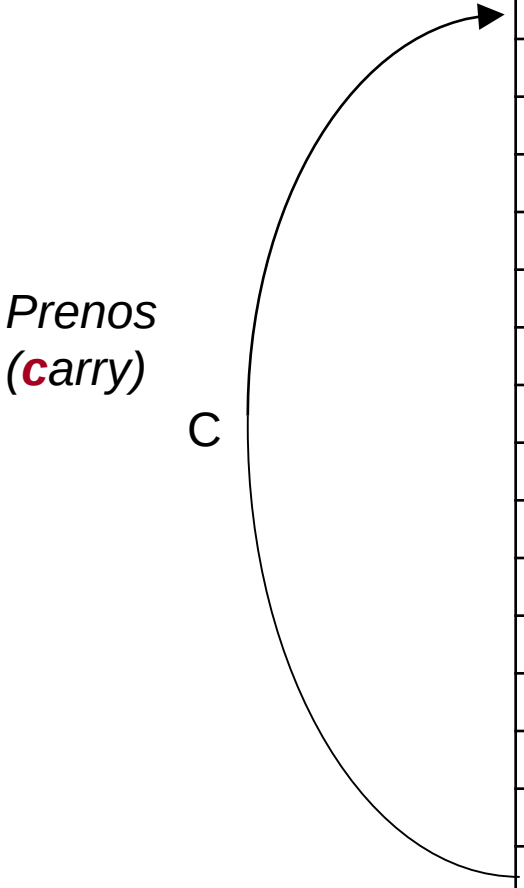
Nepredznačena in predznačena cela števila



Dvojiški zapis	Nepredznačeno	Predznačeno
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Pri odštevanju je stanje C obratno (posebnost ARM)!

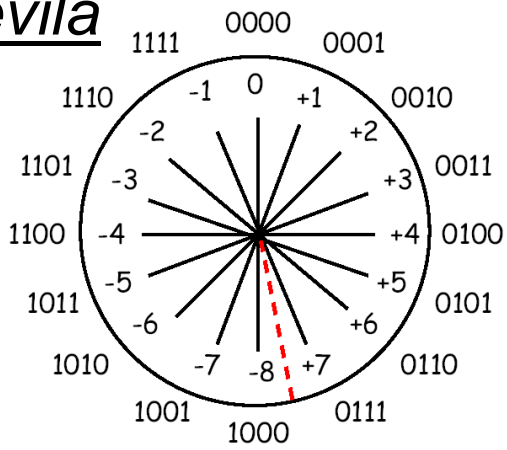
- če ne prekoračimo 0 => C=1

- če prekoračimo 0 => C=0

preliv (overflow)

$$V = A_{n-1} B_{n-1} \overline{S}_{n-1} \vee \overline{A}_{n-1} \overline{B}_{n-1} S_{n-1}$$

Nepredznačena in predznačena cela števila



preliv (overflow)

Dvojiški zapis	Nepredznačeno	Predznačeno
1000	**8**	**-8**
1001	**9**	**-7**
1010	**10**	**-6**
1011	**11**	**-5**
1100	**12**	**-4**
1101	**13**	**-3**
1110	**14**	**-2**
1111	**15**	**-1**
0000	**0**	**0**
0001	**1**	**1**
0010	**2**	**2**
0011	**3**	**3**
0100	**4**	**4**
0101	**5**	**5**
0110	**6**	**6**
0111	**7**	**7**

