“ Invaţă din greşelile altora, pentru că nu ai timp să le faci pe toate.” S. Freud

Curs 12
Instrucţiuni Speciale pentru Grafică şi Multimedia

- Cresterea performantei microprocesoarelor este datorata :

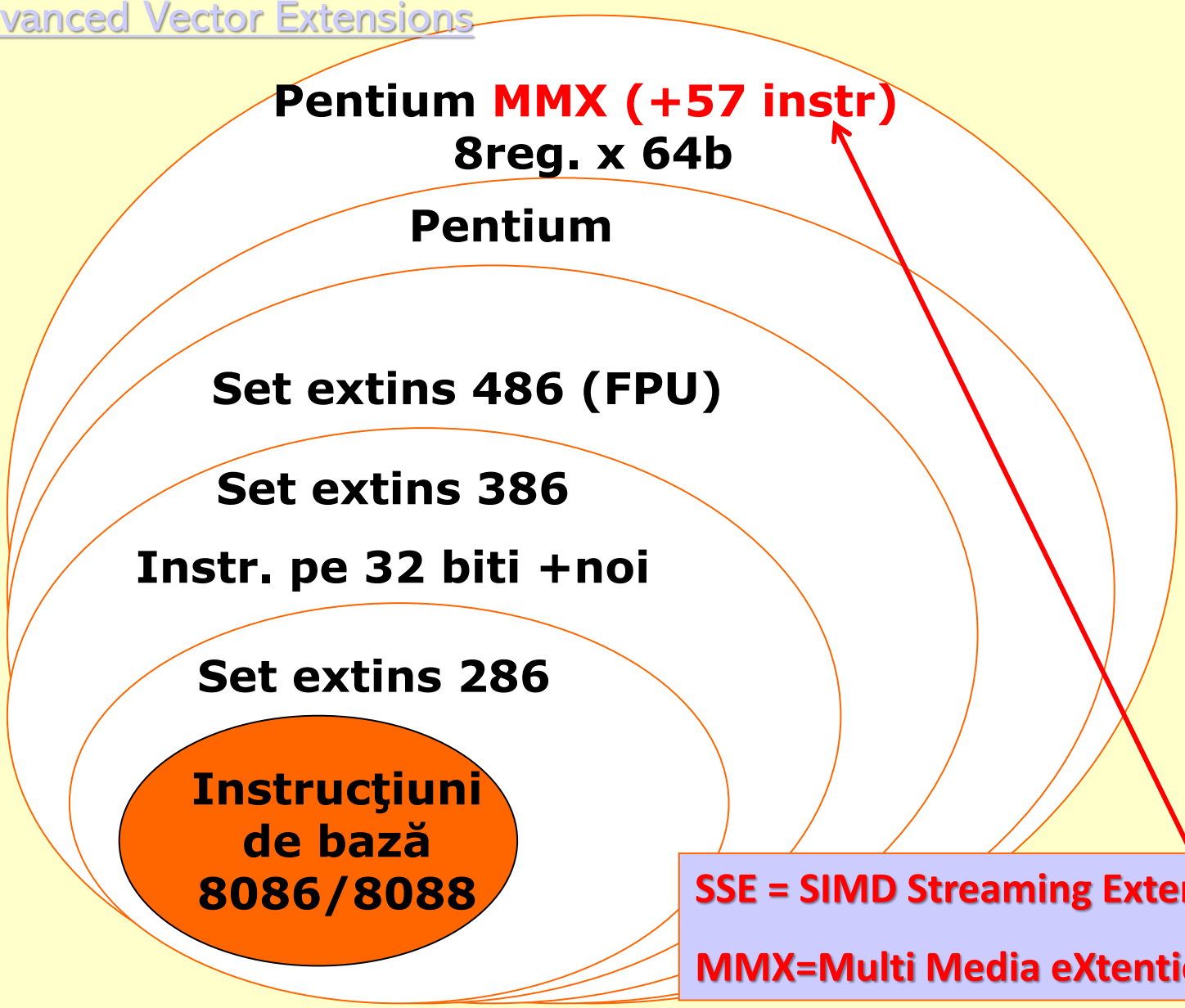
  - Cresterii fecventei de ceas
  - Imbunatatirii arhitecturii (ft. importante: pipeline/cache/SIMD)

  - Intel a analizat aplicatiile ***multimedia*** si a stabilit ce au in comun:
    - tipuri de date mici (8-bit/pixel, 16-bit /sample-audio)
    - operatii recurente
    - paralelism inerent

PIII = PII + SSE (70 instr.) – 8 reg x 128b – FP-DP

P4= PIII + SSE2 (144 instr.) + SSE3 (13 instr) +SSE4(54) +SSE5 (170) +... (AVX)

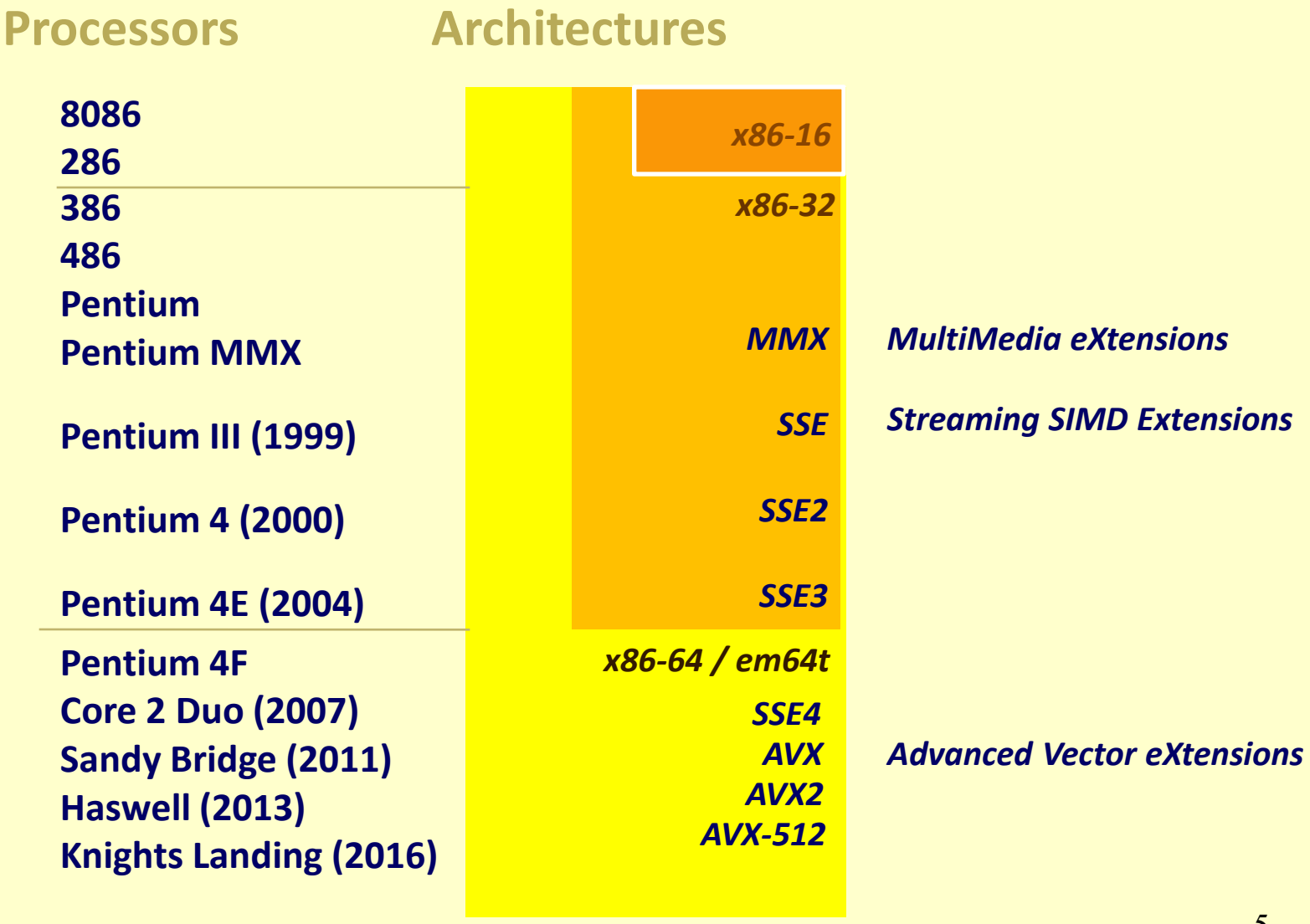
Advanced Vector Extensions



SSE = SIMD Streaming Extension

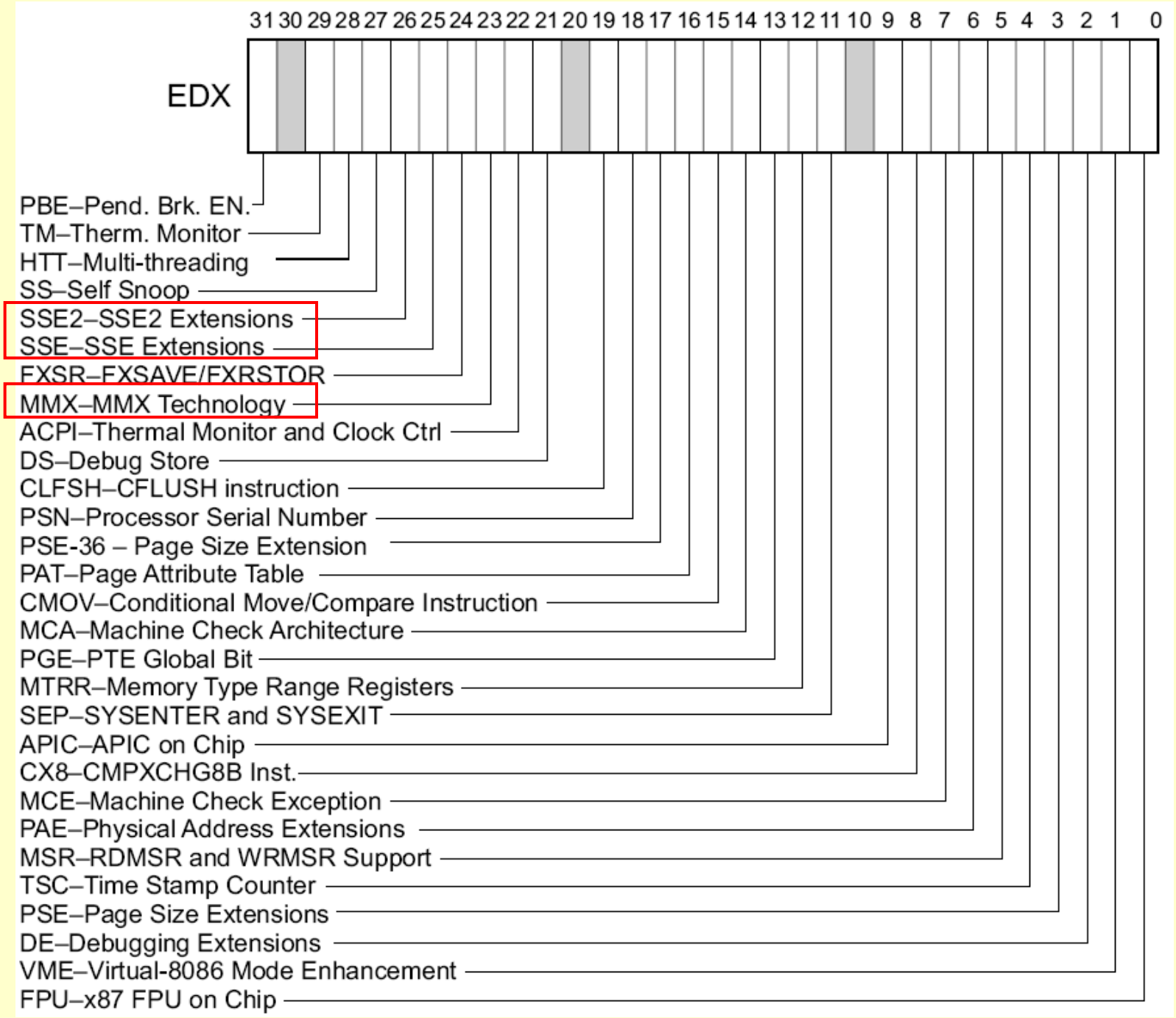
MMX=Multi Media eXtention

Intel Architectures and SIMD



Detectie MMX/SSE

```
mov    eax, 1 ; cerere info versiune
cpuid         ; suportat de la Pentium
test   edx, 00800000h ;bit 23  MMX
          ; 02000000h (bit 25) SSE
          ; 04000000h (bit 26) SSE2
jnz    HasMMX
```


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
EDX
PBE–Pend. Brk. EN.
TM–Therm. Monitor
HTT–Multi-threading
SS–Self Snoop
SSE2–SSE2 Extensions
SSE–SSE Extensions
FXSR–FXSAVE/FXRSTOR
MMX–MMX Technology
ACPI–Thermal Monitor and Clock Ctrl
DS–Debug Store
CLFSH–CFLUSH instruction
PSN–Processor Serial Number
PSE-36 – Page Size Extension
PAT–Page Attribute Table
CMOV–Conditional Move/Compare Instruction
MCA–Machine Check Architecture
PGE–PTE Global Bit
MTRR–Memory Type Range Registers
SEP–SYSENTER and SYSEXIT
APIC–APIC on Chip
CX8–CMPXCHG8B Inst.
MCE–Machine Check Exception
PAE–Physical Address Extensions
MSR–RDMSR and WRMSR Support
TSC–Time Stamp Counter
PSE–Page Size Extensions
DE–Debugging Extensions
VME–Virtual-8086 Mode Enhancement
FPU–x87 FPU on Chip

De ce Instructiuni Multimedia?

