MOVZX - zero extend
MOVSX - sign extend
} vždy do registru, menší → větší
zdroj i paměť

ADD - podle výsledku mění CF, OF, SF, ZF, AF, PF
ADC = dst + SignExt(src) + CF
SBB = (dst - src - CF) = dst - (src + CF) ← první přičte CF k src

MUL: AX = AL · src8
DX:AX = AX · src16
EDX:EAX = EAX · src32
unsigned multiplication!
mění CF, OF, pokud to šahne vyš (DX!=0, EDX!=0, AH!=0)

IMUL src8/16/32: stejně jako MUL,
CF=OF= pokud vrchní půlka != znaménko spodní (samé 0/1)
IMUL reg16/32 (dst), r/m16/32 (src)
dst = dst * src CF/OF se nastaví, pokud se výsl. nevejde
IMUL reg16/32 (dst), r/m16/32 (src1), imm8/16/32 (imm)
dst = src1 * SignExt(imm)

DIV/IDIV reg/mem - vždy operuje nad (D):A
AL = AX / src8
AH = AX % src8
AX = (DX:AX) / src16
DX = (DX:AX) % src16
EAX = (EDX:EAX) / src32
EDX = (EDX:EAX) % src32

při dělení d=a/b platí a = d·b + m
m = a%

INC, DEC - nemění CF! ostatní (včetně OF) ano!
NEG - CF je 1, pokud výsledek není 0 (neg = not & add 1)
CMP - nastaví FLAGS podle src1 - src2 (jako SUB src1, src2)

CBW: AX = SignExt(AL)
Convert Byte to Word
CWD: DX:AX = SignExt(AX)
Convert Word to Doubleword
CWDE: EAX = SignExt(AX)
Convert Word to Doubleword in EAX
CDQ: EDX:EAX = SignExt(EAX)
Convert Doubleword to Quadword

XCHG: prohodí src a dest (může být reg/reg nebo reg/mem)

AND, OR, XOR - nastaví OF=CF=0!, ZF, SF, PF podle výsledku
TEST - nastaví ZF, SF, PF podle src1 & src2

Jcc: dá se skákat podle:

CF	JC	JNC
ZF	JZ	JNZ
SF	JS	JNS
OF	JO	JNO
PF	JP/JPE (parity even)	JNP/JPO (parity odd)

+ JCXZ / JECXZ

CMOVcc: pouze reg16/32, r/m16/32

Bez znaménka:
Equal ⇔ JZ
Above ⇔ CF=0 ∧ ZF=0
Below ⇔ CF=1
AE ⇔ CF=0
BE ⇔ CF=1 ∨ ZF=1

Se znaménkem:
Equal
Greater ZF=0 ∧ SF=OF
Less SF≠OF
GE SF=OF
LE ZF=1 ∨ SF≠OF

LOOP: skočí ⇔ ECX != 0
LOOP(N)E: skočí ⇔ ECX!=0 ∧ ZF==1 (N: ZF==0)

STC Set Carry | STD Set Direct. F. | CLC Clear Carry | CLD Clear DF | CMC Complement Carry (toggle)

LAHF - load FLAGS to AH SAHF - store AH to FLAGS

SHL == SAL r/m (CL/imm8) - nastaví ZF, SF, PF + do CF stříží poslední posunutý bit
SHift Left / Shift Aritm. Left

SHR r/m (CL | imm8) — || — SAR r/m (CL | imm8) - posouvá doprava a vlevo doplňuje 0 nebo 1 podle původního MSb
SHift Right (logical)

ROL, ROR - CF má hodnotu posledního bitu přicházejícího na druhou stranu

RCL, RCR - registr se chová, jako by měl navíc / napravo navíc 1 bit - carry
Rotate through Carry R/L

MOVSx - přesouvá [ESI] → [EDI] a přidá k ESI a EDI -1/1 (podle DF) • velikost x, nemění

