Gradus Vol 5, No 2 (2018) 318-324
ISSN 2064-8014

AZ SIMD ARCHITEKTÚRA BEVEZETÉSE AZ OKTATÁSBA

INTRODUCTION OF SIMD ARCHITECTURE TO EDUCATION

Pintér István [1*]

[1] Informatika Tanszék, GAMF Műszaki és Informatikai Kar, Neumann János Egyetem, Magyarország

Kulcsszavak:
SIMD
vektor
telítéses aritmetika
kódrészlet

Keywords:
SIMD
vector
saturated arithmetic
code-snippet

Cikktörténet:
Beérkezett 2018.augusztus 01.
Átdolgozva 2018.szeptember 04.
Elfogadva 2018.október 01.

Összefoglalás

A számítógép-architektúrák egyik típusa az SIMD (Single Instruction stream Multiple Data stream). Az Intel/AMD x86 mikroprocesszorok újabb változataiban jelen van az ilyen irányú utasítás-készlet bővítés, ami az MS Visual Studio fejlesztési környezetben is elérhető. A cikk fő témája a natív skalár és vektor adattípusok, valamint a telítéses üzemmódú aritmetika. A fogalmakat C/asm kódrészletek szemléltetik.

Abstract

SIMD (Single Instruction stream Multiple Data stream) is a type of computer architectures. This ISA-extension can be found in latest versions of Intel/AMD x86 processors, and can be accessed using MS Visual Studio development environment. The main topics of the paper are the native scalar and vector data types and the saturated arithmetic. The concepts will be illustrated with C/asm code-snippets.

1. Bevezetés

Napjainkban sok számítógép működik Intel-architektúrájú mikroprocesszorral [1]. Intézményünkben az IBM PC XT/AT kompatibilis számítógépek megjelenésével egyidejűleg szerepel az oktatásban az IBM PC assembly témakör [2]. A kezdetektől fogva célunk az elektronikus digitális számítógép működésének minél mélyebb megismertetése volt. Jelenleg a számítógép-architektúrák I., II. tantárgyak gyakorlatainak anyaga ez az alapképzésben (BSc) és a felsőoktatási szakképzésben (FOSzK) mind nappali. mind levelező munkarendben. A tantárgy a 2017-es új tanterv megjelenéséig kötelező volt mindegyik munkarend és mindegyik képzési típus esetében.

A cikk példáiban fejlesztői környezetként a Microsoft Visual Studio 2013 programot használtuk, ugyanis ez egyrészt lehetővé teszi _asm{} kódrészletek írását, másrészt a gépi utasítások végrehajtása kényelmesen tanulmányozható a lépésenkénti utasítás-végrehajtással. További hasznos funkció a memóriatartalom különböző modellek szerinti megjeleníthetősége, a regiszterek, állapotjelzők tartalmának megtekinthetősége. A gépi utasítások bináris kódja is elérhető a disassembly funkcióval. A cikk témája szempontjából nézve alapvető fontosságú, hogy az _asm{} blokkon belül írható és futtatható olyan kód, amely az újabb ISA-bővítésekkel (ISA, Instruction Set Architecture) bevezetett utasításokat is tartalmazza. Ennek támogatása processzorfüggő és fordítóprogram-függő, de számítógépeinken az MMX, SSE, SSE2 utasítások működése tanulmányozható. Ez az alapja annak, hogy a számítógép-architektúrák II. tantárgyban megjelent az SIMD architektúra.

* Kapcsolattartó szerző. Tel.: +36 76 516420; fax: +36 76 516399
E-mail cím: pinter.istvan@gamf.uni-neumann.hu

A cikk felépítése a következő. A második fejezet az SIMD modellhez szükséges alapozó ismeretek összefoglalásával foglalkozik, ezt követik az MMX/SSE/SSE2 példák: itt a hangsúlyt a vektorokon végzett utasításokra helyeztük. A cikk összefoglalással, köszönetnyilvánítással és irodalomjegyzékkel fejeződik be.

