

4. Soubor instrukcí procesorů Pentium. Celočíselné instrukce.



Instrukční soubor procesorů Pentium:

- Celočíselné instrukce
 - přenosové instrukce
 - instrukce binární aritmetiky
 - instrukce pro předávání řízení
 - instrukce pro ovládání příznaků
 - instrukce pro práci se segmentovými registry
 - logické instrukce
 - instrukce posuvů a rotací
 - instrukce pro práci s bity a slabikami
 - řetězové instrukce
 - instrukce dekadické aritmetiky
- FPU instrukce
- MMX, SSE, AVX instrukce ...
- Systémové instrukce

Celočíselné instrukce

Tučně jsou označeny důležité instrukce

Normálně jsou označeny užitečné instrukce

Červeně jsou označeny instrukce, které
nebudeme v IAS používat

Přenosové instrukce (nemění příznaky)

- MOV
- PUSH
- POP
- PUSHA/PUSHAD
- POPA/POPAD
- CBW/CWDE
- CWD/CDQ
- XCHG
- IN
- OUT
- CMOV_{cc}
- MOVSX
- MOVZX
- BSWAP
- XADD
- CMPXCHG
- CMPXCHG8B

MOV

MOV dest,src

(Move Source to Destination)

88 /r MOV r/m8,r8
 89 /r MOV r/m16,r16
 89 /r MOV r/m32,r32

1	0	0	0	1	0	0	0	mod	/r	r/m
1	0	0	0	1	0	0	1	mod	/r	r/m
1	0	0	0	1	0	0	1	m o d	/r	r/m

Příklady:

mov al,bl	88D8	[1000 1000 11 011 000]
mov sp,bp	89EC	
mov [alpha],cl	880E+d16	d16 = offset of alpha
mov [bx],al	8807	
mov dx,sp	89E2	
mov [beta],bp	892E+d16	d16 = offset of beta
mov [bp+si+10],bx	895A+d8	d8 = 0AH
mov ecx,edi	66 89F9	66 = prefix
mov [gamma],ebx	66 891E+d16	d16 = offset of gamma
mov [di+gamma+20],ecx	66 898D+d16	d16 = offset of gamma + 20

MOV dest,src

(Move Source to Destination)

8A /r MOV r8,r/m8
 8B /r MOV r16,r/m16
 8B /r MOV r32,r/m32

1	0	0	0	1	0	1	0	mod	/r	r/m
1	0	0	0	1	0	1	1	m o d	/r	r/m
1	0	0	0	1	0	1	1	m o d	/r	r/m

Příklady:

mov al,bl	8AC3	[1000 1010 11 000 011]
mov cl,[alpha]	8A0E+d16	d16 = offset of alpha
mov dl,[bx]	8A17	
mov bp, [beta]	8B2E+d16	d16 = offset of beta
mov ax,[bp+si+10]	8B42+d8	d8 = 0AH
mov ebx,[gamma]	66 8B1E+d16	d16 = offset of gamma
mov ecx,[di+gamma+20]	66 8B8D+d16	d16 = offset of gamma + 20
mov ecx,[gamma+di+20]	66 8B8D+d16	d16 = offset of gamma + 20

MOV dest,src

(Move Source to Destination)

8C /sreg MOV r/m16,sreg

1	0	0	0	1	1	0	0	mod	/sreg	r/m
---	---	---	---	---	---	---	---	-----	-------	-----

Segmentový registr:	ES	CS	SS	DS	FS	GS
sreg:	000	001	010	011	100	101

Příklady:

mov cx,ds

8CD9

mov [alpha],cs

8C0E+d16

d16 = offset of alpha

mov [bx+si+330],gs

8CA8+d16

d16 = 014AH

8CA84A01

takto je to v paměti!

MOV dest,src

(Move Source to Destination)

8E /sreg MOV sreg,r/m16

1	0	0	0	1	1	1	0	mod	/sreg	r/m
---	---	---	---	---	---	---	---	-----	-------	-----

!!! sreg nesmí být CS registr !!!

Segmentový registr:	ES	CS	SS	DS	FS	GS
sreg:	000	001	010	011	100	101

Příklady:

mov es,ax

8EC0

mov fs,[bp+si+10]

8E62+d8

d8 = 0AH

mov ss,[beta+di+20]

8E95+d16

d16 = offset of beta + 20

MOV dest,src

(Move Source to Destination)

A0 MOV AL,moffs8
 A1 MOV AX,moffs16
 A1 MOV EAX,moffs32

1 0 1 0 0 0 0 0	displacement – 16 bitů
1 0 1 0 0 0 0 1	displacement – 16 bitů
1 0 1 0 0 0 0 1	displacement – 16 bitů

Pozn.: moffs (memory offset) vyjadřuje skutečnost, že v instrukci je přímo EA, tj. přímo 16ti bitový displacement bez slabiky r/m. Číslo 8, 16, nebo 32 označuje velikost operandu, který se z paměti přenesse do střádače.

Příklady:

mov al,[alpha]	A0+d16	d16 = offset of alpha
mov ax,[beta]	A1+d16	d16 = offset of beta
mov eax,[gamma]	66A1+d16	d16 = offset of gamma
mov al,[alpha+15]	A0+d16	d16 = offset of alpha + 15

MOV dest,src

(Move Source to Destination)

A2 MOV moffs8,AL

1	0	1	0	0	0	1	0	displacement – 16 bitů
---	---	---	---	---	---	---	---	------------------------

A3 MOV moffs16,AX

1	0	1	0	0	0	1	1	displacement – 16 bitů
---	---	---	---	---	---	---	---	------------------------

A3 MOV moffs32,EAX

1	0	1	0	0	0	1	1	displacement – 16 bitů
---	---	---	---	---	---	---	---	------------------------

Pozn.: moffs (memory offset) vyjadřuje skutečnost, že v instrukci je přímo EA, tj. přímo 16ti bitový displacement bez slabiky r/m. Číslo 8, 16, nebo 32 označuje velikost operandu, který se z paměti přenesse do strádače.

