Ukazi za zamenjavo/prenos

- **Ukazi za zamenjavo vrednosti dveh registrov**
  - **XGDX** - Zamenjava (swap) X <=> D
  - **XGDY** - Zamenjava (swap) Y <=> D
- **Prepisovanje vrednosti enega registra v drugega**
  - **TAB** - Prenos A => B (ni zamenjava vrednosti!)
  - **TBA** - B => A
  - **TSX, TSY** - Prenos (SP+1) => X (ali Y)
  - **TXS, TYS** - Prenos [X (ali Y)-1] => SP
  - **TPA** - Prenos CCR (reg. pogojnih kod) => akum. A
  - **TAP** - Prenos akum. A => CCR

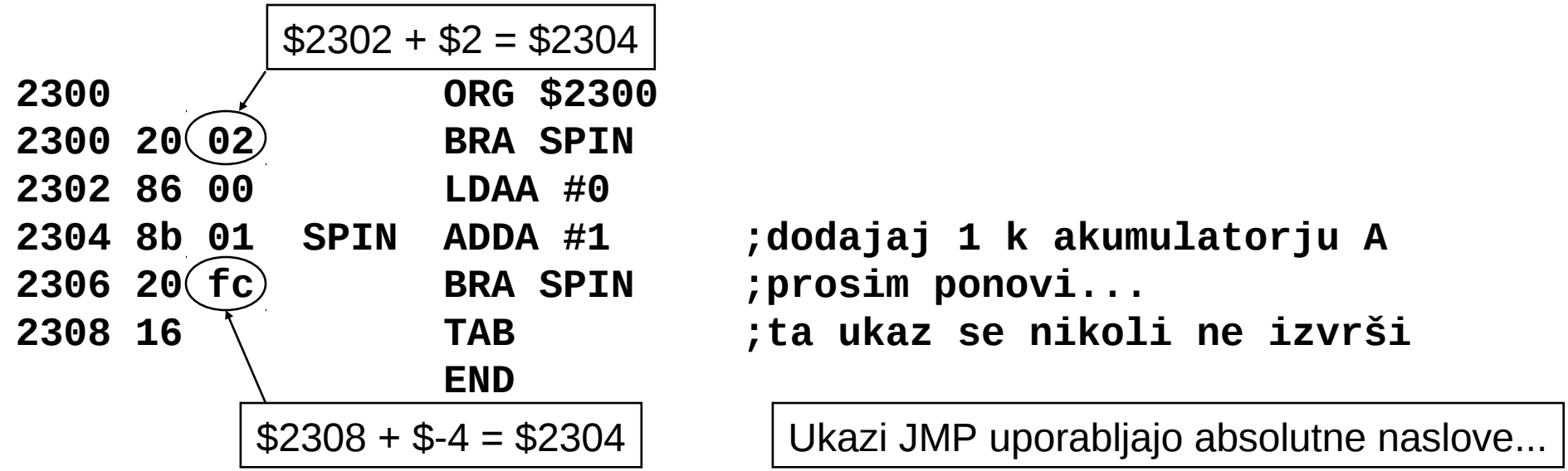
Zanke in Vejitve

- **Zanke naredimo z uporabo ukazov tipa GoTo**
  - Pogojni skoki (samo kratki - BXX)
  - Brezpogojne skoke predstavljata ukaza **BRA** (kratki) in **JMP** (dolgi)

```
        ORG    $2300
        BRA    SPIN   ;preskoči naslednji ukaz
        LDAA   #0     ;škoda, da se ta ukaz ne izvede...
SPIN    ADDA   #1     ;dodajaj 1 k akumulatorju A
        BRA    SPIN   ;prosim ponovi...
        END
```

Naslavljanje pri ukazih Bxx

- **Vejitve uporabljajo relativno naslavljanje**
  - Skočni naslov pri ukazu Bxx je
    - Določen kot odmik glede na NASLEDNJI ukaz. NASLOV=PC+ODMIK
    - Odmik je 8-bitno predznačeno št. v dvojiškem komplementu



Zastavice (Condition Codes)

- **Zastavice postavi večina aritmetičnih ukazov glede na rezultat operacije.**
  - **`ADDA $2020`** ; če pride do prenosa, se postavi zast. C
- **Zastavice so veljavne po končanem ukazu, ki jih je postavil**
  - Veljavne ostanejo, dokler jih ne spremeni kakšen drug ukaz

```
ADDA    $2020
BCC     FOO
```