• Ideea care sta la baza setului **MMX** si a noilor seturi – **Multimedia Instructions Sets'** este ca : *mai multe date de acelasi tip sunt procesate simultan (SIMD)*

• Programul executa o aceiasi operatie asupra datelor pe care le proceseaza

• In general, este mai rapid a prelucra aceste date simultan (paralel) decat secvential

• Aplicatiile Multimedia şi de Comunicatii consumă resurse de calcul semnificative

•Sprijinul oferit de hardware-ul specific permite accelerarea procesarii

Scopul MMX

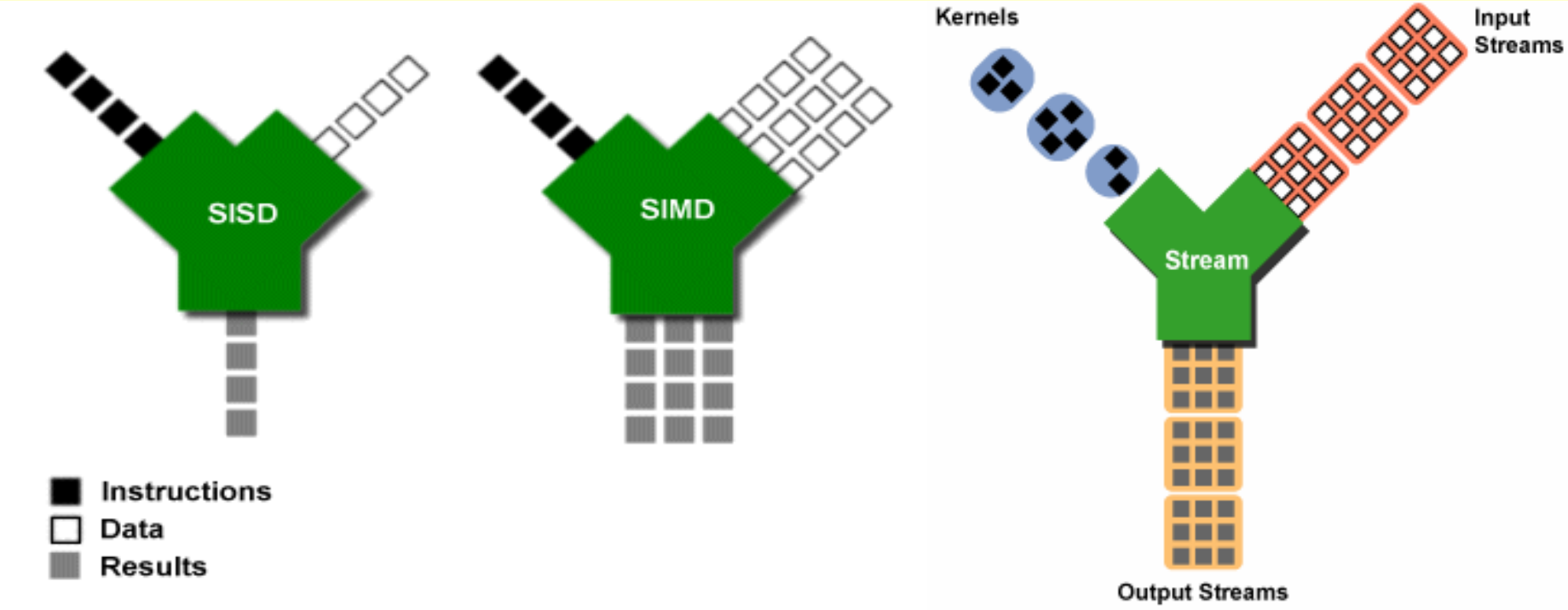
- Extensiile SIMD au fost concepute inițial pentru a accelera aplicațiile multimedia și de comunicații
  - grafica și prelucrarea imaginilor
  - prelucrarea video și audio
  - recunoaștere a vorbirii, …

- Poate fi folosit și pentru alte calcule științifice intensive

Ce reprezinta arhitectura SIMD?

• O maşină SIMD beneficiază de proprietatea fluxului de date numită *paralelismul datelor*

• Paralelismul datelor apare când avem o cantitate mare de date de acelasi tip, care necesita procesarea prin aceiaşi metodă/ instructiune



Operatii cu Vectori

- Adunare vectori Z = X + Y

```
for (i=0; i<n; i++)
z[i] = x[i] + y[i];
```

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \dots\dots \\ x_n + y_n \end{pmatrix}$$

- Scalare vectori Y = a * X

```
for(i=0; i<n; i++)
y[i] = a*x[i];
```

$$a * \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} a * x_1 \\ a * x_2 \\ \dots\dots \\ a * x_n \end{pmatrix}$$

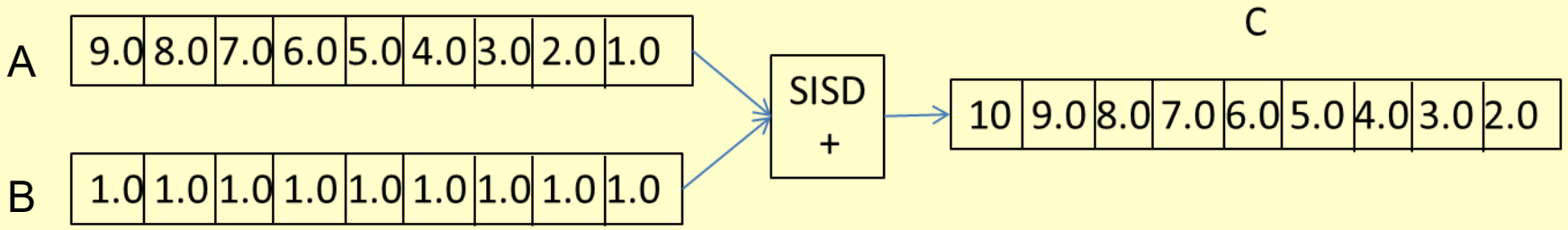
- Dot product (suma de *)

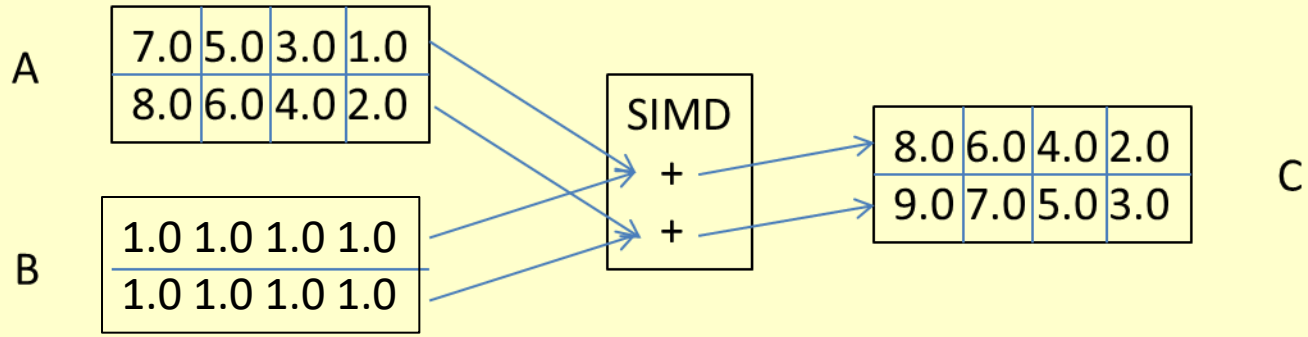
```
for(i=0; i<n; i++)
r += x[i]*y[i];
```

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \bullet \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = x_1 * y_1 + x_2 * y_2 + \dots\dots + x_n * y_n$$

Operatii cu vectori - SISD si SIMD

- C = A + B
  - For (i=0;i<n; i++) c[i] = a[i] + b[i]





Dezvoltarea IA-32 SIMD

- *MMX (MultiMedia EXtension)* a fost introdus in 1997 (Pentium MMX si Pentium II).

-SSE (Streaming SIMD Extension) a fost introdus la Pentium III.

-SSE2 a fost introdus la Pentium 4

-SSE3 a fost introdus la Pentium 4 cu suport hyper-threading technology

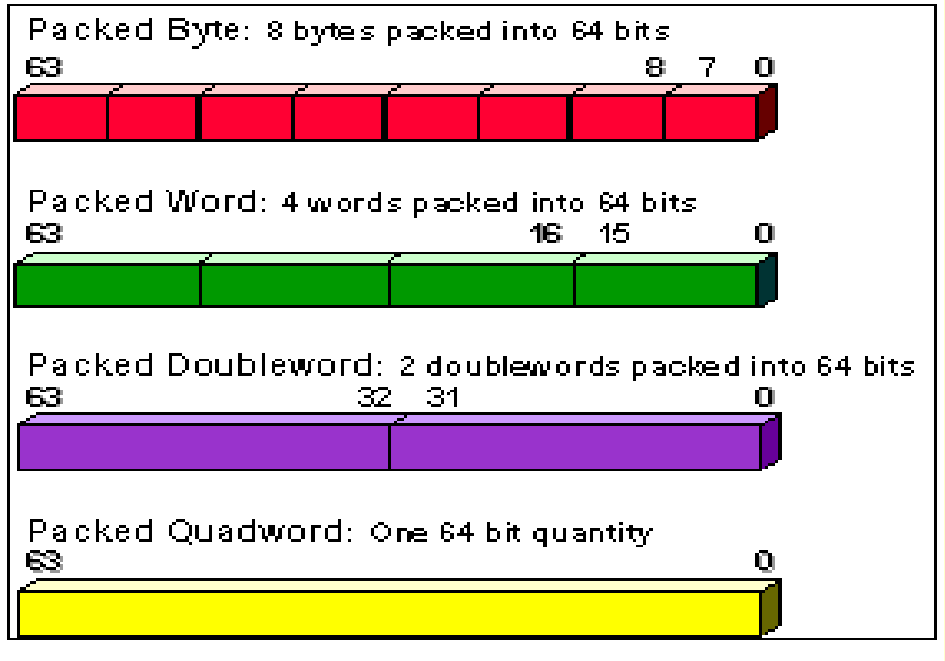
…

- După analizarea multor aplicații existente, cum ar fi ***grafica, MPEG, muzică, recunoașterea vorbirii, jocuri, prelucrare de imagini*** s-a descoperit că mulți algoritmi multimedia ***executa aceleași instrucțiuni pe mai multe pachete de date dintr-un set mare de date***.

- Elemente tipice sunt de mici dimensiuni: 8 biți/pixel, 16 biți pentru audio, 32 biți pentru grafică și de calcul general.