Příklady:

mov [alpha],al

A2+d16

d16 = offset of alpha

mov [beta+330],ax

A3+d16

d16 = offset of beta + 330

mov [gamma],eax

66A3+d16

d16 = offset of gamma

MOV dest,src

(Move Source to Destination)

B0+r MOV r8,imm8
 B8+r MOV r16,imm16
 B8+r MOV r32,imm32

1 0 1 1 0 - r -	immediate – 8bitů
1 0 1 1 1 - r -	immediate – 16bitů
1 0 1 1 1 - r -	immediate – 32 bitů

Příklady:

mov al,12	B0+i8	i8 = 0CH
mov bx,254	BB+i16	i16 = 00FEH
mov bx,-15	BB+i16	i16 = 0FFF1H
	BBF1FF	takto je to v paměti!
mov ebp,1000	66BD+d32	d32 = 000003E8H
	66BDE8030000	takto je to v paměti!
mov bp,alpha	BD+d16	d16 = offset of alpha
mov eax,beta+20	66B8+d32	d32 = offset of beta + 20

MOV dest,src

(Move Source to Destination)

C6 /0 MOV r/m8,imm8

1	1	0	0	0	1	1	0	mod	0	0	0	r/m	immediate i8
---	---	---	---	---	---	---	---	-----	---	---	---	-----	--------------

1	1	0	0	0	1	1	0	mod	0	0	0	r/m	displacement d8	immediate i8
---	---	---	---	---	---	---	---	-----	---	---	---	-----	-----------------	--------------

1	1	0	0	0	1	1	0	mod	0	0	0	r/m	displacement high	displacement low	immediate - i8
---	---	---	---	---	---	---	---	-----	---	---	---	-----	-------------------	------------------	----------------

Příklady:

mov cl,20

C6C1+i8

i8 = 14H

mov byte [bx+si],15

C600+i8

i8 = 0FH

mov byte [bx+si+20],-13

C640+d8+i8

d8 = 14H i8 = F3H

C64014F3

takto je to v paměti!

mov byte [alpha],30

C606+d16+i8

d16 = offset of alpha

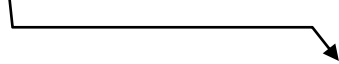
i8 = 1EH

MOV dest,src

(Move Source to Destination)

C7 /0 MOV r/m16,imm16

C7 /0 MOV r/m32,imm32



1	1	0	0	0	1	1	1	mod	0	0	0	r/m	immediate i16/i32
---	---	---	---	---	---	---	---	-----	---	---	---	-----	-------------------

1	1	0	0	0	1	1	1	mod	0	0	0	r/m	displacement d8	immediate – i16/i32
---	---	---	---	---	---	---	---	-----	---	---	---	-----	-----------------	---------------------

1	1	0	0	0	1	1	1	mod	0	0	0	r/m	displacement high	displacement low	immediate i16/i32
---	---	---	---	---	---	---	---	-----	---	---	---	-----	-------------------	------------------	-------------------

Příklady:

mov word [alpha],-150	C706+d16+i16	d16 = offset of alpha, i16 = FF6AH
mov word [bx+di],100	C701+i16	i16 = 0064H
	C7016400	takto je to v paměti!
mov word [bx+di+20],100	C741+d8+i16	d8 = 14H, i16 = 0064H
	C741146400	takto je to v paměti!
mov word [alpha],beta	C706+d16+i16	d16 = offset of alpha, i16 = offset of beta

MOV dest,src

(Move Source to Destination)

C7 /0 MOV r/m16,imm16

C7 /0 MOV r/m32,imm32

Příklady:

mov dword [alpha],-150	66C706+d16+i32	d16 = offset of alpha, i32 = FFFFFFF6AH
mov dword [bx+di],100	66C701+i132 66C70164000000	i32 = 00000064H takto je to v paměti!
mov dword [bx+di+20],100	66C741+d8+i32 66C7411464000000	d8 = 14H, i32 = 00000064H takto je to v paměti!
mov dword [alpha],beta	66C706+d16+i32	d16 = offset of alpha, i32 = offset of beta

MOV dest,src

(Move Source to Destination)

Stručný přehled jednotlivých možností:

1. MOV reg,reg ... reg ~ registr
2. MOV reg,mem ... mem ~ paměťový operand
3. MOV mem,reg
4. MOV reg16,sreg ... sreg ~ segmentový registr
5. MOV sreg,reg16 ... reg16 ~ 16bitový registr, **sreg nesmí být CS**
6. MOV sreg,mem16 ... !!!! Cílovým sreg nesmí být CS registr !!!
7. MOV reg,imm ... imm značí přímý operand
8. MOV mem,imm

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

mov ax,cx

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

mov sp,dx

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

mov bl,dh

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

mov [beta],dh

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

10010011	FFFF
00100000	FFFE
11100011	FFFD
...	...
10101010	alpha+1
01010101	alpha
...	...
10011100	beta+1
00101101	beta
...	...
11110111	0001
00011110	0000

mov [beta],dx

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

10010011	FFFF
00100000	FFFE
11100011	FFFD
...	...
10101010	alpha+1
01010101	alpha
...	...
10011100	beta+1
00101101	beta
...	...
11110111	0001
00011110	0000

mov di,[alpha]

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

10010011	FFFF
00100000	FFFE
11100011	FFFD
...	...
10101010	alpha+1
01010101	alpha
...	...
10011100	beta+1
00101101	beta
...	...
11110111	0001
00011110	0000

mov dl,[bx]

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000111	10111011

10010011	FFFF
00100000	FFFE
11100011	FFFD
...	...
10101010	alpha+1
01010101	alpha
...	...
10011100	beta+1
00101101	beta
...	...
11110111	0001
00011110	0000

XCHG

XCHG dest,src (Exchange Register/Memory with Register)

90+r	XCHG AX,r16	(XCHG EAX,r32)
90+r	XCHG r16,AX	(XCHG r32,EAX)
86 /r	XCHG r/m8,r8	
86 /r	XCHG r8,r/m8	
87 /r	XCHG r/m16,r16	(XCHG r/m32,r32)
87 /r	XCHG r16,r/m16	(XCHG r32,r/m32)

Příklady použití instrukce XCHG

```
xchg bl,cl
```

```
xchg [alpha],al
```

```
xchg al,[alpha]
```

```
xchg dx,si
```

```
xchg ax,[beta]
```

```
xchg [beta],si
```

```
xchg eax,ebx
```

XCHG dest,src (Exchange Register/Memory with Register)

Stručný přehled jednotlivých možností:

1. XCHG reg,reg ... reg značí registr
2. XCHG reg,mem ... mem značí paměťový operand
3. XCHG mem,reg

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

zdroj

00111010

cíl

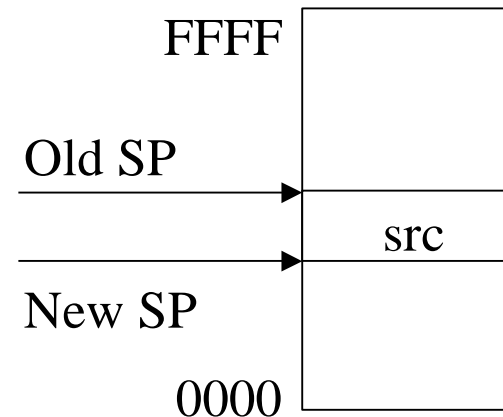
10000001

PUSH, POP

PUSH src

(Push onto Stack)

FF /6	PUSH r/m16	(PUSH r/m32)
50+r	PUSH r16	(PUSH r32)
6A	PUSH imm8	
68	PUSH imm16	(PUSH imm32)
0E	PUSH CS	
16	PUSH SS	
1E	PUSH DS	
06	PUSH ES	
0F A0	PUSH FS	
0F A8	PUSH GS	



V případě imm8 se operand před uložením do paměti znaménkově rozšíří na 16 nebo 32 bitů!

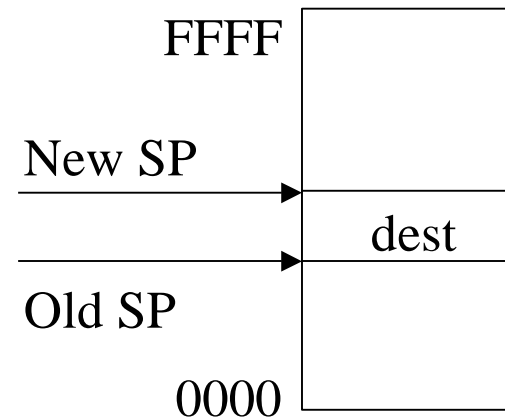
POP dest

(Pop a Value from the Stack)

8F /0	POP m16
58+r	POP r16
1F	POP DS
07	POP ES
17	POP SS
0F A1	POP FS
0F A9	POP GS

(POP m32)

(POP r32)



Příklady použití instrukcí PUSH a POP

```
push cx  
push word [beta]  
push word beta
```

push byte 5	6A05	do paměti se uloží 0005H
push word 5	680500	do paměti se uloží 0005H

```
push cs
```

```
pop ds  
pop bp  
pop word [gamma]  
pop dword [beta]
```


PUSH src

(Push onto Stack)

POP dest

(Pop a Value from the Stack)

Stručný přehled jednotlivých možností:

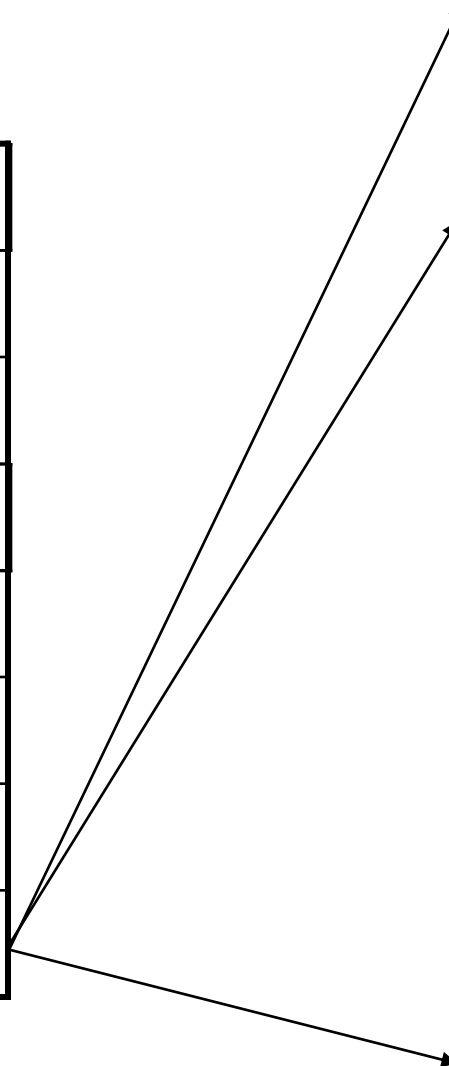
- | | | |
|--------------|-----------------|----------------------------|
| 1. PUSH reg | ... reg16/reg32 | (ukládá se na 16/32 bitů) |
| 2. PUSH mem | ... mem16/m32 | (ukládá se na 16/32 bitů) |
| 3. PUSH sreg | | (ukládá se na 16 bitů) |
| 4. PUSH imm | ... imm8/16/32 | (ukládá se na 16/32 bitů!) |

- | | |
|-------------|-----------------------------------|
| 1. POP reg | ... reg16, nebo reg32 |
| 2. POP mem | ... mem16, nebo m32 |
| 3. POP sreg | ... sreg nesmí být registr CS !!! |

push ax
push dx

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	00000000	00000000

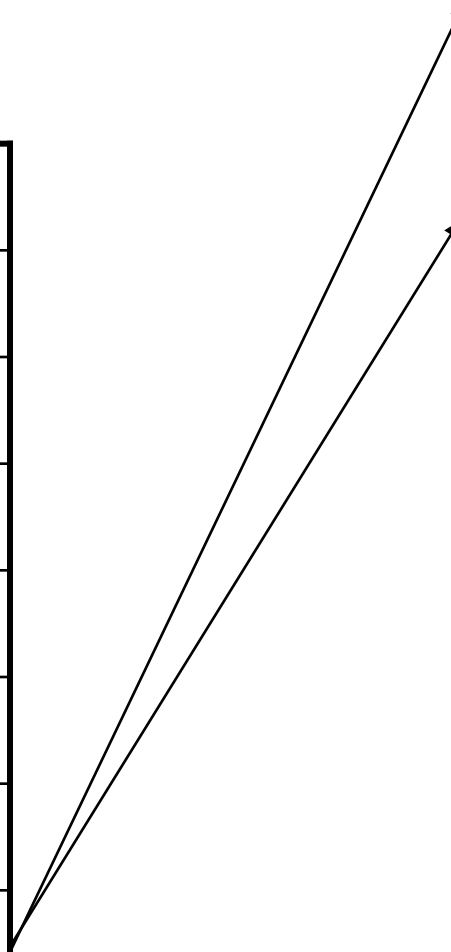
10010011	FFFF
00100000	FFFE
11100011	FFFD
00110011	FFFC
...	
...	
...	...
...	
...	
...	
11110111	0001
10100101	0000



pop bx

AX	10010011	00111001
BX	11111111	11111101
CX	11110111	00011100
DX	00011110	10000001
SI	11000110	11011011
DI	00110110	11110000
BP	00111100	00100110
SP	11111111	11111110