CMPSx - porovná [ESI] - [EDI] — || — (nastavuje FLAGS)

SCASx - čte z [EDI] a porovnává s AL/AX/EAX: A - [EDI], — || — (pouze k EDI), nastavuje (E)FLAGS
SCan String

LODSx - čte z [ESI] a ukládá do — || — : [ESI] → A, — || —, nemění př.
LOaD String

STOSx - ukládá AL/AX/EAX do [EDI]
STOre String

Prefixy:

REP
dokud ECX != 0
snižuje ECX

REPE
rep while Equal
REPZ
rep while Zero
dokud ECX != 0 ∧ ZF == 1

REPNE
rep while Not Equal
REPNZ
rep while Not Zero
dokud ECX != 0 ∧ ZF == 0

používá se s CMPSx a SCASx

nezapomeň na STD/CLD! (odzadu)/(dopředu)

BT - BTS / BTR / BTC r/m16/32 (dest), r16/32 | imm16/32 (src)
Bit Test & Set & Reset & Complement

↳ CF = dest[src]

BSF r16/32 (dest), r/m16/32 (src) - v src najde první nenulový bit „zprava" - od LSB jde doleva
Bit Scan Forward
MOV ax, 0x0004
BSF bx, ax => v bx bude hod. 2 - index bitu ($4_{10} = 0100_2$)

BSR — || — - najde první nenulový bit „zleva" - jde od MSb doprava
Bit Scan Reverse ⇒ vrací INDEX ZAČÍNAJÍCÍ V LSB! stejně jako BSF

PUSHA (D) - pushne všechny obecné reg. (A, C, D, B, SP, SI, DI)
POPA (D)
PUSHF (D) - pushne (E)FLAGS
POPF (D)

CALL = PUSH EIP
MOV EIP, nová adresa

(pokud jde o far jump, první PUSH ZeroExt(CS) a nastavuje se i CS)

RET = POP EIP

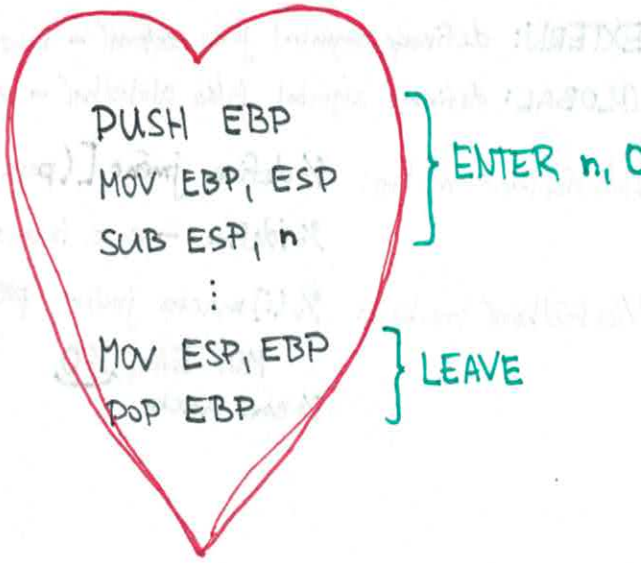
RET imm16 = POP EIP + ADD ESP, n
└ uklízení argumentů (pascal, stdcall, fastcall)

RETF = POP EIP + POP CS
(return from Far jmp)

INT imm8 = PUSHFD + PUSH ZeroExt(CS) + PUSH EIP, nastaví se far adresa ISR přerušení n

INT3 = INT 3 INTO = INT 4

IRET = navrácení se z ISR



pascal convention:
parametry zleva doprava, uklízí volaný

```
fun(a,b,c) => PUSH a
   ↓          PUSH b
RET 12        PUSH c
              CALL fun
```

cdecl conv.:
parametry zprava doleva, uklízí volající

```
fun(a,b) => PUSH b
   ↓        PUSH a
RET         CALL fun
            ADD ESP, 8
```

stdcall conv.:
parametry zprava doleva, uklízí volaný

```
fun(a,b) => PUSH b
   ↓        PUSH a
RET 8       CALL fun
```

fastcall:
první dva par. v ECX a EDX,
zbytek zprava doleva, uklízí volaný

```
fun(a,b,c,d) => MOV ECX, a
     ↓          MOV EDX, b
RET 8           PUSH d
                PUSH c
                CALL fun
```

