

3. fejezet

Az X86 Utasításkészlet

Grafikus Processzorok Tudományos Célú Programozása

- Assembly nyelv:

Bitkódok (gépikód) helyett rövid párbetűs nevek (mnemonic) az utasításoknak és a regisztereknek.

Majdnem 1→1 megfelelés a gépikódnak, könnyű megírni a fordítót:

- Assembler:

Az a program, ami a szöveges assembly nyelvű programot gépikódra fordítja.

X86 Assembly

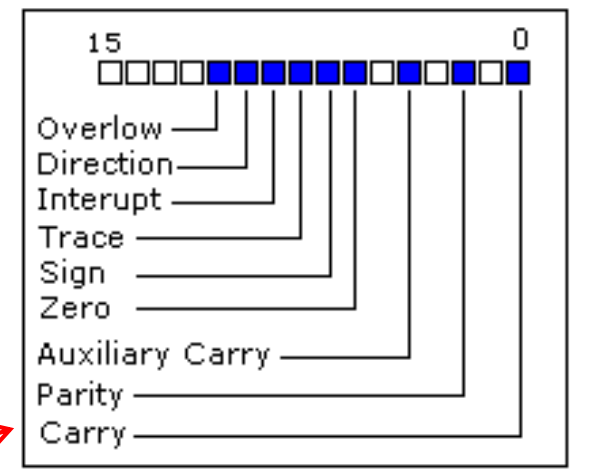
- Regiszterek:
Az ezekben tárolt adatokon végeznek műveleteket az utasítások, a memóriából először ezekbe kell mozgatni az adatokat.
- Az eredeti 16-bites regiszterek:
AX, BX, CX, DX - Általános célú
SP - Stack Pointer
(pl. a verem tetejére mutat)
BP - Base Pointer
(pl. a lokális változókra mutat)

SI - Source Index,
DI - Destination Index

Szegmens regiszterek:

CS - Kódszegmens
DS - Adat szegmens
SS - Stack szegmens
ES - Extra szegmens

Flag regiszter
IP - Instruction pointer



X86 Assembly utasítások

- Fontosabb 8086 Utasítások:

- Adatmozgatás: MOV, XCHG, PUSH, POP, IN, OUT, XLAT
- Logikai: AND, OR, NOT, XOR
- Bitműveletek: SHL, SHR, SAL, SAR, ROL, ROR
- Aritmetika: INC, DEC, ADD, ADC, SUB, SBB, MUL, IMUL, DIV, IDIV, NEG
- Logikai: CMP (kivonás), TEST (logikai ÉS)
- Elágazás: feltétel nélküli: JMP, feltételes (flag-ek alapján): J..
- Ciklus: LOOPE, LOOPNE, LOOPNZ, LOOPZ
- Szubrutin: CALL, RET

Előjeles egészek: komplementerként
ábrázolva!

X86 Assembly utasítások

- 186: Szubrutin: **ENTER**, **LEAVE** Adatmozgatás: **PUSHA**, **POPA**
- 286: főleg task és jogosultság kezelési bővítmények
- 386: 32 bites regiszterek!, kibővített bitműveletek, adatmozgatás, loop, feltételes értékadás
- 486: **BSWAP**: byte-ok cseréje, **CMPXCHG**: atomi összehasonlítás és csere, cache utasítások, **XADD**: csere és összeadás
- Pentium: **CPUID**, Model specifikus segéd/diagnosztikai regiszterek, **RDTSC**: órajelszámláló, **MMX**

X86 Assembly utasítások

- **MMX:**

[Összefoglaló az utasításkészletről](#)

Az x87 processzor regisztereinek újrafelhasználása, random-access regiszterekként. A váltás a kétféle működés között költséges!

Ezeket [SIMD](#) (single instruction, multiple data) működés

Csak Integer műveletek (8x8 bit, 4x16 bit, 2x32 bit, vagy 1x64 bit):

Széles MOV: **movd** (32 bit), **movq** (64 bit)

Logikai műveletek: **PXOR, POR, PAND, PANDN** + **shiftelés**

Aritmetika: összeadás, kivonás, szorzás, összehasonlítás, csomagolás

X86 Assembly utasítások

- Pentium Pro: **CMOV..**: feltételes (flag alapú) adatmozgatás
- **SSE**: 8 új, független, 128 bit széles regiszter, egyszeres lebegőpontos utasítások!