MMX - arhitectura software

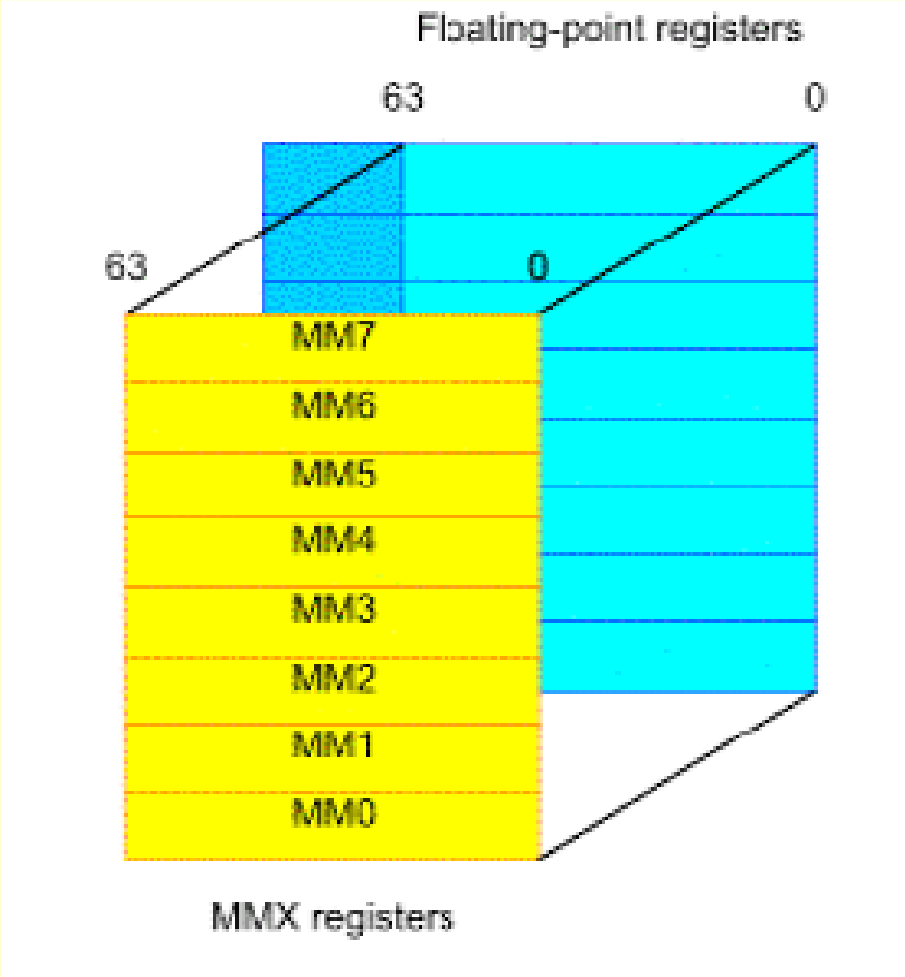
Tipuri de date



- 8 registre de 64-biti
  - Registrele pot fi subdivizate in:
    - 8 elemente x 8-biti - intregi (packed byte)
    - 4 elemente x 16-biti – intregi cu/fara semn (packed word)
    - 2 elemente x 32-biti - intregi cu/fara semn (packed double word)
- *Registrele MMX sunt partajate cu operatiile flotante* (FPU)

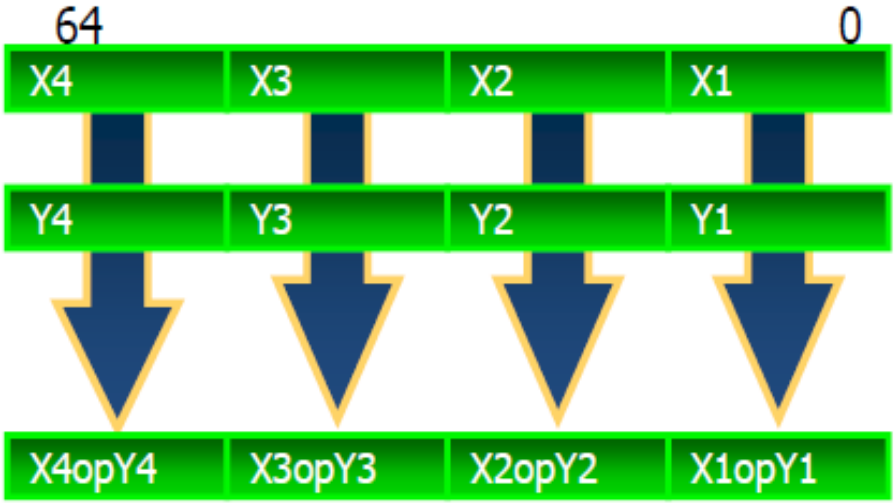
Registre MMX- Compatibilitate

- Nu s-au adaugat noi excepţii sau stări la procesor
- Registrele MMX reprezintă registre alias ale FPU: - câmpul exponent, care corespunde bitilor (64-78) ale reg. FPU şi bitul de semn (bit 79) sunt setati la “1”, făcând ca valoarea din registru să fie NaN (Not a Number) sau infinit, când este văzută ca o valoare flotantă



Exemplu de prelucrare

- Pixelii sunt în general întregi pe 8/16-biţi - se pot împacheta câte 8/4 pixeli într-un registru MMX de 64-biţi
- O instrucţiune MMX ia cei 4 pixeli deodată dintr-un registru MMX, execută operaţia aritmetică sau logică pe toate elementele in paralel şi scrie rezultatul intr-un registru MMX



MMX™

Vector size: 64bit

Data types: 8, 16 and 32 bit integers

VL: 2,4,8

For sample on the left: Xi, Yi 16 bit integers

Cum opereaza setul MMX ? Instructiuni MMX : 57

Tipuri de instrucţiuni:

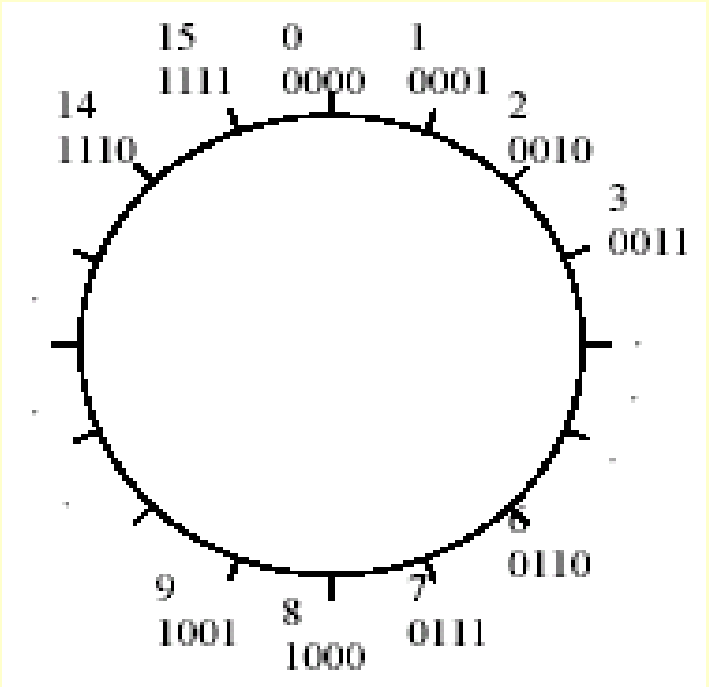
- transfer date (movq, movd)
- aritmetice (padd, psub, pmul ...)
- comparaţii (pcmpeq, pcmpgt)
- conversii (pack,punpck)
- logice (pand, pandn, por, pxor)
- deplasare (psll, psrl, psra)
- management stare (emms)

- add, subtract, multiply, compare, shift, data conversion, 64-bit data move, 64-bit logical operation and multiply-add for multiply-accumulate operations.

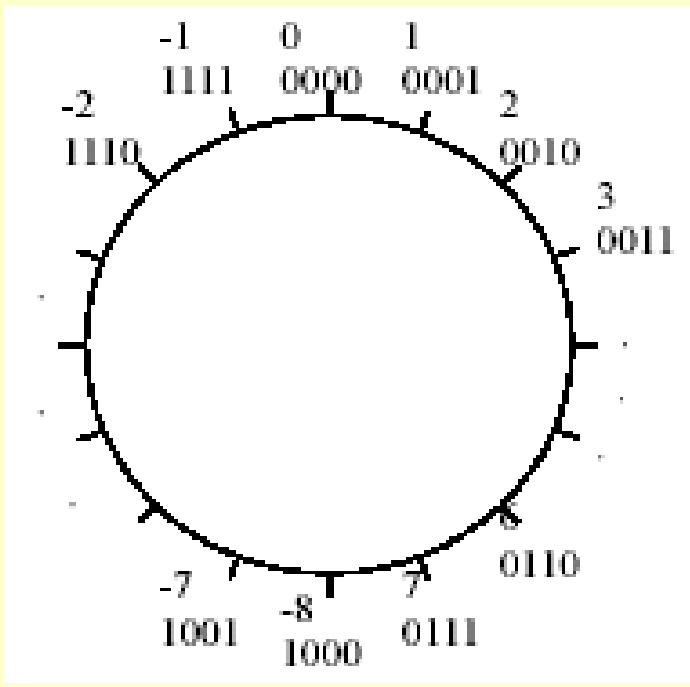
Aritmetica “wrap around” = modulo 2^n

- Cu aritmetica modulo 2^n:
- La un rezultat care este in afara domeniului (CF=1/ *overflow este ignorat*) numai biţii LSB sunt păstraţi din destinaţie (rezultat)

“Roata numerelor” fără semn reprezentate pe 4 biţi

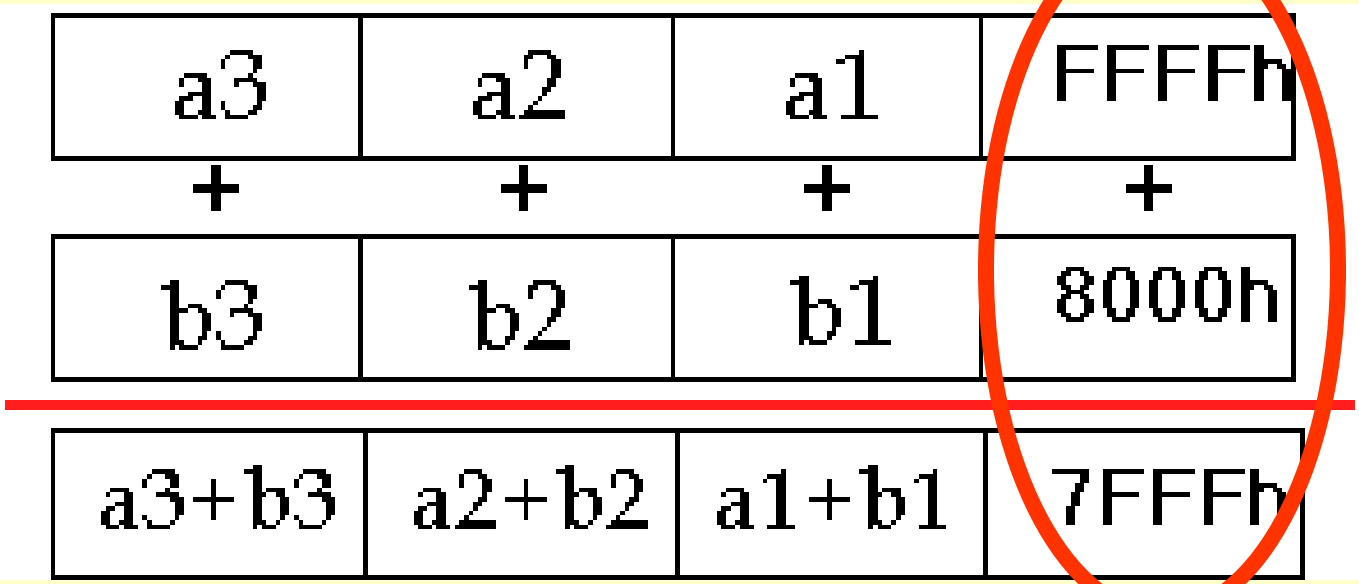


“Roata numerelor” cu semn reprezentate pe 4 biţi



Packed Add Word cu wrap around (modulo 2)

- Fiecare adunare este independentă
- La cea din dreapta apare depăşire şi trunchiere



0x00	0x00	0x57	0x7F	0xAB	0xAB	0xC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0xEB	0x08	0x3F

Obtinerea efectului wrap around pe culoarea pixelilor (modulo 2^8);

Saturarea aritmetică

- **Saturare:** dacă rezultatul operatiei este overflow/underflow, rezultatul este limitat la cea mai mare/mica valoare reprezentabila

- Aceasta este importanta la calculul pixelilor si se evita un “wrap-around” care ar cauza trecerea unui pixel alb brusc la negru (de ex.)

