Arq. Org. de Computadores 2

Instruções SIMD

(Intel MMX / SSE)

Introdução

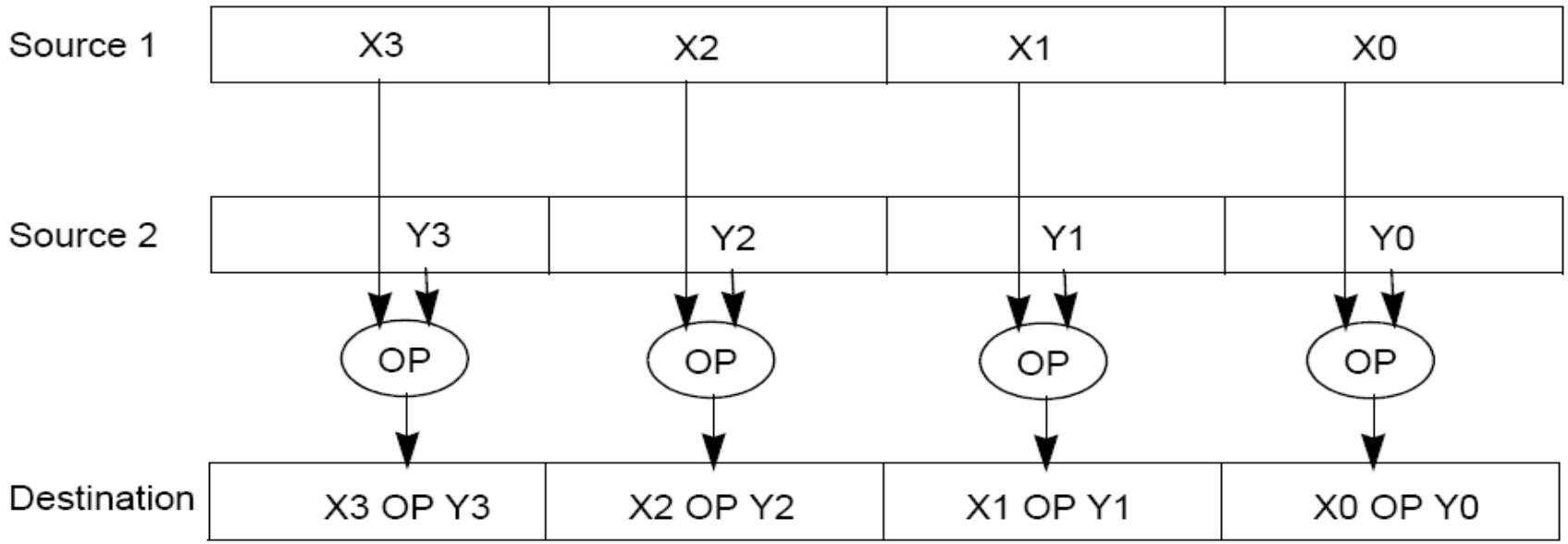
- Em muitos casos, apenas aumentar a taxa de clock não é suficiente p/ aumentar o desempenho geral da aplicação.
  - Algumas mudanças na arquitetura podem ser mais significantes (ex: pipeline/cache/SIMD).

- Na década de 90, a Intel analizou um grande número de aplicações de multimedia (imagem, som, video), e descobriu as seguites características nos códigos:
  - Uso de tipos de dados pequenos (pixels: 8-bits, áudio: 16-bits)
  - Repetição de operações
  - Paralelismo inerente
- Estas foram as bases para a inclusão de recursos de processamento SIMD em várias arquiteturas, incluindo a IA-32.

