5. Celočíselné instrukce - pokračování.

Logické instrukce (operace provádějí po bitech !)

- AND
- OR
- XOR
- NOT
- TEST

AND, OR, XOR

AND dest,src (And)

24 i8	AND AL, imm8		
25 i8	AND AX, imm16	(AND EAX, imm32)	
80 /4 i8	AND r/m8,imm8		
81 /4 i8	AND r/m16,imm16	(AND r/m32,imm32)	
83 /4 i8	AND r/m16,imm8	(AND r/m32,imm8)	*
20 /r	AND r/m8,r8		
21 /r	AND r/m16,r16	(AND r/m32,r32)	
22 /r	AND r8,r/m8		
23 /r	AND r16,r/m16	(AND r32,r/m32)	

dest AND src → dest

dest AND (sign-extend of src) → dest *

Příznaky OF a CF jsou vynulovány, příznaky SF, ZF a PF jsou nastaveny podle výsledku, hodnota příznaku AF není definována.

OR dest,src (Or)

0C i8	OR AL, imm8		
0D i8	OR AX, imm16	(OR EAX, imm32)	
80 /1 i8	OR r/m8,imm8		
81 /1 i8	OR r/m16,imm16	(OR r/m32,imm32)	
83 /1 i8	OR r/m16,imm8	(OR r/m32,imm8)	*
08 /r	OR r/m8,r8		
09 /r	OR r/m16,r16	(OR r/m32,r32)	
0A /r	OR r8,r/m8		
0B /r	OR r16,r/m16	(OR r32,r/m32)	

dest OR src → dest

dest OR (sign-extend of src) → dest *

Příznaky OF a CF jsou vynulovány, příznaky SF, ZF a PF jsou nastaveny podle výsledku, hodnota příznaku AF není definována.

XOR dest,src (Exclusive Or)

34 i8	XOR AL, imm8		
35 i8	XOR AX, imm16	(XOR EAX, imm32)	
80 /6 i8	XOR r/m8,imm8		
81 /6 i8	XOR r/m16,imm16	(XOR r/m32,imm32)	
83 /6 i8	XOR r/m16,imm8	(XOR r/m32,imm8)	*
30 /r	XOR r/m8,r8		
31 /r	XOR r/m16,r16	(XOR r/m32,r32)	
32 /r	XOR r8,r/m8		
33 /r	XOR r16,r/m16	(XOR r32,r/m32)	

dest XOR src → dest

dest XOR (sign-extend of src) → dest *

Příznaky OF a CF jsou vynulovány, příznaky SF, ZF a PF jsou nastaveny podle výsledku, hodnota příznaku AF není definována.

Příklady použití:

and ax,00ffh	původní hodnota AX:	0101010101010011
	00ffh:	0000000011111111
	nová hodnota AX:	0000000001010011
or bl,cl	původní hodnota BL:	01010101
	původní hodnota CL:	00111000
	nová hodnota BL:	01111101
xor [alpha],cl	původní hodnota [alpha]:	11100101
	původní hodnota CL:	00110011
	nová hodnota alpha:	11010110

AND dest,src	(And)
OR dest,src	(Or)
XOR dest,src	(Exclusive Or)

Stručný přehled jednotlivých možností:

1. AND/OR/XOR reg-dest,reg-src
2. AND/OR/XOR reg,mem
3. AND/OR/XOR mem,reg
4. AND/OR/XOR reg,imm
5. AND/OR/XOR mem,imm

Oba operandy musí mít stejnou velikost (8, 16, nebo 32 bitů).

NOT

NOT dest (One's Complement Negation)

F6 /2 NOT r/m8
F7 /2 NOT r/m16 (NOT r/m32)

NOT dest → dest

Nemění příznaky.

Příklad použití:

```
not al
```

původní hodnota AL: 01001101
nová hodnota AL: 10110010

TEST

TEST dest,src (Logical Compare)

A8 i8	TEST AL, imm8	
A9 i8	TEST AX, imm16	(TEST EAX, imm32)
F6 /0 i8	TEST r/m8,imm8	
F7 /0 i8	TEST r/m16,imm16	(TEST r/m32,imm32)
84 /r	TEST r/m8,r8	
85 /r	TEST r/m16,r16	(TEST r/m32,r32)

dest AND src → temp

Příznaky OF a CF jsou vynulovány, příznaky SF, ZF a PF jsou nastaveny podle výsledku, hodnota příznaku AF není definována.
Hodnoty operandů se nemění!

Pozn.: Instrukce TEST pracuje podobně, jako instrukce AND. Cílový operand však nemění a nastavuje pouze příznaky.

TEST dest,src (Logical Compare)

Stručný přehled jednotlivých možností:

1. TEST reg-dest,reg-src
2. TEST reg,mem
3. TEST mem,reg
4. TEST reg,imm
5. TEST mem,imm

Oba operandy musí mít stejnou velikost (8, 16, nebo 32 bitů).

Instrukce pro předávání řízení

- **JMP**
- **Jcc**
- **JCXZ**/JECXZ
- **LOOP**
- LOOPcc
- **CALL**
- **RET**
- INT
- INT3
- INTO
- IRET
- BOUND
- ENTER
- LEAVE

JMP

JMP – nepodmíněný skok

Nepodmíněné skoky se mohou být:

- skoky uvnitř jednoho kódového segmentu:
    - krátké (short)
    - blízké (near)
        - přímé (direct)
        - nepřímé (indirect)

- skoky do jiných kódových segmentů:
    - vzdálené (far)
        - přímé (direct)
        - nepřímé (indirect)

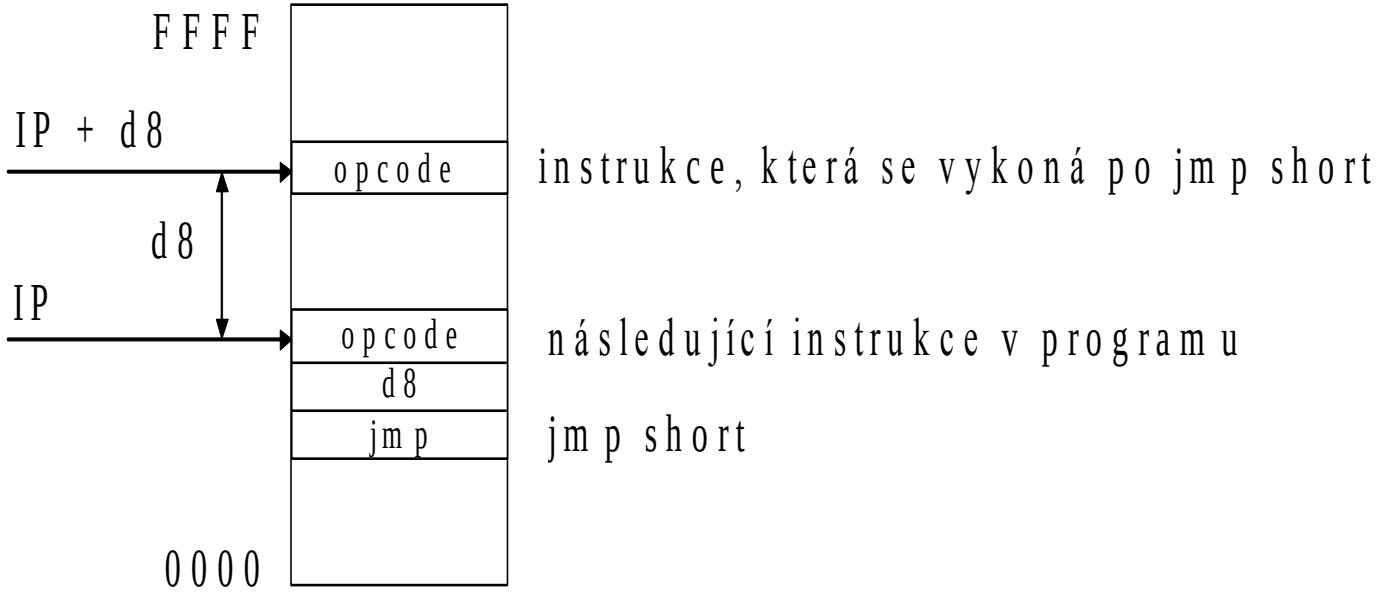
Nepodmíněné skoky nemění příznaky !!! (kromě vzdálených skoků způsobujících přepnutí úloh v chráněném režimu).