2. Előkészületek az SIMD architektúra tanulmányozásához

A számítógép-architektúrák I. tantárgyban a fixpontos és lebegőpontos aritmetikai utasításokkal, a logikai utasításokkal, a különféle címzési módokkal, a ciklus-szervezéssel és a feltételes elágazási utasításokkal foglalkozunk elsősorban. A tantárgy részletesen tárgyalja az állapotjelzőket/állapotbiteket is (átvitelbit, kölcsönbit, előjelbit, fixpontos aritmetikai túlcsordulás bit, segéd-átvitelbit, paritásbit, zérus jelző). Mivel az SIMD architektúra egyik jellemzője a telítéses üzemmódú aritmetika, az említett előismeretek birtokában meg tudják valósítani a hallgatók ezeket a számításokat nem-SIMD utasításokkal, majd ezt követőleg már értékelni képesek a vonatkozó SIMD utasítások előnyeit.

A mintapéldákban a CPU által közvetlenül kezelhető (natív) adattípusok a következők: bit, bájt (byte, B), szó (word, W, 16 bites), duplaszó (double word, D, 32 bites), négyszó (quad word, Q, 64 bites), sztring (bájtsorozat, 0x00 végjellel), az IEEE-754 szabvány szerinti 32 és 64 bites lebegőpontos számok. A számábrázolásokra hagyományosan nagy hangsúlyt fektetünk. A hallgatók megismerik a fixpontos aritmetikai utasításokról szóló tananyagrészben az előjel nélküli egész és a kettes komplemens kódú számokat. Az előjeles abszolút-értékes és a b-többletes kódú számábrázolás a lebegőpontos számokkal foglalkozó anyag témája. A sztring kivételével a többi adattípus lényegében *számok* bináris ábrázolását jelenti, ezért ezeket nevezhetjük *skalár*oknak is. Az oktatási tapasztalatok szerint ez az alapozás fontos, mert az SIMD architektúrában elérhető natív vektor adattípusban a komponensek éppen az említett ábrázolású számok (skalárok).

3. Az MMX utasításkészlet-bővítés [1]

Az MMX az angol Multimedia Extension szavakból alkotott betűszó. Az MMX technológiával rendelkező Pentium processzorban és a Pentium II-ben megjelenő ISA-bővítés és a hozzá tartozó számítási modell jelentette e tekintetben az újdonságot. Ennek célja elsősorban az volt, hogy a multimédia alkalmazásokhoz szükséges jel- és képfeldolgozási algoritmusokat is hatékonyan lehessen megvalósítani ezeken a processzorokon futó programokkal. Fontos, hogy futás közben a program meg tudja állapítani, hogy van-e az adott processzoron MMX támogatás. Ez a megfelelően paraméterezett CPUID utasítással ismerhető meg (azokon a processzorokon, amelyek ismerik ezt az utasítást).

Az MMX modell regiszterei 64 bitesek, számuk 8. Különlegességük, hogy a lebegőpontos egység (FPU, Floating Point Unit) 80 bites regisztereinek alsó 64 bites részeként valósították meg őket. Ez azt jelenti, hogy vagy az FPU regiszter/vermet, vagy az MMX regisztereket lehet használni, a kettőt együtt nem, továbbá gondoskodni kell az FPU állapotának mentéséről/visszaállításáról is. Erre szolgálnak az FSAVE/EMMS/FRSTOR utasítások.

Az MMX modell utasításainak operandusai fixpontosak, nincsenek lebegőpontos utasítások. A számos utasítás-csoport közül ebben a cikkben az aritmetikai utasításokkal foglalkozunk, de hadd említsük meg a 64 bites bitenkénti logikai utasításokat, különösen a bitenkénti NAND (NEM-ÉS) utasítást.

3.1. Összeadás, kivonás, az aritmetikai egység üzemmódjai

Az MMX modell vektorokkal számol. A szakirodalom szerint [1] emiatt az SIMD számítási modell jellemző, mert ekkor a CPU az adott gépi utasítás (Single Instruction) végrehajtása során *egyidejűleg* több adattal (Multiple Data) végez műveletet.

Az MMX modell a 64 bites adatot háromféle felbontásban értelmezi vektorként:

- kétdimenziós vektor, 32 bites komponensekkel (D),
- négydimenziós vektor, 16 bites komponensekkel (W),
- 8 dimenziós vektor, 8 bites komponensekkel (B).