10010011	FFFF
00100000	FFFE
11100011	FFFD
00110011	FFFC
...	
...	
...	...
...	
...	
...	
11110111	0001
10100101	0000



PUSHA, POPA

PUSHA/PUSHAD (Push All General-Purpose Registers)

60 **PUSHA** push AX, CX, DX, BX, original SP, BP, SI,
 and DI

60	PUSHAD	push EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI
----	--------	--

POPA/POPAD (Pop All General-Purpose Registers)

61 POPA pop DI, SI, BP, -, BX, DX, CX, and AX

61 POPAD pop EDI, ESI, EBP, -, EBX, EDX, ECX,
 and EAX

CBW/CWDE, CWD/CDQ

CBW/CWDE

(Convert Byte to Word /
Convert Word to Doubleword)

98 CBW

sign-extend of AL \rightarrow AX

98 CWDE

sign-extend of AX \rightarrow EAX

CWD/CDQ

(Convert Word to Doubleword /
Convert Doubleword to Quadword)

99 CWD

sign-extend of AX \rightarrow DX:AX

99 CDQ

sign-extend of EAX \rightarrow EDX:EAX

cbw

ax

00000000	00111001
----------	----------

cbw

ax

00110010	10111001
----------	----------

IN, OUT

IN src		(Input from Port)
E4 i8	IN AL, imm8	input byte from imm8 I/O port address into AL
E5 i8	IN AX, imm8	input byte from imm8 I/O port address into AX
E5 i8	IN EAX, imm8	input byte from imm8 I/O port address into EAX
EC	IN AL,DX	input byte from I/O port address in DX into AL
ED	IN AX,DX	input word from I/O port address in DX into AX
ED	IN EAX,DX	input doubleword from I/O port address in DX into EAX

OUT dest

(Output to Port)

E6 i8	OUT imm8, AL	output byte in AL to I/O port address imm8
E7 i8	OUT imm8, AX	output word in AX to I/O port address imm8
E7 i8	OUT imm8, EAX	output doubleword in EAX to I/O port address imm8
EE	OUT DX, AL	output byte in AL to I/O port address in DX
EF	OUT DX, AX	output word in AX to I/O port address in DX
EF	OUT DX, EAX	output doubleword in EAX to I/O port address in DX

CMOV

CMOVcc dest,src (Conditional Move Words or Doublewords)

0F 47 /r	CMOVA r,r/m	move if above (CF=0 and ZF=0)
0F 43 /r	CMOVAE r,r/m	move if above or equal (CF=0)
0F 42 /r	CMOVB r,r/m	move if below (CF=1)
0F 46 /r	CMOVBE r,r/m	move if below or equal (CF=1 or ZF=1)
0F 42 /r	CMOVC r,r/m	move if carry (CF=1)
0F 44 /r	CMOVE r,r/m	move if equal (ZF=1)
0F 4F /r	CMOVG r,r/m	move if greater (ZF=0 and SF=OF)
0F 4D /r	CMOVGE r,r/m	move if greater or equal (SF=OF)
0F 4C /r	CMOVL r,r/m	move if less (SF<>OF)
0F 4E /r	CMOVLE r,r/m	move if less or equal (ZF=1 or SF<>OF)
0F 46 /r	CMOVNA r,r/m	move if not above (CF=1 or ZF=1)
0F 42 /r	CMOVNAE r,r/m	move if not above or equal (CF=1)
0F 43 /r	CMOVNB r,r/m	move if not below (CF=0)
0F 47 /r	CMOVNBE r,r/m	move if not below or equal (CF=0 and ZF=0)

0F 43 /r	CMOVNC r,r/m	move if not carry (CF=0)
0F 45 /r	CMOVNE r,r/m	move if not equal (ZF=0)
0F 4E /r	CMOVNG r,r/m	move if not greater (ZF=1 or SF<>OF)
0F 4C /r	CMOVNGE r,r/m	move if not greater or equal (SF<>OF)
0F 4D /r	CMOVNL r,r/m	move if not less (SF=OF)
0F 4F /r	CMOVNLE r,r/m	move if not less or equal (ZF=0 and SF=OF)
0F 41 /r	CMOVNO r,r/m	move if not overflow (OF=0)
0F 4B /r	CMOVNP r,r/m	move if not parity (PF=0)
0F 49 /r	CMOVNS r,r/m	move if not sign (SF=0)
0F 45 /r	CMOVNZ r,r/m	move if not zero (ZF=0)
0F 40 /r	CMOVO r,r/m	move if overflow (OF=1)
0F 4A /r	CMOVP r,r/m	move if parity (PF=1)
0F 4A /r	CMOVPE r,r/m	move if parity even (PF=1)
0F 4B /r	CMOVPO r,r/m	move if parity odd (PF=0)
0F 48 /r	CMOVS r,r/m	move if sign (SF=1)
0F 44 /r	CMOVZ r,r/m	move if zero (ZF=1)

MOVSX, MOVZX

MOVSX dest,src (Move with Sign Extension, 386)

0F BE /r	MOVSX r16,r/m8	move byte to word with sign extension
0F BE /r	MOVSX r32,r/m8	move byte to doubleword, sign extension
0F BF /r	MOVSX r32,r/m16	move word to doubleword, sign extension

MOVZX dest,src (Move with Zero Extension, 386)

0F B6 /r	MOVZX r16,r/m8	move byte to word with zero extension
0F B6 /r	MOVZX r32,r/m8	move byte to doubleword, zero extension
0F B7 /r	MOVZX r32,r/m16	move word to doubleword, zero extension

BSWAP, XADD,
CMPXCHG, CMPXCHG8B

BSWAP r32 (Byte Swap, i486™)

0F C8+r BSWAP r32 reverses bytes of r32 register

XADD dest,src (Exchange and Add, i486™)

0F C0 /r XADD r/m8,r8

0F C1 /r XADD r/m16,r16 (XADD r/m32,r32)

src + dest → temp

dest → src

temp → dest

CMPXCHG dest,srcr (Compare and Exchange, i486™)

0F B0 /r CMPXCHG r/m8,r8

0F B1 /r CMPXCHG r/m16,r16 (CMPXCHG r/m32,r32)

if accumulator = dest then begin

1 → ZF

src → dest

end

else begin

0 → ZF

dest → accumulator

end;

CMPXCHG8B m64 (Compare and Exchange 8 Bytes, Pentium®)

0F C7 /1 CMPXCHG8B m64

if EDX:EAX = dest then begin

1 → ZF

ECX:EBX → dest

end

else begin

0 → ZF

dest → EDX:EAX

end;

Instrukce binární aritmetiky (mění příznaky!)

