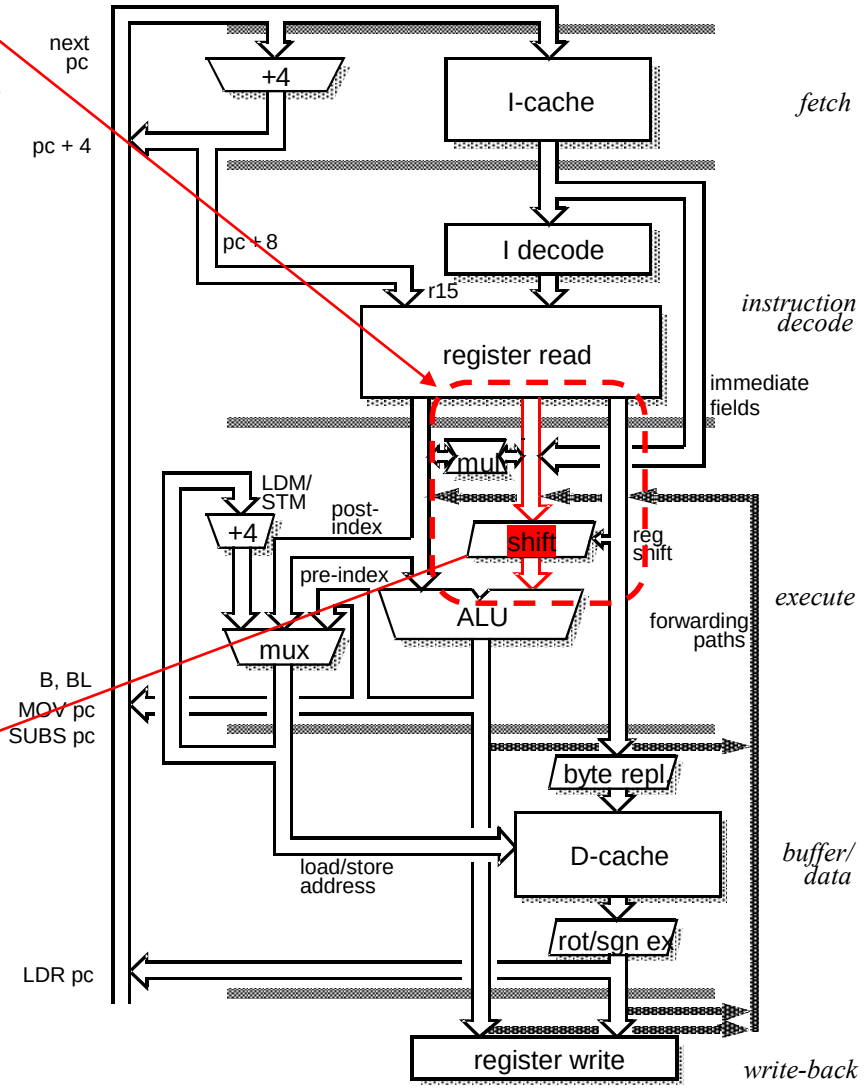
ARM ima v podatkovni poti **hitri pomikalnik:**

- hitro pomikamo vsebino **drugega** operanda
- operacije pomika, množenja/delj. s potenco št. 2

Možni pomiki drugega operanda:

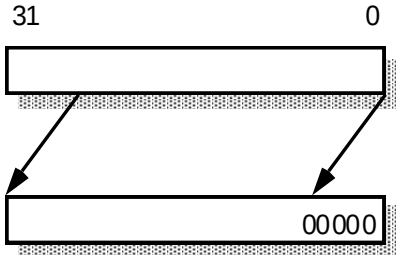
LSL: logični pomik v levo za 0-31 mest
LSR: log. pomik v desno za 0-31 mest
ASL: enako kot LSL
ASR: aritmetični pomik v desno
ROR: rotacija v desno za 0-31 mest
RRX: Rotate Right Extended.