- Nu exista *"saturation mode bit”:* ptr. ca un nou bit de mod necesita modificarea SO

- Se folosesc instructiuni distincte ptr. a obtine rezultat *“wrap-around”* sau *cu saturatie*

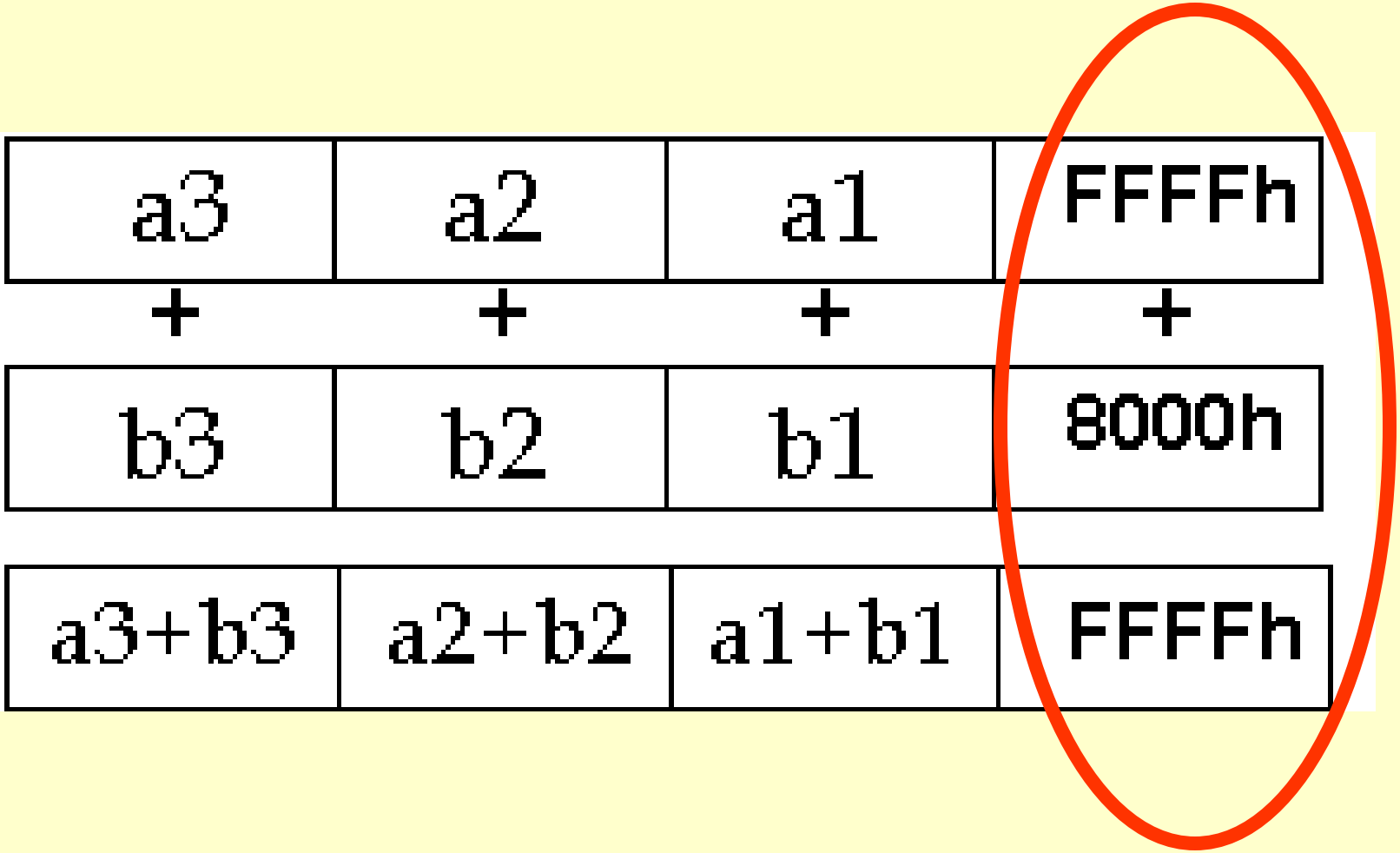
Gamele de saturare

Data Type	Lower Limit		Upper Limit	
	Hexadecimal	Decimal	Hexadecimal	Decimal
Signed Byte	80H	-128	7FH	127
Signed Word	8000H	-32,768	7FFFH	32,767
Unsigned Byte	00H	0	FFH	255
Unsigned Word	0000H	0	FFFFH	65,535

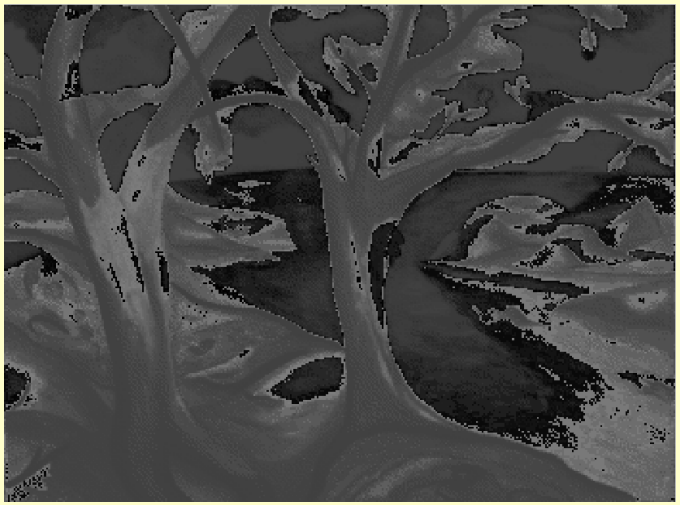
- Saturarea aritmetica dă un răspuns pentru multe situaţii de depăşire.
- de ex., la calculul culorilor, saturarea determină ca o culoare să rămână „pură” (negru/alb) fără a permite inversarea

0x00	0x00	0x57	0x7F	0xAB	0xAB	0xC8	0xFF
+	+	+	+	+	+	+	+
0x40	0x40	0x40	0x40	0x40	0x40	0x40	0x40
=	=	=	=	=	=	=	=
0x40	0x40	0x97	0xBF	0xEB	0XEB	0xFF	0xFF

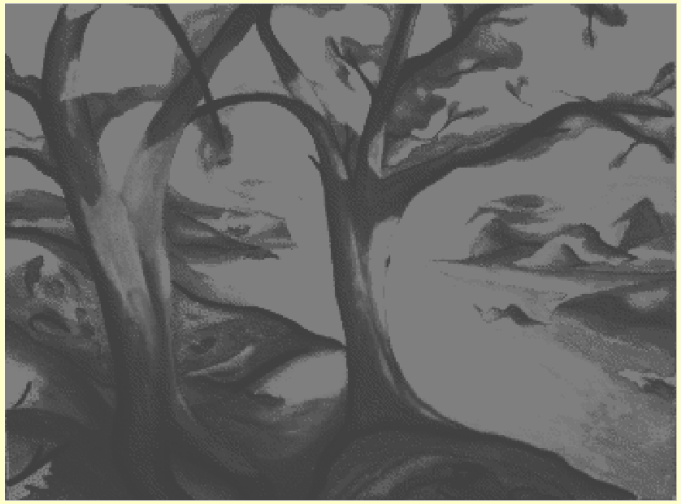
Packed Add Word cu unsigned saturation



Imaginea originală pe nivele de gri;



Efectul Wraparound;



Efectul de saturaţie

Setul de Instructiuni MMX

Category		Wraparound Modulo 2^n	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW PSUBSB, PSUBSW	PADDUSB, PADDUSW PSUBUSB, PSUBUSW
	Subtraction	PSUBB, PSUBW, PSUBD		
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB

Setul de Instructiuni MMX

Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		
		Packed		**Full Quadword**
Logical	And And Not Or Exclusive OR			PAND PANDN POR PXOR
Shift	Shift Left Logical Shift Right Logical Shift Right Arithmetic	PSLLW, PSLLD PSRLW, PSRLD PSRAW, PSRAD		PSLLQ PSRLQ

Setul de Instructiuni MMX

		Doubleword Transfers	Quadword Transfers
Data Transfer	Register to Register	MOVD	MOVQ
		MOVD	MOVQ
	Load from Memory	MOVD	MOVQ
	Store to Memory		
Empty MMX State		EMMS	

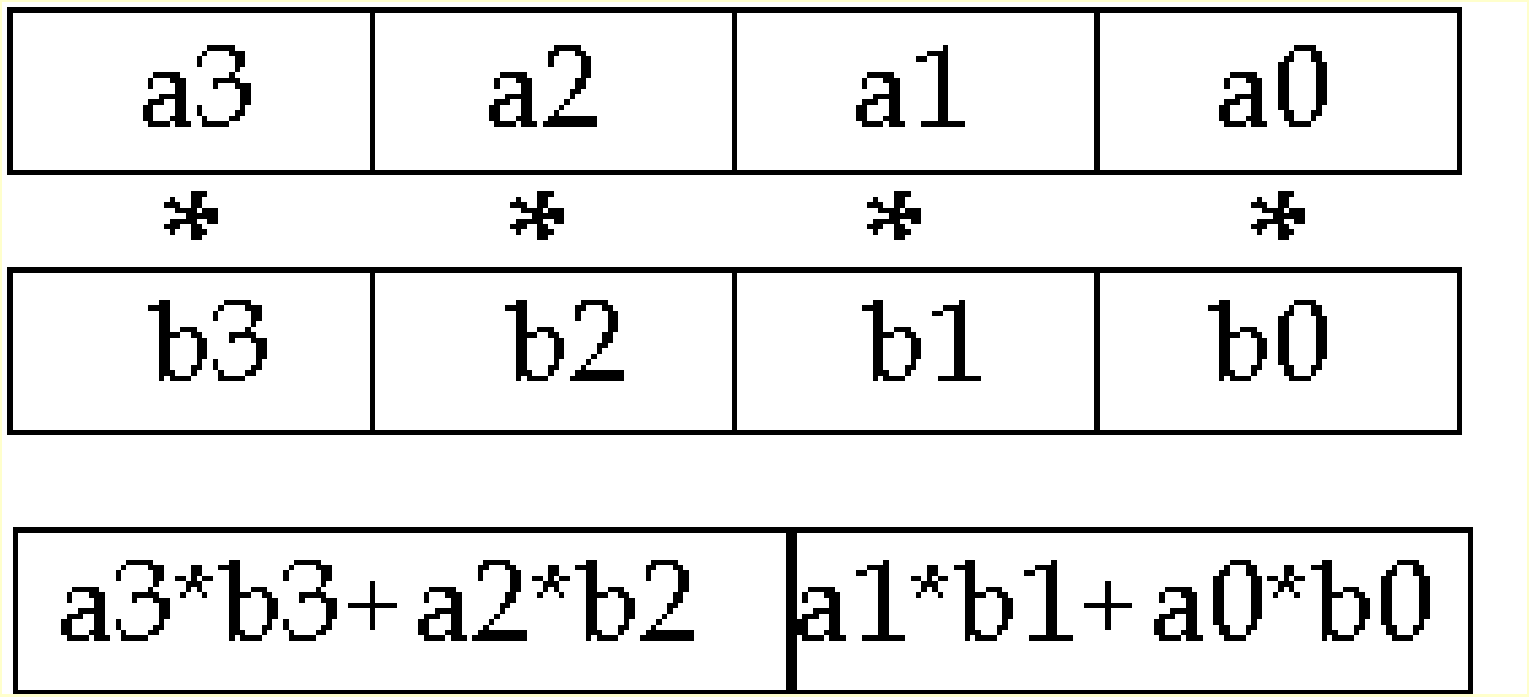
http://tommesani.com/index.php/simd/39-mmx-conversion.html

Ex: adun o constanta la un vector

```
char d[]={15, 15, 15, 15, 15, 15, 15, 15};8 bytes
char vect[]={75,76,78,...,97,98}; 24 bytes
_asm{
    movq mm1, d              ; mm1=d
    mov cx, 3                ; cx=3
    mov esi, 0               ; esi=0
L1: movq mm0, vect[esi]      ; mm0=vect (8 valori)
    paddb mm0, mm1           ; mm0=vect+d
    movq vect[esi], mm0      ; vect=vect+d
    add esi, 8               ; esi=esi+8
    loop L1                  ; if cx≠0 go to L1
    emms                     ; exit MMX
}
```

Multiply-Accumulate

- Operaţiile MAC (* +) sunt fundamentale pentru mulţi algoritmi de prelucrare a semnalelor ca *: produse de vectori, produse de matrici, filtre FIR, IIR, FFT, DCT etc.*



Packed Compare

- Nu există flaguri noi de condiţie
- *Nici unul din flagurile de condiţie nu este afectat* de această instrucţiune
- Rezultatele pot fi folosite ca măşti ptr. selectarea diferitelor intrări folosind operaţii logice, *eliminând salturile.*

23	45	16	34
gt ?	gt ?	gt ?	gt ?
31	7	16	67
0000h	FFFFh	0000h	0000h

Conditional Select

If (cond) True
Ra := Rb
else Ra := Rc

Presupunem că Rx=11111...111b, dacă conditia este adevarată
şi Rx=00000...000b, dacă condiţia este falsă.
Ra poate fi calculat cu expresia logică :

$$Ra = (Rb \text{ AND } Rx) \text{ OR } (Rc \text{ NAND } Rx)$$

• Chroma Keying, de exemplu, demonstrează cum selecţia condiţionată, folosind setul de instrucţiuni MMX, elimină predicţiile greşite ale salturilor, în plus realizând mai multe operaţii de selecţie în paralel.