- **ADD**
- **ADC**
- **SUB**
- **SBB**
- **IMUL**
- **MUL**
- **IDIV**
- **DIV**
- **INC**
- **DEC**
- **NEG**
- **CMP**

ADD, ADC, SUB, SBB

ADD dest,src

(Integer Add)

04 i8	ADD AL, imm8	
05 i8	ADD AX, imm16	(ADD EAX, imm32)
80 /0 i8	ADD r/m8,imm8	
81 /0 i8	ADD r/m16,imm16	(ADD r/m32,imm32)
83 /0 i8	ADD r/m16,imm8	(ADD r/m32,imm8) *
00 /r	ADD r/m8,r8	
01 /r	ADD r/m16,r16	(ADD r/m32,r32)
02 /r	ADD r8,r/m8	
03 /r	ADD r16,r/m16	(ADD r32,r/m32)

dest + src → dest

dest + (sign-extend of src) → dest *

Nastavuje příznaky OF, SF, ZF, AF, PF a CF (podle výsledku).

ADC dest,src

(Add with Carry)

14 i8	ADC AL, imm8	
15 i8	ADC AX, imm16	(ADC EAX, imm32)
80 /2 i8	ADC r/m8, imm8	
81 /2 i8	ADC r/m16,imm16	(ADC r/m32,imm32)
83 /2 i8	ADC r/m16,imm8	(ADC r/m32,imm8) *
10 /r	ADC r/m8,r8	
11 /r	ADC r/m16,r16	(ADC r/m32,r32)
12 /r	ADC r8,r/m8	
13 /r	ADC r16,r/m16	(ADC r32,r/m32)

dest + src + CF → dest

dest + (sign-extend of src) + CF → dest *

Nastavuje příznaky OF, SF, ZF, AF, PF a CF (podle výsledku).

SUB dest,src

(Subtract)

2C i8	SUB AL, imm8	
2D i8	SUB AX, imm16	(SUB EAX, imm32)
80 /5 i8	SUB r/m8, imm8	
81 /5 i8	SUB r/m16,imm16	(SUB r/m32,imm32)
83 /5 i8	SUB r/m16,imm8	(SUB r/m32,imm8) *
28 /r	SUB r/m8,r8	
29 /r	SUB r/m16,r16	(SUB r/m32,r32)
2A /r	SUB r8,r/m8	
2B /r	SUB r16,r/m16	(SUB r32,r/m32)

dest – src → dest

dest – (sign-extend of src) → dest *

Nastavuje příznaky OF, SF, ZF, AF, PF a CF (podle výsledku).

SBB dest,src

(Subtract with Borrow)

1C i8	SBB AL, imm8	
1D i8	SBB AX, imm16	(SBB EAX, imm32)
80 /3 i8	SBB r/m8, imm8	
81 /3 i8	SBB r/m16,imm16	(SBB r/m32,imm32)
83 /3 i8	SBB r/m16,imm8	(SBB r/m32,imm8) *
18 /r	SBB r/m8,r8	
19 /r	SBB r/m16,r16	(SBB r/m32,r32)
1A /r	SBB r8,r/m8	
1B /r	SBB r16,r/m16	(SBB r32,r/m32)

$\text{dest} - (\text{src} + \text{CF}) \rightarrow \text{dest}$

$\text{dest} - (\text{sign-extend of src} + \text{CF}) \rightarrow \text{dest}$ *

Nastavuje příznaky OF, SF, ZF, AF, PF a CF (podle výsledku).

Příklady použití instrukcí ADD (totéž platí pro ADC, SUB, SBB):

```
add al,45
```

```
adc ax,-542
```

```
add eax,2000
```

```
add [alpha], byte 8
```

```
add byte [alpha],8
```

```
add word [beta], -798
```

```
add dword [gamma], -1
```

```
add al,bl
```

```
add [beta],cx
```

```
add di,cx
```

```
add sp,[beta]
```

```
add ebp,esi
```

```
add [gamma],eax
```

ADD dest,src	(Integer Add)
ADC dest,src	(Add with Carry)
SUB dest,src	(Subtract)
SBB dest,src	(Subtract with Borrow)

Stručný přehled jednotlivých možností:

1. ADD/ADC/SUB/SBB reg,reg
2. ADD/ADC/SUB/SBB reg,mem
3. ADD/ADC/SUB/SBB mem,reg
4. ADD/ADC/SUB/SBB reg,imm
5. ADD/ADC/SUB/SBB mem,imm

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

add al,bl

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 1 0 0 0 1 1 0	70	70

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

adc al,bl

+ čísla bez znaménka
± čísla se znaménkem

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 1 0 0 0 1 1 0	70	70

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

adc al,bl

+ čísla bez znaménka
± čísla se znaménkem

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 1 0 0 0 1 1 1	71	71

OF	SF	ZF	AF	PF	CF
0	0	0	1	1	0

add al,bl

		+	±
AL	0 1 1 1 1 0 1 0	122	122
BL	0 1 0 0 1 1 0 1	77	77
AL	1 1 0 0 0 1 1 1	199	-57

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
1	1	0	1	0	0

add al,bl

		+	±
AL	1 1 1 1 1 1 1 1	255	-1
BL	1 1 1 1 1 1 0 1	253	-3
AL	1 1 1 1 1 1 0 0	252	-4

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
0	1	0	1	1	1

add al,bl

		+	±
AL	1 1 1 1 1 1 1 1	255	-1
BL	0 0 0 0 0 0 0 1	1	1
AL	0 0 0 0 0 0 0 0	0	0

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
0	0	1	1	1	1

sub al,bl

+ čísla bez znaménka
 ± čísla se znaménkem

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 0 1 0 1 1 0 0	44	44

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

sbb al,bl

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 0 1 0 1 1 0 0	44	44

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

sbb al,bl

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13
AL	0 0 1 0 1 0 1 1	43	43

+ čísla bez znaménka
 ± čísla se znaménkem

OF	SF	ZF	AF	PF	CF
0	0	0	1	1	0

sub al,bl

+ čísla bez znaménka
 ± čísla se znaménkem

		+	±								
AL	<table><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	0	1	1	1	1	0	1	0	122	122
0	1	1	1	1	0	1	0				
BL	<table><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	1	0	1	1	0	0	1	1	179	-77
1	0	1	1	0	0	1	1				
AL	<table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	0	0	0	1	1	1	199	-57
1	1	0	0	0	1	1	1				