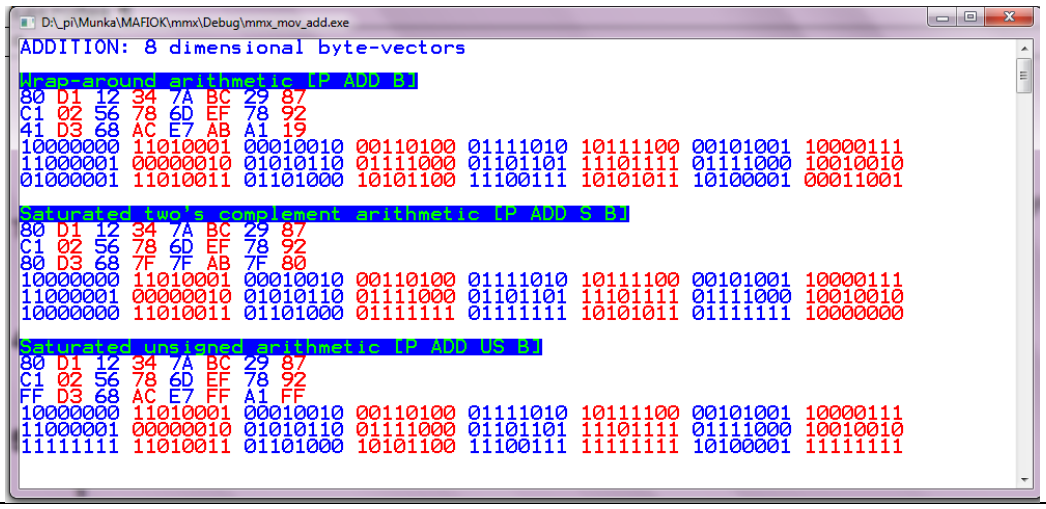
A komponensek vagy előjel nélküli egész, vagy kettes komplemens kódú számok. Az aritmetikai egység üzemmódjainak száma is három. A szokásos körbenforgó üzemmódon kívül telítéses (szaturációs) üzemmódban is képes számolni, mindkét számábrázolás esetén. Hétféle összeadó és hétféle kivonó utasítás van (nem kilenc féle), mert a telítéses üzemmódban csak bájt vagy szó méretű komponensek lehetnek. Ennek következménye, hogy az adott számítási feladatnak megfelelő gépi utasítást ki kell ezek közül választani. Az utasítás-mnemonikok logikája a következő. P-vel kezdődnek, ami arra utal, hogy a vektort pakolt (Packed) adatnak is nevezi a szakirodalom [1]. A következő rész az aritmetikai műveletre utal (ADD (összeadás) illetve SUB (kivonás)), ezt követi az üzemmódra utaló betű(hiány) (betűhiány a körbenforgó esetben, S a telítéses kettes komplemens kódú esetben, US a telítéses, előjel nélküli egészeknél. Végül a komponensek méretére utaló betű következik a szokásos Intel-es elnevezésekre utalván (B: 8 bites bájt, W: 16 bites szó, D: 32 bites duplaszó). Például a PADDUSW utasítás-mnemonik esetében a művelet 4 dimenziós vektor-összeadás, 16 bites előjel nélküli egész komponensekkel, telítéses üzemmódú aritmetikai egységgel.

A fenti leírás is arra utal, hogy az MMX utasítások végrehajtásának tanulmányozása körültekintést igényel. Ezt megkönnyítendő az SIMD-s tananyagrészhez írtunk olyan függvényeket, amikkel az operandusokat és az eredményt az előírt granulációval (B, W, D, Q), színesen ki lehet írni a képernyőre mind bináris, mind hexadecimális alakban. A továbbiakban bemutatott példák így készültek. Az 1. és 2. ábrán 8 dimenziós vektorok összeadása látható.

```
__int64 x64, y64, z64;
x64 = 0x80D112347ABC2987;
y64 = 0xC10256786DEF7892;
z64 = 0x0000000000000000;
_asm{
      movq mm0, x64;//64 bites adatmozgatás
      movq mm1, y64;//64 bites adatmozgatás
      paddb mm0, mm1;//összeadás, byte granuláció, körbenforgó üzemmód
      movq z64, mm0;//a 64 bites eredmény tárolása

}
```

1. ábra. Kódrészlet: 8 dimenziós vektorok összeadása körbenforgó üzemmódú aritmetikával.