Postavi zastavice C, Z, N, V, in H

Pogojni skok glede na zastavice, ki jih je postavil ADD (Branch Carry Clear)

- **Zastavice**
  - C - Prenos (za aritmetične ukaze)
  - Z - Zero (postavljen, če je rezultat operacije nič)
  - N - Negative (postavljen, če je rezultat operacije negativen)
  - V - Preliv (za aritmetične ukaze)
  - H - Half Carry prenos od bita 3 do 4

Ukazi Bxx

BCC	Carry Clear	$\overline{C}$
BCS	Carry Set	C
BEQ	Equal	Z
BNE	Not Equal	$\overline{Z}$
BMI	Minus	N
BPL	Plus	$\overline{N}$
BVC	Overflow clear	$\overline{V}$
BVS	Overflow set	V

Ukazi so smiselni, če je zastavice postavil ukaz, ki odšteje $N_1 - N_2$

BHI	Higher	$\overline{C} \bullet \overline{Z}$
BHS	Higher or Same	$\overline{C}$
BLO	Lower	C
BLS	Lower or Same	C + Z

Za aritmetiko z nepredznačenimi števili

BGE	Greater or Equal	$N \bullet V + \overline{N} \bullet \overline{V}$
BGT	Greater	$(N \bullet V + \overline{N} \bullet \overline{V}) \bullet \overline{Z}$
BLE	Less than or Equal	$Z + N \bullet \overline{V} + \overline{N} \bullet V$
BLT	Less than	$N \bullet \overline{V} + \overline{N} \bullet V$

Za aritmetiko v dvojiškem komplementu

Ukaz za primerjanje (Compare)

- **Ukaz compare se uporablja za postavitev zastavic**
  - **`CMPA $2030`**

Postavi zastavice enako, kot če bi se izvedel ukaz A - mem[$2030]

- **CMP pripravi zastavice za pogojni skok, ki sledi**

```
LOOP    ADDA    #1
        CMPA    #32
        BNE     LOOP
```

Ponavljaj, dokler ni izpolnjen pogoj A == 32

Preveri ali Z==0 Branch Not Equal

Različice: **`CMPA, CMPB, CBA, CPD, CPX, CPY`**

Uporaba CMP in Bxx

Izračuna A - <K>

Izvedi ukaz **CMPA <K>. Če mu sledi ukaz** Bxx, to pomeni:

Ukaz	Skoči če:
BEQ	A = K
BNE	A ≠ K
BLO	A < K
BLS	A ≤ K
BHI	A > K
BHS	A ≥ K

Za **nepredznačena števila**.

Ukaz	Skoči če:
BEQ	A = K
BNE	A ≠ K
BLT	A < K
BLE	A ≤ K
BGT	A > K
BGE	A ≥ K

Za **predznačena števila**.

Ukazi Bxx se običajno uporabljajo za ukazom CMP.

Ukaz TST

TSTA ; enako postavi zastavice kot operacija A - 0

TSTA je podoben kot CMPA #0
Vpliva na zastavici N in Z; V in C imata vrednost 0.

Ukaz	Skoči če:
BEQ	A = 0
BNE	A ≠ 0
BLT	A < 0
BLE	A ≤ 0
BGT	A > 0
BGE	A ≥ 0

Pozor: Pri nepredznačenih številih sta smiselna samo BEQ in BNE

Različice: **`TST <opr>, TSTA, TSTB`**

Logični ukazi

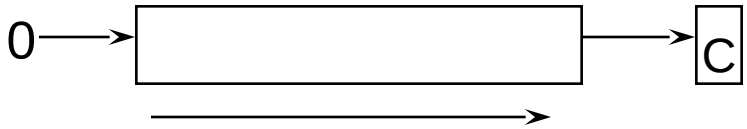
- **AND - logični IN**
- **OR - logični ALI**
- **EOR - ekskluzivni ALI**
- **COM - eniški komplement**
- **NEG – dvojiški komplement**

Vsi logični ukazi delujejo nad biti (bitwise). Izjema je NEG.

```
    10100101
OR  00110011
    10110111

    10100101
AND 00110011
    00100001

    10100101
EOR 00110011
    10010110

COM 10100101
    01011010

NEG 10100101
    01011011
```