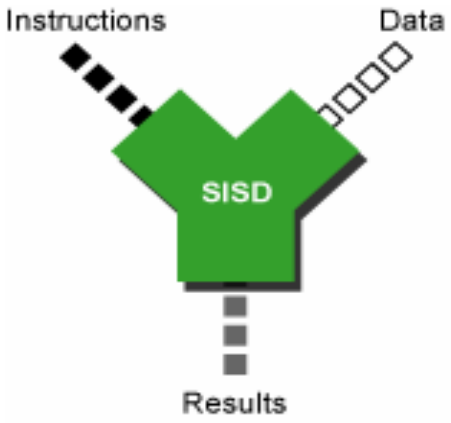
SIMD

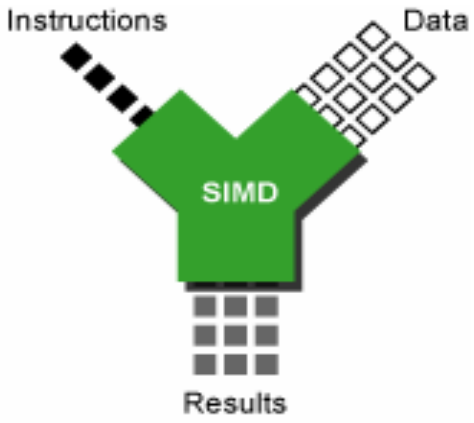
Arquiteturas SIMD (single instruction multiple data) executam a mesma operação em multiplos elementos de dados, paralelamente.

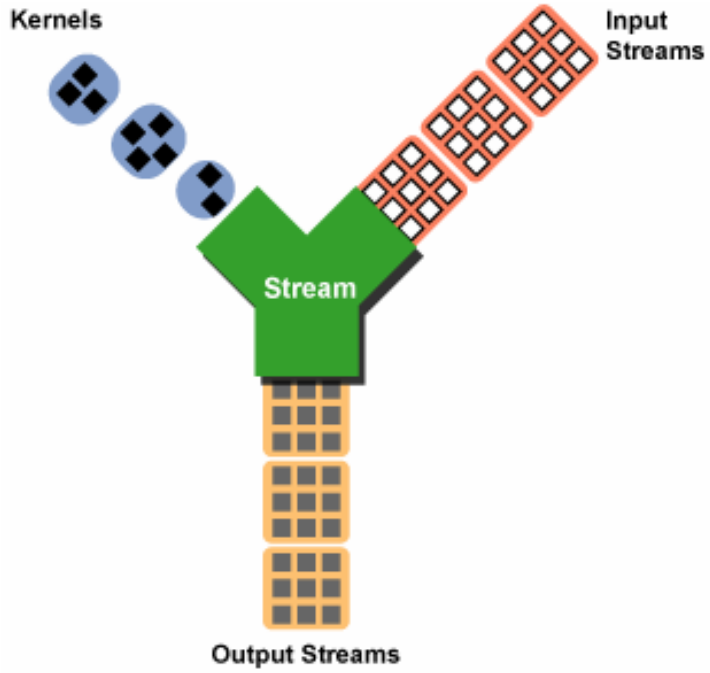
```
PADDW MM0, MM1
```



SISD/SIMD/Streaming







IA-32 SIMD

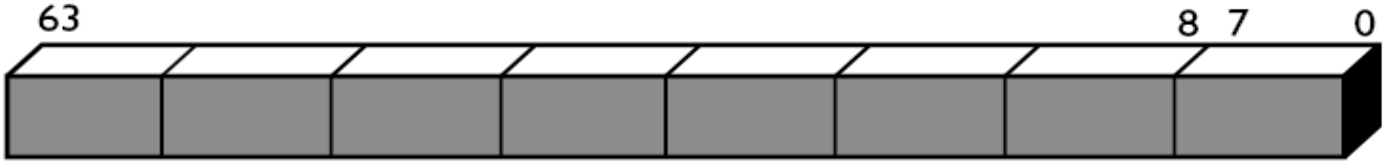
- MMX (Multimedia Extension): 1996 (Pentium with MMX and Pentium II).
- SSE (Streaming SIMD Extension): Pentium III.
- SSE2: Pentium 4.
- SSE3: Pentium 4 hyper-threading
- SSE4: 2007

MMX

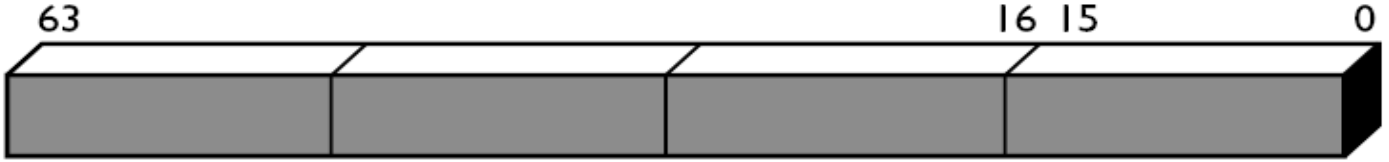
- Novo tipo de dados: 64-bits packed (empacotado).
- Operações com números inteiros apenas
- Reutiliza registradores de ponto flutuante
  - Não é possível misturar instruções MMX c/ operações de ponto flutuante (sem grande perda de desempenho).