• Text overlay (subtitrarea) pe video ar putea beneficia de această tehnică

OBS. Chroma keying este o tehnică pentru a mixa (fuziona) două imagini sau cadre împreună, în care o culoare (sau o mica gama de culori) de la o imagine este eliminată (sau făcută transparent), şi dezvăluie o altă imagine în spatele ei.

Conditional Select – aplicatie Chroma Keying

x y New_image

```
for (i=0; i<image_size; i++) {
if (x[i] == Blue) new_image[i] =y[i];
                  else new_image[i] = x[i];
}
```

green

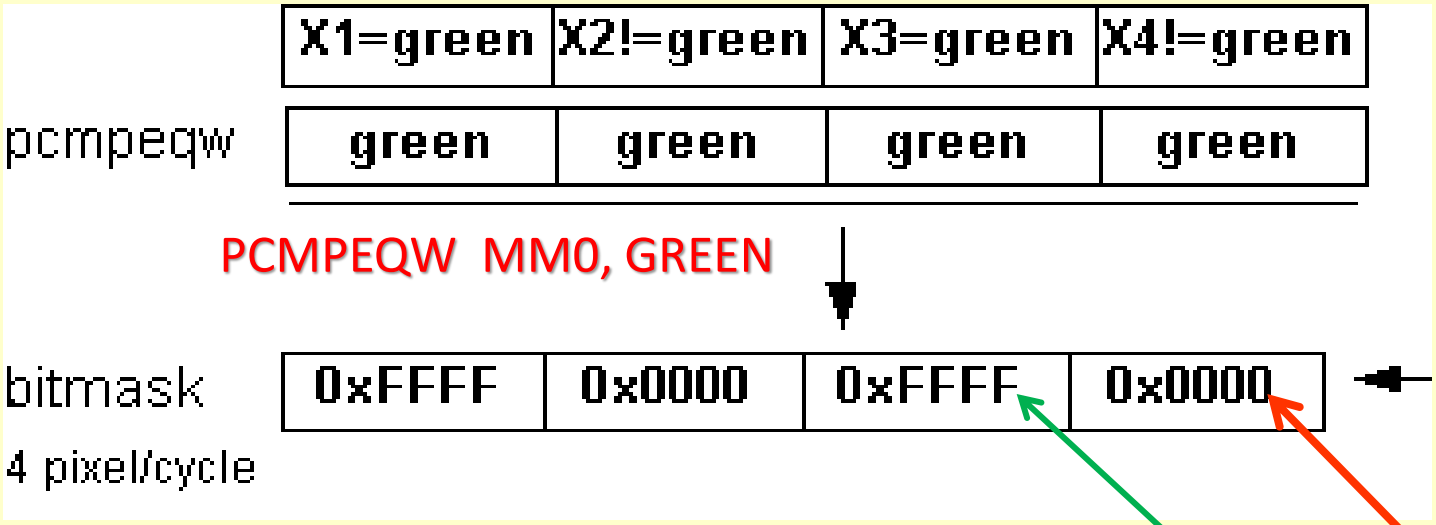
Procesarea

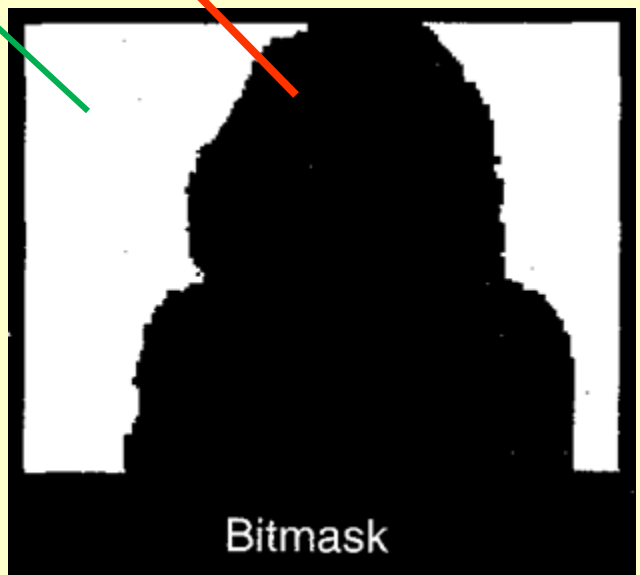
- se iau pixelii din imaginea x (cu femeia pe fond verde)
- O instructiune de comparare construieşte o mască pentru aceste date.
- Această mască este o secvenţă de biti care sunt fie toţi “1” fie toţi „0”
- Acum ştim care este fondul nedorit şi ce dorim să păstrăm

Creare Masca

(packed compare for equality)

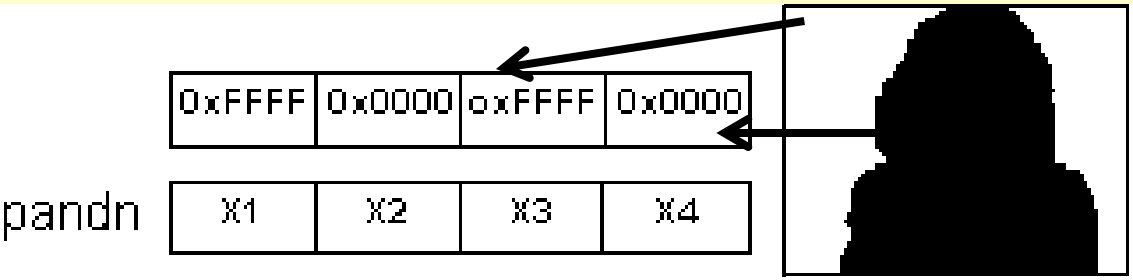
Presupunem că pixelii alternează green/not_green

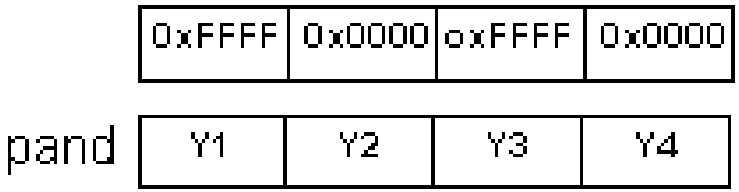


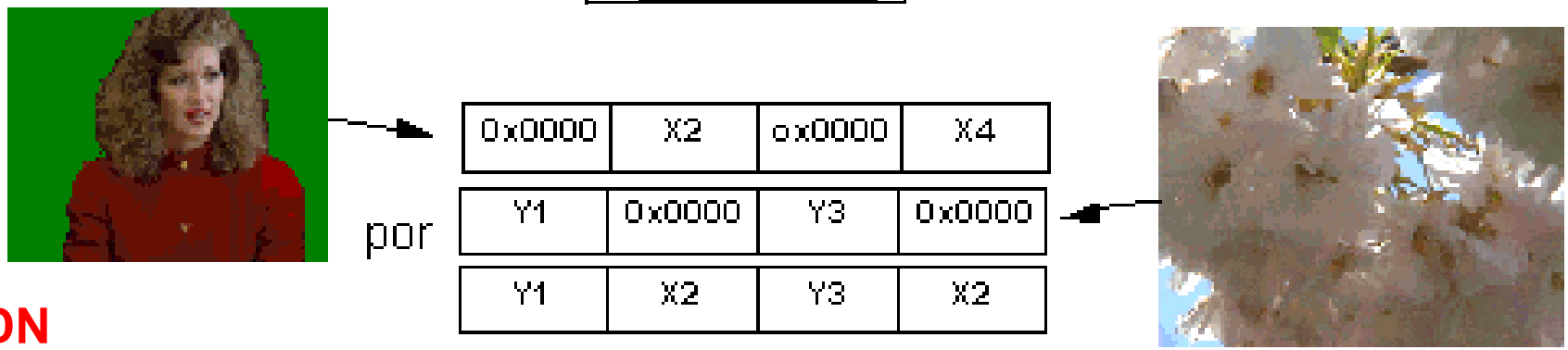


Conditional Select

Combine: PANDN, PAND, POR





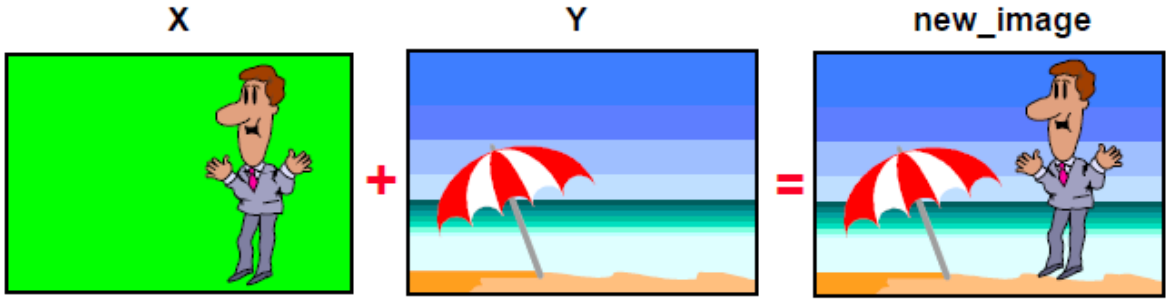


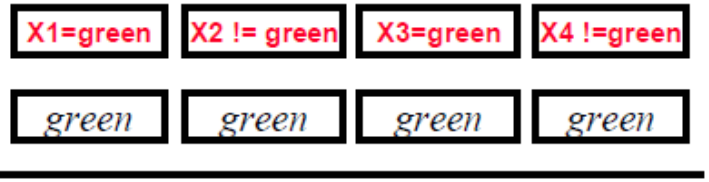
PANDN

DEST ← (NOT DEST) AND SRC;

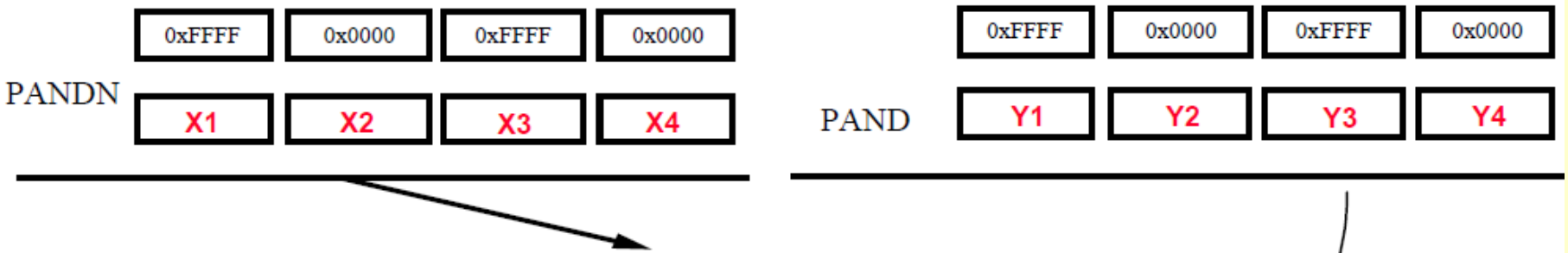
Example: Conditional Select / Branch Removal

- Packed Comparison (PCMPCC) and the logical instructions enable conditional select operations in parallel and without data dependent branches.

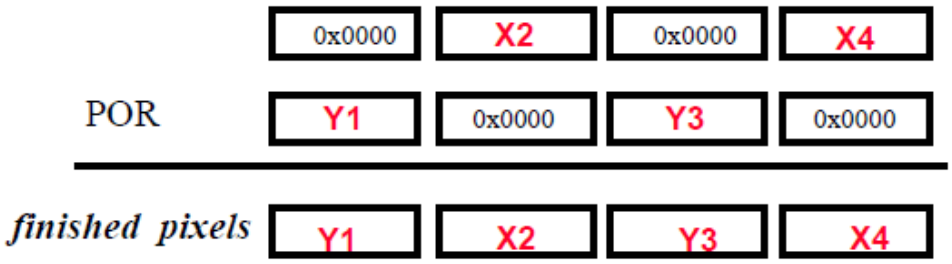





```
if (X[i] != green) then
    new_image[i] = X[i]
else
    new_image[i] = Y[i]
```




```
MOVQ      MM1, X
PCMPEQW   MM1, GREEN
MOVQ      MM2, MM1
PANDN     MM1, X
PAND      MM2, Y
POR       MM1, MM2
MOVQ      New, MM1
```



Conditional Select !!!