C ↓

V

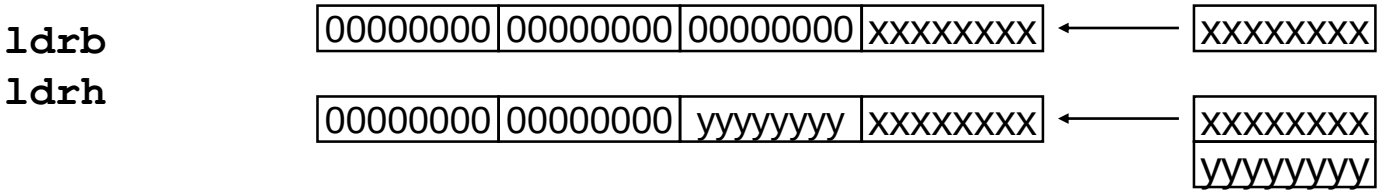
$$V = A_{n-1}B_{n-1}\overline{S}_{n-1} \vee \overline{A}_{n-1}\overline{B}_{n-1}S_{n-1}$$

Vir slike: https://www.doc.ic.ac.uk/~eedwards/compsys/arithmetic/index.html

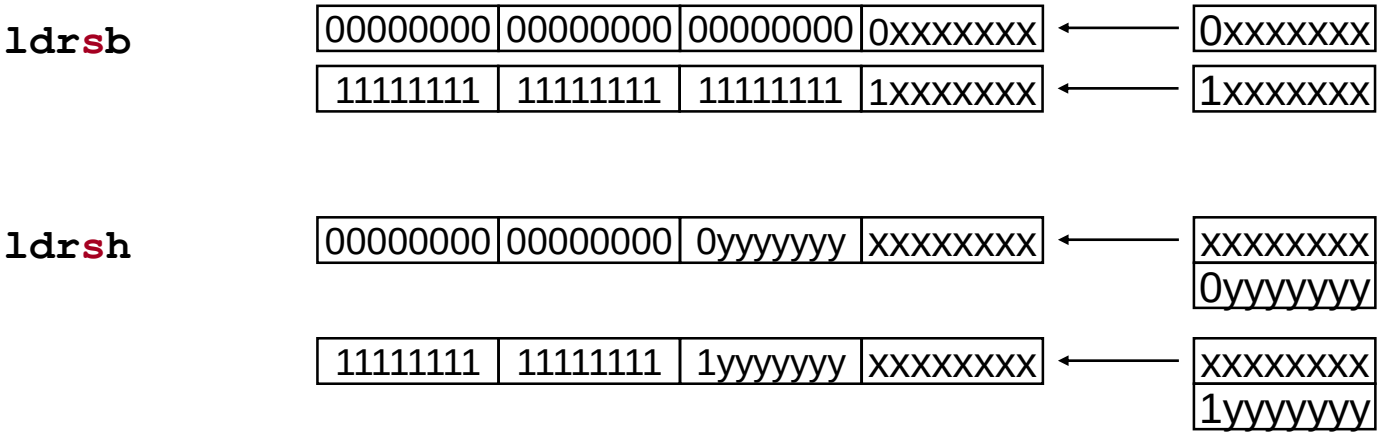
Razširitev ničle / razširitev predznaka

• pri nalaganju 8 in 16 – bitnih pomnilniških operandov je potrebno razširiti predznak ali ničlo (ker so registri in ALE operacije 32 bitni). S pomnilniškimi operandi delata samo load in store (ldr, str).

• pri nepredznačenih operandih je potrebno razširiti ničlo:

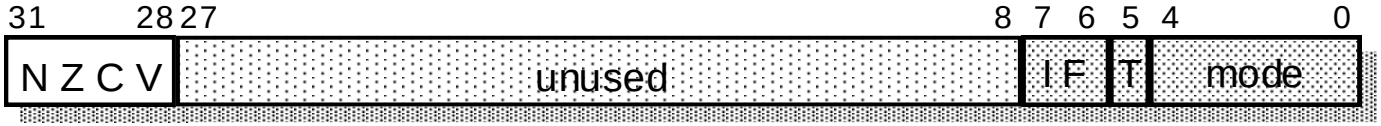


• pri predznačenih operandih je potrebno razširiti predznak:



Zastavice

- so štirje biti v registru CPSR, za vsak bit velja :
- 1 – zastavica je postavljena.
- 0 – zastavica ni postavljena



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: **+**: C = 1: rezultat je povzročil prenos, C = 0: rezultat ni povzročil prenosa (*Carry*)

-: C = **0**: rezultat je povzročil prenos, C = **1**: rezultat ni povzročil prenosa (*Carry*)

V = 1: rezultat je povzročil preliv, V = 0: rezultat ni povzročil preliva (*oVerflow*)

Pri odštevanju je stanje C obratno (posebnost ARM)!