MMX: Tipos de Dados

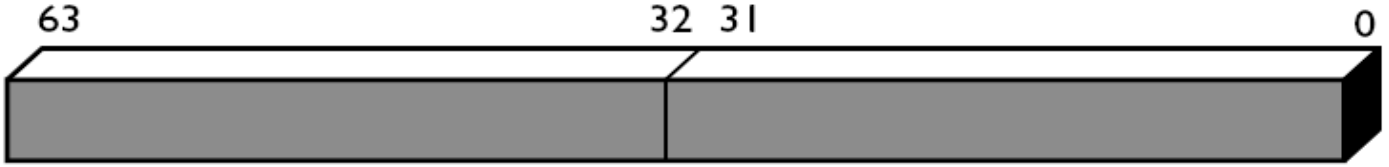
Packed Byte: 8 bytes packed into 64 bits



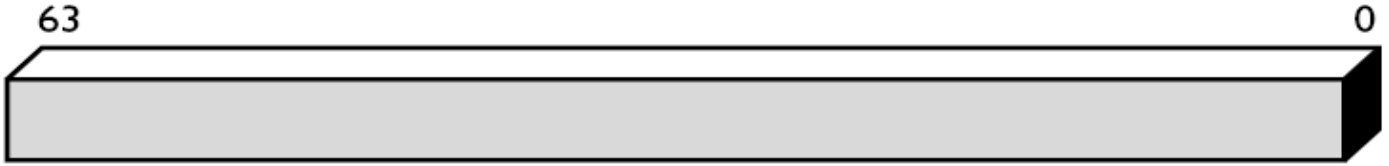
Packed Word: 4 words packed into 64 bits



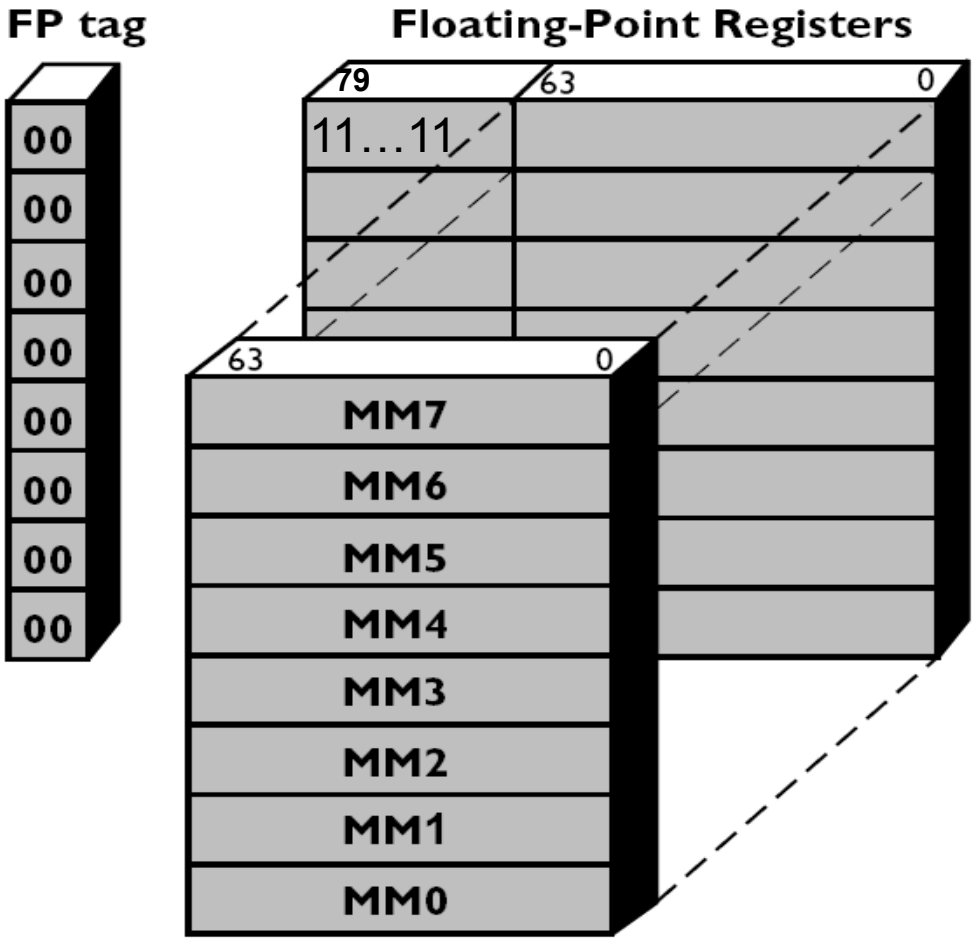
Packed Doubleword: 2 doublewords packed into 64 bits