Alignment!

(Az integer utasítások (+, *, min, max, átlag) továbbra is az MMX regisztereken hatnak!)

CPU: szelektív mozgatás, prefetch, fence,
non-temporal (NT) műveletek

SSE utasítások:

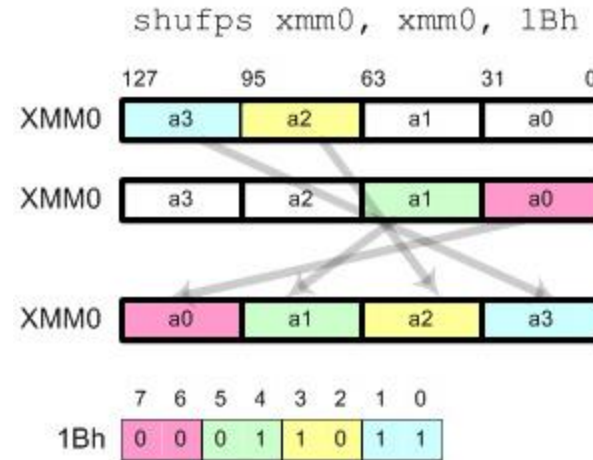
Aritmetika: +, -, *, /, reciprok, négyzetgyök, 1/gyök, max, min, átlag, shuffle, unpack!

Logikai, Összehasonlító műveletek: hasonlóak mint az MMX-nél.

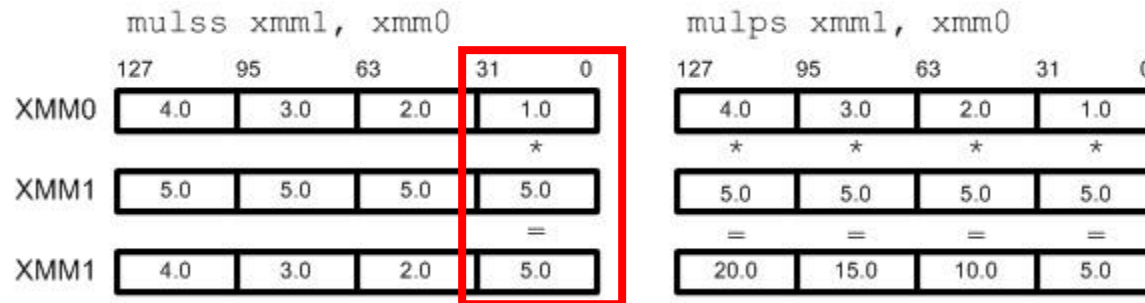
Adatmozgatás, Cache + fence, átalakítások (egész-lebegőpontos).

SSE példák:

- **Shuffle**
a harmadik operandus kódolja az átrendezést



- Szorzás egy értéken, vagy az egész regiszteren:



- **SSE2**: +8 új regiszter, az MMX utasítások kiterjesztése az SSE regiszterekre, 64 bites lebegőpontos műveletek, néhány 128 bites integer művelet.
 - Bővül minden SSE művelet a megfelelő adattípusokra, és kiterjesztik a shuffle-t is.
 - Közelítő reciprok számolás
 - Read, Read/Write fence
 - PAUSE - várakozás egy lock-ra (bővebben a párhuzamosításnál)
 - Még több Non-temporal és cache-el kapcsolatos művelet