JMP dest (Jump)

EB d8 JMP rel8 jump short, relative

Krátký relativní skok uvnitř kódového segmentu v rozsahu <–128, +127> slabik od aktuálního obsahu registru (E)IP:

(E)IP + d8 → (E)IP.



Příklad použití krátkého skoku:

```
            …
            jmp  short  tam
zde:        mov  ax,bx
            add  ax,20
            …
tam:        push  ax
            …
```

Pozn.: Přičítání relativního posunutí k obsahu IP (IP obsahuje adresu operačního kódu instrukce bezprostředně následující za instrukcí jmp short) se provádí jako standardní sečítání znaménkových čísel v doplňkovém kódu - relativní posunutí d8 se znaménkově rozšíří na 16 bitů:

Původní IP	001A	001A	001A	FFEE
Posunutí d8	77	FE	82	7F
Rozšířené posunutí	0077	FFFE	FF82	007F
Nový IP	0091	0018	FF9C	006D

JMP dest (Jump)

E9 d16	JMP rel16	jump near, relative
E9 d32	JMP rel32	jump near, relative

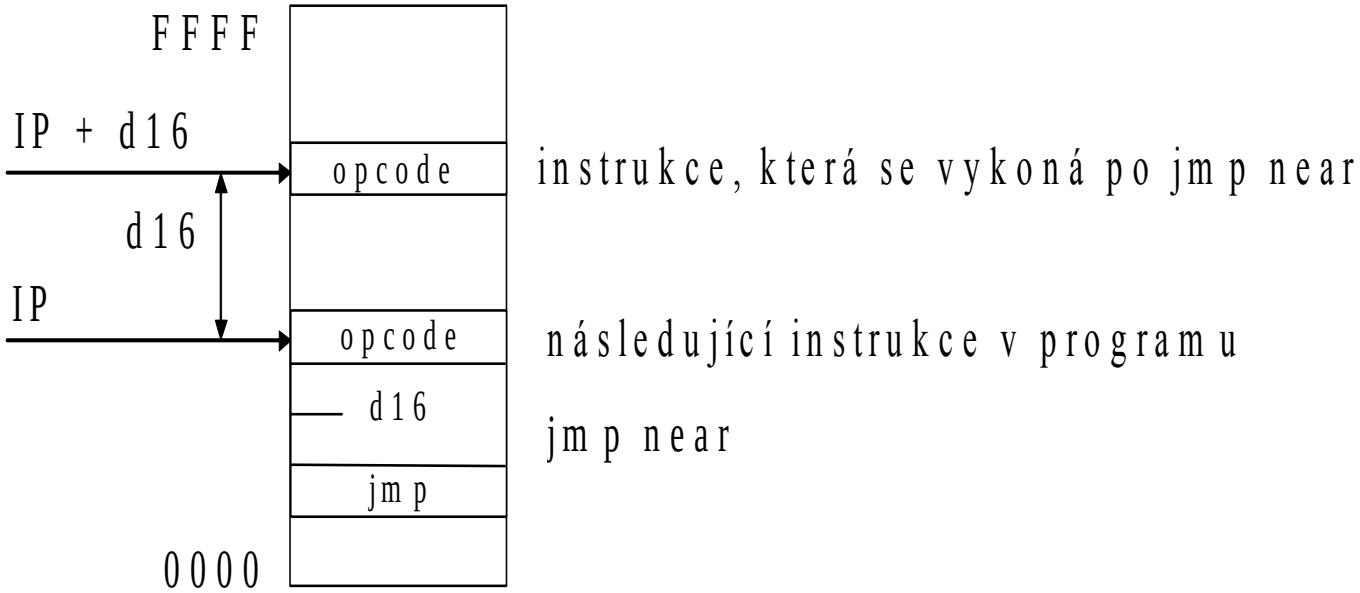
Přímý blízký (relativní) skok uvnitř kódového segmentu v rozsahu <–32768, +32767> slabik od aktuálního obsahu registru (E)IP :

(E)IP + d16(32) → (E)IP,

Pozn.: přímý relativní skok near je podobný skoku short, pouze relativní posunutí je šestnáctibitové (je uloženo na dvou slabikách za operačním kódem) a skočit se tak dá kamkoliv v rámci segmentu:

Původní IP	002B	002B	(7FFF je maximální kladné číslo)
Posunutí d16	7FFF	8000	(8000 je minimální záporné číslo)
Nový IP	802A	802B	(adresa 802B bezprostředně následuje za adresou 802A)

Blízký skok přímý:



Příklady použití blízkého přímého skoku:

```
zpet:   …
        …
                jmp  near  zpet     ; blízký skok je implicitní, a proto
        …
                jmp  zpet           ; slovo near se nemusí uvádět
```

JMP dest (Jump)

FF /4	JMP r/m16	jump near, absolute indirect
FF /4	JMP r/m32	jump near, absolute indirect

Nepřímý blízký (absolutní) skok uvnitř kódového segmentu:
offset z r/m16 (r/m32) nahradí původní obsah registru (E)IP.

Příklady použití blízkého nepřímého skoku (všechny uvedené skoky směřují na návěští *tam*):

```
        jmp  near  [tam_offset]     ; přes paměť - offset je v paměti
        jmp  [tam_offset]           ; totéž
        mov  bx,tam_offset          ; přes paměť adresovanou registrem
        jmp  [bx]                   ; jinak totéž
        mov  bx,tam                 ; přes registr - offset návěští „tam“ se uloží do
        jmp  bx                     ; registru
        mov  bx,[tam_offset]        ; totéž – do registru se uloží obsah
        jmp  bx                     ; slova paměti s adresou „tam_offset“, což
        …                           ; je offset návěští „tam“
tam:    …
        …
        …
tam_offset dw  tam
```

JMP dest (Jump)

EA d16:d16	JMP ptr16:16	jump far, absolute
EA d16:d32	JMP ptr16:32	jump far, absolute

Přímý vzdálený (absolutní skok) - ukazatel uložený v instrukci nahradí původní obsah CS:(E)IP.

Příklad použití vzdáleného přímého skoku na návěští *tam* (následující tři symbolické instrukce jsou zcela ekvivalentní) :

```
jmp  far  tam
jmp  (seg tam): tam
jmp  seg tam: tam
```

JMP dest (Jump)

FF /5	JMP m16:16	jump far, absolute indirect
FF /5	JMP m16:32	jump far, absolute indirect

Nepřímý vzdálený (absolutní) skok - ukazatel z m16:16 (m16:32) nahradí původní obsah CS:(E)IP.

Příklad použití vzdáleného nepřímého skoku:

```
                mov  bx,tam_offset      ; adresa cílového místa (ukazatel) je
                jmp  far  [bx]          ; v paměti adresované registrem bx
                jmp  far  [tam_offset]  ; adresa cílového místa (ukazatel) je
                …                       ; v paměti adresované posunutím d16
                …
                …
tam_offset      dw  tam,seg tam
```

JMP dest (Jump)

Stručný přehled jednotlivých možností:

1. JMP short dest
2. JMP (near) dest
3. JMP (near) mem16
4. JMP (near) reg16
5. JMP far dest32
6. JMP far mem32

Příklady ze souboru jmp.asm (skoky short a near):

```
1.      jmp short jedna
2.      jmp dva
        jmp near tri
3.      jmp [o_ctyri]
        jmp near [o_pet]
        mov bx,o_sest
        jmp [bx]
        mov bx,o_sedm
        jmp near [bx]
4.      mov bx,osm
        jmp bx
        mov bx,[o_devet]
        jmp bx
        …

jedna:  …
        …
dva:    …
        …
tri:    …
        …
ctyri:  …
        …
pet:    …
        …
sest:   …
        …
sedm:   …
        …
osm:    …
        …
devět   …
        …

o_ctyri dw ctyri        ; definice offsetu symbolu/návěští ctyri,
o_pet   dw pet          ; pet,
o_sest  dw sest         ; sest,
o_sedm  dw sedm         ; sedm,
o_osm   dw osm          ; osm,
o_devet dw devět        ; a devět.
```

Příklady ze souboru jmp.asm (skoky far):

```
5.                  jmp far pism_a
6.                  jmp far [o_p_b]
                    mov bx,o_p_c,
                    jmp far [bx]
                    …
                    …
          pism_a    …
                    …
          pism_b    …
                    …

          o_p_b     dw pism_b, seg pism_b     ; definice offsetu a segmentu
                                              ; symbolu pism_b
          o_p_c     dw pism_c, seg pism_c     ; definice offsetu a segmentu
                                              ; symbolu pism_c
```

Jcc

Jcc dest (Jump Short if Condition is Met)

Jcc - podmíněný krátký nebo přímý blízký skok, který se uskuteční pouze v případě splnění podmínky naznačené písmeny *cc* za písmenem *J*.