Packed Quadword: One 64-bit quantity

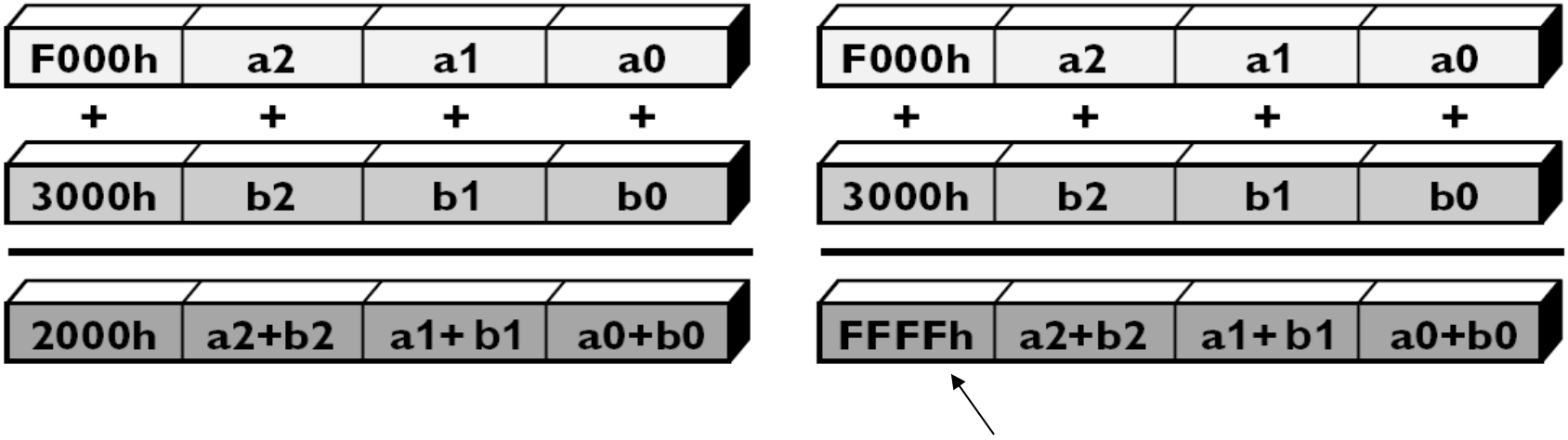


MMX: Integração c/ ISA



Aritmética de Saturação

- Útil em aplicações gráficas.
- Quando uma operação resulta em overflow (ou underflow), o resultado é o maior (ou menor) inteiro possível de se representar com um dado número de bits.
- Pode ser com ou sem sinal (signed and unsigned saturation)



Operação convenvional
(wrap-around)

Operação c/ saturação

MMX: Instruções

- 57 instruções MMX executam operações em paralelo sobre múltiplos elementos de dados, "empacotados" (packed) em tipos de dados de 64 bits.
  - **add, subtract,multiply**
  - **compare**
  - **shift**
  - **data conversion,**
  - **64-bit data move, 64-bit logical operation**
  - **multiply-add** p/ operações do tipo multiply-accumulate (MAC)
- Todas as instruções usam apenas registradores MMX (exceto moves).
- Maior abrangência p/ operações sobre 16-bits.