- če ne prekoračimo 0 => C=1

- če prekoračimo 0 => C=0

Če želimo, da ALE ukaz vpliva na zastavice, mu dodamo s:

```
movs r1, #3              @ r1 ← 3
adds r2, r7, #0x20       @ r2 ← r7 + 32
subs r4, r5,#1           @ r4 ← r5 - 1
```

Primerjave

Za spreminjanje zastavic lahko uporabimo ukaze za primerjanje (spadajo med ALE ukaze):

cmp (Compare): postavi zastavice glede na rezultat Rn - Op2

cmp R1, #10 @ R1-10

cmn (Compare negated): postavi zastavice glede na rezultat Rn + Op2

cmn R1, #10 @ R1+10

Ukaza vplivata samo na zastavice, vrednosti registrov **ne spreminjata**. Ker se uporabljata zgolj za spreminjanje zastavic, jima ne dodajamo pripone S.

Primerjave nepredznačenih števil

Zgled: primerjanje dveh nepredznačenih števil:

- **opazujemo zastavici C in Z**

```
mov r1,#11
cmp r1,#10 @ C=1, Z=0

mov r1,#10
cmp r1,#10 @ C=1, Z=1

mov r1,#9
cmp r1,#10 @ C=0, Z=0
```

Torej:

r1 > 10	C=1 in Z=0	Higher
r1 >= 10	C=1	Higher or Same
r1 = 10	Z=1	Equal
r1 < 10	C=0	Lower
r1 <= 10	C=0 ali Z=1	Lower or Same

Primerjave predznačenih števil

Ker gre pri primerjanju za odštevanje/seštevanje, ki je za predznačena števila enako kot za nepredznačena, tudi za primerjanje predznačenih števil uporabimo iste ukaze, opazovati pa moramo druge zastavice!

- Opazovati je potrebno zastavice V, Z in N

Zgled:

```
mov r1,#0
cmp r1,#-1 @ C=0, Z=0, V=0, N=0
```

Zastavice ne ustrezajo pogoju > za nepredznačena števila (C=1 in Z=0)!

Pogoj **> za predznačena** števila je drugačen od pogoja **> za nepredznačena** števila. Pravilen pogoj je: Z = 0 in N = V

Oznake pogojev

Oznaka pogoja	Pomen	Stanje zastavic, ob katerem se ukaz izvede
EQ	Equal / equals zero	Z set
NE	Not equal	Z clear
CS	Carry set	C set
CC	Carry clear	C clear
MI	Minus / negative	N set
PL	Plus / positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HS	Unsigned higher or same	C set
LO	Unsigned lower	C clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set
GE	Signed greater than or equal	N equals V
LT	Signed less than	N is not equal to V
GT	Signed greater than	Z clear and N equals V
LE	Signed less than or equal	Z set or N is not equal to V

Skočni ukazi

Skok je ukaz tipa GOTO oznaka - pri skokih se sklicujemo na oznake. Naslov ukaza, ki stoji za oznako se zapiše v PC.

b (Branch)

```
zanka:      …
            sub r1, r1, #1
            b zanka @ GOTO zanka
```

Zanka se bo ponavljala v nedogled. r1 se bo neprestano zmanjševal, ko bo prišel do 0, bo prišlo do prenosa, v r1 pa bo 0xffffffff.

Če želimo narediti zanko, ki se bo nehala ponavljati, ko bo r1 prišel do 0, potrebujemo ukaz tip **IF pogoj THEN GOTO oznaka**. Ukaz b se bo torej izvedel samo, če bo pogoj ustrezen.