Podmíněné skoky testují, ale nemění příznaky!!!

Pozn.: Skoky podmíněné blízké lze realizovat podmíněným krátkým skokem s opačnou podmínkou a nepodmíněným blízkým skokem:

```
        jz  tam:          ≡          jnz  sem
        …                            jmp  tam
        …                            …
        …                  sem:      …
tam:    …                            …
                           tam:      …
```

Skoky na základě příznaků:

70 d8	JO rel8	jump short if overflow (OF=1)
71 d8	JNO rel8	jump short if not overflow (OF=0)
72 d8	JC rel8	jump short if carry (CF=1)
73 d8	JNC rel8	jump short if not carry (CF=0)
74 d8	JZ rel8	jump short if zero (ZF=1)
75 d8	JNZ rel8	jump short if not zero (ZF=0)
78 d8	JS rel8	jump short if sign (SF=1)
79 d8	JNS rel8	jump short if not sign (SF=0)
7A d8	JP rel8	jump short if parity (PF=1)
7A d8	JPE rel8	jump short if parity even (PF=1)
7B d8	JNP rel8	jump short if not parity (PF=0)
7B d8	JPO rel8	jump short if parity odd (PF=0)

Skoky po porovnání bezznaménkových čísel (CMP dest,src), zkratka se vztahuje k operandu *dest*:

77 d8	JA rel8	jump short if above (CF=0 and ZF=0)
73 d8	JAE rel8	jump short if above or equal (CF=0)
72 d8	JB rel8	jump short if below (CF=1)
76 d8	JBE rel8	jump short if below or equal (CF=1 or ZF=1)
74 d8	JE rel8	jump short if equal (ZF=1)
75 d8	JNE rel8	jump short if not equal (ZF=0)
76 d8	JNA rel8	jump short if not above (CF=1 or ZF=1)
72 d8	JNAE rel8	jump short if not above or equal (CF=1)
73 d8	JNB rel8	jump short if not below (CF=0)
77 d8	JNBE rel8	jump short if not below or equal (CF=0 and ZF=0)

Skoky po porovnání znaménkových čísel (CMP dest,src),
zkratka se vztahuje k operandu *dest*:

7F d8	JG rel8	jump short if greater (ZF=0 and SF=OF)
7D d8	JGE rel8	jump short if greater or equal (SF=OF)
7C d8	JL rel8	jump short if less (SF<>OF)
7E d8	JLE rel8	jump short if less or equal (ZF=1 or SF<>OF)
74 d8	JE rel8	jump short if equal (ZF=1)
75 d8	JNE rel8	jump short if not equal (ZF=0)
7E d8	JNG rel8	jump short if not greater (ZF=1 or SF<>OF)
7C d8	JNGE rel8	jump short if not greater or equal (SF<>OF)
7D d8	JNL rel8	jump short if not less (SF=OF)
7F d8	JNLE rel8	jump short if not less or equal (ZF=0 and SF=OF)

Jcc dest (Jump Near if Condition is Met, 386)

(řazeno abecedně, w … word = d16, d … doubleword = d32)

0F 87 w/d	JA rel16/32	jump near if above (CF=0 and ZF=0)
0F 83 w/d	JAE rel16/32	jump near if above or equal (CF=0)
0F 82 w/d	JB rel16/32	jump near if below (CF=1)
0F 86 w/d	JBE rel16/32	jump near if below or equal (CF=1 or ZF=1)
0F 82 w/d	JC rel16/32	jump near if carry (CF=1)
0F 84 w/d	JE rel16/32	jump near if equal (ZF=1)
0F 8F w/d	JG rel16/32	jump near if greater (ZF=0 and SF=OF)
0F 8D w/d	JGE rel16/32	jump near if greater or equal (SF=OF)
0F 8C w/d	JL rel16/32	jump near if less (SF<>OF)
0F 8E w/d	JLE rel16/32	jump near if less or equal (ZF=1 or SF<>OF)
0F 86 w/d	JNA rel16/32	jump near if not above (CF=1 or ZF=1)
0F 82 w/d	JNAE rel16/32	jump near if not above or equal (CF=1)
0F 83 w/d	JNB rel16/32	jump near if not below (CF=0)
0F 87 w/d	JNBE rel16/32	jump near if not below or equal (CF=0 and ZF=0)

0F 83 w/d	JNC rel16/32	jump near if not carry (CF=0)
0F 85 w/d	JNE rel16/32	jump near if not equal (ZF=0)
0F 8E w/d	JNG rel16/32	jump near if not greater (ZF=1 or SF<>OF)
0F 8C w/d	JNGE rel16/32	jump near if not greater or equal (SF<>OF)
0F 8D w/d	JNL rel16/32	jump near if not less (SF=OF)
0F 8F w/d	JNLE rel16/32	jump near if not less or equal (ZF=0 and SF=OF)
0F 81 w/d	JNO rel16/32	jump near if not overflow (OF=0)
0F 8B w/d	JNP rel16/32	jump near if not parity (PF=0)
0F 89 w/d	JNS rel16/32	jump near if not sign (SF=0)
0F 85 w/d	JNZ rel16/32	jump near if not zero (ZF=0)
0F 80 w/d	JO rel16/32	jump near if overflow (OF=1)
0F 8A w/d	JP rel16/32	jump near if parity (PF=1)
0F 8A w/d	JPE rel16/32	jump near if parity even (PF=1)
0F 8B w/d	JPO rel16/32	jump near if parity odd (PF=0)
0F 88 w/d	JS rel16/32	jump near if sign (SF=1)
0F 84 w/d	JZ rel16/32	jump near if zero (ZF=1)

Příklady použití podmíněných skoků

a) Skoky závislé na nastavení příznaků:

```
        add ax,bx           ; sečítání bezznaménkových čísel
        jc error            ; chybný výsledek indikuje příznak CF
        …
        add cl,[alpha]      ; sečítání znaménkových čísel
        jo error            ; chybný výsledek indikuje příznak OF
        …
        sub cl,[alpha]      ; odečítání znaménkových čísel
        jo error            ; chybný výsledek indikuje opět příznak OF
        js minus            ; skok v případě záporného výsledku
        jz nula             ; skok v případě nulového výsledku
        …                   ; zde pokračuje v případě kladného výsledku,
nula:   …                   ; zde v případě nulového výsledku
        …
minus:  …                   ; a zde v případě záporného výsledku
        …
error:  …                   ; ošetření chyb (např. výpis chybové zprávy)
        …
```

Příklady použití podmíněných skoků

b) Skoky po porovnání bezznaménkových čísel:

```
        cmp bx,dx       ; porovnání bezznaménkových čísel
        jb mensi        ; skok, je-li první číslo menší než druhé
        ja vetsi        ; skok, je-li první číslo větší než druhé
        …               ; zde se pokračuje, jsou-li obě čísla stejná
        …
        jmp dale
mensi:  …               ; zde se pokračuje, je-li první číslo menší
        …
        jmp dale
vetsi:  …               ; zde se pokračuje, je-li první číslo větší
        …
dale:   …
        …
```

Příklady použití podmíněných skoků

b) Skoky po porovnání znaménkových čísel:

```
        cmp bx,dx       ; porovnání bezznaménkových čísel
        jl mensi        ; skok, je-li první číslo menší než druhé
        jg vetsi        ; skok, je-li první číslo větší než druhé
        …               ; zde se pokračuje, jsou-li obě čísla stejná
        …
        jmp dale
mensi:  …               ; zde se pokračuje, je-li první číslo menší
        …
        jmp dale
vetsi:  …               ; zde se pokračuje, je-li první číslo větší
        …
dale:   …
        …
```