MMX: Instruções

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	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
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		

MMX instructions

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

Utilize quando for mudar o contexto, de MMX p/ Ponto Flutuante
(Operação demorada)

Aritmética

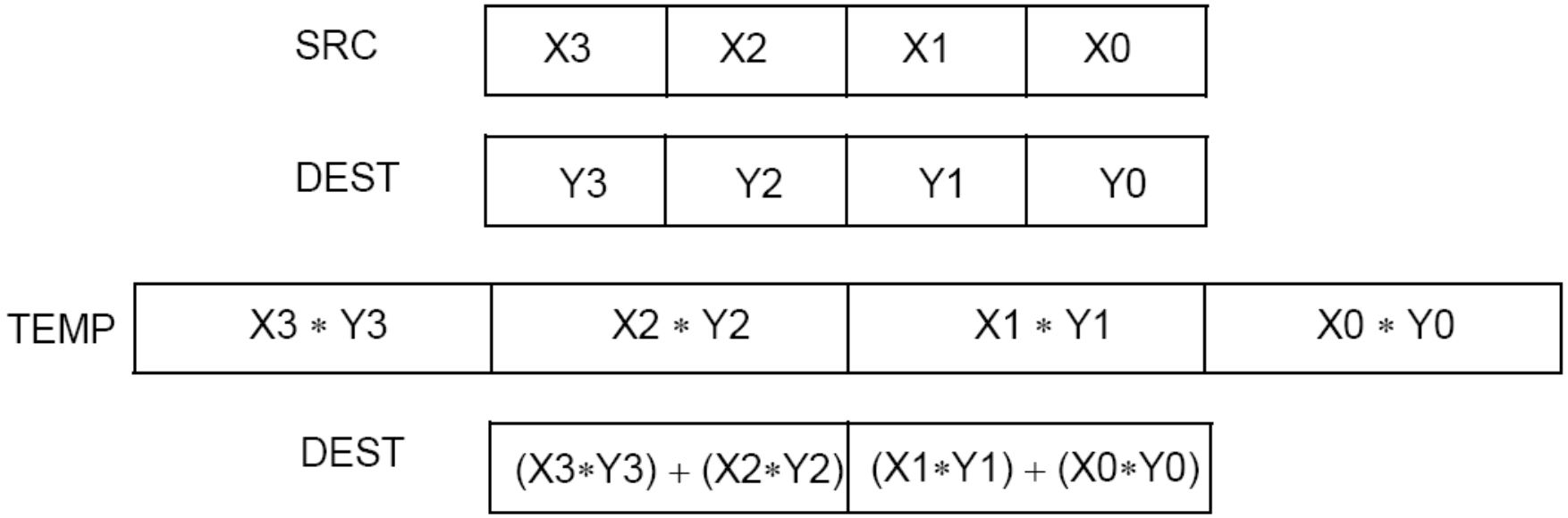
- **PADDB/PADDW/PADDD:** soma dois registradores contendo números empacotados.
  - Não modifica valor das EFLAGS, o programador deve garantir que overflow não ocorre.

- Multiplicação- dois passos:
- **PMULLW:** multiplica quatro words e armazena as 4 metades menos significativas (lo words) do resultado.
- **PMULHW/PMULHUW:** multiplica quatro words e armazena as 4 metades mais significativas (hi words) do resultado
  - ***PMULHUW** → unsigned.

Aritmética

Multiplica e Acumula: PMADDWD

DEST[31:0] ← (DEST[15:0] * SRC[15:0]) + (DEST[31:16] * SRC[31:16]);
DEST[63:32] ← (DEST[47:32] * SRC[47:32]) + (DEST[63:48] * SRC[63:48]);