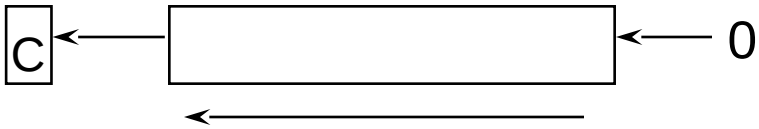
Ukazi za pomikanje

Logični pomik v DESNO



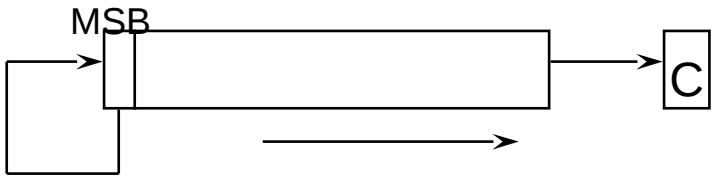
00001111 ⇨ 00000111
LSR
C = 1

Logični pomik v LEVO



00001111 ⇨ 00011110
LSL
C = 0

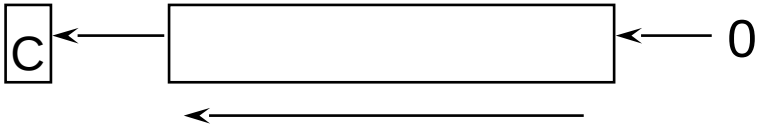
Aritmetični pomik v DESNO



00001111 ⇨ 00000111
ASR C=1

10100011 ⇨ 11010001
ASR C=1

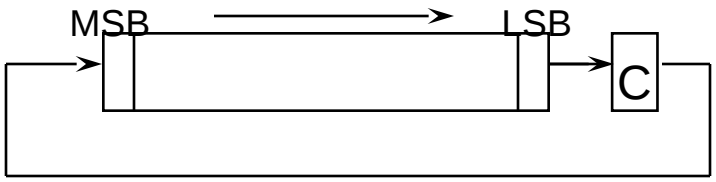
Aritmetični pomik v LEVO



00001111 ⇨ 00011110
ASL C=0

Ukazi za kroženje (rotate)

Zakroži v DESNO



C=0

00001111 ⇨ 00000111 C=1

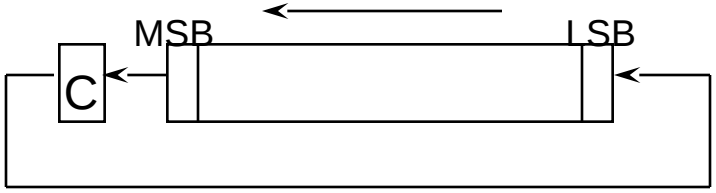
ROR

C=1

10100010 ⇨ 11010001 C=0

ROR

Zakroži v LEVO



C=0

00001111 ⇨ 00011110 C=0

ROL

C=1

10100010 ⇨ 01000101 C=1

ROL

Delo z biti

- **Ukaz deluje nad posameznimi biti**
  - **`BSET <opr>, <mask>`** ; postavi <mask> bite
    - **`BSET $20,#5`** ; postavi bita 0 in 2
    - Če je bila vrednost pom[$20] 00101001, bo po ukazu 00101101.
  - **`BCLR <opr>, <mask>`** ; briše <mask> bite
    - **`BCLR 0,X,#$C0`** ; briši bita 6 in 7
- **Opomba:**
  - Bite štejemo od nič dalje (ne od ena)!
  - Razširjeno neposredno naslavljanje pomnilnika ni mogoče! Za naslove nad 255 ($FF) morate obvezno uporabiti bazno (indeksno) naslavljanje!

Ukaz za preverjanje bitov

- **Ukaza BITA, BITB postavita zastavice**
  - **`BITA  $2030`**

Postavi zastavice enako, kot če bi se izvedel ukaz A • pom[$2030]

- **BITA, BITB pripravi zastavice za pogojni skok, ki sledi**

```
LOOP    LDAA    PIA_DRA
        BITA    #%00000001
        BEQ     LOOP
```

Ponavljaj, dokler je bit z najnižjo težo v akumulatorju A enak 0

Preveri ali Z==1 Branch Equal

Ostali ukazi

- **BRCLR, BRSET - Skoči, če so biti določeni z masko vsi 0 ali vsi 1.**
  - `BRSET  $20 #$81 SKIP ;skoči,če sta bita 0 in 7`
    `;na naslovu[$20] postavljena`
  - `BRCLR  5,X #$04 SKIP`
    - **Podpira samo pomnilniško neposredno (nerazširjeno) in indeksno naslavljanje**
- **SEC, CLC, SEV, CLV, SEI, CLI – postavljanje in brisanje posameznih zastavic**
- **SWI - Programska prekinitev (Software Interrupt)**
  - Vrnitev v IDT (debugger)...