Pseudoinstrukce NASMu:

Definice inicializovaných dat: DB DW DD DQ DT (80b float) DO (128b) DY (256b) DZ (512

-||- neinicializovaných dat: RESB RESW RESD ...

Externí binární soubor: INCBIN "filename", SKIP, LIMIT

Konstanta: EQU : msg db 'ahoj', 0 msgLen EQU $-msg

Opakování: TIMES : arr TIMES 32 db 0

EXTERN: definuje symbol jako externí – musí jej najít linker

GLOBAL: definuje symbol jako globální – viditelný pro další moduly

Jednořádkové makro: %define jméno [(parametry)]

%idefine – case insensitive

Víceřádkové makro: %(i)macro jméno počet Parametrů
    MOV EAX, %0
%endmacro

} umí spoustu blbůstek

Indexové registry:
- ESP
- EBP
- ESI
- EDI

Segmentové registry: - používá se v 16 bit módu
- CS - kódový segment => CS:[IP]
- DS - datový segment => DS:[si], DS:[var]
- SS - zásobníkový seg. => SS:[SP], SS:[BP]
- ES, FS, GS } extra segmenty

Endianita:

BIG endian = na nejnižší adrese je MSB

little endian = na nejnižší adrese je LSB

v x86 se používá little endian!

adresa	BIG end.	little end.
0x05	0x78	0x12
0x04	0x56	0x34
0x03	0x34	0x56
0x02	0x12	0x78

číslo 0x 12 34 56 78
MSB LSB

Segmentová adresa v 16b režimu: adresovatelný 1 MB => 2^{20} B - chybí 4 bity

= vzdálená adresa **sreg:[offset]**

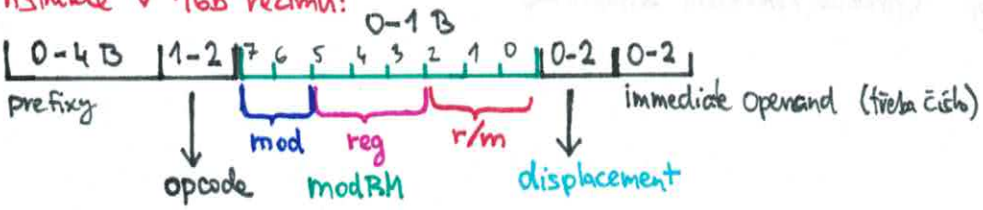
↳ přímá hod. imm = číslo
↳ nepřímá hod. = obsah registru

} **Efektivní adresa EA** = offset vyjádřený konkr. hodnotou z reg.

Výpočet fyzické adresy:

FA = segment · 16 + offset (EA)

Instrukce v 16b režimu:



000	AL	AX	BX+SI *
001	CL	CX	BX+DI
010	DL	DX	BP+SI
011	BL	BX	BP+DI
100	AH	SP	SI
101	CH	BP	DI
110	DH	SI	d16 / BP
111	BH	DI	BX
r/m (reg)	8b	16b	adresa bez displacementu

→ bez operandů: jen opcode, příp. prefixy

→ s jedním operandem:
- v registru -> buď přímo v opcode, nebo v r/m
- v paměti -> podle mod a r/m
  - -> mod = 00 => EA = * (110 => d16)
  - = 01 => EA = * + d8 (110 => BP + d8)
  - = 02 => EA = * + d16 (110 => BP + d16)