Detectando MMX/SSE no processador

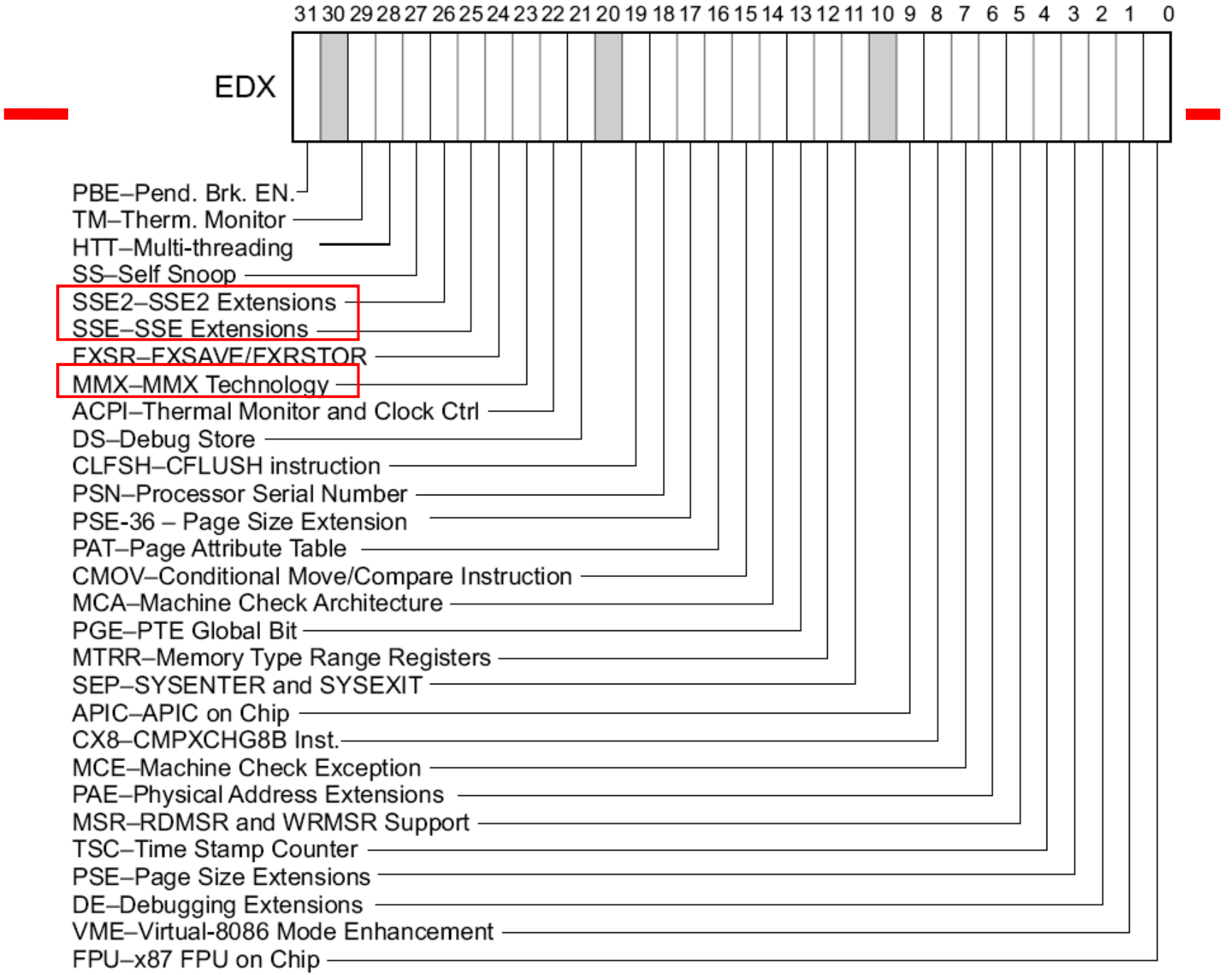
```
mov   eax, 1 ; request version info
cpuid;
test  edx, 00800000h ;(bit 23) MMX
         ; 02000000h (bit 25) SSE
         ; 04000000h (bit 26) SSE2
jnz   HasMMX
```

Instrução CPUID

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX	Maximum Input Value for Basic CPUID Information (see Table 3-13)
	EBX	"Genu"
	ECX	"ntel"
	EDX	"inel"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5)
	EBX	Bits 7-0: Brand Index Bits 15-8: CLFLUSH line size (Value ∗ 8 = cache line size in bytes) Bits 23-16: Maximum number of logical processors in this physical package. Bits 31-24: Initial APIC ID
	ECX	Extended Feature Information (see Figure 3-6 and Table 3-15)
	EDX	Feature Information (see Figure 3-7 and Table 3-16)
02H	EAX	Cache and TLB Information (see Table 3-17)
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information