Jcc (Jump if Condition is Met)

Pravidla syntaxe:

- Instrukce vždy začíná písmem J
- Druhým znakem instrukce může být písmeno N, pak jde o negaci podmínky dané následujícím znakem, resp. znaky

Jcc (Jump if Condition is Met)

- Testují-li se aktuální příznaky v registru Flags je dalším znakem instrukce písmeno:
  - O testuje-li se příznak OF
  - C testuje-li se příznak CF
  - Z testuje-li se příznak ZF
  - S testuje-li se příznak SF
  - P testuje-li se příznak PF

  Pozn.: Dvě instrukce pro testování příznaků představují výjimku z výše uvedených pravidel: JPE ≡ JP, JPO ≡ JNP

Jcc (Jump if Condition is Met)

- Testuje-li se výsledek porovnání dvou bezznaménkových čísel (po instrukci CMP), pak musí následovat písmeno nebo dvojice písmen:
  - A skok se provede, bylo-li první číslo větší než druhé (dest > src)
  - E skok se provede, byla-li obě čísla stejná
  - B skok se provede, bylo-li první číslo menší než druhé (dest < src)
  - AE skok se provede pro (dest ≥ src)
  - BE skok se provede pro (dest ≤ src)

Jcc (Jump if Condition is Met)

- Testuje-li se výsledek porovnání dvou znaménkových čísel (po instrukci CMP), pak musí následovat písmeno nebo dvojice písmen:
  - G skok se provede, bylo-li první číslo větší než druhé (dest > src)
  - E skok se provede, byla-li obě čísla stejná
  - L skok se provede, bylo-li první číslo menší než druhé (dest < src)
  - GE skok se provede pro (dest ≥ src)
  - LE skok se provede pro (dest ≤ src)

JCXZ, JECXZ, **LOOP,** LOOPcc

JCXZ/JECXZ dest (Jump Short if Condition is Met)

E3 d8	JCXZ rel8	jump short if CX=0
E3 d8	JECXZ rel8	jump short if ECX=0

Příznaky nemění.

LOOP dest (Loop According to (E)CX Counter)

E2 d8 LOOP rel8 loop

Instrukce LOOP dekrementuje čítač (CX v 16-bitovém režimu, ECX ve 32-bitovém režimu) a pokud je nová hodnota čítače nenulová provede skok. Příznaky nemění.

Typické je použití pro počítaný cyklus (počet opakování je dán bezznaménkovým číslem v CX registru (0…65535)):

```
            …
            jcxz cyklus_end
cyklus:     …                   ; tělo cyklu
            …                   ; nesmí měnit obsah registru CX !!!
            loop cyklus
cyklus_end:
            …
```

LOOPcc dest (Loop According to (E)CX Counter)

E1 b	LOOPE rel8	loop if equal
E1 b	LOOPZ rel8	loop if zero
E0 b	LOOPNE rel8	loop if not equal
E0 b	LOOPNZ rel8	loop if not zero

Instrukce LOOPE/LOOPZ dekrementuje čítač a provede skok, je-li splněna podmínka: (nová hodnota čítače # 0) and (ZF = 1).
Instrukce LOOPNE/LOOPNZ dekrementuje čítač a provede skok, je-li splněna podmínka: (nová hodnota čítače # 0) and (ZF = 0).
Příznaky nemění.

CALL, RET

Call – skok do procedury

Skoky do procedur mohou být:

- Skoky do procedur definovaných ve stejném kódovém segmentu:
    - blízké
        - přímé
        - nepřímé

- Skoky do procedur definovaných v jiných kódových segmentech:
    - vzdálené
        - přímé
        - nepřímé

Skoky do procedur nemění příznaky !!! (kromě vzdálených skoků způsobujících přepnutí úloh v chráněném režimu).

CALL dest		(Call Procedure)
E8 d16	CALL rel16	call near, relative
E8 d32	CALL rel32	call near, relative
FF /2	CALL r/m16	call near, absolute indirect
FF /2	CALL r/m32	call near, absolute indirect

Call near – uloží do zásobníku (E)IP (ten ukazuje na instrukci bezprostředně následující po instrukci CALL) a provede:

- (E)IP + d16(32) → (E)IP
  jde-li o přímý blízký (relativní) skok, nebo
- offset z r/m16 (r/m32) → (E)IP
  jde-li o nepřímý (absolutní) skok.

Příklady použití (možných zápisů) jsou podobné jako u blízkých nepodmíněných skoků.

CALL dest (Call Procedure)

9A d16:d16	CALL ptr16:16	call far, absolute
9A d16:d32	CALL ptr16:32	call far, absolute
FF /3	CALL m16:16	call far, absolute indirect
FF /3	CALL m16:32	call far, absolute indirect

Call far – uloží do zásobníku CS:(E)IP a provede

- d16:d16(32) → CS:(E)IP,
  jde-li o přímý (absolutní) vzdálený skok, nebo
- ukazatel z m16:16 (m16:32) → CS:(E)IP,
  jde-li o nepřímý (absolutní) vzdálený skok.

Příklady použití (možných zápisů) jsou podobné jako u vzdálených nepodmíněných skoků.

CALL dest (Call Procedure)

Stručný přehled jednotlivých možností:

1. CALL (near) dest
2. CALL (near) mem16
3. CALL (near) reg16
4. CALL far dest
5. CALL far mem32

Příklady ze souboru proc.asm (volání procedury near):

```
1.      call mypnear
        call near mypnear
2.      call [mypn_o]
        call near [mypn_o]
        mov bx,mypn_o
        call [bx]
        mov bx,mypn_o
        call near [bx]
3.      mov bx,mypnear
        call bx
        mov bx,[mypn_o]
        call bx
        …
        …
mypn_o  dw mypnear          ; definice offsetu návěští mypnear
```

```
                …
mypnear         …
                …
                …
                ret
                …
```

Příklady ze souboru proc.asm (volání procedury far):

```
4.              call far mypfar
5.              call far [mypf_os]
                mov bx,mypf_os
                call far [bx]
                …
                …
mypf_os     dw mypfar,seg mypfar        ; definice ofsetu a segmentu návěští
```

Jiný segment:

```
                …
mypfar:         …
                …
                …
                retf
                …
```

RET (Return from Near Procedure)

C3	RET	near return
C2 i16	RET imm16	near return and pop imm16 bytes

Near return: POP IP

POP EIP

POP IP, SP + imm16 → SP

POP EIP, ESP + imm16 → ESP

Příznaky nemění.

RETF (Return from Far Procedure)

