4. Architektura procesoru - přenosy, aritmetické a logické instrukce

Instrukční soubor procesorů IA-32

- Celočíselné instrukce
  - přenosové instrukce
  - aritmetické instrukce (binární aritmetika)
  - logické instrukce
  - instrukce posuvů a rotací
  - bitové a slabikové instrukce
  - instrukce předávání řízení
  - instrukce ovládání příznaků
  - instrukce pro práci se segmentovými registry
  - řetězové instrukce
  - instrukce dekadické aritmetiky
- Instrukce FPU
- Instrukce MMX, SSE, SSE2, SSE3, SSSE3, AVX, …
- Instrukce systémové
- Další instrukce

Celočíselné instrukce

Tučně zeleně jsou označeny instrukce důležité pro ISU
Normálně jsou označeny dalsí užitečné instrukce
Červeně jsou označeny instrukce, které nebudeme v ISU používat

Zelené a *normální* budou na zkoušce!!!

Pro účely předmětu ISU bude za implicitní velikost operandů brána hodnota 32 bitů! Taktéž adresování bude implicitně 32bitové.
=> instrukce s 16bitovými operandy musí mít prefix 0x66

Přenosové instrukce (nemění příznaky)

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
MOV	**PUSH/POP** **PUSHF/POPF** **PUSHA/POPA** **PUSHAD/POPAD** **CBW/CWDE** **CWD/CDQ** MOVSX/MOVZX XCHG CMOV*cc*	IN OUT BSWAP XADD CMPXCHG CMPXCHG8B …

Instrukce pro práci se segmentovými registry

☯	☹	☺ … zajímavé
		LDS/LES/LFS/LGS/LSS

Aritmetické instrukce (mění příznaky)

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
ADD **SUB** **MUL/IMUL** **DIV/IDIV** **CMP**	**ADC** **SBB** **INC** **DEC** **NEG**	Instrukce dekadické aritmetiky: DAA, DAS AAA, AAS, AAM, AAD

Logické instrukce (mění příznaky)

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
AND **OR** **XOR** **NOT** **TEST**		

Instrukce ovládání příznaků

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
STC/CLC/CMC **STD/CLD**	LAHF/SAHF	STI/CLI

Instrukce posuvů a rotací

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
SHR/SHL,SAR/SAL **ROR/ROL,RCR/RCL**		SHRD/SHLD

Ostatní instrukce

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
	LEA NOP XLAT/XLATB ENTER/LEAVE	UD2 CPUID

Instrukce pro práci s bity

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
	BT/BTS/BTR/BTC BSF/BSR	SETcc

Řetězové instrukce

☯ … životně důležité	☹	☺ … zajímavé
REP/REPE/REPZ/REPNE/REPNZ MOVS,CMPS,SCAS LODS/STOS		INS/OUTS

Instrukce pro předávání řízení

☯ … životně důležité	☹ … musím znát	☺ … zajímavé
JMP,Jcc,LOOPcc CALL,RET	INT,IRET	INT3,INTO

MOV

přenosová instrukce

MOV r/m,reg

Instrukce	Kód (32bitový režim)	prefix	opcode	ModR/M			SIB	d	imm
MOV r/m8,r8	88 /r	---	10001000	mod	r8	r/m8	?	?	---
MOV r/m16,r16	66 89 /r	01100110	10001001	mod	r16	r/m16	?	?	---
MOV r/m32,r32	89 /r	---	10001001	mod	r32	r/m32	?	?	---

r/m = reg … zkopíruje obsah registru reg do paměti nebo do jiného registru r/m.

MOV AL,BL	88 D8	10001000	11011000		
MOV SP,BP	66 89 EC	01100110	10001001	11101100	
MOV ESP,EBP	89 EC	10001001	11101100		
MOV [p1],CL	88 0D p1	10001000	00001101	p1(d32)	
MOV [p1],CX	66 89 0D p1	01100110	10001000	00001101	p1(d32)
MOV [EBX],AL	88 03	10001000	00000011		
MOV EDX,ESP	89 E2	10001001	11100010		
MOV [p2],EBP	89 2D p2	10001001	00101101	p2(d32)	
MOV [EBP+ESI*2+7],EDI	89 7C 75 07	10001001	01111100	01110101	00000111
MOV [ESP],EDI					
MOV [ESP+nic*1],EDI	89 3C 24	10001001	00111100	00100100	
MOV [ESP+nic*2],EDI	89 3C 64	10001001	00111100	01100100	

MOV reg,r/m

Instrukce	Kód (32bitový režim)	prefix	opcode	ModR/M			SIB	d	imm
MOV r8,r/m8	8A /r	---	10001010	mod	r8	r/m8	?	?	---
MOV r16,r/m16	66 8B /r	01100110	10001011	mod	r16	r/m16	?	?	---
MOV r32,r/m32	8B /r	---	10001011	mod	r32	r/m32	?	?	---

reg = r/m … zkopíruje obsah paměti nebo jiného registru r/m do registru reg.