OF	SF	ZF	AF	PF	CF
1	1	0	0	0	1

sub al,bl

+ čísla bez znaménka
 ± čísla se znaménkem

		+	±
AL	0 0 0 0 0 0 0 0 1	1	1
BL	0 0 0 0 0 0 0 0 1	1	1
AL	0 0 0 0 0 0 0 0 0	0	0

OF	SF	ZF	AF	PF	CF
0	0	1	0	1	0

Struktura programu v NASM (v IAS)


```
%include "rw32.inc"

[segment .data use32]
    ; data

[segment .code use32]
prologue

    ; program

epilogue
```

Soubor **rw32.inc**, který se vloží do uživatelského programu direktivou **%include**, obsahuje definici maker **prologue** a **epilogue** a funkce pro čtení z klávesnice a výpis na obrazovku.

Čtení z klávesnice:

ReadChar	znak do AL
ReadUInt8	bezznaménkové 8bitové číslo do AL
ReadInt8	znaménkové 8bitové číslo do AL
ReadUInt16	bezznaménkové 16bitové číslo do AX
ReadInt16	znaménkové 16bitové číslo do AX
ReadUInt32	bezznaménkové 32bitové číslo do EAX
ReadInt32	znaménkové 32bitové číslo do EAX
ReadFloat	reálné 32bitové číslo do EAX
ReadDouble	reálné 64bitové číslo do ST0 (FPU)
ReadString	řetěz znaků na adresu předem uloženou v EDI, maximální počet znaků je v registru EBX, řetězec je ukončen nulou (hodnota 0)
Příklad volání:	

```
call ReadUInt8
mov edi, muj_skvely_retezec
mov ebx, 10
call ReadString
```

Výpis na obrazovku:

WriteChar	znak z AL
WriteNewLine	vypíše konec řádku (odskočí na další řádek)
WriteBin8	8bitové číslo z AL v binární formě
WriteUInt8	bezznaménkové 8bitové číslo z AL
WriteInt8	znaménkové 8bitové číslo z AL
WriteBin16	16bitové číslo z AX v binární formě
WriteUInt16	bezznaménkové 16bitové číslo z AX
WriteInt16	znaménkové 16bitové číslo z AX
WriteBin32	32bitové číslo z EAX v binární formě
WriteUInt32	bezznaménkové 32bitové číslo z EAX
WriteInt32	znaménkové 32bitové číslo z EAX
WriteString	řetěz znaků z paměti počínaje adresou, která je uložena v ESI
WriteFlags	obsah příznakového registru
WriteFloat	reálné 32bitové číslo z EAX
WriteDouble	reálné 64bitové číslo z ST(0)

Některé direktivy použitelné v datovém segmentu:

[jméno[:]] db imm8[,imm8[,...]]
[jméno[:]] db 'řetěz znaků'
[jméno[:]] dw imm16[,imm16[,...]]
[jméno[:]] dd imm32[,imm32[,...]]
[jméno[:]] resb n
[jméno[:]] resw n
[jméno[:]] resd n

Příklady:

alfa db 20
beta dw -28, 198
pole resb 500
zprava db 'Vse OK'

Příklad programu (přičte desítku k zadanému bezznaménkovému číslu bez kontroly přetečení a nové číslo zobrazí):

```
%include "rw.asm"
segment code
prologue
    call ReadUInt8
    call WriteNewLine
    add  al,10
    call WriteUInt8
    call WriteNewLine
epilogue
```

IMUL, MUL, IDIV, DIV

MUL

(Unsigned Multiply)

F6 /4	MUL r/m8	$AL * r/m8 \rightarrow AX$
F7 /4	MUL r/m16	$AX * r/m16 \rightarrow DX:AX$
F7 /4	MUL r/m32	$EAX * r/m32 \rightarrow EDX:EAX$

Pokud lze součin uložit do paměťového prostoru daného velikostí implicitního operandu (AL, AX, EAX), tak ($0 \rightarrow CF$, $0 \rightarrow OF$), jinak ($1 \rightarrow CF$, $1 \rightarrow OF$). Hodnoty příznaků SF, ZF, AF a PF nejsou definovány.

Příklady použití:

mul bl	mul byte [bx]
mul dx	mul word [bx+di+65]
mul ebx	mul dword [gamma]

MUL src

(Unsigned Multiply)

Stručný přehled jednotlivých možností:

1. MUL reg ... 8/16/32
2. MUL mem ... 8/16/32

IMUL

(Signed Multiply)

F6 /5 IMUL r/m8

 $AL * r/m8 \rightarrow AX$

F7 /5 IMUL r/m16

 $AX * r/m16 \rightarrow DX:AX$

F7 /5 IMUL r/m32

 $EAX * r/m32 \rightarrow EDX:EAX$

Pokud lze součin uložit do paměťového prostoru daného velikostí implicitního operandu (AL, AX, EAX), tak ($0 \rightarrow CF$, $0 \rightarrow OF$), jinak ($1 \rightarrow CF$, $1 \rightarrow OF$). Hodnoty příznaků SF, ZF, AF a PF nejsou definovány.

Příklady použití:

imul bl

imul byte [bx]

imul dx

imul word [beta]

imul ebx

imul dword [bx+si+5]

IMUL

(Signed Multiply)

0F AF /r IMUL r16,r/m16

 $r16 * r/m16 \rightarrow r16$

0F AF /r IMUL r32,r/m32

 $r32 * r/m32 \rightarrow r32$

Pokud lze součin uložit do paměťového prostoru daného velikostí cílového operandu, tak ($0 \rightarrow CF$, $0 \rightarrow OF$), jinak ($1 \rightarrow CF$, $1 \rightarrow OF$). Hodnoty příznaků SF, ZF, AF a PF nejsou definovány.