Pogojni skoki

V zbirniku ARM je pogoj vedno določen s stanjem zastavic! Oznake pogojev smo že spoznali.

Preden uporabimo pogojni skok moramo primerno postaviti zastavice. To lahko naredimo z ukazi za primerjavo, zelo pogosto pa kar z enim izmed ostalih ALE ukazov.

Zanka, ki se ustavi, ko r1 pride do 0 bi lahko bila realizirana tako:

b (Branch)

```
zanka:      …
            sub r1, r1, #1
            cmp r1, #0
            bne zanka @ IF Z=0 THEN GOTO zanka
```

Ukazu b smo dodali pripono, ki določa, ob kakšnem stanju zastavic se skok izvede. Če stanje zastavic ni ustrezno, se ukaz ne izvede!

Ker se skok izvede le ob določenem pogoju, mu pravimo **pogojni skok**.

Pogojni skoki

Ukaz cmp v prejšnjem zgledu je torej pripravil zastavico Z, ki je predstavljala pogoj za pogojni skok. Zastavico bi lahko postavili že pri zmanjševanju r1. Ukazu sub je potrebno dodati s:

b (Branch)

```
            mov r1, #10
zanka:      …
            subs r1, r1, #1 @ postavi zastavice!
            bne zanka @ IF Z=0 THEN GOTO zanka
            mov r2, #10
```

Zanka se bo ponovila desetkrat. Ko bo r1 prišel do 0, bo subs zastavico Z postavil na Z=1. Pogojni skok se takrat ne bo izvršil, izvedel se bo ukaz `mov r2, #10` za pogojnim skokom. Pogojni skok torej deluje na način:

IF pogoj THEN PC ← oznaka
ELSE PC ← PC+4

Pogojni skoki

Ukazu b lahko dodamo katerokoli oznako pogoja iz tabele na prosojnici 35. Tako dobimo vse možne pogojne skoke:

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

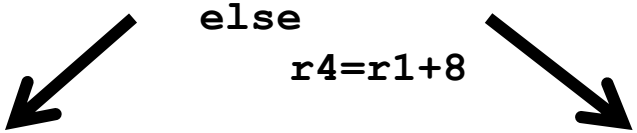
Pogojno izvajanje ukazov

Pogojni skoki so le poseben primer pogojnega izvajanja ukazov. Tudi za druge ukaze je mogoče z dodajanjem ustreznih končnic določiti, da se izvedejo le ob določenem pogoju.

```
        cmp r0, #5
        beq SKOK
        add r1,r1,r0
        sub r1,r1,r2
SKOK    ..
```

```
cmp   r0, #5
addne r1,r1,r0
subne r1,r1,r2
```

```
if (r1<10)
    then
        r4=r1+5
    else
        r4=r1+8
```

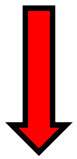


```
        cmp r1, 10
        blo MANJ
        add r4,r1,#8
        b NAPREJ
MANJ    add r4,r1,#5
NAPREJ  ...
```

```
cmp   r1,#10
addlo r4,r1,#5
addhs r4,r1,#8
```

Pogojno izvajanje ukazov

```
if ((r0==r1) AND (r2==r3)) then r4=r4+1
```



```
cmp   r0,r1     ; postavi Z, ce je r0=r1
cmpeq r2,r3     ; primerjaj le, ce je Z=1 in
                ;   spet postavi Z, ce je r2=r3
addeq r4,r4,#1  ; sestej le, ce je Z=1 (r0=r1 in r2=r3)
```

- večino if-then-else stavkov je mogoče implementirati s pogojnim izvajanjem!
- if-then-else stavke, ki vsebujejo AND ali OR lahko implementiramo z uporabo pogojnih primerjanj.
- uporaba pogojnega izvajanja je pogosto (za krajše veje) bolj učinkovita kot uporaba skokov!
- Pozor: izvajanje ukaza vedno traja vsaj 1 periodo, tudi če pogoj ni izpolnjen!