MOV AL,BL	8A C3	10001010	11000011		
MOV SP,BP	66 8B E5	01100110	10001011	11100101	
MOV ESP,EBP	8B EC	10001011	11101100		
MOV CL,[p1]	8A 0D p1	10001010	00001101	p1(d32)	
MOV CX,[p1]	66 8B 0D p1	01100110	10001011	00001101	p1(d32)
MOV AL,[EBX]	8A 03	10001010	00000011		
MOV EDX,ESP	8B D4	10001011	11010100		
MOV EBP,[p2]	8B 2D p2	10001011	00101101	p2(d32)	
MOV EDI,[EBP+ESI*2+7]	8B 7C 75 07	10001011	01111100	01110101	00000111
MOV EDI,[ESP]					
MOV EDI,[ESP+nic*1]	8B 3C 24	10001011	00111100	00100100	
MOV EDI,[ESP+nic*2]	8B 3C 64	10001011	00111100	01100100	

MOV[1] r/m,sreg
MOV[2] sreg,r/m

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

Instrukce	Kód (32bitový režim)	prefix	opcode	ModR/M			SIB	d	imm
MOV r16,sreg	66 8C /sreg	01100110	10001100	mod	sreg	r16	?	?	---
MOV r/m32,sreg	8C /sreg	---	10001100	mod	sreg	r/m32	?	?	---
MOV sreg,r16	66 8E /sreg	01100110	10001100	mod	sreg	r16	?	?	---
MOV sreg,r/m32	8E /sreg	---	10001100	mod	sreg	r/m32	?	?	---

1) r/m = sreg … zkopíruje obsah segmentového registru sreg do paměti nebo do jiného 16/32bitového registru r/m (nuluje horních 16 b).

2) sreg = r/m … zkopíruje 16 bitů z paměti nebo z 16/32bitového registru r/m (dolních 16 bitů) do segmentového registru sreg.

MOV CX,CS	66 8C C9	01100110	10001100	11001001	
MOV EAX,SS	8C D0	10001001	11010000		
MOV [p1],FS	8C 25 p1	10001100	00100101	p1(d32)	
MOV [EBP+ESI*2+7],CS	8C 4C 75 07	10001100	01001100	01110101	00000111
MOV ES,CX	66 8E C1	01100110	10001100	11000001	
MOV SS,EAX	8E D0	10001001	11010000		
MOV FS,[p1]	8E 25 p1	10001100	00100101	p1(d32)	
~~MOV CS,[EBP+ESI*2+7]~~	~~8E 4C 75 07~~	~~10001100~~	~~01001100~~	~~01110101~~	~~00000111~~

MOV[1] AL/AX/EAX,moffs
MOV[2] moffs,AL/AX/EAX

Instrukce	Kód (32bitový režim)	prefix	opcode	ModR/M			SIB	d	imm
MOV AL,moffs	A0	---	10100000	--	---	---	-	d32	---
MOV AX,moffs	66 A1	01100110	10100001	--	---	---	-	d32	---
MOV EAX,moffs	A1	---	10100001	--	---	---	-	d32	---
MOV moffs,AL	A2	---	10100010	--	---	---	-	d32	---
MOV moffs,AX	66 A3	01100110	10100011	--	---	---	-	d32	---
MOV moffs,EAX	A3	---	10100011	--	---	---	-	d32	---

1) AL/AX/EAX = [moffs] … zkopíruje 8/16/32 bitů z (efektivní) adresy moffs do registru AL/AX/EAX.

2) [moffs] = AL/AX/EAX … zkopíruje obsah registru AL/AX/EAX do paměti na (efektivní) adresu moffs.

MOV AL,[0x11223344]	A0 44 33 22 11	10100000 01000100 00110011 00100010 00010001
MOV AX,[p1]	66 A1 p1	01100110 10100001 p1(d32)
MOV EAX,[0x99AABBCC]	A1 CC BB AA 99	10100001 11001100 10111011 10101010 10011001
MOV [0x11223344],AL	A2 44 33 22 11	10100010 01000100 00110011 00100010 00010001
MOV [p2],AX	66 A3 p2	01100110 10100011 p2(d32)
MOV [0x99AABBCC],EAX	A3 CC BB AA 99	10100011 11001100 10111011 10101010 10011001

MOV[1] reg,imm
MOV[2] r/m,imm

Instrukce	Kód (32bitový režim)	prefix	opcode	ModR/M			SIB	d	imm
MOV reg8,imm8	B0+rb	---	10110???	--	---	---	-	---	imm8
MOV reg16,imm16	66 B8+rw	01100110	10111???	--	---	---	-	---	imm16
MOV reg32,imm32	B8+rd	---	10111???	--	---	---	-	---	imm32
MOV r/m8,imm8	C6 /0	---	11000110	mod	000	r/m8	?	?	imm8
MOV r/m16,imm16	66 C7 /0	01100110	11000111	mod	000	r/m16	?	?	imm16
MOV r/m32,imm32	C7 /0	---	11000111	mod	000	r/m32	?	?	imm32

1) reg = imm ... zkopíruje konstantu imm do registru reg.

2) r/m = imm ... zkopíruje konstantu imm do registru nebo do paměti r/m.

MOV AL,-1	B0 FF	10110000 11111111
MOV CX,0x1122	66 B9 22 11	01100110 10111001 00100010 00010001
MOV EAX,0x11223344	B8 44 33 22 11	10111000 01000100 00110011 00100010 00010001
MOV EAX,0x11223344	C7 44332211	11000111 11000000 $(44\ 33\ 22\ 11)_{16}$
MOV [EAX],byte 0x12	C6 00 12	11000110 00000000 00010010
MOV [EBP+ESI*2+7], byte 0x12	C6 44 75 12	11000110 01000100 01110101 00010010

MOV [EDI+ESI*8+0xFFEEDDCC],dword 0x11223344

C7 84 F7 CC DD EE FF 44 33 22 11

11000111 10000100 11110111 11001100 11011101 11101110 11111111 01001000 00110011 00100010 00010001