CB	RETF	far return
CA i16	RETF imm16	far return and pop imm16 bytes

Far return: POP IP, POP CS

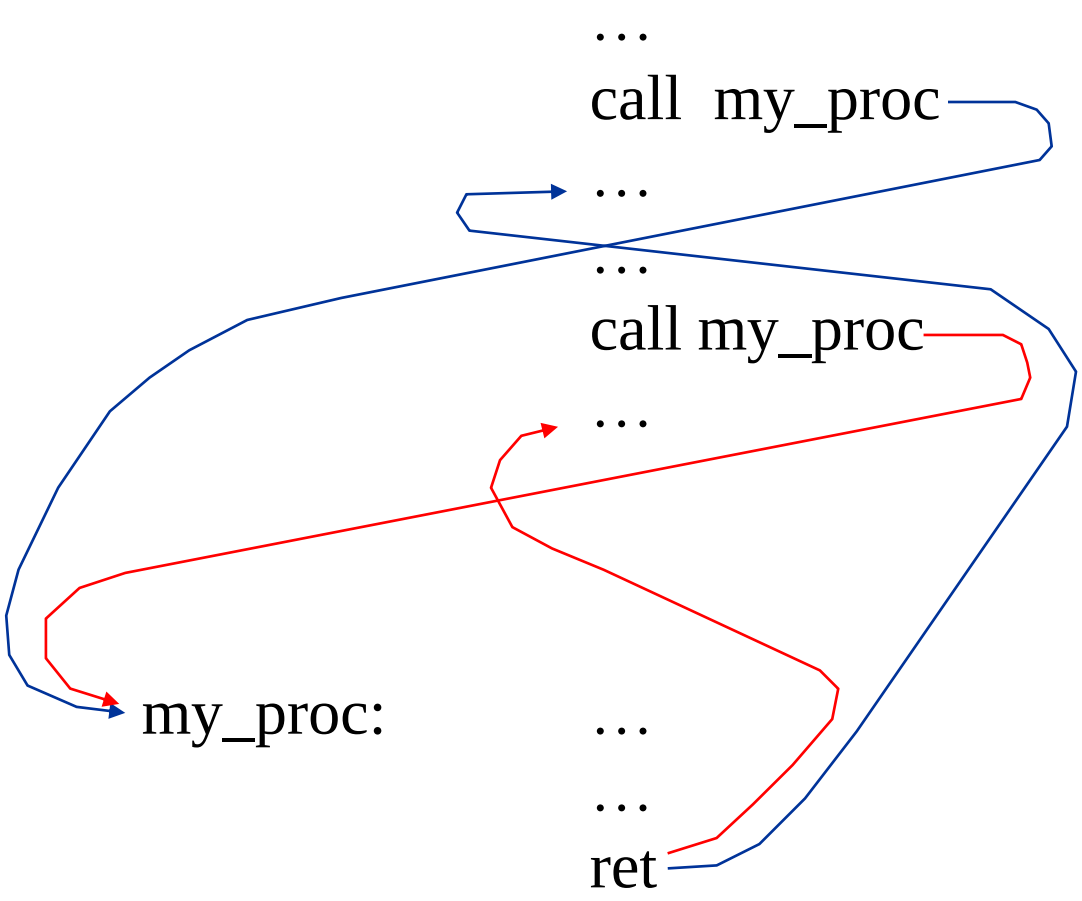
POP EIP, POP CS

POP IP , POP CS, SP + i16 → SP

POP EIP, POP CS, ESP + imm16 → ESP

Příznaky nemění.

Příklad volání procedury a návratu zpět do volajícího programu:



Příklad procedury (E := *max*(A,B,C,D))

```
segment code
…
push word [A]
push word [B]
push word E
call vrat_vetsi
push word [E]
push word [C]
push E
call vrat_vetsi
push word [E]
push word [D]
push word E
call vrat_vetsi
…
```

```
segment DATA
…
A       resw 1
B       resw 1
C       resw 1
D       resw 1
E       resw 1
…
```

Příklad procedury (E := *max*(A,B,C,D))

```
; procedura vrat_vetsi
vrat_vetsi:
        push bp                 ; úschova registru bp do zásobníku
        mov bp,sp               ; Stack: BP|IP|EA výsledku|2.cislo|1.cislo|...
        mov ax,[bp+8]           ; první číslo do AX
        cmp ax,[bp+6]           ; porovnání čísel
        jge vetsi               ; skok, je-li první větší nebo stejné
        mov ax,[bp+6]           ; druhé číslo do AX (je větší než první)
vetsi:  mov bx,[bp+4]           ; efektivní adresa výsledku do bx
        mov [bx],ax             ; vetší z čísel na zadanou adresu
        pop bp                  ; obnova registru bp
        ret 6                   ; odstranění parametrů ze zásobníku
```

INT, INT3, INTO, IRET

INT 3 / INT n / INTO (Call to Interrupt Procedure)

CC	INT3	interrupt 3 - trap to debugger
CD i8	INT imm8	interrupt vector number specified by immediate byte
CE	INTO	interrupt 4 - if overflow flag is 1

Programové přerušení:

PUSHF, $0 \rightarrow$ TF, $0 \rightarrow$ IF, CALL FAR (vector number $*$ 4)

Příznaky CF, OF, SF, ZF, AF a PF nemění.

IRET (Interrupt Return)

CF IRET (IRETD)

Návrat (far return) do přerušeného programu:

POP IP, POP CS, POPF, resp. POP EIP, POP CS, POPFD

Pozn.: Výše jde pouze o symbolický popis činnosti instrukce IRET.
Samostatná instrukce POP CS neexistuje!!!

BOUND

BOUND dest,src (Check Array Index Against Bounds, 186)

62 / r BOUND r16,m16&16 (BOUND r32,m32&32)

Kontroluje, zda obsah *dest* leží v mezích (včetně) určených nepřímo druhým operandem – první slovo (dvouslovo) adresované paměti obsahuje dolní mez, druhé slovo (dvouslovo) pak horní mez. Pokud obsah *dest* leží mimo tyto meze, generuje se výjimka 5.

Příznaky nemění.

ENTER, LEAVE

ENTER o1,o2 (Make Stack Frame for Procedure Parameters, 186)

C8 w b ENTER imm16,imm8

imm8 mod 32 → NestingLevel (imm8 je operand o2)
PUSH(BP) (PUSH(EBP))
SP → FrameTemp (ESP → FrameTemp)
if NestingLevel > 0 then begin
    for i := 1 to (NestingLevel - 1) do begin
        BP – 2 → BP (EBP – 4 → EBP)
        Push([BP]) (* word push *) (Push([EBP])) (* dword push *)
    end
    Push(FrameTemp); (* word push *) (* dword push *)
end
FrameTemp → BP (FrameTemp → EBP)
BP – Size → SP (EBP – Size → ESP)
(* Size je velikost rámce určená operandem o1 *)
Příznaky nemění.

LEAVE (High Level Procedure Exit , 186)

C9 LEAVE

BP → SP (EBP → ESP)
POP(BP) (POP(EBP))

Příznaky nemění.

Instrukce pro ovládání příznaků

- **STC**
- **CLC**
- **CMC**
- **STD**
- **CLD**
- STI
- CLI
- LAHF
- SAHF
- **PUSHF**/PUSHFD
- **POPF**/POPFD

STC, CLC, CMC

STD, CLD

STI, CLI

STC (Set Carry Flag)

F9 STC $1 \rightarrow CF$

Ostatní příznaky nemění.

CLC (Clear Carry Flag)

F8 CLC $0 \rightarrow CF$

Ostatní příznaky nemění.

CMC (Complement Carry Flag)

F5 CMC $\neg CF \rightarrow CF$

Ostatní příznaky nemění.

STD (Set Direction Flag)

FD STD $1 \rightarrow DF$

Ostatní příznaky nemění.

CLD (Clear Direction Flag)

FC CLD $0 \rightarrow DF$

Ostatní příznaky nemění.

STI (Set Interrupt Flag)

FB STI $1 \rightarrow IF$

Ostatní příznaky nemění.

CLI (Clear Interrupt Flag)

FA CLI $0 \rightarrow IF$

Ostatní příznaky nemění.

LAHF, SAHF

LAHF (Load Status Flags into AH Register)

9F LAHF FLAGS(SF:ZF:0:AF:0:PF:1:CF) → AH

Příznaky nemění.

SAHF (Store AH into FLAGS Register)

9E SAHF AH → FLAGS(SF:ZF:0:AF:0:PF:1:CF)

PUSHF/PUSHFD, **POPF**/POPFD

PUSHF/PUSHFD (Push (E)FLAGS Register onto the Stack)

9C PUSHF (PUSHFD)

Příznaky nemění.

POPF/POPFD (Pop Stack into (E)FLAGS Register)

9D POPF (POPFD)

Instrukce pro práci se segmentovými registry

- **LDS**
- **LES**
- **LFS**
- **LGS**
- **LSS**

LDS, LES, LFS, LGS, LSS

LDS/LES/LFS/LGS/LSS dest,src (Load Far Pointer)

C5 /r	LDS r16,m16:16	(LDS r32,m16:32)
C4 /r	LES r16,m16:16	(LES r32,m16:32)
0F B4 /r	LFS r16,m16:16	(LFS r32,m16:32)
0F B5 /r	LGS r16,m16:16	(LGS r32,m16:32)
0F B2 /r	LSS r16,m16:16	(LSS r32,m16:32)

Naplní segmentový registr určený druhým a třetím písmenem instrukce a cílový registr ukazatelem (segment:offset) z paměti adresované zdrojovým operandem.

Příznaky nemění.