⋮

⋮


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

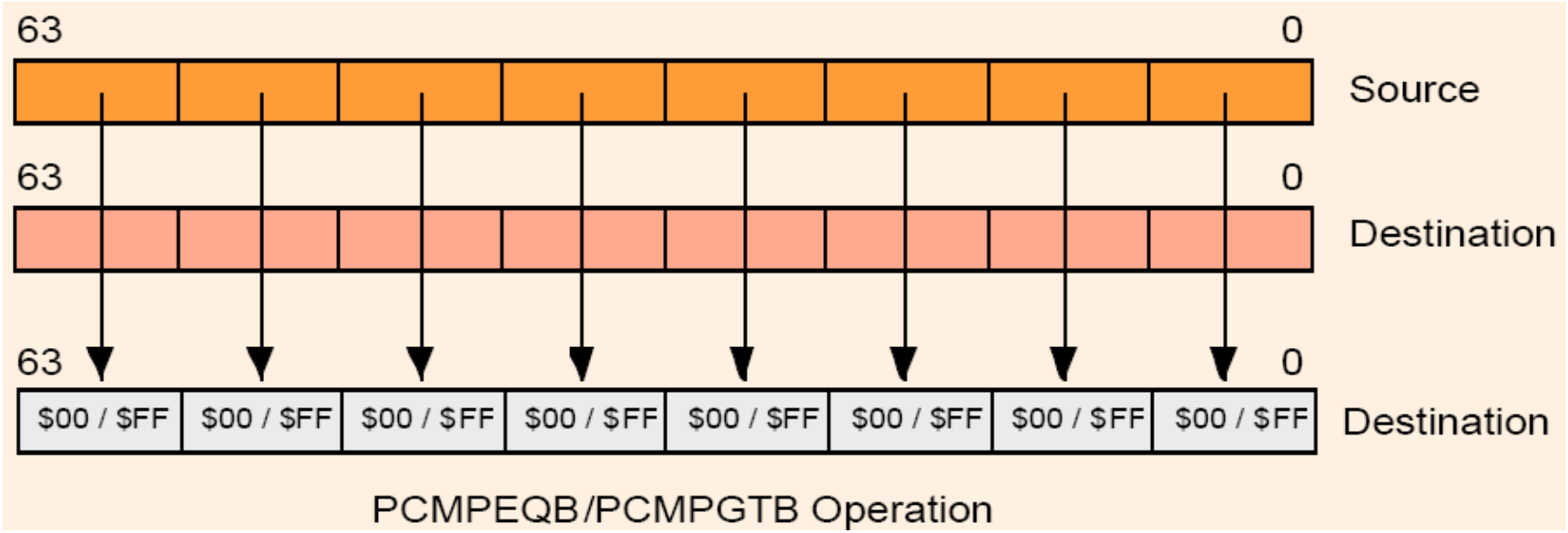
Exemplo: Somar uma constante a um Vetor

```
char d[]={5, 5, 5, 5, 5, 5, 5, 5};
char clr[]={65,66,68,...,87,88}; // 24 bytes
__asm{
     movq mm1, d
     mov cx, 3
     mov esi, 0
L1:  movq mm0, clr[esi]
     paddb mm0, mm1
     movq clr[esi], mm0
     add esi, 8
     loop L1
     emms
}
```

clr[i]= clr[i] + d[i%8]

Compração

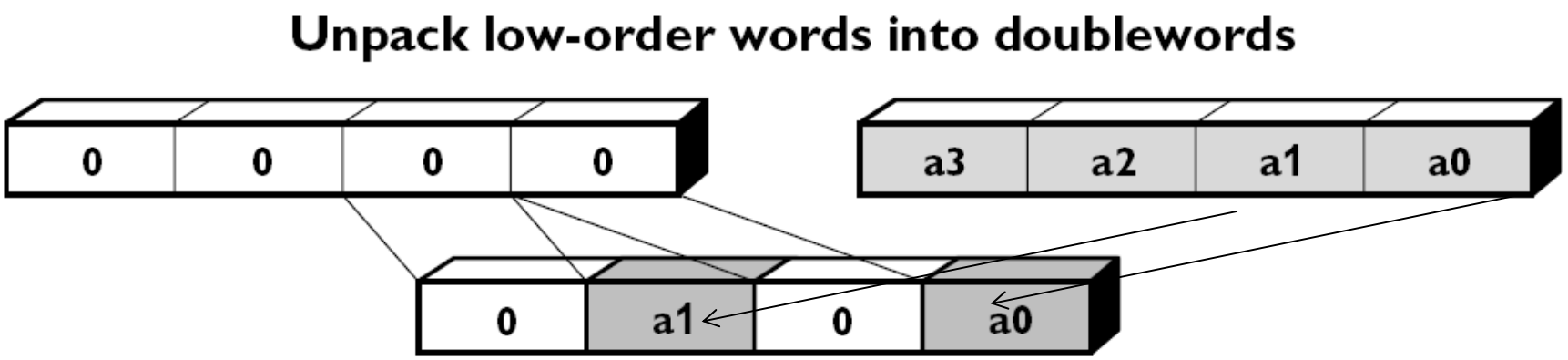
- Não modificam CFLAGS
- Teste Igualdade (EQ) → PCMPEQB
- Teste Maior que (GT) → PCMPGTB
- Teste Menor que (LT) → não disponível