MOV – příklady (hodnoty jsou uvedeny hexadecimálně)

```
EAX = 00 00 93 19
ECX = 10 20 F7 1C
   MOV EAX,ECX
EAX = 10 20 F7 1C
```

```
ESP = 84 21 07 BB
EDX = 10 20 F7 1C
   MOV SP,DX
ESP = 84 21 F7 1C
```

```
EBX = 31 C8 FF FD
EDX = 00 00 1E 81
   MOV BL,DH
EBX = 31 C8 FF 1E
```

```
EAX = 00 00 93 19
ECX = 10 20 F7 1C
   MOV AH,CL
EAX = 00 00 1C 19
```

```
EAX = 00 00 93 19
p2  = 00 00 10 00
   byte [p2+3] = 64
   MOV [p2+3],AH
   byte [p2+3] = 93
```

```
EAX = 00 00 93 19
p2  = 00 00 10 00
word [p2] = 00 00 66 65
   MOV [p2],AX
word [p2] = 00 00 93 19
```

```
EBX = 00 00 01 00
EAX = 00 00 05 00
  dword [p1] = FFFFFFFF
   MOV [EBX],EAX
dword [p1] = 00 00 05 00
```

	adresa *adr*	mem[*adr*]	
	0xFFFFFFFF	67	67
	0xFFFFFFFE	66	66
	0xFFFFFFFD	65	65
	0xFFFFFFFC	64	64
p2 + 3	0x00001003	64	93
p2 + 2	0x00001002	FF	FF
p2 + 1	0x00001001	66	93
p2	0x00001000	65	19
p1 + 3	0x00000103	FF	00
p1 + 2	0x00000102	FF	00
p1 + 1	0x00000101	FF	05
p1	0x00000100	FF	00
	0x00000003	45	45
	0x00000002	25	25
	0x00000001	FF	FF
	0x00000000	12	12

MOV dst,src

(*Move Source to Destination*)

```
1. MOV reg,reg
2. MOV (s)reg,mem
3. MOV mem,(s)reg
4. MOV reg,imm
5. MOV mem,imm
6. MOV r16/r32,sreg
7. MOV sreg,r16/r32    …      (sreg != CS)
```

Oba operandy musí mít stejnou velikost (8, 16, nebo 32 bitů), kromě případu přesunu segmentového registru (č. 6 a 7).

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
r8 ∈ {AL,BL,CL,DL,AH,BH,CH,DH}, r16 ∈ {AX,BX,CX,DX,SI,DI,BP,SP}
r32 ∈ {EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}, sreg ∈ {CS,DS,ES,FS,GS,SS}
(s)reg = reg ∪ sreg, mem = paměť, imm = přímý operand (konstanta)

Znaménkové rozšíření × rozšíření nulami

(*Sign Extension* × *Zero Extension*)

Zero Extend 8 b -> 16 b	0001 1001 -> 0000 0000 0001 1001 25 -> 25
Zero Extend 16 b -> 32 b	1101 1011 0101 1001 -> 0000 0000 0000 0000 1101 1011 0101 1001 56 153 -> 56 153
Zero Extend 8 b -> 16 b	1111 1111 -> 0000 0000 1111 1111 -1 (255) -> 255
Sign Extend 8 b -> 16 b	0011 1001 -> 0000 0000 0011 1001 +57 -> +57
Sign Extend 16 b -> 32 b	1101 1011 0101 1001 -> 1111 1111 1111 1111 1101 1011 0101 1001 -9 383 -> -9 383
Sign Extend 8 b -> 16 b	1111 1111 -> 1111 1111 1111 1111 -1 -> -1
Sign Extend 16 b -> 32 b	0000 0001 0101 0001 -> 0000 0000 0000 0000 0000 0001 0101 0001 +337 -> +337

ADD, SUB
ADC, SBB
aritmetické instrukce

mění příznaky:	OF	SF	ZF	AF	CF	PF

ADD dst,src (*Add*)
ADC dst,src (*Add with carry*)
SUB dst,src (*Subtract*)
SBB dst,src (*Integer subtraction with borrow*)

Oba operandy musí mít stejnou velikost (8, 16, nebo 32 bitů), kromě případu č. 6, kdy je 8bitová konstanta příslušně znaménkově rozšířena na 16 nebo 32 bitů.

Mění příznaky OF, SF, ZF, AF, PF a CF podle výsledku instrukce.

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
6. ADD/ADC/SUB/SBB reg16/reg32,imm8

```
ADD dst,src
dst = dst + SignExtend(src);
```

```
SUB dst,src
dst = dst – SignExtend(src);
```

```
ADC dst,src
dst = dst + SignExtend(src) + CF;
```

```
SBB dst,src
dst = dst – SignExtend(src) – CF;
```

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
r16 ∈ {AX,BX,CX,DX,SI,DI,BP,SP}, r32 ∈ {EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť, imm = přímý operand (konstanta)

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

```
ADD AL,0x45          04 45
ADD AX,-1         66 05 FF FF

ADD AL,BL         … atd.
ADD [p2],CX
ADD DI,CX
ADD SP,[p2]
ADD EBP,ESI
ADD EAX,-2000
ADD [p3],EAX
ADD [p1], byte 8
ADD byte [p1],8
ADD word [p2], -798
ADD dword [p3], -1
```

AL = 0010 1101 45 +45 0x2D

0000 1101 100 +100 0x0D

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?

ADD AL,100

AL = 1001 0001 145 -111 0x91

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

AL = 0010 1101 45 +45 0x2D

CL = 1111 1111 255 -1 0xFF

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?