2. ábra. Futási eredmény: 8 dimenziós vektorok összeadása mindhárom üzemmód esetén.

3.2. Szorzás és szorzatok összegzése

Az MMX utasításkészlet-bővítésben található a PMULLW és a PMULHW utasítás. Ezekkel négydimenziós vektorok komponensenkénti szorzását végezhetjük el – a komponensek kettes komplemens kódú számok. Mivel két N bites szám szorzata 2N bites, a szorzat alsó 16 bitje a PMULLW utasítással állítható elő a megfelelő komponens pozíciójában, míg a szorzat fölső 16 bitje

a PMULHW utasítással számítható ki. A 4. ábra példájában: 0x8000 és 0x8001 szorzata 0x3FFF8000. Ha a számok decimális alakjával végezzük az ellenőrzést, akkor -32768 és -32767 szorzatát kell kiszámítani, majd az eredményt 32 bites kettes komplemens kódúvá kell alakítani. A szorzat 1073709056, az átalakítás után 0x3FFF8000 adódik.

Hasznos művelet a szorzatok összegzése. Kettő 2 dimenziós vektor skaláris szorzatát számíthatjuk ki a PMADDWD utasítással. A megfelelő indexű komponensek 32 bites szorzatainak összege található az eredmény fölső illetve alsó 32 bitjén. A 3. ábra kódrészletei szemléltetik a műveleteket, a futási eredményt a 4. ábra mutatja. A PMADDWD esetében a két skaláris szorzat:

[0x3004; 0x400D] és [0x2005; 0x5007] skaláris szorzata 0x1A07406F,

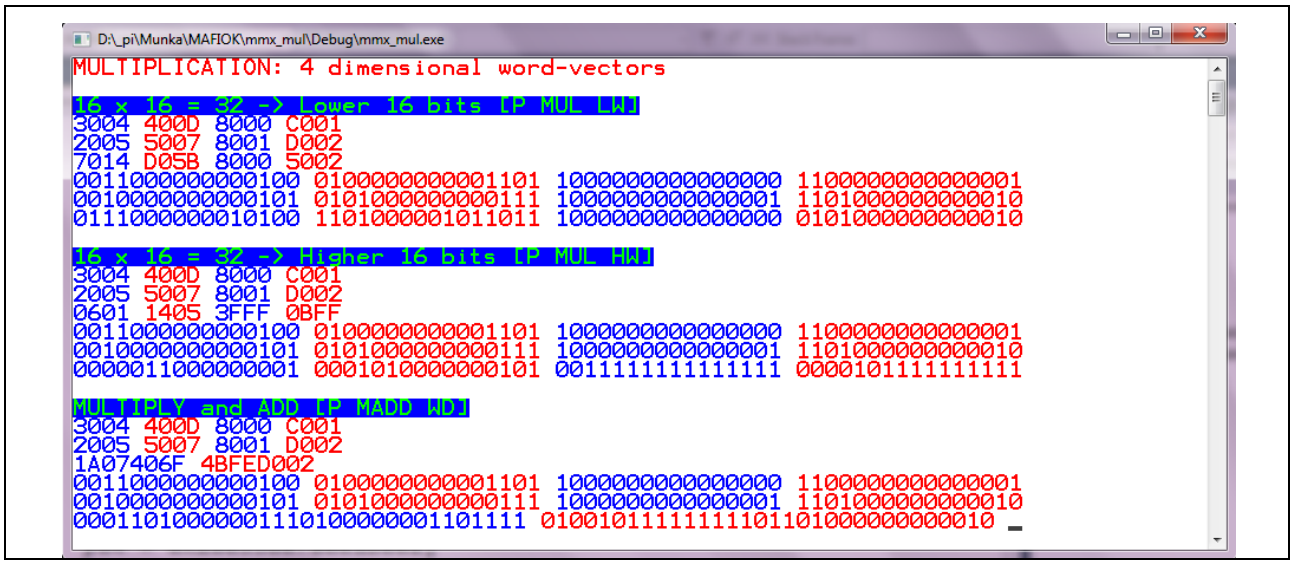
[0x8000; 0xC001] és [0x8001; 0xD002] skaláris szorzata 0x4BFED002.

```
__int64 x64, y64, z64;
x64 = 0x3004400D8000C001;
y64 = 0x200550078001D002;
z64 = 0x0000000000000000;
_asm{
      movq mm0, x64;
      movq mm1, y64;
      pmullw mm0, mm1;
      movq z64, mm0;
}
```

```
__int64 x64, y64, z64;
x64 = 0x3004400D8000C001;
y64 = 0x200550078001D002;
z64 = 0x0000000000000000;
_asm{
      movq mm0, x64;
      movq mm1, y64;
      pmulhw mm0, mm1;
      movq z64, mm0;
}
```

```
__int64 x64, y64, z64;
x64 = 0x3004400D8000C001;
y64 = 0x200550078001D002;
z64 = 0x0000000000000000;
_asm{
      movq mm0, x64;
      movq mm1, y64;
      pmaddwd mm0, mm1;
      movq z64, mm0;
}
```