Příklady použití:

```
imul  bx,dx
```

```
imul  dx,word [beta]
```

```
imul  ebx,ecx
```

```
imul  eax,dword [gamma+bx]
```

IMUL

(Signed Multiply)

6B /r i8	IMUL r16,r/m16,imm8	$r/m16 * \text{sign-ext. imm8} \rightarrow r16$
6B /r i8	IMUL r32,r/m32,imm8	$r/m32 * \text{sign-ext. imm8} \rightarrow r32$
6B /r i8	IMUL r16,imm8	$r16 * \text{sign-ext. imm8} \rightarrow r16$
6B /r i8	IMUL r32,imm8	$r32 * \text{sign-ext. imm8} \rightarrow r32$

Pokud lze součin uložit do paměťového prostoru daného velikostí cílového operandu, tak ($0 \rightarrow \text{CF}$, $0 \rightarrow \text{OF}$), jinak ($1 \rightarrow \text{CF}$, $1 \rightarrow \text{OF}$). Hodnoty příznaků SF, ZF, AF a PF nejsou definovány.

Příklady použití:

```
imul bx,ax, byte 5
imul ecx,edx, byte 10
imul cx, byte 10
imul edx, byte 10
```

IMUL (Signed Multiply)

69 /r i8	IMUL r16,r/m16,imm16	$r/m16 * imm16 \rightarrow r16$
69 /r i8	IMUL r32,r/m32,imm32	$r/m32 * imm32 \rightarrow r32$
69 /r i8	IMUL r16,imm16	$r/m16 * imm16 \rightarrow r16$
69 /r i8	IMUL r32,imm32	$r/m32 * imm32 \rightarrow r32$

Pokud lze součin uložit do paměťového prostoru daného velikostí cílového operandu, tak ($0 \rightarrow CF$, $0 \rightarrow OF$), jinak ($1 \rightarrow CF$, $1 \rightarrow OF$). Hodnoty příznaků SF, ZF, AF a PF nejsou definovány.

Příklady použití:

```
imul bx,ax, word 5
```

```
imul bx,ax,5
```

```
imul ecx,ebx,5
```

```
imul cx,10      =  imul cx,cx,10
```

```
imul ecx,10     =  imul ecx,ecx,10
```

IMUL src (Signed Multiply)
IMUL dest,src
IMUL dest,src1,src2

Stručný přehled jednotlivých možností:

- | | |
|------------------------------|-------------|
| 1. IMUL reg | ... 8/16/32 |
| 2. IMUL mem | ... 8/16/32 |
| 3. IMUL reg-dest,reg-src | ... 16/32 |
| 4. IMUL reg,mem | ... 16/32 |
| 5. IMUL reg,imm8 | ... 16/32 |
| 6. IMUL reg,imm | ... 16/32 |
| 7. IMUL reg-dest,reg-src,imm | ... 16/32 |
| 8. IMUL reg,mem,imm | ... 16/32 |

Pro dva nebo tři operandy platí, že musí mít stejnou velikost (16/32 bitů), výjimkou je přímý operand imm8.

imul bl / mul bl

		AX	AL
AX	x x x x x x x x 0 0 0 1 0 1 0 0	?	20
BX	x x x x x x x x 0 0 0 0 0 1 0 1	?	5
AX	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0	100	100

OF	SF	ZF	AF	PF	CF
0	?	?	?	?	0

imul bl

		AX	AL																	
AX	<table><tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	x	x	x	x	x	x	x	x	x	0	0	0	1	1	0	1	1	?	27
x	x	x	x	x	x	x	x	x	0	0	0	1	1	0	1	1				
BX	<table><tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	x	x	x	x	x	x	x	x	x	0	0	0	0	0	1	0	1	?	5
x	x	x	x	x	x	x	x	x	0	0	0	0	0	1	0	1				
AX	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	135	-121
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1				

OF	SF	ZF	AF	PF	CF
x	x	x	x	x	x

mul bl

		AX	AL
AX	x x x x x x x x 0 0 0 1 1 0 1 1	?	27
BX	x x x x x x x x 0 0 0 0 0 1 0 1	?	5
AX	0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1	135	135

OF	SF	ZF	AF	PF	CF
0	?	?	?	?	0

imul bl

		AX	AL																
AX	<table><tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	x	x	x	x	x	x	x	x	1	1	1	1	1	0	1	1	?	-5
x	x	x	x	x	x	x	x	1	1	1	1	1	0	1	1				
BX	<table><tr><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	x	x	x	x	x	x	x	x	0	0	0	0	0	1	0	1	?	5
x	x	x	x	x	x	x	x	0	0	0	0	0	1	0	1				
AX	<table><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	-25	-25
1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1				

OF	SF	ZF	AF	PF	CF
0	x	x	x	x	0

mul bl

		AX	AL
AX	<div> <div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>1</div><div>1</div><div>1</div><div>1</div><div>1</div><div>0</div><div>1</div><div>1</div> </div>	?	251
BX	<div> <div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>x</div><div>0</div><div>0</div><div>0</div><div>0</div><div>0</div><div>0</div><div>1</div><div>0</div><div>1</div> </div>	?	5
AX	<div> <div>0</div><div>0</div><div>0</div><div>0</div><div>0</div><div>1</div><div>0</div><div>0</div><div>1</div><div>1</div><div>1</div><div>0</div><div>0</div><div>1</div><div>1</div><div>1</div> </div>	1255	231

OF	SF	ZF	AF	PF	CF
1	1	1	1	1	1

IDIV

(Signed Divide)

F6 /7	IDIV r/m8	$AX \text{ div } r/m8 \rightarrow AL$ $AX \text{ mod } r/m8 \rightarrow AH$
F7 /7	IDIV r/m16	$DX:AX \text{ div } r/m16 \rightarrow AX$ $DX:AX \text{ mod } r/m16 \rightarrow DX$
F7 /7	IDIV r/m32	$EDX:EAX \text{ div } r/m32 \rightarrow EAX$ $EDX:EAX \text{ mod } r/m32 \rightarrow EDX$

Pokud podíl nelze uložit do paměťového prostoru daného velikostí cílového operandu generuje se výjimka 0 – chyba dělení.

Hodnoty příznaků CF, OF, SF, ZF, AF a PF nejsou definovány.

Příklady použití:

idiv bl

idiv byte [bx]

idiv bx

idiv word [bp+5]

idiv edx

idiv dword [gamma+bx+si]

IDIV src

(Signed Divide)

Stručný přehled jednotlivých možností:

1. IDIV reg ... 8/16/32
2. IDIV mem ... 8/16/32

DIV

(Unsigned Divide)

F6 /6

DIV r/m8

AX div r/m8 \rightarrow ALAX mod r/m8 \rightarrow AH

F7 /6

DIV r/m16

DX:AX div r/m16 \rightarrow AXDX:AX mod r/m16 \rightarrow DX

F7 /6

DIV r/m32

EDX:EAX div r/m32 \rightarrow EAXEDX:EAX mod r/m32 \rightarrow EDX

Pokud podíl nelze uložit do paměťového prostoru daného velikostí cílového operandu generuje se výjimka 0 – chyba dělení.