ADD AL,CL

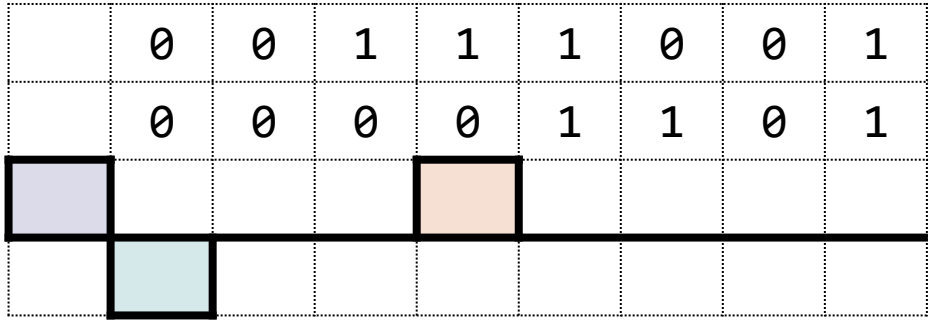
AL = 0010 1100 44 +44 0x2C

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

AL = 0011 1001 57 +57 0x39

BL = 0000 1101 13 +13 0x0D

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?



ADD AL,BL

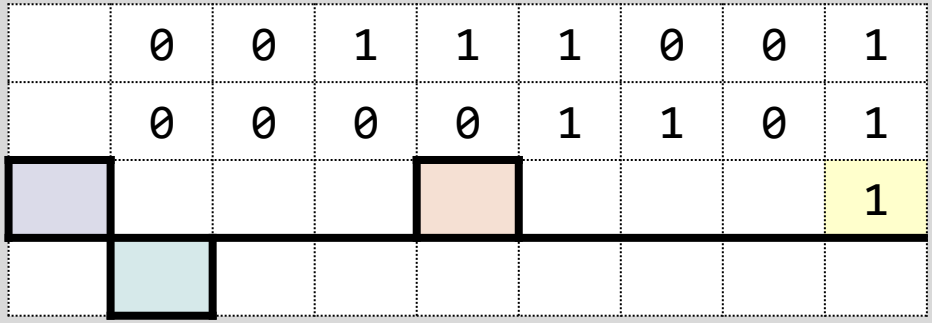
AL = 0100 0110 70 +70 0x50

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

AL = 0011 1001 57 +57 0x39

BL = 0000 1101 13 +13 0x0D

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	1



ADC AL,BL

AL = 0100 0111 71 +71 0x47

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

AL = 0111 1010 122 +122 0x7A

BL = 0100 1101 77 +77 0x4D

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?

	0	1	1	1	1	0	1	0
	0	1	0	0	1	1	0	1

ADD AL,BL

AL = 1100 0111 199 -57 0xC7

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

AL = 1111 1111 255 -1 0xFF

BL = 1111 1101 253 -3 0xFD

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?

	0	0	1	1	1	0	0	1
	0	0	0	0	1	1	0	1
								1

ADD AL,BL

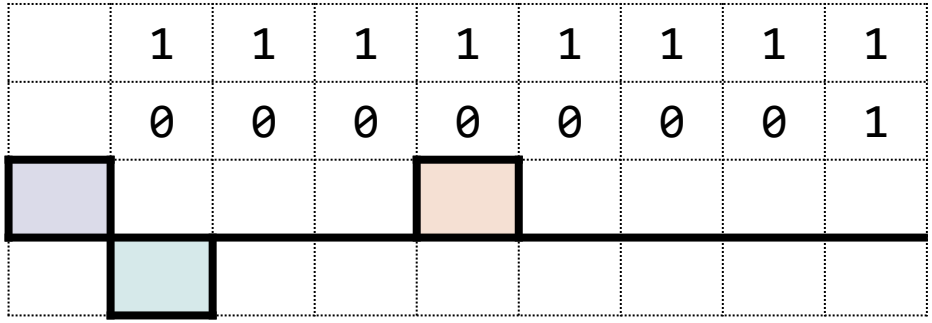
AL = 1111 1100 252 -4 0xFC

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

AL = 1111 1111 255 -1 0xFF

BL = 0000 0001 1 +1 0x01

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?



ADD AL,BL

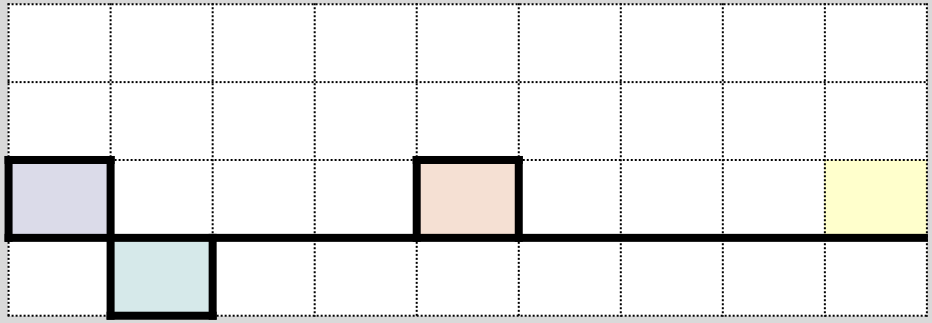
AL = 0000 0000 0 0 0x00

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

AL =

BL =

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?