- **Eliminarea ramificaţiilor (salturi conditionate)**

• Fără tehnologia MMX fiecare pixel este procesat separat si necesită un salt condiţionat

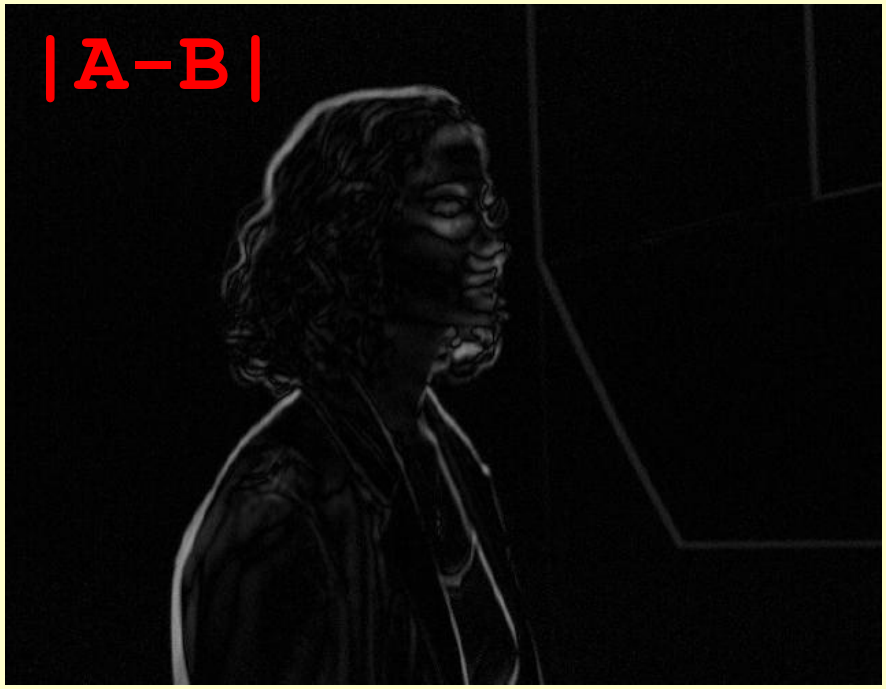
• Folosind setul MMX: (8 pixeli x 8 biti) pot fi procesaţi în paralel si nu implică folosirea salturilor !!!

```
MOVQ      MM1, X
PCMPEQW   MM1, GREEN
MOVQ      MM2, MM1
PANDN     MM1, X
PAND      MM2, Y
POR       MM1, MM2
MOVQ      New, MM1
```

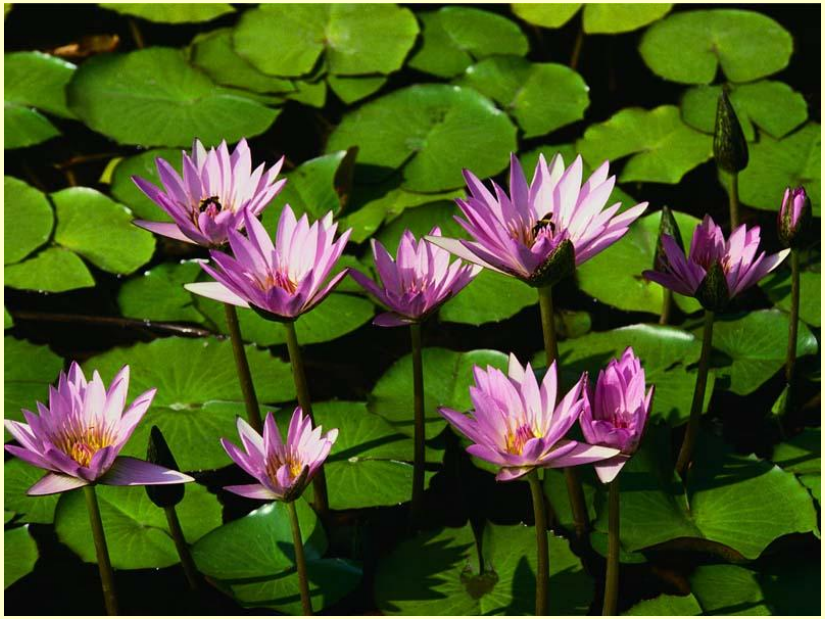
Aplicatie: diferenta cadrelor

(A-B) or (B-A)

```
MOVQ      mm1, A     //move 8 pixels din imag. A
MOVQ      mm2, B     //move 8 pixels of imag. B
MOVQ      mm3, mm1   // mm3=A
PSUBSB    mm1, mm2   // mm1=A-B
PSUBSB    mm2, mm3   // mm2=B-A
POR       mm1, mm2   // mm1=|A-B|
```



Ex. *image fade-in-fade-out*



A

B

$$A*\alpha+B*(1-\alpha) = B+\alpha(A-B)$$

$\alpha=0.75$

$\alpha=0.5$

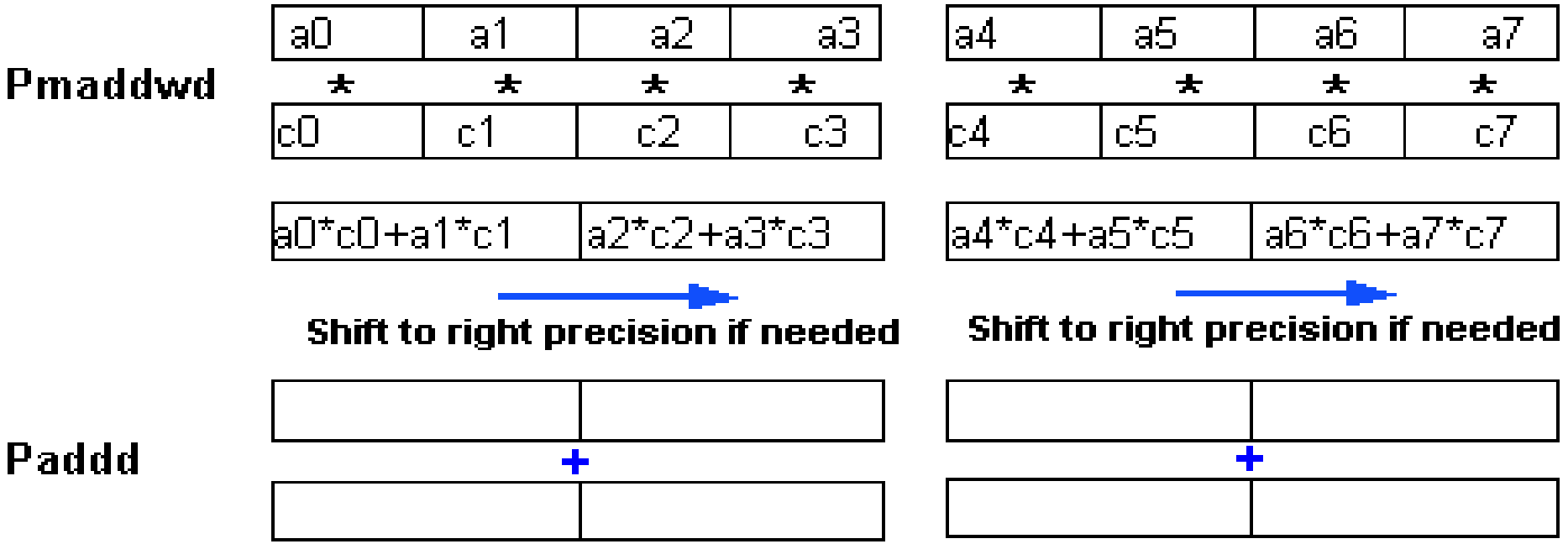
$\alpha=0.25$

$\alpha=0$

Vector Dot Product (suma de produse)

- Vector dot product este unul dintre cei mai utilizaţi algoritmi în prelucrarea semnalelor (imagini, audio, video şi voce)
- PMADD realizează 4 * si 2 + simultan, iar cuplat cu PADD, 8 operatii multiply-accumulate pot fi executate: 2 PMADD şi 2 PADD

$$x = \sum a(i) * c(i)$$



Vector Dot Product - Comparatie

	fara MMX	cu MMX
Load	16	4
Multiply	8	2
Shift	8	2
Add	7	1
Miscellaneous	--	3
Store	1	1
Total	40	13

MMX

Pro si Contra

Pro:

- Realizeaza mai multe calcule simultan
- Poate utiliza diferite tipuri de date

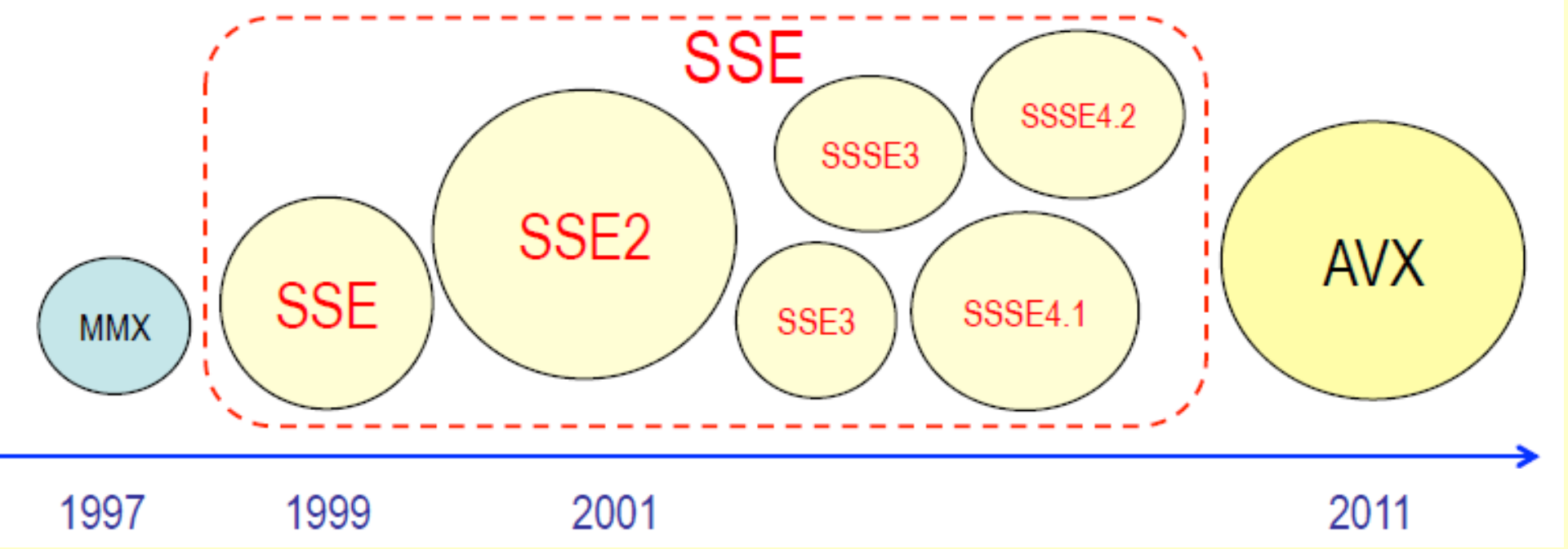
Contra:

- Datorită partajării registrelor nu poate executa in acelaşi timp operaţii *MMX şi în virgulă flotantă*

```
                        ; void EndMMX()
                        ; global _EndMMX
_EndMMX:  emms          ; Allow CPU to use floating point
          ret
```

- Necesita compilatoare noi ptr execuţia instructiunilor MMX
- Operează numai cu valori intregi

MMX, SSE and AVX



AVX – Advanced Vector Extensions

- Operatiile MMX sunt limitate la valori intregi
  – extensiile SSE and AVX permit si operatii cu date in virgula mobila
- MMX sunt rar folosite in procesoarele moderne – se folosesc in schimb SSE sau AVX

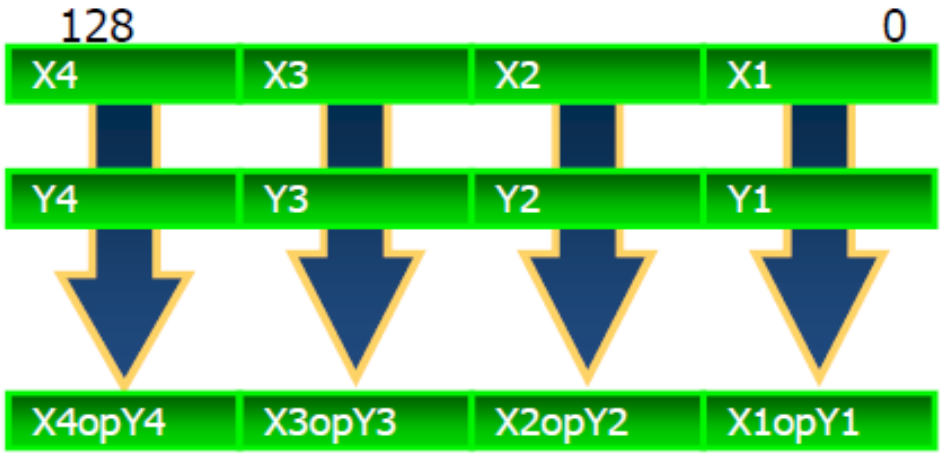
- **SSE –** Streaming SIMD Extension s-a introdus la Pentium III, si este suportat de majoritatea procesoarelor moderne