X86 Assembly utasítások

- **SSE3**: Vízszintes (egy regiszteren belüli) műveletek (+, -), unaligned move, duplikálás, float→integer konverzió kerekítés nélkül.
- **SSSE3**: még több vízszintes +, - utasítás, abszolút érték, egészekre szorzás-és-összeadás!, in place shuffle, feltételes negálás, bit manipuláció.
- **SSE4**: feltételes másolás, különböző kerekítések, összehasonlítás (string!), konverziók, skaláris szorzat, abszolút eltérés összeg, CRC-ellenőrző összeg számítás, bit számlálás, vezető 0 számlálás
- **AES**: titkosítás lépéseit és kulcsgenerálást támogató utasítások
- **AVX**: regiszter bővítés 256 bitre, 3 operandusú utasítások, de főleg csak adatmozgatás, és shuffle
FMA3/4: fused-multiply-and-add
- **AVX2**: integer műveletek kiterjesztései 256 bitre, beolvasás nem folytonos memóriából, feltételes beolvasás, shift és shuffle variánsok.
- **AVX-512**: 512 bit széles regiszterek, okosabb mask és feltételes műveletek, ciklus kezelési műveletek, közelítő és pontosabb reciprok, 1/gyök, hatvány, számolás.
 - AVX-512 kiterjesztések: Neurális hálókhoz 4 lépéses skaláris szorzat, illetve FMA

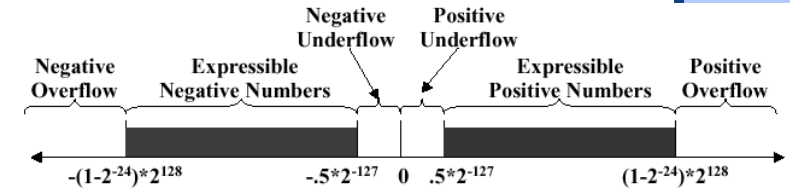
IEEE 754 - Lebegőpontos szabvány

- **1960-1970:** minden lebegőpontos számolás teljesen szoftveres emulációval történt
teljes káosz az ábrázolásokban, teljesen abszurd hibák:
 $1 \cdot x \neq x$, $\frac{x}{x} \neq 1$, $x - y = 0$, *pedig* $x \neq y$ mindez csak bizonyos x, y értékekre!
Eredmény: teljesen hordozhatatlan numerika, a kódok tele voltak workaroundokkal!
- **1976:** Az Intel elkezdett tervezni egy lebegőpontos co-processzort a 8086 mellé. A cél a „legjobb” aritmetika volt!
- **1977:** Az anarchia felszámolására meetingek kezdődtek, sok mikroprocesszor gyártó is képviseltette magát (Intel, National Semiconductor, Zilog, Motorola, IBM).

Az Intel bemutatja a meetingen a co-processzor egyes funkcióit és hogy miért azok mellett döntöttek.

IEEE 754 - Lebegőpontos szabvány

- **1977-1981:** Hosszas vita a gradual underflow-ról
Ez sokkal súlyosabb probléma, mint az ember gondolná, sok numerikus kód vált az underflow 0-ra kerekítése miatt használhatatlanná...



Ez egy egyetemi számolásban kisebb gond, de amikor egy elhárító rakéta téveszt célt, az [életekbe kerül...](#)

[Bővebben az IEEE 754 történetéről](#)

- **1980-1985:** A szabvány még csak draft, de a legtöbb gyártó már elkezdte implementálni: Intel, AMD, Apple, ELXSI, IBM, Motorola, National Semiconductor, Weitek, Zilog, AT&T...
- **1985:** Hivatalosan is publikálják a szabványt.
[Amit minden programozónak tudnia kellene a lebegőpontos aritmetikáról!](#)

Milyen pontosság érhető el?

Megjegyzések:

- Az IEEE definiál más pontosságokat is.
- Hardveresen CPU-kon a single és a double pontosság érhető el
- GPU-kon van half is (5 bit exponens, 11 bit törtrész, kb. 3-4 tizedes jegy)
- Ha ennél nagyobb (kisebb) de ismert pontosságra van szükségünk, azt szoftveresen tudjuk megoldani, ami sokkal költségesebb (kb. 2-5x nagyobb költség)
- Ha ráadásul még a pontosságot dinamikusan akarjuk változtatni, akkor a költség sokszorosára nő (100x-1000x függően a szám méretétől)