ADD AL,BL

AL =

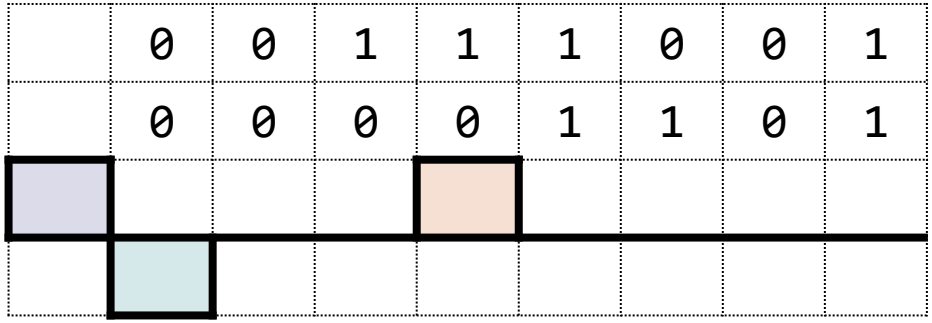
FLAGS =	OF	SF	ZF	AF	PF	CF

AL = 0011 1001 57 +57 0x39

BL = 0000 1101 13 +13 0x0D

FLAGS =

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



SUB AL,BL

AL = 0010 1100 44 +44 0x2C

FLAGS =

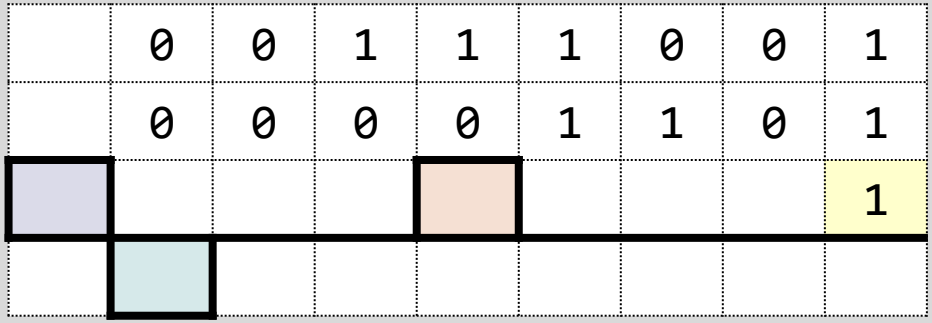
OF	SF	ZF	AF	PF	CF
0	0	0	1	0	0

AL = 0011 1001 57 +57 0x2D

BL = 0000 1101 13 +13 0x0D

FLAGS =

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



SBB AL,BL

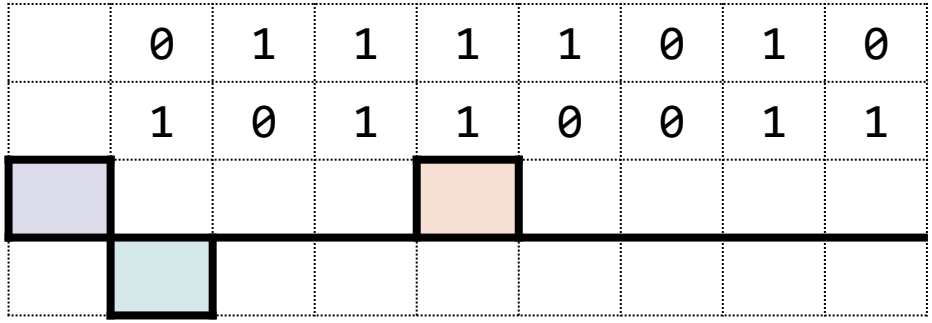
AL = 0100 1011 43 +43 0x2B

FLAGS =

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

AL =	0111 1010	122	+122	0x7A	
BL =	1011 0011	179	-77	0xB3	

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?



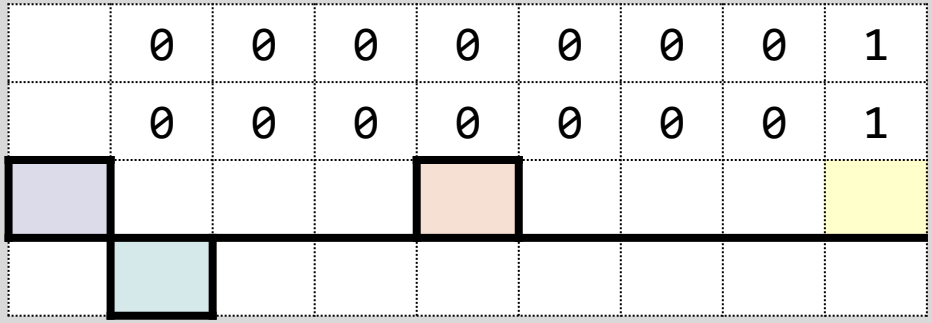
SUB AL,BL

AL =	1100 0111	199	-57	0xC7

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

AL =	0000 0001	1	+1	0x01	
BL =	0000 0001	1	+1	0x01	

FLAGS =	OF	SF	ZF	AF	PF	CF
	?	?	?	?	?	?