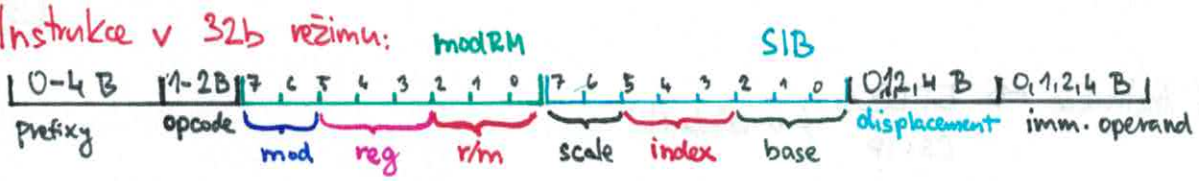
→ se dvěma operandy:
- dva registry → ID registrů v reg a r/m
- registr a paměť → registr v reg, EA

∧ mod = 11!

Efektivní adresa: EA = base + index + displacement
- base: BX, BP, Ø (od zač. segm.)
- index: SI/DI

} operandy mohou být jen kombinace BX, BP, SI, DI => nelze MOV [AX], word

Instrukce v 32b režimu:



r/m reg base	16/32b mod 11	i \ mod 00/01/10	mem 100
000	(E)AX	[EAX]	[EAX + i·2^s]
001	(E)CX	[ECX]	[ECX + i·2^s]
010	(E)DX	[EDX]	[EDX + i·2^s]
011	(E)BX	[EBX]	[EBX + i·2^s]
100	(E)SP	podle SIB nic	[ESP + i·2^s]
101	(E)BP	d32 / [EBP]	[d32 / EBP + i·2^s]
110	(E)SI	[ESI]	[ESI + i·2^s]
111	(E)DI	[EDI]	[EDI + i·2^s]
	*A	*B	*C	*D

index označuje registr podle *A→*B, ale 100 je neplatná hod. ⇒ index nelze přečíst z ESP! i je hodnota v tom reg.

S jedním operandem:
- v registru → buď přímo v opcode, nebo v r/m (ad *A→*B)
- v paměti → podle mod, r/m a příp. SIB
  - → mod = 00 ⇒ EA = *C
    - = 01 ⇒ EA = *C + d8
    - = 10 ⇒ EA = *C + d32
  - → mod = XX ∧ r/m = 100
    - ⇒ adresa se spočítá podle SIB
      - mod = 00 ⇒ EA = *D
      - = 01 ⇒ EA = *D + d8
      - = 10 ⇒ EA = *D + d32

Se dvěma operandy:
- dva registry → podle reg a r/m (*A→*B)
- registr a paměť → registr v reg, EA jako →

Efektivní adresa: $EA = base + index \cdot 2^{scale} + displacement$
(base: *B; index: *B bez ESP)

⇒ ve 32b režimu se dají použít všechny běžné registry: MOV [eax], word 5
pozor na [eax + esp*1] ⇒ invalid effective address

příklad: MOV [EBP + ESI*4 + 7], EDI : ⇒ 89 7C B5 07
(base, index, scale, d8)

MOV = 89

r/m = 100
mod = 01 (potřebuje 8b displacement)
scale = 10
base = 101
index = 110
reg = 111

⇒ 01 111 100 | 10 110 101 | 00000111
7C | B5 | 07

MOV [ESP], EDI : pro ESP nemám *C, musím jít přes mem=100 ∧ index=100

= MOV [ESP + nic*2 (+0)], EDI
(base, index, scale, bez d, reg = 111)

mod = 00 (bez d) ⇒ 00 111 100 XX 100 100
(XX je jedno)

Výjimky a interrupty:

interrupt ⇒ dokončení instrukce ⇒ EFLAGS, CS:IP } → zásobník

⇒ z tabulky přerušení se přečte adresa ISR ⇒ adresa → CS:IP

⇒ provede se ISR ⇒ pop původní CS:IP a EFLAGS ⇒ pokračuje původní prog.