Příklad použití:

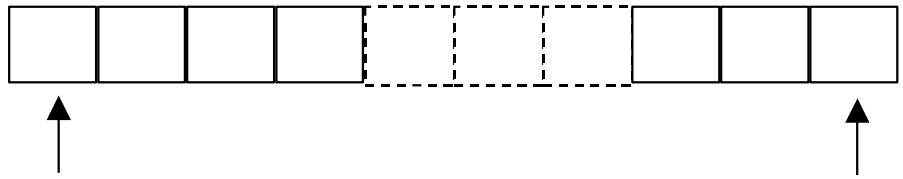
```
            …
            les di,[p_cislo]
            mov al,[es:di]
            call Write_ShortInt
            call WriteLn
            …

            segment moje_data
cislo       db -31

            segment DATA
p_cislo     dw cislo
            dw seg cislo
```

Instrukce posuvů a rotací

- **SAR**
- **SHR**
- **SAL/SHL**
- SHRD
- SHLD
- **ROR**
- **ROL**
- **RCR**
- **RCL**

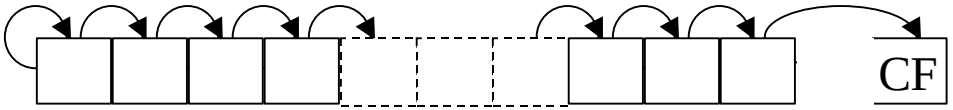


Bity se číslují zprava počínaje nulou, tzn., že LSB má č. 0, MSB slabiky má č. 7, MSB slova č. 15 a MSB dvouslova č. 31.

SAR, SHR, SAL, SHL

SAR dest,count (Shift Arithmetic Right)

D0 /7	SAR r/m8,1	
D2 /7	SAR r/m8,CL	
C0 /7 i8	SAR r/m8,imm8	
D1 /7	SAR r/m16,1	(SAR r/m32,1)
D3 /7	SAR r/m16,CL	(SAR r/m32,CL)
C1 /7 i8	SAR r/m16,imm8	(SAR r/m32,imm8)



Počet posuvů *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet posuvů je vyčíslen pomocí vztahu (*count* and 1FH), t.j. je omezen hodnotou 31. Posouvá se obsah cílového operandu.

Pro *count* = 0 se příznaky nemění. Jinak se příznaky SF, ZF, a PF nastavují podle výsledku a hodnota příznaku AF není definována. Příznak OF je pro *count* = 1 vynulován, jinak není jeho hodnota definována.

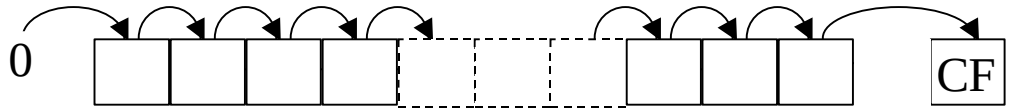
Příklady použití:

SAR … aritmetický posuv doprava: každý posuv znamená celočíselné dělení znaménkového čísla dvěma, v případě lichých čísel s kladným zbytkem +1 (číslo -1 se proto nemění!!!).

sar bl,1	původní hodnota BL:	00011010	26 / 2
	nová hodnota BL:	00001101	= 13
	původní hodnota BL:	00011011	27 / 2
	nová hodnota BL:	00001101	= 13
sar cl,1	původní hodnota CL:	11111100	-4 / 2
	nová hodnota CL:	11111110	= -2
	původní hodnota CL:	11111101	-3 / 2
	nová hodnota CL:	11111110	= -2
	původní hodnota CL:	11111111	-1 / 2
	nová hodnota CL:	11111111	= -1 !!!
sar [alpha],4	původní hodnota [alpha]:	00011110	30 / 16
	nová hodnota [alpha]:	00000001	= 1
mov cl,4			
sar [alpha],cl	totéž jako sar [alpha],4		

SHR dest,count (Shift Logical Right)

D0 /5	SHR r/m8,1	
D2 /5	SHR r/m8,CL	
C0 /5 i8	SHR r/m8,imm8	
D1 /5	SHR r/m16,1	(SHR r/m32,1)
D3 /5	SHR r/m16,CL	(SHR r/m32,CL)
C1 /5 i8	SHR r/m16,imm8	(SHR r/m32,imm8)



Počet posuvů *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet posuvů je vyčíslen pomocí vztahu (*count* and 1FH), t.j. je omezen hodnotou 31. Posouvá se obsah cílového operandu.

Pro *count* = 0 se příznaky nemění. Jinak se příznaky SF, ZF, a PF nastavují podle výsledku a hodnota příznaku AF není definována. Příznak OF je pro *count* = 1 nastaven na původní hodnotu MSB cílového operandu, jinak není jeho hodnota definována.

Příklady použití:

SHR … logický posuv doprava: každý posuv znamená celočíselné dělení bezznaménkového čísla dvěma.

shr bl,1	původní hodnota BL:	00011010	26 / 2
	nová hodnota BL:	00001101	= 13
	původní hodnota BL:	00011011	27 / 2
	nová hodnota BL:	00001101	= 13
shr cl,1	původní hodnota CL:	11111100	252 / 2
	nová hodnota CL:	01111110	= 126
	původní hodnota CL:	11111101	253 / 2
	nová hodnota CL:	01111110	= 126
	původní hodnota CL:	11111111	255 / 2
	nová hodnota CL:	01111111	= 127
shr [alpha],4	původní hodnota [alpha]:	00011110	30 / 16
	nová hodnota [alpha]:	00000001	= 1

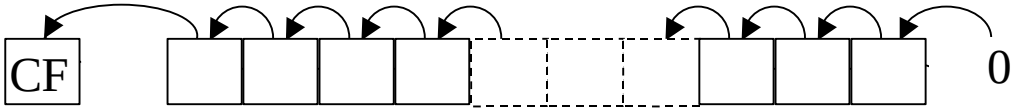
mov cl,4
shr [alpha],cl totéž jako sar [alpha],4

SAL/SHL dest,count (Shift Arithmetic/Logical Left)

D0 /4	SAL r/m8,1	
D2 /4	SAL r/m8,CL	
C0 /4 i8	SAL r/m8,imm8	
D1 /4	SAL r/m16,1	(SAL r/m32,1)
D3 /4	SAL r/m16,CL	(SAL r/m32,CL)
C1 /4 i8	SAL r/m16,imm8	(SAL r/m32,imm8)

D0 /4	SHL r/m8,1	
D2 /4	SHL r/m8,CL	
C0 /4 i8	SHL r/m8,imm8	
D1 /4	SHL r/m16,1	(SHL r/m32,1)
D3 /4	SHL r/m16,CL	(SHL r/m32,CL)
C1 /4 i8	SHL r/m16,imm8	(SHL r/m32,imm8)

SAL/SHL dest,count (continuation)



Počet posuvů *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet posuvů je vyčíslen pomocí vztahu (*count* and 1FH), t.j. je omezen hodnotou 31. Posouvá se obsah cílového operandu.

Pro *count* = 0 se příznaky nemění. Jinak se příznaky SF, ZF, a PF nastavují podle výsledku a hodnota příznaku AF není definována. Příznak OF je pro *count* = 1 vynulován, jestliže MSB výsledku má stejnou hodnotu jako CF (jestliže je hodnota dvou nejvyšších bitů cílového operandu před posuvem stejná), jinak je tento příznak nastaven (OF=1).

Příklady použití:

SAL … aritmetický posuv doleva: každý posuv znamená celočíselné násobení znaménkového čísla dvěma. Přetečení výsledku indikuje příznak OF.

Znaménková čísla:			CF	**OF**
sal bl,1	původní hodnota BL:	00011010 26 * 2		
	nová hodnota BL:	00110100 = 52	0	0
	původní hodnota BL:	11111110 = -2 * 2		
	nová hodnota BL:	11111100 = -4	1	0
	původní hodnota BL:	10000011 -125 * 2		
	nová hodnota BL:	00000110 = 6	1	1

Příklady použití:

SHL … logický posuv doleva: každý posuv znamená celočíselné násobení bezznaménkového čísla dvěma. Přetečení výsledku indikuje příznak CF.