X87 Assembly utasítások

- **x87**: A matematikai co-processor utasításkészlete
- 8 db 80 bit széles lebegőpontos regiszter, stack szerkezetben, ezen operálnak a műveletek, 32/64/80 bites lebegőpontos számokon, 16/32 bites egészeken

Műveletek:

Adat / stack mozgatás

+, -, *, /, összehasonlítás, 0-ra tesztelés
négyzetgyök, abszolút érték, maradék, tan, atan, 2 hatványozás,
 $2^x - 1$, $y \cdot \log_2 x$, $y \cdot \log_2(x + 1)$

matematikai konstansok (0, 1, π , $\ln 2$, $\log_2 e$, $\log_2 10$, $\log_{10} 2$) kerekítés

- **387**: Sin, Cos, SinCos
- **Pentium Pro**: conditional move
- **SSE3**: integerre trunkálás, kerekítés nélkül

X86 utasítások időzítései

Művelet	Latency
Shift / Rot	1-4
AND / OR / XOR	1-4
Compare/test	1-4
Call (Ret)	5 (8)
Integer add/sub	1-8
Integer mul	3-18
Integer div	32-103

Művelet	Latency
MMX	1-9
SSE Integer	1-9
SSE single ált.	1-12
SSE single mul	3-7
SSE single div/sqrt	10-40
SSE2 double ált.	1-12
SSE2 double mul	3-7
SSE2 double div/sqrt	14-70
SSE2 128bit int	1-10
SSE3 / SSE4	1-14

AVX arányok kb. az SSE-vel egyeznek

Művelet	Latency
FADD/FSUB/FABS	2-6
FMUL	7-8
FDIV	23-44
FSQRT	23-44
FSIN, FCOS	160-280
FSINCOS	160-250
FPTAN	225-300
FPATAN	150-300
FSCALE	60
FYL2X/FYL2XP1	100-250

A műveleteknek az egymáshoz viszonyított aránya mérvadó csak!

Időköltségek A modern számítástechnikában

Művelet	Időköltség [ns]	Időköltség [ms]
1 órajel egy 3 GHz-es processzoron	1	1e-6
L1 cache elérése	0.5	5e-7
Elágazás téves jóslása	5	5e-6
L2 cache elérése	7	7e-6
Mutex lock/unlock	25	2,5e-5
RAM elérése	100	0,0001
1kB adat tömörítése Snappy -val	3000	0,003
1kB adat átküldése 1 Gbps-es hálózaton	10000	0,01
4 kB adat random olvasása SSD-ről	150000	0,15
1 MB adat folytonos beolvasása RAM-ból	250000	0,25
Oda-vissza út egy adatközponton belül	500000	0,5
1 MB adat folytonos olvasása SSD-ről	1e6	1
Merevlemez ,seek' (keresési) ideje	1e7	10
1 MB adat folytonos olvasása lemezeről	2e7	20
TCP/IP csomag kontinensek között	1,5e8	150

Forrás és ajánlott néznivaló

- Ma a teljesítmény nagy részét azzal érjük el, hogy a vektor utasításkészletet használjuk, ezek ugyanis sokkal optimálisabbak és hatékonyabbak, mint az eredeti x86/x87 utasítások
- Ahhoz, hogy a vektor regiszterekbe át tudjuk tölteni a 4-8-16 komponenses értékeket, megfelelő adatlokalitás szükséges a memóriában!

X86 Assembly utasítások - Összefoglalás

- Az x86 egy évtizedek óta fejlődő és bonyolódó, de végig visszafelé kompatibilis utasítás készletet rejt.
- Az összes különböző lehetséges utasítás száma több ezer.
- Az egyre szélesebb SIMD regiszterek és utasítások lassan önálló nyelvvé fejlődnek, amiknek az optimális kihasználása nem könnyű (a fordítóra vagyunk bízva legtöbbször).
- Az új utasítások a leggyakrabban használt műveleteket gyorsítják.

Intel Manualok:

- [Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 1](#)
- [Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3A](#)

Agner Fog [weboldala](#) ahol részletes [optimalizációról](#) szóló írások, tesztek és forráskódok találhatóak.