- Registre de 128 biti - suport ptr. operatii single-precision floating point
- **SSE2** – Streaming SIMD Extension 2 s-a introdus la Pentium 4
- Suporta si operatii double-precision floating point
- Alte extensii: SSE3, SSSE3, SSE4.1, SSE4.2 …

- Instructiunile cu vectori au fost introduse treptat si extinse in mai multe generatii de procesoare

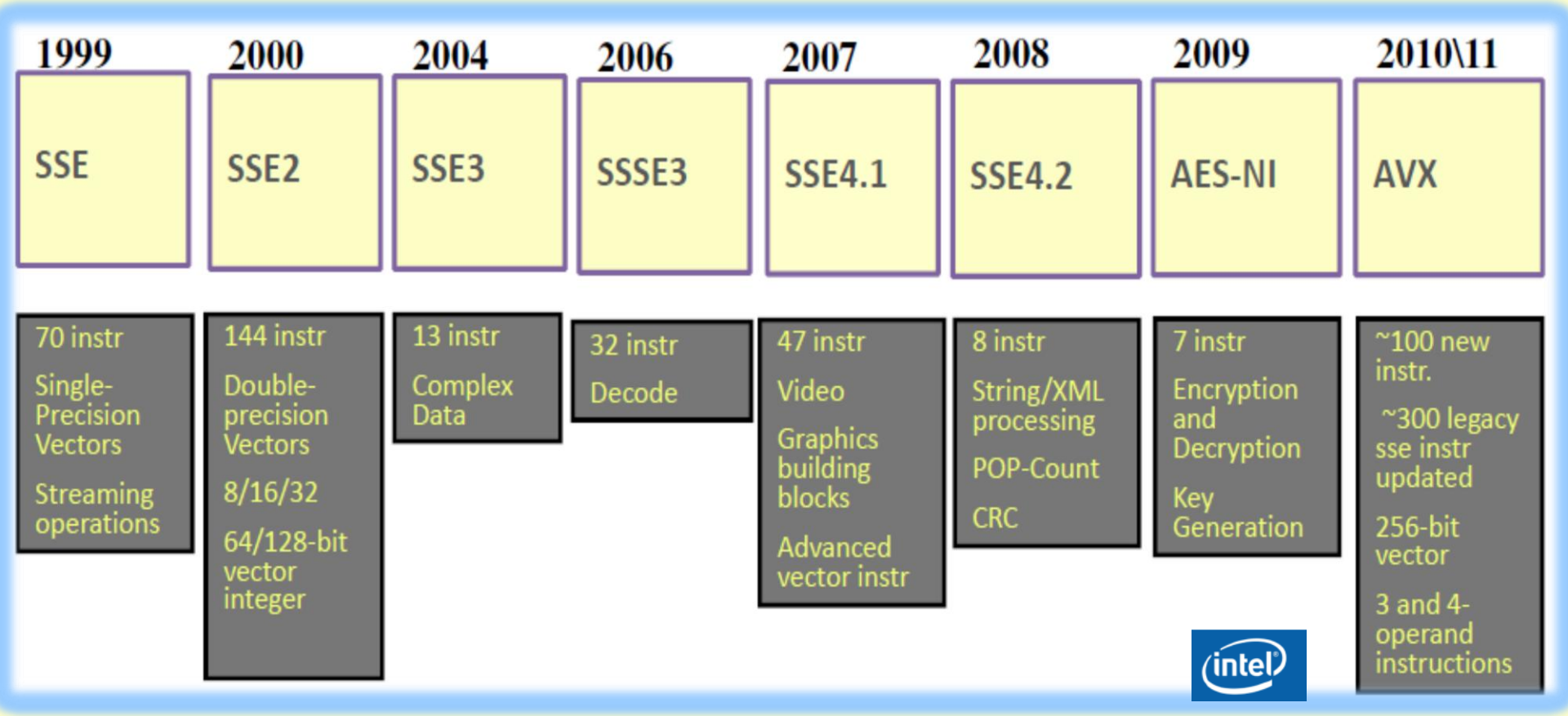
– extensiile SSE impreuna includ peste 400 de instructiuni

- Programarea cu AVX este similara, dar lungimea vectorilor si numele instructiunilor difera



Intel® SSE
Vector size: 128bit
Data types:
8,16,32,64 bit integers
32 and 64bit floats
VL: 2,4,8,16
Sample: Xi, Yi bit 32 int / float

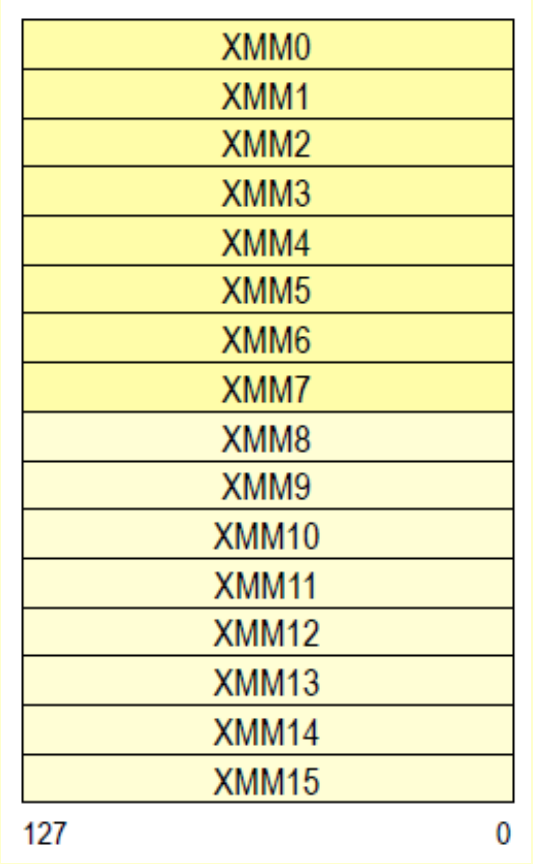
Evolutia continua a setului SSE



Registre vector SSE

- SSE introduce un nou set de 8/16 reg. vector de 128-biti , XMM0-XMM15
  - 8 reg. XMM in mod non 64-biti
  - 16 reg. XMM in mod 64-biti
- Reg. XMM sunt registre fizice reale nu reg. alias (ca MM0-MM7)
  - independente de reg. de uz general sau de reg.FPU/MMX
- Registrele XMM pot fi accesate in mod 32-bit, 64-bit sau 128-bit
  - Numai ptr. operatii pe date, *nu pe adrese*
- MXCSR este un reg. de 32 biti de control si stare

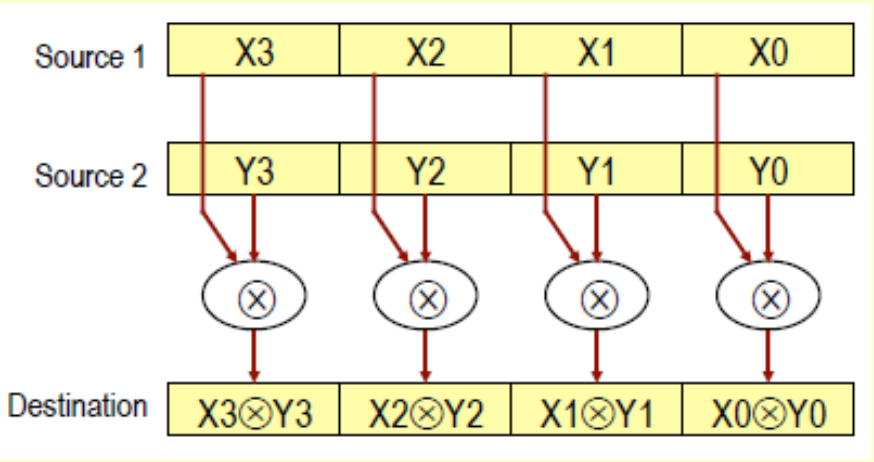
•Permite operații pe numere impachetate în virgulă mobilă simplă precizie

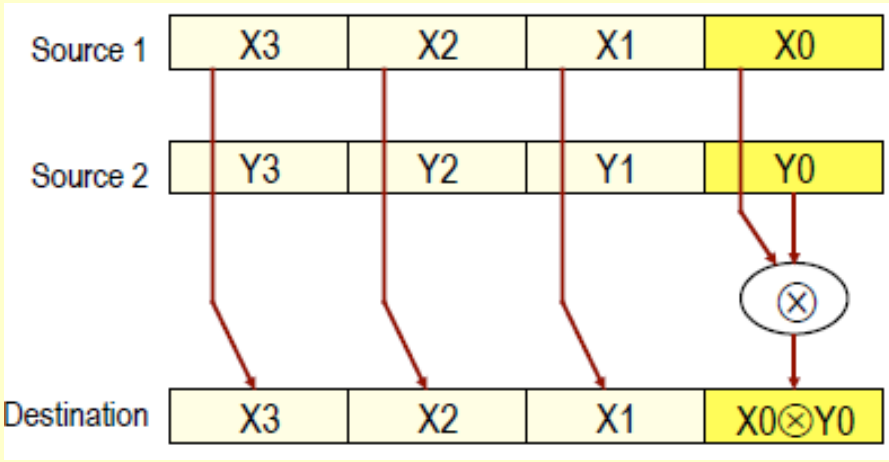


Instructiuni SSE

- Extensia originala SSE adauga 70 de instructiuni noi la setul de instructiuni x86
  - 50 ptr. operatii SIMD single-precision floating-point (SPFP)
  - 12 ptr. Operatii SIMD cu intregi
  - 8 ptr. Control cache
  - extensiile urmatoare au adaugat mai multe instructiuni
- Supporta ambele tipuri de instructiuni packed/ scalar SPFP
  - operatii pe valori impachetate de 32-bit floating-point (sufix P)
  - operatii pe scalar de 32-bit floating-point (32 LSB), (sufix S)

- ***Compilatoarele folosesc instr. SSE scalare ptr. operatiile flotante in locul instructiunilor x87 ale FPU***





Tipuri de date ptr. SSE/SSE2

Figure 1. SSE/SSE2 data types

SSE2

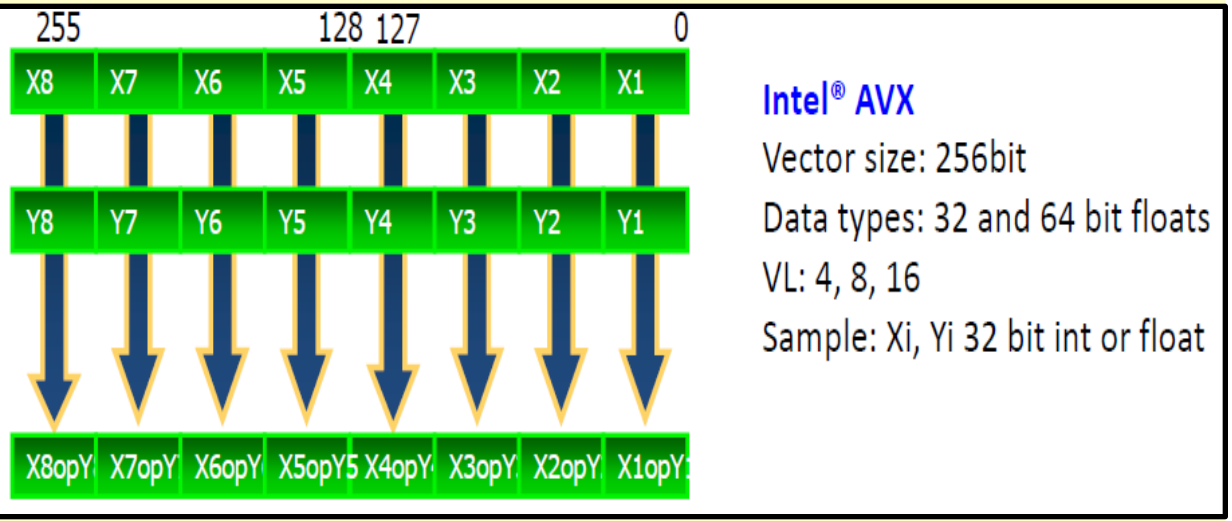
■ Streaming SIMD Extension 2 (Pentium 4)
- extinde SSE și înlocuiește instrucțiunile vectoriale MMX cu intregi