3. ábra. Kódrészlet: szorzás és a skaláris szorzat számítása.



4. ábra. Futási eredmény: szorzás és a skaláris szorzat számítása.

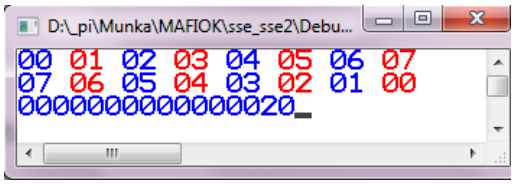
4. Az SSE és SSE2 utasításkészlet-bővítés [1]

Az MMX modellben bevezetett SIMD architektúra továbbfejlesztett változata az SSE (Streaming SIMD Extensions). A programozási modell a számos új utasításon kívül nyolc, egyenként 128 bites regiszterrel bővült, és rendelkezésre áll a 4 dimenziós natív vektor adattípus, melynek komponensei az IEEE-754 szabványnak megfelelő 32 bites lebegőpontos számok. Az SSE2 ennek tovább fejlesztett változata (Streaming SIMD Extensions 2).

Az MMX regisztereket érintő utasítások közül kiemeljük itt a PSADBW utasítást. Ez kiszámítja két 8 dimenziós, előjel nélküli egész számokból álló vektor Manhattan-távolságát (Packed Sum of Absolute Differences), és az eredményt egy MMx regiszterbe írja (5. ábra).

```
__int64 x, y, z;
x = 0x0001020304050607;
y = 0x0706050403020100;
z = 0x0000000000000000;
_asm{
      movq mm0, x;
      movq mm1, y;
      psadbw mm0, mm1;
      movq z, mm0;

}
```



5. ábra. Kódrészlet és futási eredmény: a komponensek különbsége abszolút-értékének összege.

Az SSE modell 128 bites XMM regisztereiben négy, egyenként 32 bites, IEEE-754 szabvány szerint lebegőpontos szám található a natív vektor adattípus esetében. Az SSE modellnél idézzük fel a számítógép-architektúrák I. tantárgyban már tanult fogalmat: a 128 bites regisztertartalom a memóriában vagy igazított (aligned), vagy nem igazított (unaligned) kezdőcímű tartományba tárolható, ilyen címről írható. Például a MOVAPS (Move Aligned Packed Single-precision) utasítás megköveteli az igazított címet, míg a MOVUPS nem. Az SSE modell esetében az igazított cím bájt-szervezésű memória-modellre vonatkozik, és 16-tal maradék nélkül osztható címet jelent. (Ezt még az i8086/8088 mikroprocesszorra utalva paragrafus-határnak is nevezik.) Mivel két egymást követő igazított cím között 16 bájt a memória-terület, éppen elfér benne egy 128 bites XMM regiszter tartalma (16 x 8 bit = 128 bit).