Množenje

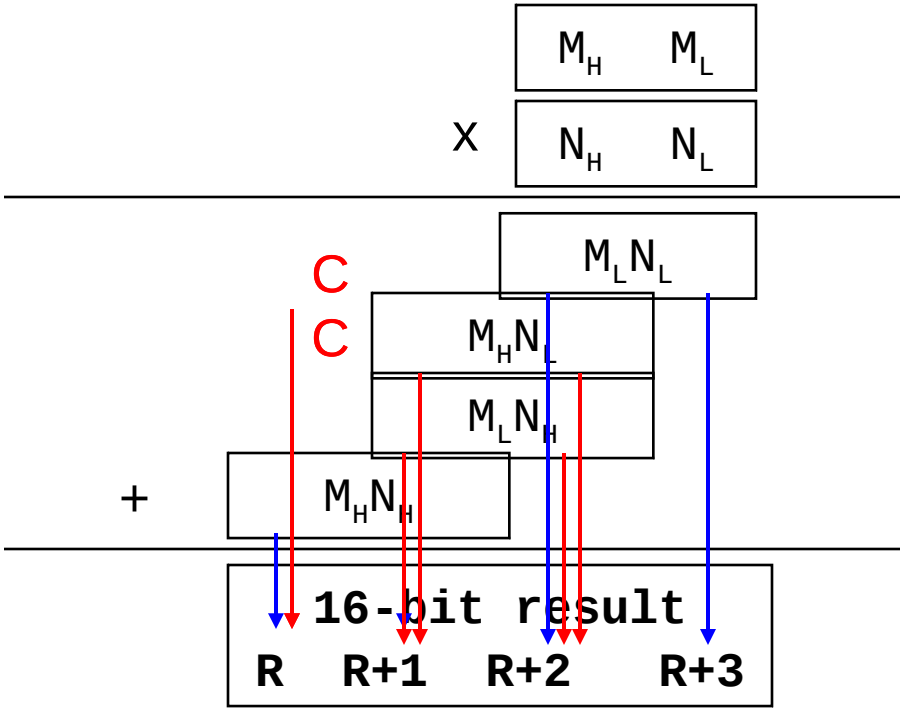
Ukaz MUL pomnoži ak. A z ak. B (8 bitov X 8 bitov). Rezultat (16 biten) je v registru D.

```
        ORG     $2000
RESULT  RMB     2
START   LDAA    #$15
        LDAB    #$32
        MUL
        STD     RESULT
        END

        ORG     $FFFE           ;RESET Vektor
        FDB     START
```

Pozor: Vrednost akumulatorjev A in B se spremeni!

Množenje 16-bitnih števil

Zmnoži dve 16-bitni števili, M_HM_L and N_HN_L



1. Zmnoži M_L x N_L --> D
2. Shrani D v R+2, R+3
3. Zmnoži M_H x N_H --> D
4. Shrani D v R, R+1
5. Zmnoži M_H x N_L --> D
6. Prištej D k R+1, R+2
7. Prištej ***prenos*** k R
8. Zmnoži M_L x N_H --> D
9. Prištej D k R+1, R+2
10. Prištej ***prenos*** k R

Deljenje

Rezultat celoštevilčnega deljenja je količnik in ostanek.

```
 00 09  = 2, ostanek 1
-----
 00 04
```

Ukaz **IDIV** deli 16-bitni ak. D s 16-bitnim indeksnim registrom X

Po operaciji je količnik v X, ostanek v D.

```
        ORG     $2300
START   LDD     #$0009
        LDX     #$0004
        IDIV            ; deli 9/4
        STX     $2000   ; količnik -> $2000
        STD     $2002   ; ostanek -> $2002

        ORG     $FFFE
        FDB     START
```

Odprite pomnilniško okno ($2000) in po korakih izvedite program.