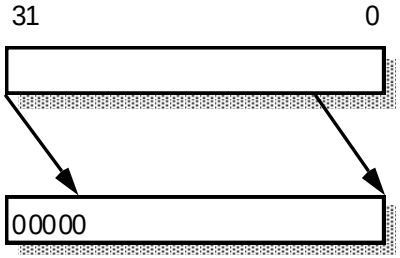
Možni pomiki drugega operanda

LSL/ASL: logični/aritm. pomik v levo za 0-31 mest



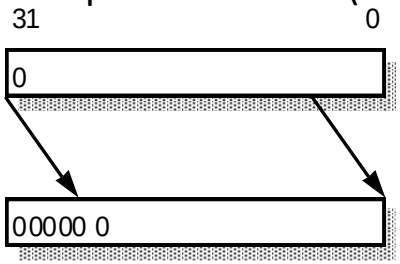
LSL #5

LSR: log. pomik v desno za 0-31 mest

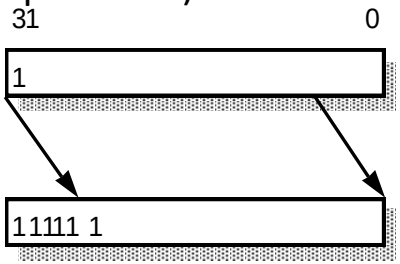


LSR #5

ASR: aritmetični pomik v desno (na levi se širi predznak)