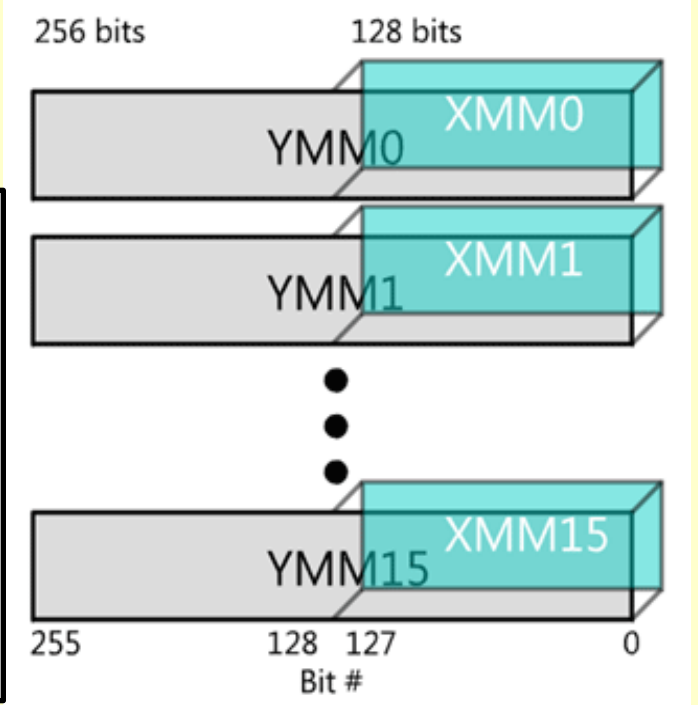
■ Extinde setul SSE cu suport pentru:
- valori în virgulă mobilă cu **dublă precizie** impachetate
- valori **întregi impachetate** - adaugă peste 70 de instrucțiuni noi setului de instrucțiuni

■ Funcționează pe entități de 128 biți din registrele XMM
- datele trebuie aliniate la limitele de 16 biți, atunci când sunt stocate în memorie
- restricțiile privind alinierea au fost ulterior relaxate
- au fost introduse instrucțiuni de load/store nealiniate

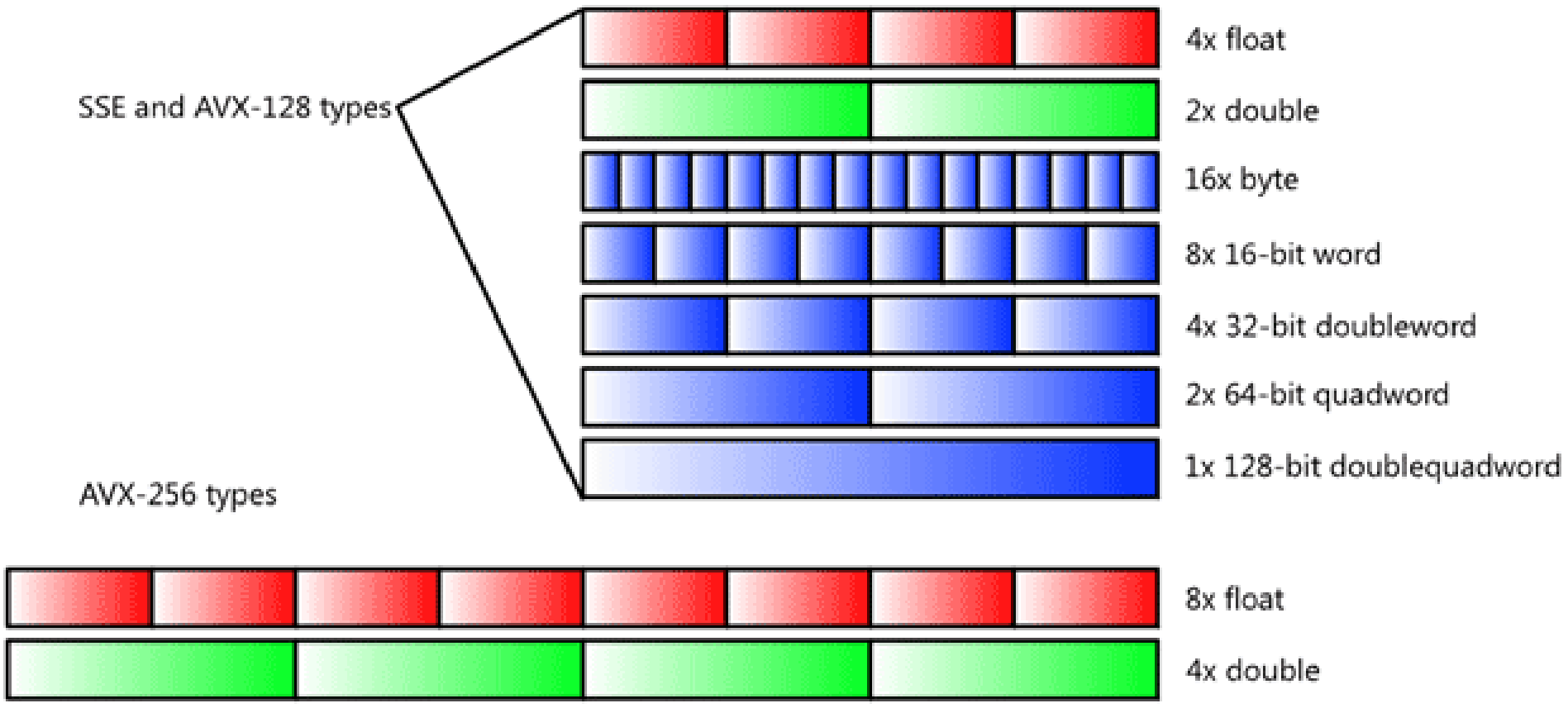
AVX – Advanced Vector Extensions

– suportat de procesorul Intel Sandy Bridge(2011) si urmatoarele
– extinde reg. vectoriali la 256 biti, YMM0 – YMM15
- Instrucțiunile SSE funcționează pe jumătatea inferioară a registrelor YMM
■ Introduce noi instrucțiuni cu trei operanzi
- o destinație și doi operanzi sursă: c = a ⊗ b
■ AVX-512 este suportat în microarhitectura Knight Landing
■ Procesorul Intel Xeon Phi acceptă o unitate de procesare vectorială de 512biți
- nu este compatibil cu SSE sau AVX





Tipuri de date ptr. AVX



Caracteristici ale AVX

Key Intel® Advanced Vector Extensions (Intel® AVX) Features

KEY FEATURES	BENEFITS
• Wider Vectors – Increased from 128 to 256 bit – Two 128-bit load ports	• Up to 2x peak FLOPs (floating point operations per second) output with good power efficiency
• Enhanced Data Rearrangement – Use the new 256 bit primitives to broadcast, mask loads and permute data	• Organize, access and pull only necessary data more quickly and efficiently
• Three and four Operands: Non Destructive Syntax for both AVX 128 and AVX 256	• Fewer register copies, better register use for both vector and scalar code
• Flexible unaligned memory access support	• More opportunities to fuse load and compute operations
• Extensible new opcode (VEX)	• Code size reduction

Intel® AVX is a general purpose architecture, expected to supplant SSE in all applications used today

intel Software

Exemple de generare cod

```
static double A[1000], B[1000],
              C[1000];
void add() {
  int i;
  for (i=0; i<1000; i++)
    if (A[i]>0)
      A[i] += B[i];
    else
      A[i] += C[i];
}
```

```
.B1.2::
  movaps    xmm2, A[rdx*8]
  xorps     xmm0, xmm0
  cmpltpd   xmm0, xmm2
  movaps    xmm1, B[rdx*8]
  andps     xmm1, xmm0
  andnps    xmm0, C[rdx*8]
  orps      xmm1, xmm0
  addpd     xmm2, xmm1
  movaps    A[rdx*8], xmm2
  add       rdx, 2
  cmp       rdx, 1000
  jl        .B1.2
```

SSE2

```
.B1.2::
  vmovaps   ymm3, A[rdx*8]
  vmovaps   ymm1, C[rdx*8]
  vcmpgtpd  ymm2, ymm3, ymm0
  vblendvpd ymm4, ymm1,B[rdx*8], ymm2
  vaddpd    ymm5, ymm3, ymm4
  vmovaps   A[rdx*8], ymm5
  add       rdx, 4
  cmp       rdx, 1000
  jl        .B1.2
```

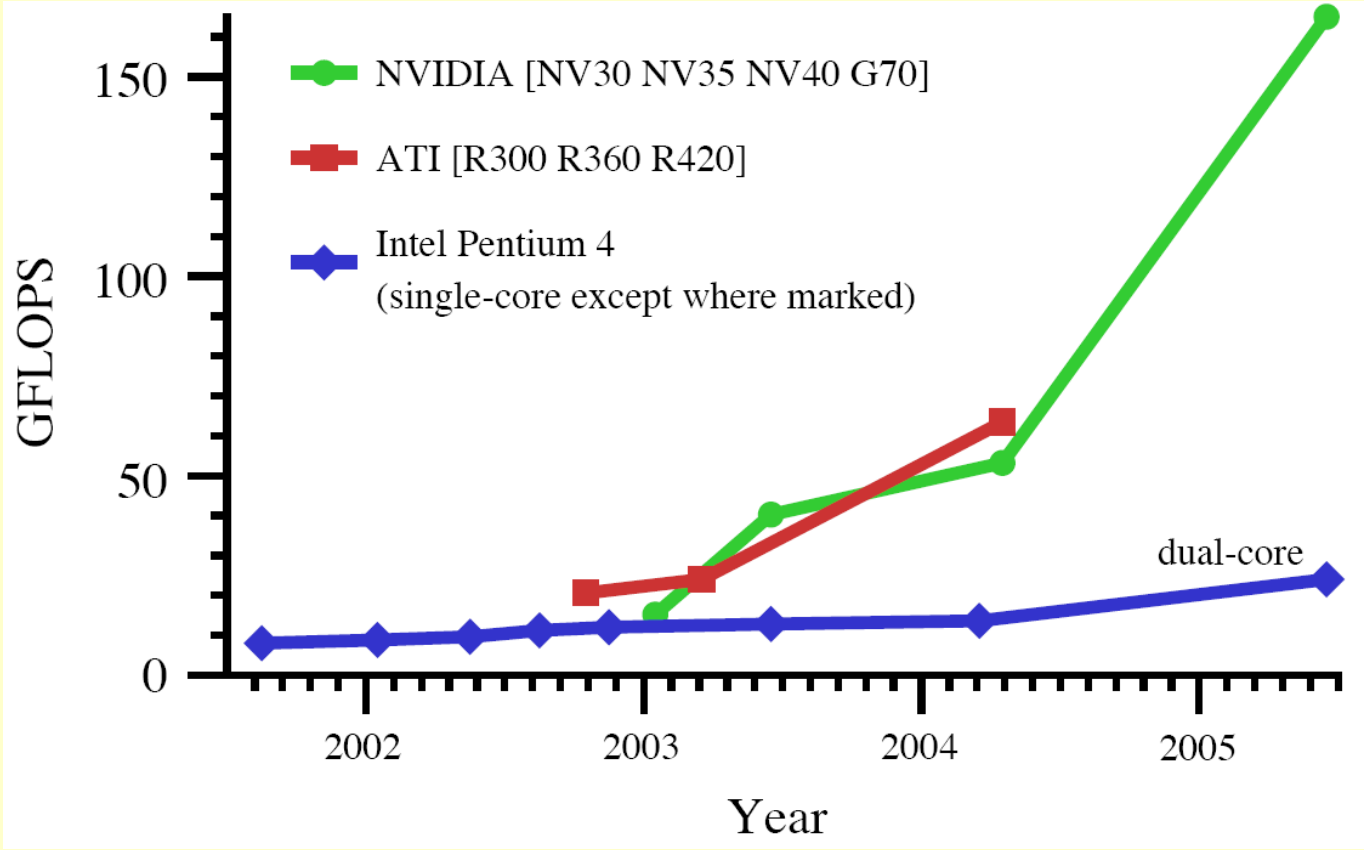
AVX

```
.B1.2::
  movaps    xmm2, A[rdx*8]
  xorps     xmm0, xmm0
  cmpltpd   xmm0, xmm2
  movaps    xmm1, C[rdx*8]
  blendvpd  xmm1, B[rdx*8], xmm0
  addpd     xmm2, xmm1
  movaps    A[rdx*8], xmm2
  add       rdx, 2
  cmp       rdx, 1000
  jl        .B1.2
```

SSE4.1

Alte arhitecturi SIMD

- Graphics Processing Unit (GPU): nVidia 7800, 24 pipelines (8 vector/16 fragment)



Procesoare cu caracteristici SIMD

Intel

- **MMX:**
  - Pentium MMX
  - Pentium Pro
  - Pentium II
  - Celeron
- **SSE:**
  - Pentium III
  - Pentium IV
  - Celeron 2

AMD

- 3DNow!
  - K6-2
  - K6-3
- Advanced 3DNow!
  - Athlon
  - Duron