Az SSE utasítások tanulmányozásához hasznos lenne például az __int128 adattípus, hasonlóan az __int64-hez az MMX esetében. Ennek hiányában igazított címen kezdődő, egyszeres pontosságú (float) típusú lebegőpontos számokból álló tömbbel oldható meg a feladat:

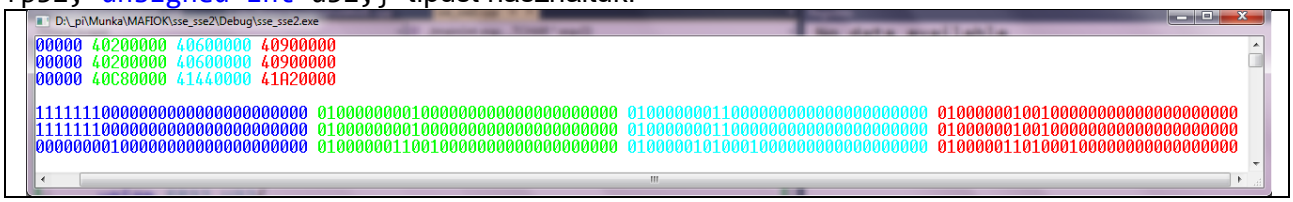
```
__declspec(align(16)) float fp32ArrayX[4] = { 1.5, 2.5, 3.5, 4.5 };
```

A négy aritmetikai alapművelet elvégzéséhez rendelkezésre állnak az ADDPS, SUBPS, MULPS, DIVPS utasítások. Vektorok komponensenkénti szorzása látható a 6. és 7. ábrán:

```
_asm{
      movaps xmm0, fp32ArrayX;//128 bites adatmozgatás igazított címről
      movaps xmm1, fp32ArrayY;/ 128 bites adatmozgatás igazított címről
      mulps xmm0, xmm1;//4 dimenziós, FP32 komponensű vektorok szorzása
      movaps fp32ArrayZ, xmm0;//a 128 bites eredmény tárolása
}
```

6. ábra. Kódrészlet: négydimenziós vektorok komponensenkénti szorzása.

A komponensenkénti hexadecimális és bináris megjelenítéshez az `union FP32_U32{float fp32; unsigned int u32;}` típust használtuk:



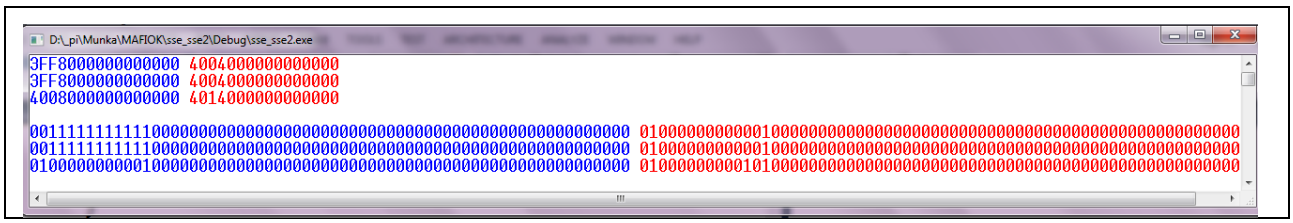
7. ábra. Futási eredmény: négydimenziós vektorok komponensenkénti szorzása.

A komponensenkénti értelemben vett négy alapműveleten kívül rendelkezésre áll például a négyzetgyökvonó (SQRTPS) és a reciprok-képző utasítás is (RCPPS). Ezzel kapcsolatban említjük meg, hogy a számítógép-architektúrák II. tantárgyban a bináris aritmetikai algoritmusok anyagrészben szerepel az iteratív reciprok-képző és az iteratív négyzetgyökvonó algoritmus és a hozzá tartozó számítási struktúra.