Bezznaménková čísla:					**CF**	OF
shl cl,1	původní hodnota CL:	00001111		15 * 2		
	nová hodnota CL:	00011110	=	30	0	0
	původní hodnota CL:	01111100		124 * 2		
	nová hodnota CL:	11111000	=	248	0	1
	původní hodnota CL:	11111110		254 * 2		
	nová hodnota CL:	11111100	=	252	1	0

Příklady použití:

Pozor na posuvy doleva, jejichž počet je větší než 1 !!!

Znaménková čísla				CF	OF
sal byte [alpha],4	původní hodnota [alpha]:	00011110	30 * 16	1	-
	nová hodnota [alpha]:	11100000 =	-32		

Beznaménková čísla

mov cl,5					
shl bl,cl	původní hodnota BL:	11000010	194 * 32	0	-
	nová hodnota BL:	01000000 =	64		

SAL dest,count	(Shift Arithmetic Left)
SHL dest,count	(Shift Logic Left)
SAR dest,count	(Shift Arithmetic Right)
SHR dest,count	(Shift Logic Right)

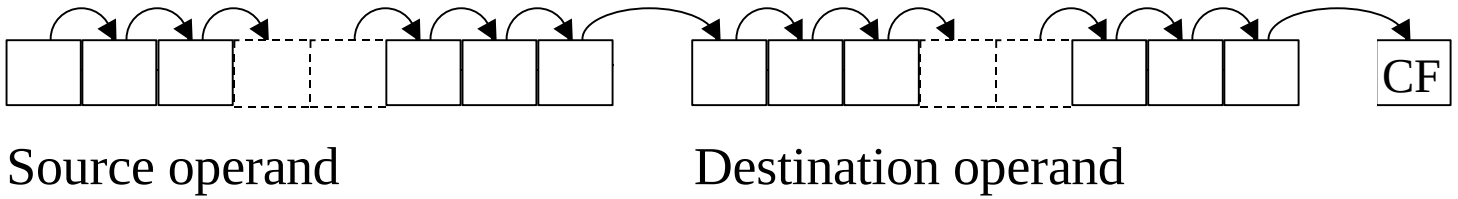
Stručný přehled jednotlivých možností:

1.	SAL/SHL/SAR/SHR	reg,1	reg8/16/32
2.	SAL/SHL/SAR/SHR	reg,cl	reg8/16/32
3.	SAL/SHL/SAR/SHR	reg,imm8	reg8/16/32
4.	SAL/SHL/SAR/SHR	mem,1	mem8/16/32
5.	SAL/SHL/SAR/SHR	mem,cl	mem8/16/32
6.	SAL/SHL/SAR/SHR	mem,imm8	mem8/16/32

SHRD, SHLD

SHRD dest,src,count (Double Precision Shift Right, 386)

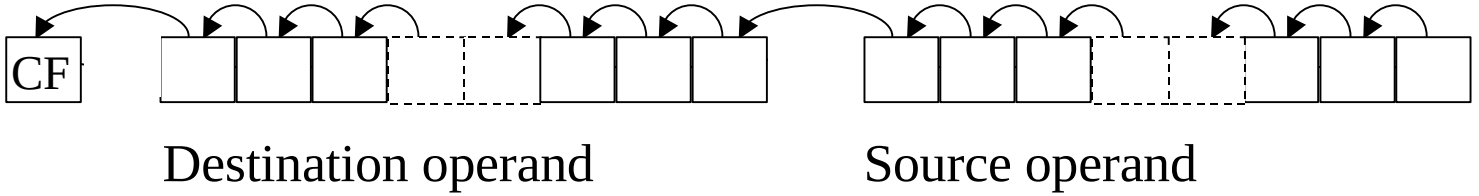
0F AC … SHRD r/m16,r16,imm8 (SHRD r/m32,r32,imm8)
0F AD … SHRD r/m16,r16,CL (SHRD r/m32,r32,CL)



Počet posuvů *count* je dán buď přímým operandem nebo obsahem registru CL. Skutečný počet posuvů je vyčíslen pomocí vztahu (*count* mod 32). Tento počet musí být menší než velikost cílového operandu, jinak nejsou hodnoty cílového operandu a příznaků CF, OF, SF, ZF, AF a PF definovány.
Posouvá se obsah cílového operandu, zdrojový operand se nemění!
Pro *count* = 0 se příznaky nemění. Jinak se příznaky SF, ZF, a PF nastavují podle výsledku a hodnota příznaku AF není definována. Hodnota příznaku OF pro *count* = 1 závisí na hodnotě MSB cílového operandu: změní-li se MSB je OF nastaven (OF=1), nezmění-li se MSB je OF vynulován (OF=0). Pro *count* > 1 není hodnota příznaku OF definována.

SHLD dest,src,count (Double Precision Shift Left, 386)

0F A4… SHRD r/m16,r16,imm8 (SHRD r/m32,r32,imm8)
0F A5… SHRD r/m16,r16,CL (SHRD r/m32,r32,CL)

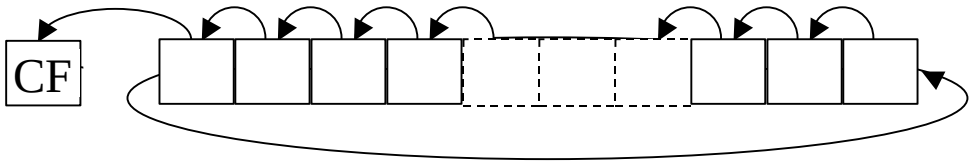


Počet posuvů *count* je dán buď přímým operandem nebo obsahem registru CL. Skutečný počet posuvů je vyčíslen pomocí vztahu (*count* mod 32). Tento počet musí být menší než velikost cílového operandu, jinak nejsou hodnoty cílového operandu a příznaků CF, OF, SF, ZF, AF a PF definovány.
Posouvá se obsah cílového operandu, zdrojový operand se nemění!
Pro *count* = 0 se příznaky nemění. Jinak se příznaky SF, ZF, a PF nastavují podle výsledku a hodnota příznaku AF není definována. Hodnota příznaku OF pro *count* = 1 závisí na hodnotě MSB cílového operandu: změní-li se MSB,je OF nastaven (OF=1), nezmění-li se MSB je OF vynulován (OF=0). Pro *count* > 1 není hodnota příznaku OF definována.

ROL, ROR, RCL, RCR

ROL dest,count (Rotate Left)

D0 /0	ROL r/m8,1	
D2 /0	ROL r/m8,CL	
C0 /0 i8	ROL r/m8,imm8	
D1 /0	ROL r/m16,1	(ROL r/m32,1)
D3 /0	ROL r/m16,CL	(ROL r/m32,CL)
C1 /0 i8	ROL r/m16,imm8	(ROL r/m32,imm8)

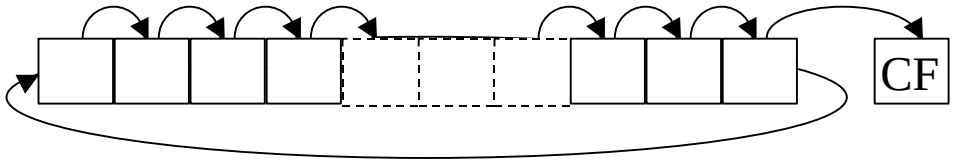


Počet rotací *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet rotací je vyčíslen pomocí vztahu (*count* mod size), t.j. je omezen hodnotou 7/15/31. Rotuje se obsah cílového operandu.

Příznaky SF, ZF, AF a PF se nemění, pro *count* = 0 se nemění ani příznaky CF a OF. Pro *count* = 1 je hodnota příznaku OF dána funkcí XOR příznaku CF a MSB výsledného cílového operandu. Pro *count* > 1 není hodnota příznaku OF definována.

ROR dest,count (Rotate Right)

D0 /1	ROR r/m8,1	
D2 /1	ROR r/m8,CL	
C0 /1 i8	ROR r/m8,imm8	
D1 /1	ROR r/m16,1	(ROR r/m32,1)
D3 /1	ROR r/m16,CL	(ROR r/m32,CL)
C1 /1 i8	ROR r/m16,imm8	(ROR r/m32,imm8)

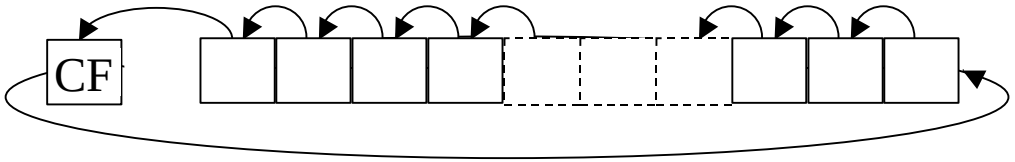


Počet rotací *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet rotací je vyčíslen pomocí vztahu (*count* mod size), t.j. je omezen hodnotou 7/15/31. Rotuje se obsah cílového operandu.