SUB AL,BL

AL =	0000 0000	0	0	0x00

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

MUL, IMUL
aritmetické instrukce

mění příznaky:	OF	SF	ZF	AF	CF	PF

MUL src (*Unsigned multiply*)
IMUL src (*Signed multiply*)
IMUL dst,src (*Signed multiply*)
IMUL dst,src1,src2 (*Signed multiply*)

1. IMUL/MUL reg … výsledek v AX/DX:AX/EDX:EAX
2. IMUL/MUL mem
3. IMUL reg16/32,mem16/32 … výsledek v cílovém registru
4. IMUL reg16/32,reg16/32
5. IMUL reg16/32,mem16/32,imm8
6. IMUL reg16/32,reg16/32,imm16/32
7. ~~MUL reg,mem~~ … **NELZE**
8. ~~IMUL/MUL reg,imm~~ … **NELZE**

reg ∈
{AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,
SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
r16 ∈ {AX,BX,CX,DX,SI,DI,BP,SP}
r32 ∈ {EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť, imm = přímý operand (konstanta)

V případech č. 3, 4 a 6 musí mít operandy stejnou velikost (16 nebo 32 bitů), v případě č. 5 je 8bitová konstanta příslušně znaménkově rozšířena na 16 nebo 32 bitů.

Mění příznaky OF a CF podle výsledku instrukce, ostatní příznaky mají nedefinovanou hodnotu.

MUL/IMUL src

MUL src

```
if (sizeof(src) == 8) {
  AX = AL*src; flag = (AH != 0);
}  else if (sizeof(src) == 16) {
  DX:AX = AX*src; flag = (DX != 0);
} else {
  EDX:EAX = EAX*src; flag = (EDX != 0);
}
CF = OF = flag; //SF, ZF, AF, PF = ?
```

IMUL src

```
if (sizeof(src) == 8) {
  AX = AL*src;
  flag = SignExtend16(AL) != AX;
}  else if (sizeof(src) == 16) {
  DX:AX = AX*src;
  flag = SignExtend32(AX) != DX:AX;
} else {
  EDX:EAX = EAX*src; flag = EDX;
  flag = SignExtend64(EAX) != EDX:EAX;
}
CF = OF = flag; //SF, ZF, AF, PF = ?
```

IMUL dst,src

IMUL dst,src1,src2

IMUL dst,src

```
dst = dst*src;
tmp = dst*src;
// sizeof(tmp) = 2*sizeof(dst)
CF = OF = tmp != dst;
// SF, ZF, AF, PF = ?
```

IMUL dst,src1,src2

```
dst = src1*SignExtend(src2);
tmp = src1* SignExtend(src2);
// sizeof(tmp) = 2*sizeof(src1)
CF = OF = tmp != dst;
// SF, ZF, AF, PF = ?
```

MUL,IMUL - příklady

```
MUL BL …         AX =AL*BL     CF=OF=(AH!=0)
MUL CX …      DX:AX =AX*CX     CF=OF=(DX!=0)
MUL EDI …   EDX:EAX=EAX*EDI   CF=OF=(EDX!=0)
MUL byte [pB] … AX=AL*BL      CF=OF=(AH!=0)
MUL word [pW] … DX:AX=AX*CX   CF=OF=(DX!=0)

analogicky pro IMUL, podmínku pro IMUL lze zapsat například i takto:
   (byte)0 - sign( AL) == AH
   (word)0 - sign( AX) == DX
  (dword)0 - sign(EAX) == EDX
```

```
~~IMUL BL,CL~~ …              NELZE
IMUL CX,[pW] …       CX = CX*m16[pW] t16H:t16L = CX*m16[pW], CF=OF=(t16L != CX)
IMUL ECX,EAX …      ECX = ECX*EAX    t32H:t32L = ECX*EAX, CF=OF=(t32L != ECX)
IMUL ECX,EAX,5 …    ECX = EAX*5      t32H:t32L = EAX*5, CF=OF=(t32L != ECX)
IMUL EDI,EBX,-10 … EDI = EBX*-10     t32H:t32L = EBC*-10, CF=OF=(t32L != EDI)
IMUL BX,AX,17 …      BX = AX*17      t16H:t16L = AX*17, CF=OF=(t16L != BX)
IMUL ECX,dword [pD],7 … ECX =[pD]*7 t32H:t32L = [pD]*7, CF=OF=(t32L != ECX)
```

AX = ???? ???? 0001 0100 AL = 20

BX = ???? ???? 0000 0101 BL = 5

FLAGS =

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

MUL BL (IMUL BL)

AX = 0000 0000 0110 0100 AX = 100

FLAGS =

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

	_H								_L							
AX	?	?	?	?	?	?	?	?	0	0	0	1	0	1	0	0
BX	?	?	?	?	?	?	?	?	0	0	0	0	0	1	0	1
AX	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
SignExtend16(AL)	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0

AX = ???? ???? 0001 1011 AL = 27

BX = ???? ???? 0000 0101 BL = 5

FLAGS =

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

MUL BL

AX = 0000 0000 1000 0111 135 (AL = 135)

FLAGS =

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

	_H								_L							
AX	?	?	?	?	?	?	?	?	0	0	0	1	1	0	1	1
BX	?	?	?	?	?	?	?	?	0	0	0	0	0	1	0	1
AX	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
SignExtend16(AL)	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1

AX = ???? ???? 0001 1011 AL = 27

BX = ???? ???? 0000 0101 BL = 5

FLAGS =

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

IMUL BL

AX = 0000 0000 1000 0111 135 (AL = -121)

FLAGS =

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

	_H								_L							
AX	?	?	?	?	?	?	?	?	0	0	0	1	1	0	1	1
BX	?	?	?	?	?	?	?	?	0	0	0	0	0	1	0	1
AX	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
SignExtend16(AL)	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1

AX = ???? ???? 1111 1011 AL = 251

BX = ???? ???? 0000 0101 BL = 5

FLAGS =

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

MUL BL

AX = 0000 0100 1110 0111 1255 (AL = 231)