0 - chyba dělení (DIV, IDIV)
1 - ladění
2 - externí nemaskovatelné přerušení
3 - breakpoint
4 - int overflow
5 - překročení hranice
6 - invalid opcode
7 - nedostupné FPU
8 - „dvojitá chyba" - chyba při obsluze chyby / interruptu
12 - stack segment fault
13 - general protection fault
14 - page fault
16 - chyba FPU

EFLAGS:

Carry	CF
Parity	PF
Auxiliary Carry	AF
Zero	ZF
Sign	SF
Direction	DF
OverFlow	OF

FPU tag reg.:

00 – platné nenulové číslo
01 – nula
10 – špatné číslo
11 – volno

FPU status reg.:

Busy	B
Condition Code	C0/1/2
Top	T
Error Summary Status	ES
Stack Fault	SF – sticky (přetečení/podtečení zásobníku)
Precision Exception	PE
Underflow	UE
Overflow	OE
Zero Divide	ZE
Denormalized Op.	DE
Invalid Op.	IE

FPU control reg.:

Rounding Control	nejbližší číslo / $-\infty$ / $+\infty$ / 0
Precision Control	přesnost mantisy: 23 / – / 52 / 64
masky	

Převody čísel b-10 → b-z (pevná řádová čárka)

$\hat{n}$: počet cifer celé části $\hat{m}$: počet cifer necelé části (zadán předem, může jít do nekonečna)

Celá část: vydělím celočíselně → zapíšu zbytek ---> přečtu zbytky odzadu

```
while (n!=0) { a[i] = n % z ; n = n/z ; i++ ; }   n je celá část čísla z základ ; i=0
```

Necelá část: vynásobím základem, co vyleze před tečku, zapíšu, dokud nejsem na m číslicích nebo na nule ---> přečtu zepředu

```
while (m!=0) { x = m * z ; a[] = (int) x ; m = x - a[] ; }   i=-1
```

=> pak $(n.m)_{10} == (a[i-1]\ a[i-2] \dots a[0] . a[-1]\ a[-2] \dots a[1])_z$

Přímý převod: pokud je základ z_1 mocninou základu z_2, jeden znak soustavy z_1 odpovídá N znakům soust. z_2 (kde $z_1 = z_2^N$):

$(3D0)_{16} = \begin{array}{|c|c|c|} 3 & D & 0 \\ 0011 & 1101 & 0000 \end{array} = (001111010000)_2$

Kódování čísel

rozsah: interval všech zobrazitelných čísel rozlišitelnost: nejmenší kladné zobrazitelné číslo

přesnost: počet platných dekadických číslic, které je v daném prostoru možné zobrazit

Pevná řád. čárka: k-bitový prostor s n bity pro celou č. a m bity pro necelou, k = m+n

→ bez znaménka: rozsah $\langle 0; 2^n - 2^{-m} \rangle$, rozliš. 2^{-m}, přesnost $(n+m) \cdot \log_{10} 2$

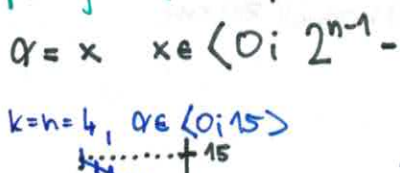
- dnes se používá prakticky pouze pro celá čísla (=> m = 0), rozsah je $\langle 0; 2^n - 1 \rangle$, rozliš. = 1

→ se znaménkem:

Přímý kód: - záporné číslo je pouze označeno hodnotou MSB (1 = záp, 0 = kkd.)