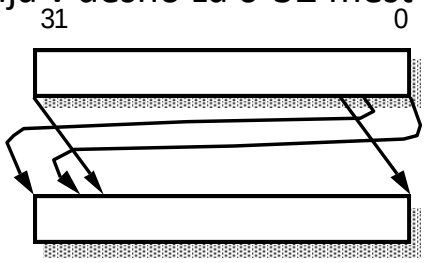


ASR #5, positive operand



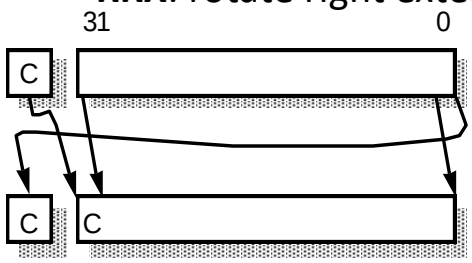
ASR #5, negative operand

ROR: rotacija v desno za 0-31 mest



ROR #5

RRX: rotate right extended

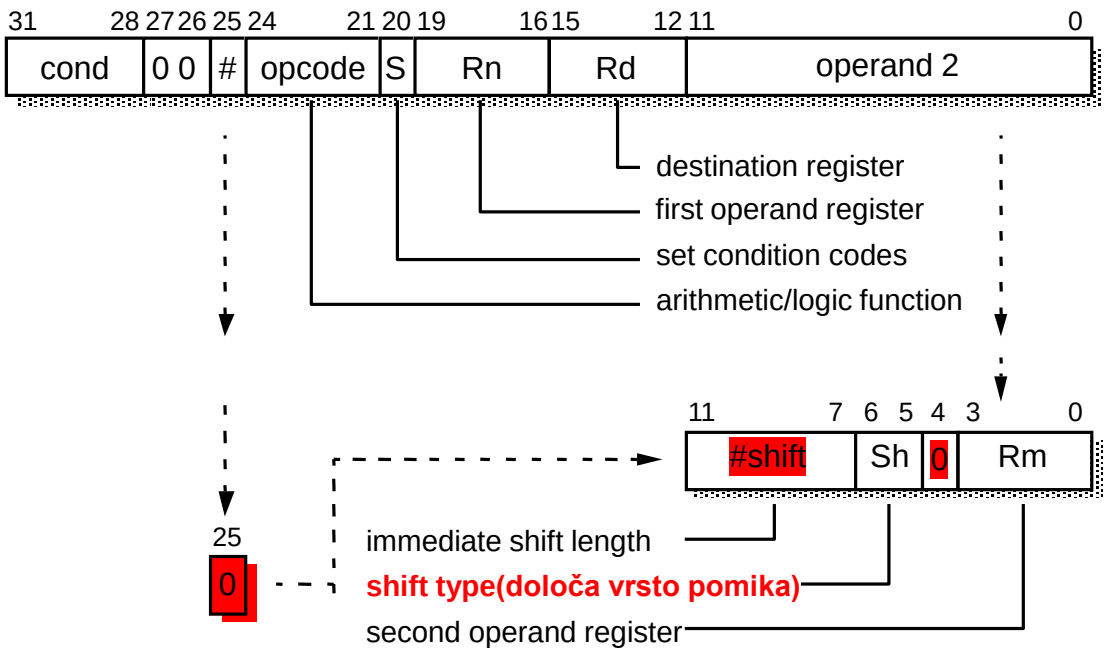


RRX

Pri RRX rotaciji:

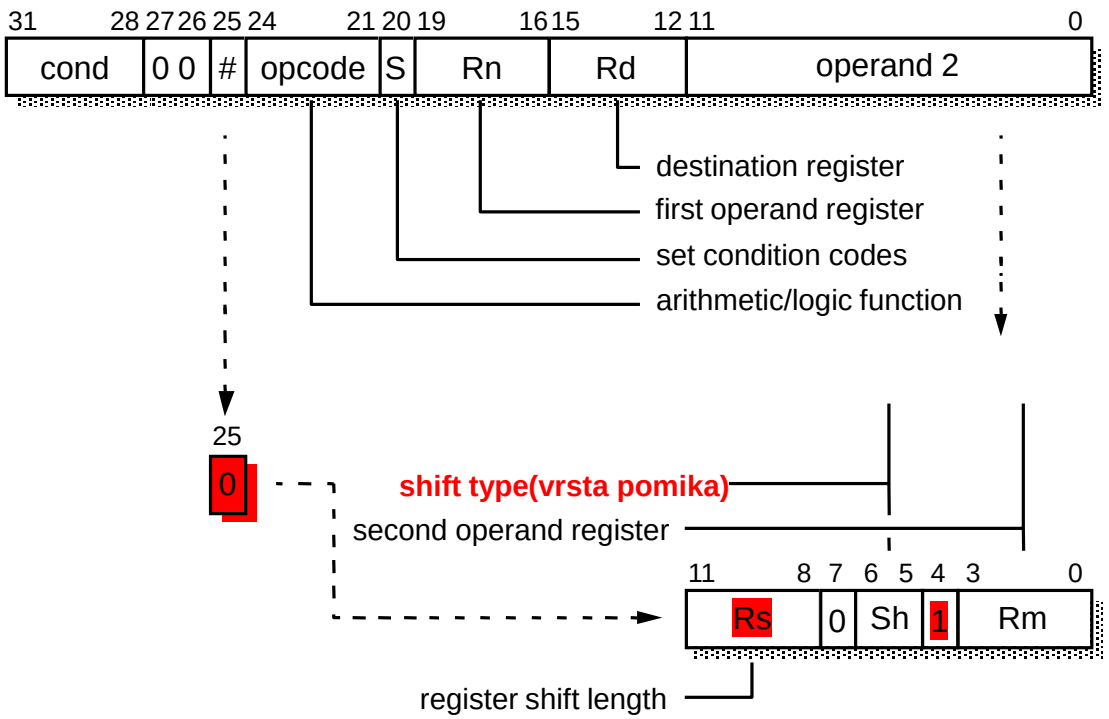
- se na najvišje mesto vpiše C bit,
- v C gre bit 0,
- ostali biti se pomaknejo v desno za eno mesto.

Aritmetično-logični ukazi (pomik določa takojšnji operand)




```
mov r1, r4, LSL #2     ; r1=r4*4
mov r1, r4, LSR #1     ; r1=r4/2 (nepredznačeno)
mov r1, r4, ASR #2     ; r1=r4/4 (predznačeno)
```

Aritmetično-logični ukazi (pomik določa register)




```
add r3, r1, r6, LSL r5 ; r3=r1+r6*(2^r5)
```

Load/store – dodatni načini naslavljanja

Posredno naslavljanje (s konstantnim odmikom) – že prej (S#21 - S#27)

```
ldr r0,[r1,#5]     @ r0 <-mem32[r1+5]
```