Ulomki (Fractional Division)

Rezultat je količnik v obliki realnega števila.

```
 00 03      = .375(10)
-------     = .011(2)
 00 08      = 0110 0000 0000 0000
              (16 bitov, dec. pika na začetku)
```

Ukaz **FDIV** deli 16-bitni akum. D s 16-bitnim indeksnim reg. X

--> *Predpostavljamo, da je števec < imenovalec!*

Količnik je v IX, ostanek je v D

```
ORG    $2300
LDD    #$0003
LDX    #$0008
FDIV             ; deli 3/8
STX    $2000     ; količnik -> $2000
STD    $2002     ; ostanek -> $2002
```

Izračuna se tudi ostanek. Uporabno predvsem, da vemo ali se je deljenje izšlo.

Čas izvajanja zanke

Vsak cikel izvajanja ukaza HC11 ustreza enemu ciklu ure E.

Frekvenca ure E je enaka eni četrtini frekvence kristala, priključenega na vhoda XTAL in EXTAL

Običajno ima ura E frekvenco 1-4 MHz. V našem primeru je frekvenca E enaka 1.2288 MHz.

```
                        cikli
      LDX   #6144       3
LOOP  NOP               2
      NOP               2      izvede se
      DEX               3      6144 krat
      BNE   LOOP        3
```

Čas = [3 + (6144 * 10)] * 1/1.2288MHz

= [61443 / 1.2288e6] s

= 0.05000244 s

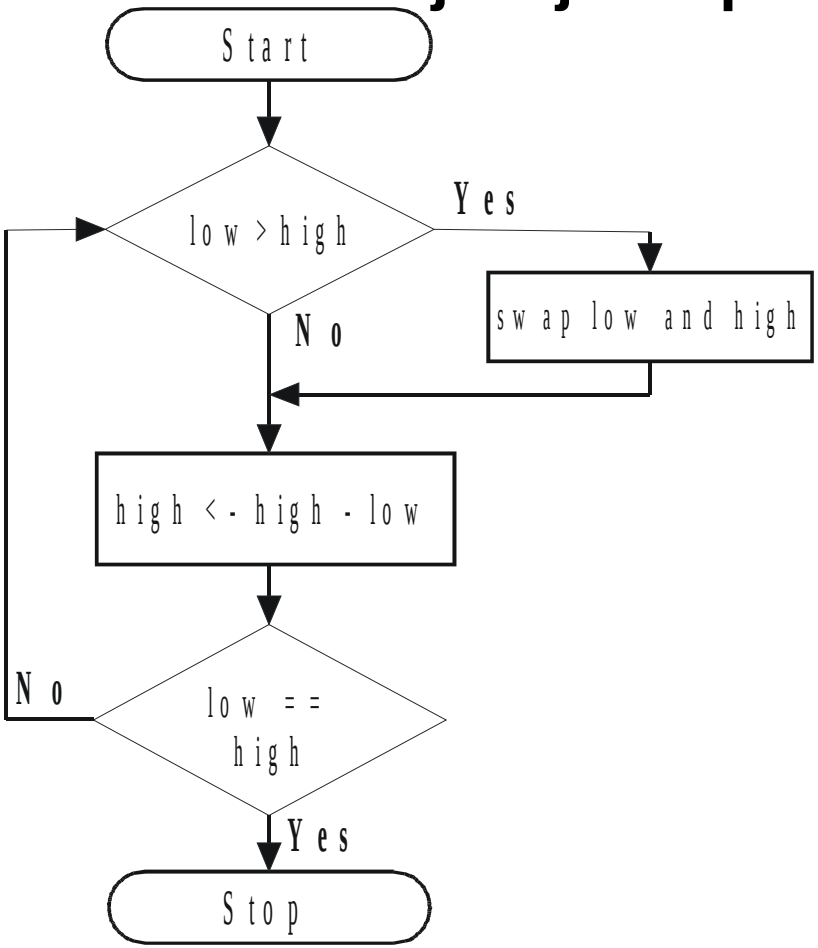
≈ 50 ms

Natančna zakasnitev 50ms

```
      LDX   #6144       3
      NOP               2      10
      NOP               2
      DEX               3
LOOP  NOP               2
      NOP               2      6143
      DEX               3      * 10
      BNE   LOOP        3
```

Evklidov algoritem

- **Poišče največji skupni delitelj (GCD).**



Psevdokoda:

```
Euclid
 while high is not equal to low
    If low > high
       swap high and low
    end if
    high <- high - low
  end while
end Euclid
```