$\alpha = x \quad x \in \langle 0; 2^{n-1} - 2^{-m} \rangle$; $\alpha = 2^{n-1} - x \quad x \in \langle -(2^{n-1} - 2^{-m}); 0 \rangle$

$k = n = 4$, $\alpha \in \langle 0; 15 \rangle$

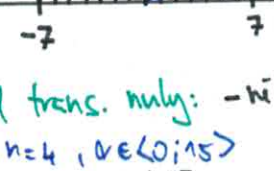


7: 0111 ($\alpha = 7$)
-7: 1111 ($\alpha = 15$)
1: 0001
-1: 1001 ($\alpha = 9$)

- existuje „záporná nula" (1.... 0 vs. 0.... 0)
- nelze provádět aritmetické op. (10 + 01 = 11, -0 + 1 = 3)

Kód trans. nuly: - někam se posune 0, např. doprostřed => $\alpha = 2^{n-1} + x \quad x \in \langle -2^{n-1}; 2^{n-1} - 2^{-m} \rangle$

$k = n = 4$, $\alpha \in \langle 0; 15 \rangle$



-8: $\alpha = 8 + (-8) = 0$ → 0000

7: $\alpha = 8 + 7 = 15$: 1111

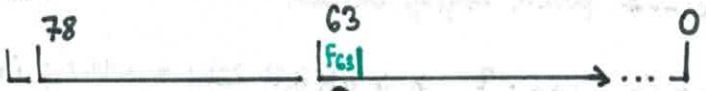
0: $\alpha = 8 + 0 = 8$ → 1000

- kladná čísla a nula mají MSB = 1, záp. 0
- nelze provádět aritmetické op.

FPU

Typy operandů: word int 16b | short int 32b | long int 64b | BCD 80b (18 číslic + sign)

Extended Real (long double) 80b



S | exponent 15 bitů | f mantisa 64 bitů

- u normálních čísel je $f_{63} == 1$ (kromě nuly)

$0 < e < 32767$ $\quad f = 1,x \Rightarrow n = (-1)^s \cdot 2^{e-16383} \cdot 1,x$

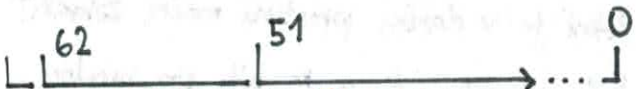
$e = 0$ $\quad f = 0,0 \Rightarrow n = 0$

$e = 0$ $\quad f = 0,x$ ($x \neq 0$) $\Rightarrow n = (-1)^s \cdot 2^{-16382} \cdot 0,x$ denormalizované

$e = 32767$ $\quad f = 1,0 \Rightarrow n = (-1)^s \infty$

$f = 1,x$ $(x \neq 0) \Rightarrow n = NaN$

Long Real (double) 64b



S | e 11 bitů | f 52 bitů + jeden fake bit (podle e), chová se jako f_{63} ↑

$0 < e < 2047$ $\quad f = x \Rightarrow n = (-1)^s \cdot 2^{e-1023} \cdot 1,x$

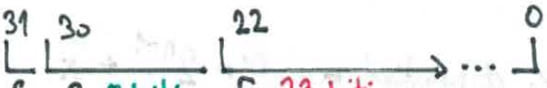
$e = 0$ $\quad f = 0 \Rightarrow n = 0$

$e = 0$ $\quad f = x$ $(x \neq 0) \Rightarrow n = (-1)^s \cdot 2^{-1022} \cdot 0,x$ denormalizované

$e = 2047$ $\quad f = 0 \Rightarrow n = \infty$

$f \neq 0$ $(x \neq 0) \Rightarrow n = NaN$

Short Real (Float) 32b



S | e 8 bitů | f 23 bitů

hraniční $e = 255$ $\quad n = (-1)^s \cdot 2^{e-127} \cdot 1,x$

(denorm: $2^{-126} \cdot 0,x$)

Převod: 1) výchozí číslo běžnou metodou s tečkou: $134,625 \rightarrow 10000110,101$

2) posunout tečku k první jedničce (normalizace): $1,0000110101$ $e' = 7$

3) zjistit e: pro Float $e = 127 + e' = 134 = 10000110$

4) nezapomeň na sign: 0 | 10000110 | 00001101000... 0 — 1 je tam implicitně