Hodnoty příznaků CF, OF, SF, ZF, AF a PF nejsou definovány.

Příklady použití:

div bl

div byte [bx]

div bx

div word [bp+5]

div edx

div dword [gamma+bx+si]

DIV src

(Unsigned Divide)

Stručný přehled jednotlivých možností:

1. DIV reg ... 8/16/32
2. DIV mem ... 8/16/32

idiv bl / div bl

		X/H	L
AX	0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1	917	-121
BX	x x x x x x x x 0 0 0 0 1 0 1 0	?	10
AX	0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1	7	91

OF	SF	ZF	AF	PF	CF
⌘	⌘	⌘	⌘	⌘	⌘

idiv bl

		X/H	L
AX	1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0	-246	10
BX	x x x x x x x x 0 1 1 0 0 1 0 0	?	100
AX	1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0	-46	-2

OF	SF	ZF	AF	PF	CF
⌘	⌘	⌘	⌘	⌘	⌘

div bl

		X/H	L
AX	1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0	65290	10
BX	x x x x x x x x 0 1 1 0 0 1 0 0	?	100
AX	? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	??	??

Divide overflow

OF	SF	ZF	AF	PF	CF
⌘	⌘	⌘	⌘	⌘	⌘

U celočíselného dělení čísel se znaménkem se podíl a zbytek vyhodnocují takto:

a	b	$a \operatorname{div} b$	$a \operatorname{mod} b$
114	5	22	4
-114	5	-22	-4
114	-5	-22	4
-114	-5	22	-4

INC, DEC

INC dest (Increment)

FE /0	INC r/m8	
FF /0	INC r/m16	(INC r/m32)
40+r	INC r16	(INC r32)

$\text{dest} + 1 \rightarrow \text{dest}$

Nastavuje příznaky OF, SF, ZF, AF, PF (podle výsledku). **Nemění příznak CF!**

Příklady použití:

inc al	inc byte [bx]
inc dx	inc word [bp+2]
inc esp	inc dword [bx+si]

DEC dest (Decrement)

FE /1	DEC r/m8	
FF /1	DEC r/m16	(DEC r/m32)
48+r	DEC r16	(DEC r32)

$\text{dest} - 1 \rightarrow \text{dest}$

Nastavuje příznaky OF, SF, ZF, AF, PF (podle výsledku). **Nemění příznak CF!**

Příklady použití:

dec al	dec byte [bx]
dec dx	dec word [bp+2]
dec esp	dec dword [bx+si]

INC dest	(Increment)
DEC dest	(Decrement)

Stručný přehled jednotlivých možností:

- | | |
|----------------|-------------|
| 1. INC/DEC reg | ... 8/16/32 |
| 2. INC/DEC mem | ... 8/16/32 |

inc bl

BL

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

OF

SF

ZF

AF

PF






CF

0	0	0	0	0	X
---	---	---	---	---	---

dec cl

CL

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PF	CF
					x

NEG

NEG dest (Two's Complement Negation)

F6 /3 NEG r/m8

F7 /3 NEG r/m16 (NEG r/m32)

if dest = 0 then 0 → CF else 1 → CF

– dest → dest

Dále nastavuje příznaky OF, SF, ZF, AF a PF (podle výsledku).

Příklady použití:

neg bl

neg ax

neg edi

neg byte [bx+si]

neg word [bp+2]

neg dword [gamma]

NEG dest (Two's Complement Negation)

Stručný přehled jednotlivých možností:

- | | |
|------------|-------------|
| 1. NEG reg | ... 8/16/32 |
| 2. NEG mem | ... 8/16/32 |

neg dl

DL

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PF	CF
0	1	0	1	1	1

neg bl

BL

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	1

neg cl

CL

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PF	CF
0	0	1	0	1	0

neg ah

AH

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

OF	SF	ZF	AF	PF	CF
⌘	⌘	0	0	0	⌘

CMP

CMP dest,src

(Compare Two Operands)

3C i8	CMP AL, imm8	
3D i8	CMP AX, imm16	(CMP EAX, imm32)
80 /7 i8	CMP r/m8, imm8	
81 /7 i8	CMP r/m16,imm16	(CMP r/m32,imm32)
83 /7 i8	CMP r/m16,imm8	(CMP r/m32,imm8) *
38 /r	CMP r/m8,r8	
39 /r	CMP r/m16,r16	(CMP r/m32,r32)
3A /r	CMP r8,r/m8	
3B /r	CMP r16,r/m16	(CMP r32,r/m32)

dest – src → temp

dest – (sign-extend of src) → temp *

Nastavuje příznaky OF, SF, ZF, AF, PF a CF (podle výsledku).

Instrukce CMP pracuje podobně jako instrukce SUB. Výsledek operace však neukládá, nastavuje pouze příznaky!!!

Příklady použití:

```
cmp al,5  
cmp bx,12afh  
cmp bl,cl  
cmp si,dx  
cmp ax,[alpha]  
cmp eax,edx
```

CMP dest,src

(Compare Two Operands)

Stručný přehled jednotlivých možností:

1. CMP reg1,reg2
2. CMP reg,mem
3. CMP mem,reg
4. CMP reg,imm
5. CMP mem,imm

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

cmp al,bl

		+	±
AL	0 0 1 1 1 0 0 1	57	57
BL	0 0 0 0 1 1 0 1	13	13

OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

cmp bl,cl

		+	±
BL	0 1 1 1 1 0 1 0	122	122
CL	1 0 1 1 0 0 1 1	179	-77

OF	SF	ZF	AF	PF	CF
1	1	0	0	0	1

cmp al,ah

		+	±
AL	0 0 0 0 1 0 1 0	10	10
AH	1 1 1 1 1 0 1 0	250	-6

OF	SF	ZF	AF	PF	CF
0	0	0	0	0	1

cmp al,ah

		+	±
AL	1 0 0 0 1 0 1 0	138	-118
AH	0 0 0 0 1 1 1 0	14	14

OF	SF	ZF	AF	PF	CF
1	0	0	1	0	0