FLAGS =

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

	_H								_L							
AX	?	?	?	?	?	?	?	?	1	1	1	1	1	0	1	1
BX	?	?	?	?	?	?	?	?	0	0	0	0	0	1	0	1
AX	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1
SignExtend16(AL)	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1

AX = ???? ???? 1111 1011 AL = -5

BX = ???? ???? 0000 0101 BL = 5

FLAGS =

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

IMUL BL

AX = 1111 1111 1110 0111 -25 (AL = -25)

FLAGS =

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

	_H								_L							
AX	?	?	?	?	?	?	?	?	1	1	1	1	1	0	1	1
BX	?	?	?	?	?	?	?	?	0	0	0	0	0	1	0	1
AX	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
SignExtend16(AL)	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1

DIV, IDIV
aritmetické instrukce

(ne?)mění příznaky:	OF	SF	ZF	AF	CF	PF

IDIV/DIV src

1. IDIV/DIV reg
2. IDIV/DIV mem

... podíl v AL/AX/EAX a zbytek v AH/DX/EDX

Hodnoty příznaků nejsou definovány. Při dělení nulou nebo v případě, kdy se podíl „nevejde“ do cílového registru, je generována výjimka „*Divide Error*“ (#DE).

(I)DIV BL …	AL=AX/BL
	AH=AX%BL
(I)DIV CX …	AX=DX:AX/CX
	DX=DX:AX%CX
(I)DIV EDI …	EAX=EDX:EAX/EDI
	EDX=EDX:EAX%EDI
(I)DIV byte [pB] …	AL=AX/[pB]
	AH=AX%[pB]
(I)DIV word [pW] …	AX=DX:AX/[pW]
	DX=DX:AX%[pW]

```
if (src == 0) throw(#DE);
if (sizeof(src) == 8) {
  tmp = AX / src;
  if (tmp > 0xFF) throw(#DE); else {
  if (tmp > 0x7F || tmp < 0x80) throw(#DE); else {
    AL = tmp; AH = AX % src;
  }
} else if (sizeof(src) == 16) {
 tmp = DX:AX / src;
 if (tmp > 0xFFFF) throw(#DE); else {
 if (tmp > 0x7FFF || tmp < 0x8000) throw(#DE); else {
    AX = tmp; DX = DX:AX % src;
  }
} else {
 tmp = EDX:EAX / src;
 if (tmp > 0xFFFFFFFF) throw(#DE); else {
 if(tmp>0x7FFFFFFF||tmp<0x80000000) throw(#DE);else {
    EAX = tmp; EDX = EDX:EAX % src;
  }
}
```

reg ∈ {AL, BL, CL, DL, AH, BH, CH, DH, AX, BX, CX, DX, SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}

mem = paměť

Připomenutí – podíl a zbytek

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

a	**b**	**d** = a / b	**m** = a % b
114	5	22	4
-114	5	-22	-4
114	-5	-22	4
-114	-5	22	-4

Platí: **a = d*b + m**

```
AX  =   0000 0011 1001 0101        AX = 917
BL  =              0000 1010       BL =  10
```

DIV BL (IDIV BL)

```
AL  =              0101 1011     podíl: AL = 91
AH  =              0000 0111     zbytek: AH = 7
```

```
AX  =   1111 1111 0000 1010        AX = -250
BL  =              0110 0100       BL =  100
```

IDIV BL

```
AL  =              1111 1110     podíl: AL = -2
AH  =              1101 0010    zbytek: AH = -50
```

```
AX  =   0000 0011 1001 0101        AX = 917
BL  =              0000 0000       BL =  0
```

DIV CL (IDIV CL)

„Divide Error“ (#DE)

```
AL  =              ???? ????     podíl: AL = ?
AH  =              ???? ????     zbytek: AH = ?
```

```
AX  =   1111 1111 0000 1010       AX = 65 290
BL  =              0110 0100      BL =  100
```

DIV BL

„Divide Error“ (#DE)

```
tmp =        0010 1000 1100 = 652 > 255 (0xFF)
AL  =              ???? ????     podíl: AL = ?
AH  =              ???? ????     zbytek: AH = ?
```

CBW, CWDE, CWD, CDQ
přenosové aritmetické instrukce

CBW
AX = SignExtend(AL) … 8 b -> 16 b

CWD
DX:AX = SignExtend(AX) … 16 b -> 2×16 b

CWDE
EAX = SignExtend(AX) … 16 b -> 32 b

CDQ
EDX:EAX = SignExtend(EAX) … 32 b -> 2×32 b

```
EAX =  FA 35 11 FF
        CBW
EAX =  FA 35 FF FF
```

```
EAX =  FA 35 11 FF
        CWDE
EAX =  00 00 11 FF
```

```
EAX =  FA 35 F1 FF
        CWDE
EAX =  FF FF F1 FF
```

```
EDX =  10 DF 95 00
EAX =  FA 35 11 FF
        CWD
EDX =  10 DF 00 00
EAX =  FA 35 11 FF
```

```
EDX =  10 DF 95 00
EAX =  FA 35 11 FF
        CDQ
EDX =  FF FF FF FF
EAX =  FA 35 11 FF
```

```
EDX =  10 DF 95 00
EAX =  FA 35 81 FF
        CWD
EDX =  10 DF FF FF
EAX =  FA 35 81 FF
```

```
EDX =  10 DF 95 00
EAX =  0A 35 11 FF
        CDQ
EDX =  00 00 00 00
EAX =  0A 35 11 FF
```

MOVZX, MOVSX
přenosové instrukce

MOVZX dst,src

(*Move with zero-extend*)