Příznaky SF, ZF, AF a PF se nemění, pro *count* = 0 se nemění ani příznaky CF a OF. Pro *count* = 1 je hodnota příznaku OF dána funkcí XOR dvou nejvyšších bitů výsledného cílového operandu. Pro *count* > 1 není hodnota příznaku OF definována.

RCL dest,count (Rotate through Carry Left)

D0 /2	RCL r/m8,1	
D2 /2	RCL r/m8,CL	
C0 /2 i	RCL r/m8,imm8	
D1 /2	RCL r/m16,1	(RCL r/m32,1)
D3 /2	RCL r/m16,CL	(RCL r/m32,CL)
C1 /2 i	RCL r/m16,imm8	(RCL r/m32,imm8)

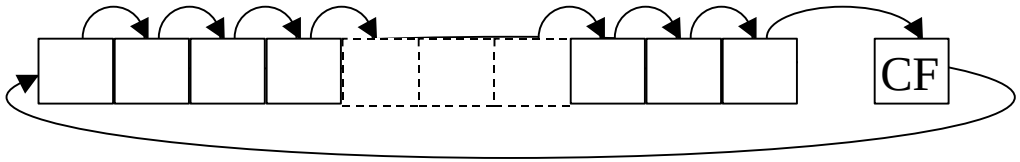


Počet rotací *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet rotací je vyčíslen pomocí vztahu (*count* mod (size+1)), t.j. je omezen hodnotou 8/16/32. Rotuje obsah cílového operandu.

Příznaky SF, ZF, AF a PF se nemění, pro *count* = 0 se nemění ani příznaky CF a OF. Pro *count* = 1 je hodnota příznaku OF dána funkcí XOR příznaku CF a MSB výsledného cílového operandu. Pro *count* > 1 není hodnota příznaku OF definována.

RCR dest,count (Rotate through Carry Right)

D0 /3	RCR r/m8,1	
D2 /3	RCR r/m8,CL	
C0 /3 i	RCR r/m8,imm8	
D1 /3	RCR r/m16,1	(RCR r/m32,1)
D3 /3	RCR r/m16,CL	(RCR r/m32,CL)
C1 /3 i	RCR r/m16,imm8	(RCR r/m32,imm8)



Počet rotací *count* je dán zdrojovým operandem: buď je 1, nebo je dán obsahem registru CL, nebo přímým operandem. Skutečný počet rotací je vyčíslen pomocí vztahu (*count* mod (size+1)), t.j. je omezen hodnotou 8/16/32. Rotuje se obsah cílového operandu.

Příznaky SF, ZF, AF a PF se nemění, pro *count* = 0 se nemění ani příznaky CF a OF. Pro *count* = 1 je hodnota příznaku OF dána funkcí XOR dvou nejvyšších bitů výsledného cílového operandu. Pro *count* > 1 není hodnota příznaku OF definována.

Příklady použití:

			CF
rol byte [alpha],4	původní hodnota [alpha]:	00011110	-
	nová hodnota [alpha]:	11100001	1
ror ax,8	původní hodnota AX:	1100110011110001	
	nová hodnota AX:	1111000111001100	1
	(vzájemná výměna obsahů registrů AH a AL)		
mov cl,6			
rcr bx,cl	původní hodnota BX a CF:	1110001100101011	0
	nová hodnota BX:	0101101110001100	1
	(v CF je nyní hodnota bitu č. 5 původního obsahu registru BX)		

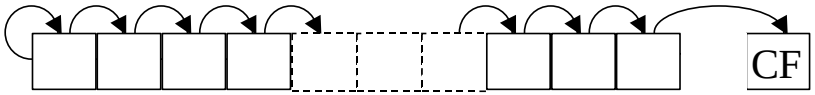
ROL dest,count	(Rotate Left)
ROR dest,count	(Rotate Right)
RCL dest,count	(Rotate through Carry Left)
RCR dest,count	(Rotate through Carry Right)

Stručný přehled jednotlivých možností:

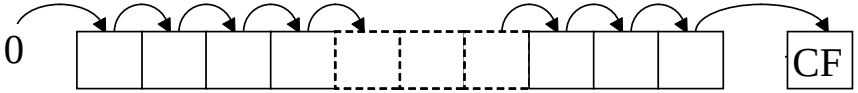
1.	ROL/ROR/RCL/RCR	reg,1	reg8/16/32
2.	ROL/ROR/RCL/RCR	reg,cl	reg8/16/32
3.	ROL/ROR/RCL/RCR	reg,imm8	reg8/16/32
4.	ROL/ROR/RCL/RCR	mem,1	mem8/16/32
5.	ROL/ROR/RCL/RCR	mem,cl	mem8/16/32
6.	ROL/ROR/RCL/RCR	mem,imm8	mem8/16/32

Stručný přehled všech posuvů a rotací:

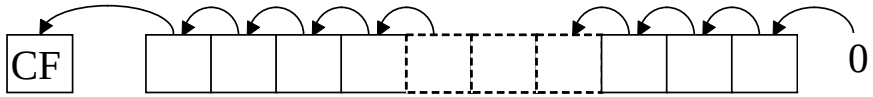
SAR



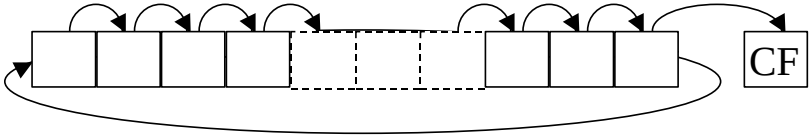
SHR



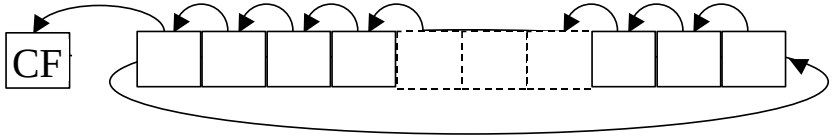
SAL/SHL



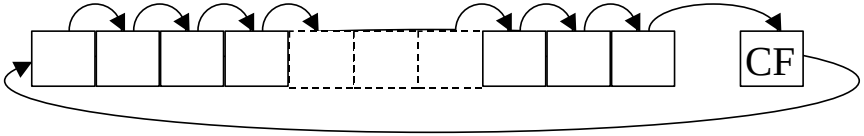
ROR



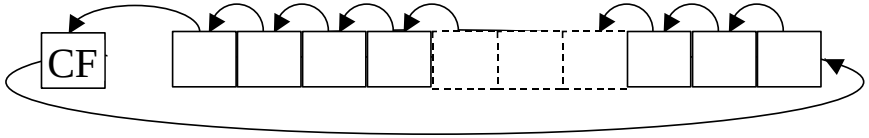
ROL



RCR



RCL



Posuvy - stručný přehled jednotlivých možností:

1.	SAL/SHL/SAR/SHR reg,1	reg8/16/32
2.	SAL/SHL/SAR/SHR reg,cl	reg8/16/32
3.	SAL/SHL/SAR/SHR reg,imm8	reg8/16/32
4.	SAL/SHL/SAR/SHR mem,1	mem8/16/32
5.	SAL/SHL/SAR/SHR mem,cl	mem8/16/32
6.	SAL/SHL/SAR/SHR mem,imm8	mem8/16/32

Rotace - stručný přehled jednotlivých možností:

1.	ROL/ROR/RCL/RCR reg,1	reg8/16/32
2.	ROL/ROR/RCL/RCR reg,cl	reg8/16/32
3.	ROL/ROR/RCL/RCR reg,imm8	reg8/16/32
4.	ROL/ROR/RCL/RCR mem,1	mem8/16/32
5.	ROL/ROR/RCL/RCR mem,cl	mem8/16/32
6.	ROL/ROR/RCL/RCR mem,imm8	mem8/16/32