Ami az SSE2-t illeti, azt emelnénk ki, hogy itt a fixpontos felbontás kibővül például a kétdimenziós, 64 bites komponensű vektorokkal, ami az MMX-es természetes folytatása. A programozási modell kibővült az általános célú regiszterekkel is (GPR, General Purpose Registers). Ezek a számítógép-architektúrák I. tárgyban szerepelnek (EAX, EBX, ECX, EDX, EBP, ESI, EDI, and ESP). Újdonságként duplapontos számokból álló kétdimenziós vektorokkal is számolhatunk. A cikk témájának megfelelően összeadásra láthatunk példát a 8., 9. ábrán.

```
      _asm{
            movapd xmm0, fp64ArrayX;//128 bites adatmozgatás igazított címről
            movapd xmm1, fp64ArrayY;//128 bites adatmozgatás igazított címről
            addpd xmm0, xmm1;//2 dimenziós, FP64 komponensű vektorok
összeadása
            movapd fp64ArrayZ, xmm0;//a 128 bites eredmény tárolása
      }
```

8. ábra. Kódrészlet: kétdimenziós vektorok összeadása.



9. ábra. Futási eredmény: kétdimenziós vektorok összeadása.

Megjegyzendő, hogy az utóbbi esetben az

```
union FP64_U64{double fp64; unsigned __int64 u64;};
```

típust használtuk a megjelenítéshez.

5. Összefoglalás

A cikkben az SIMD architektúra oktatásba történő bevezetésével foglalkoztunk. Az oktatásban célunk elsősorban a gépi utasítás-végrehajtás tanulmányozása, az alapismeretek megtanítása.

Az áttekintő jellegű bevezető után ismertettük az előkészületeket a számítógép-architektúrák I. tantárgy vonatkozásában, majd elsősorban a natív vektor adattípussal, a vektorokkal végzett műveletekkel foglalkoztunk.

Az MMX modell tárgyalásakor ismertettünk háromféle aritmetikai üzemmódot, az SSE modell esetében tértünk ki az igazított címen kezdődő adat fogalmára, példát mutattunk be 32 bites lebegőpontos számokból négydimenziós vektorokkal végzett műveletre. Az SSE2 modellről szóló részben a 64 bites, kétdimenziós vektorokkal végzett aritmetikai műveltre adtunk példát. Eddigi oktatási tapasztalataink szerint a tananyag elsajátítását segítik azok a függvények, amikkel az eredményeket kívánt granulációval, hexadecimálisan és binárisan lehet megjeleníteni.

A téma iránt tovább érdeklődő hallgatók megítélésünk szerint sikerrel dolgozhatják fel például a [3] szakkönyv anyagát.

A 2014-2017 közötti tanévek során a számítógép-architektúrák II. tárgyat összesen 492 hallgató vette fel. A részletes adatok az 1. Táblázatban találhatók mindkét munkarendre és mindkét képzési típusra vonatkozóan.

Az oktatási tapasztalatok alapján a témakör kisebb létszámú kurzusokon tárgyalható hatékonyabban: a FOSzK képzésben ismeretterjesztő jelleggel, a BSc-n specializáció választás után mélyebb áttekintésben.

A jelenlegi tanévtől kezdve az új tantervben már eszerint történik az oktatás a Számítógép-architektúrák II. tantárgy részeként, mely a FOSzK képzésben kötelező, a BSc képzésben kötelezően választható.

1. Táblázat. Hallgatói létszámok

A képzés típusa	BSc		FOSzK	
Tanév/félév	nappali	levelező	nappali	levelező
2014/15/1	74	25	16	11
2015/16/1	74	31	20	11
2016/17/1	68	30	15	17
2017/18/1	49	23	17	11
Összesen	**265**	**109**	**68**	**50**
	374		**118**	
	492			

Köszönetnyilvánítás

Köszönettel tartozunk a kutatás támogatásáért, amely az EFOP-3.6.1-16-2016-00006 „A kutatási potenciál fejlesztése és bővítése a Neumann János Egyetemen” pályázat keretében valósult meg. A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával, a Széchenyi 2020 program keretében valósul meg.

Irodalomjegyzék

[1] Intel® 64 and IA-32 Architectures Software Developer Manuals. Published on October 12, 2016, updated May 18, 2018. Available: https://software.intel.com/en-us/articles/intel-sdm [Megtekintés: 05-Jun-2018].
[2] Pintér I.: IBM PC assembly alapismeretek. H-207 jegyzet. GAMF, 1990.
[3 D. Kusswurm: Modern X86 Assembly Language Programming. Apress, 2014., ISBN 978-1-4842-0065-0.