5. **Posredno naslavljanje z registrskim odmikom**

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

```
    Zgled:
        adr r1,tabela
        mov r2,#0 @ v r2 je indeks elementa tabele
    zanka:
        ldrb r0,[r1,r2] @ dostopamo do r2-tega elementa
        add r2,r2,#1    @ naslednji element
        cmp r2,#10      @ v tabeli je 10 elementov
        bne zanka
```

6. **Posredno naslavljanje s pomaknjenim registrskim odmikom**

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

```
    Zgled: delo s tabelo 32-bitnih elementov – v prejšnjem
        zgledu se spremeni le ukaz load:

        ldr r0,[r1,r2, lsl #2] @ dostopamo do r2-tega elementa
```

Load/store – načini naslavljanja

Pogosto je potrebno dostopati do pomnilnika in nato pred naslednjim dostopom ***spremeniti naslov v baznem registru****:*

- *npr. preberemo trenutni element iz zabele in se pomaknemo na naslednjega.*

To je pogosto mogoče narediti z enim ukazom – ***avtomatsko indeksiranje****.*

Bazni naslov se lahko spremeni:

- *pred dostopom do pomnilnika (****pred-indeksiranje****) ali*
- *po dostopu do pomnilnika (****po-indeksiranje****).*

Pred-indeksiranje :

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];
```

Load/store – načini naslavljanja

Po-indeksiranje : **Primerno za delo s tabelami**

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
```

Psevdo ukaz – nalaganje 32-bitne konstante

Psevdo ukaz :

```
ldr r0,=vrednost_32b
```

Se realizira z drugim ukazom :

- **„krajša vrednost“ (ustreza pogojem za vrednost tak. operanda) :**

```
mov r0,=127  se realizira z :

mov r0,#127
```

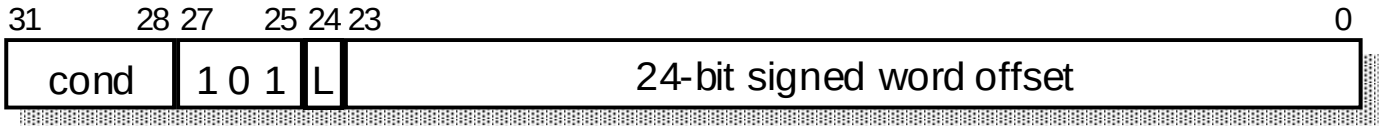
- **„daljša vrednost“ (ne ustreza vrednosti tak. operanda) :**

```
mov r0,=0x12345678   se realizira z :

temp:     .word 0x12345678      (operand v pomnilniku)
          ldr r0,temp           (prenos v r0)

Običajno:ldr r0,temp  se realizira z ldr r0,[pc, +-odmik].
```

Podprogrami



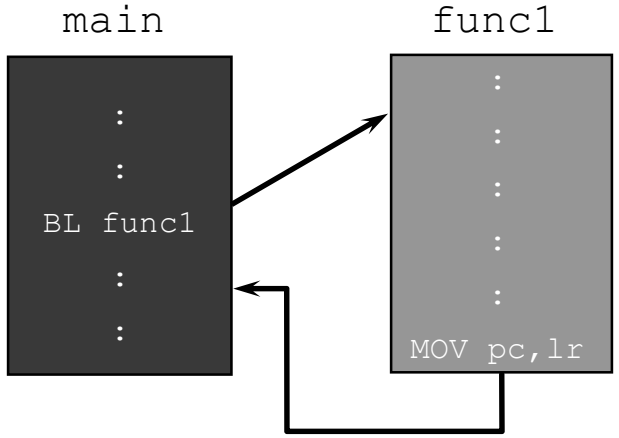
Pri klicanju podprogramov si je potrebno zapomniti povratni naslov.
ARM :

- **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 (L = 1) - shrani povratni naslov v r14.

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



Pogojni klici podprogramov:

- bleq, blhi, … POZOR: zastavice se lahko v podprog. spremenijo

Problem gnezdenja klicev podprogramov – zahteva drugačno rešitev!