1. MOVZX r16/r32,r8
2. MOVZX r32,r16
3. MOVZX r16/r32,mem8
4. MOVZX r32,mem16

MOVZX dst,src

```
dst = ZeroExtend(src)
```

```
EDX = 10 DF 95 05
EAX = FA 35 11 FF
   MOVZX AX,DL
EDX = 10 DF 95 05
EAX = FA 35 00 05
```

```
EDX = 10 DF 95 85
EAX = FA 35 11 FF
   MOVZX EAX,DL
EDX = 10 DF 95 85
EAX = 00 00 00 85
```

MOVSX dst,src

(*Move with sign-extension*)

1. MOVSX r16/r32,r8
2. MOVSX r32,r16
3. MOVSX r16/r32,mem8
4. MOVSX r32,mem16

MOVSX dst,src

```
dst = SignExtend(src)
```

```
EDX = 10 DF 95 05
EAX = FA 35 11 FF
   MOVSX AX,DL
EDX = 10 DF 95 05
EAX = FA 35 00 05
```

```
EDX = 10 DF 95 85
EAX = FA 35 11 FF
   MOVSX EAX,DL
EDX = 10 DF 95 85
EAX = FF FF FF 85
```

r8 ∈ {AL,BL,CL,DL,AH,BH,CH,DH}, r16 ∈ {AX,BX,CX,DX,SI,DI,BP,SP}
r32 ∈ {EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť

INC, DEC, NEG
aritmetické instrukce

1. INC/DEC/NEG reg
2. INC/DEC/NEG mem

mění příznaky:	OF	SF	ZF	AF	CF	PF

INC dst

```
dst = dst + 1;
CF se nemění, ostatní dle výsledku
```

DEC dst

```
dst = dst - 1;
CF se nemění, ostatní dle výsledku
```

NEG dst

```
dst = -dst;
CF = (dst != 0); ostatní dle výsledku
```

```
INC BL          DEC BL
INC CX          DEC CX
INC EDI         DEC EDI
INC byte [pB]    …
INC word [pW]
NEG EAX
NEG dword [pD]
NEG CH
```

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť

XCHG
přenosová instrukce

1. XCHG reg,reg
2. XCHG reg,mem
3. XCHG mem,reg

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

```
XCHG dst,src
tmp = dst;
dst = src;
src = tmp;
```

```
XCHG EAX,ECX
XCHG BX,AX
XCHG [EBP-8],EDX
XCHG BL,DH
```

```
EDX = 10 DF 95 85
EAX = FA 35 11 FF
    XCHG EAX,EDX
EDX = FA 35 11 FF
EAX = 10 DF 95 85
```

```
EDX = 10 DF 95 05
EAX = FA 35 11 FF
    XCHG AL,DL
EDX = 10 DF 95 FF
EAX = FA 35 00 05
```

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť

AND, OR, XOR
logické instrukce

mění příznaky:	OF	SF	ZF	AF	CF	PF

AND dst,src (*Logical AND*)
OR dst,src (*Logical inclusive OR*)
XOR dst,src (*Logical exclusive OR*)

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

Nastavuje OF=CF=0, příznaky ZF, SF a PF nastaví podle výsledku instrukce a příznak AF není definován.

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

```
AND dst,src
dst = dst & src;
```

```
OR dst,src
dst = dst | src;
```

```
XOR dst,src
dst = dst ^ src;
```

		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť, imm = přímý operand (konstanta)

NOT
logická instrukce

1. NOT reg
2. NOT mem

nemění příznaky

```
NOT dst
dst = ~dst;
nemění příznaky
```

```
NOT EAX
NOT dword [pD]
NOT CH
```

reg ∈ {AL,BL,CL,DL,AH,BH,CH,DH,AX,BX,CX,DX,SI,DI,BP,SP,EAX,EBX,ECX,EDX,ESI,EDI,ESP,EBP}
mem = paměť

TEST

logická instrukce

mění příznaky:	OF	SF	ZF	AF	CF	PF

```
1. TEST reg,reg
2. TEST reg,mem
3. TEST mem,reg
4. TEST reg,imm
5. TEST mem,imm
```

```
TEST src1,src2
tmp = src1 & src2;
- nastaví příznaky EFLAGS podle
  výsledku „tmp“
- nemění obsah registrů
```

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

Nastavuje OF=CF=0, příznaky ZF, SF a PF nastaví podle výsledku instrukce a příznak AF není definován.

```
TEST AL,5
TEST BL,CL
TEST AX,[p1]
TEST EAX,EDX
TEST [p2],byte 10
```

CMP
aritmetická instrukce

mění příznaky:	OF	SF	ZF	AF	CF	PF

```
1. CMP reg,reg
2. CMP reg,mem
3. CMP mem,reg
4. CMP reg,imm
5. CMP mem,imm
6. CMP reg16/reg32,imm8
```

```
CMP src1,src2
tmp = src1 – SignExtend(src2);
- nastaví příznaky EFLAGS podle
  výsledku „tmp“
- nemění obsah registrů
```

```
CMP AL,5
CMP BL,CL
CMP AX,[p1]
CMP EAX,EDX
CMP [p2],byte 10
```

Oba operandy musí mít stejnou velikost (8, 16, nebo 32 bitů), kromě případu č. 6, kdy je 8bitová konstanta příslušně znaménkově rozšířena na 16 nebo 32 bitů.

Mění příznaky OF, SF, ZF, AF, PF a CF podle výsledku instrukce.