PCMPEQB/PCMPGTB Operation

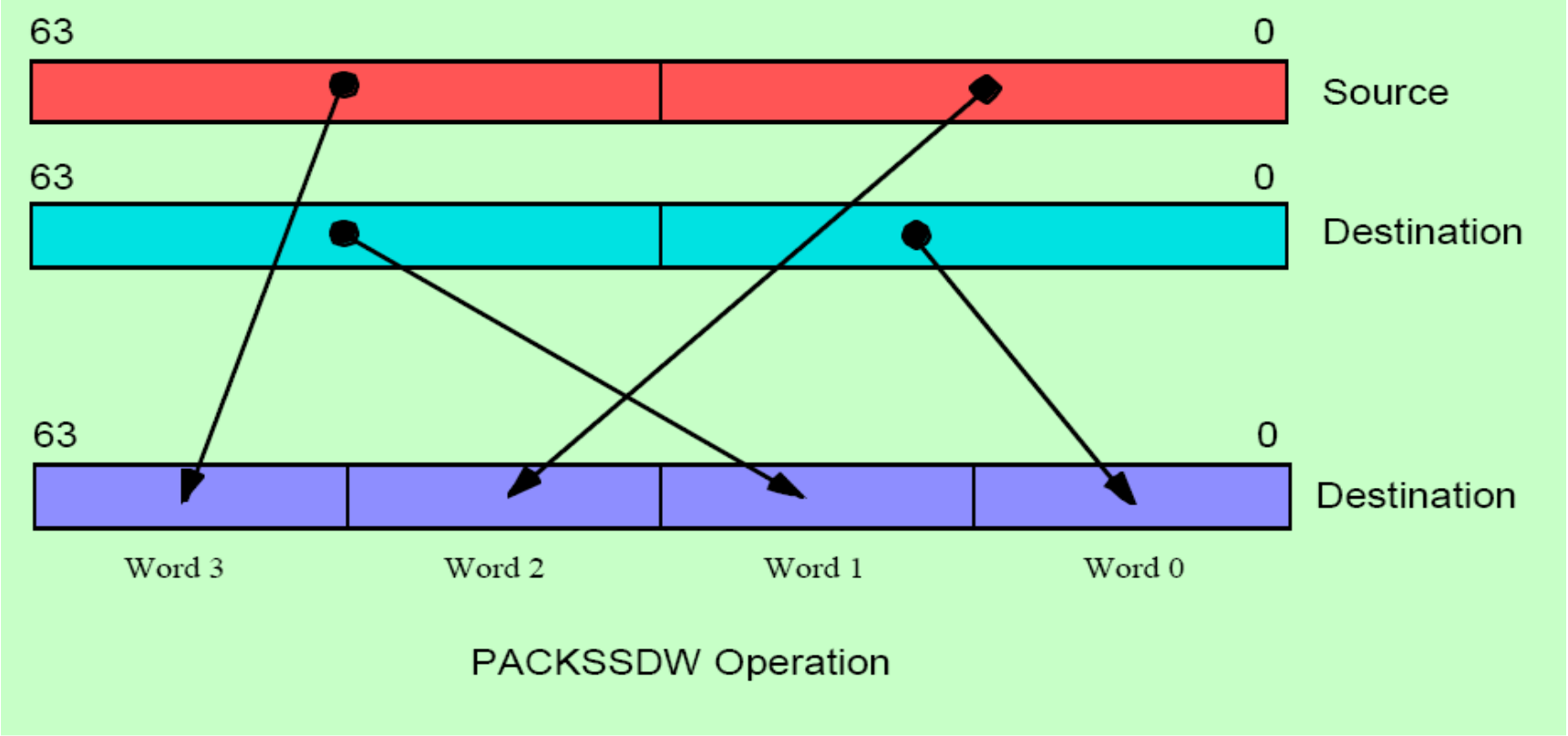
Convertendo tipos de dados

- Pack: converte um tipo de dado maior no próximo tipo menor
  Unpack: entrelaça dois operandos, o que na prática expande um tipo de dado menor em um maior.



Empacotamento c/ saturação e sinal

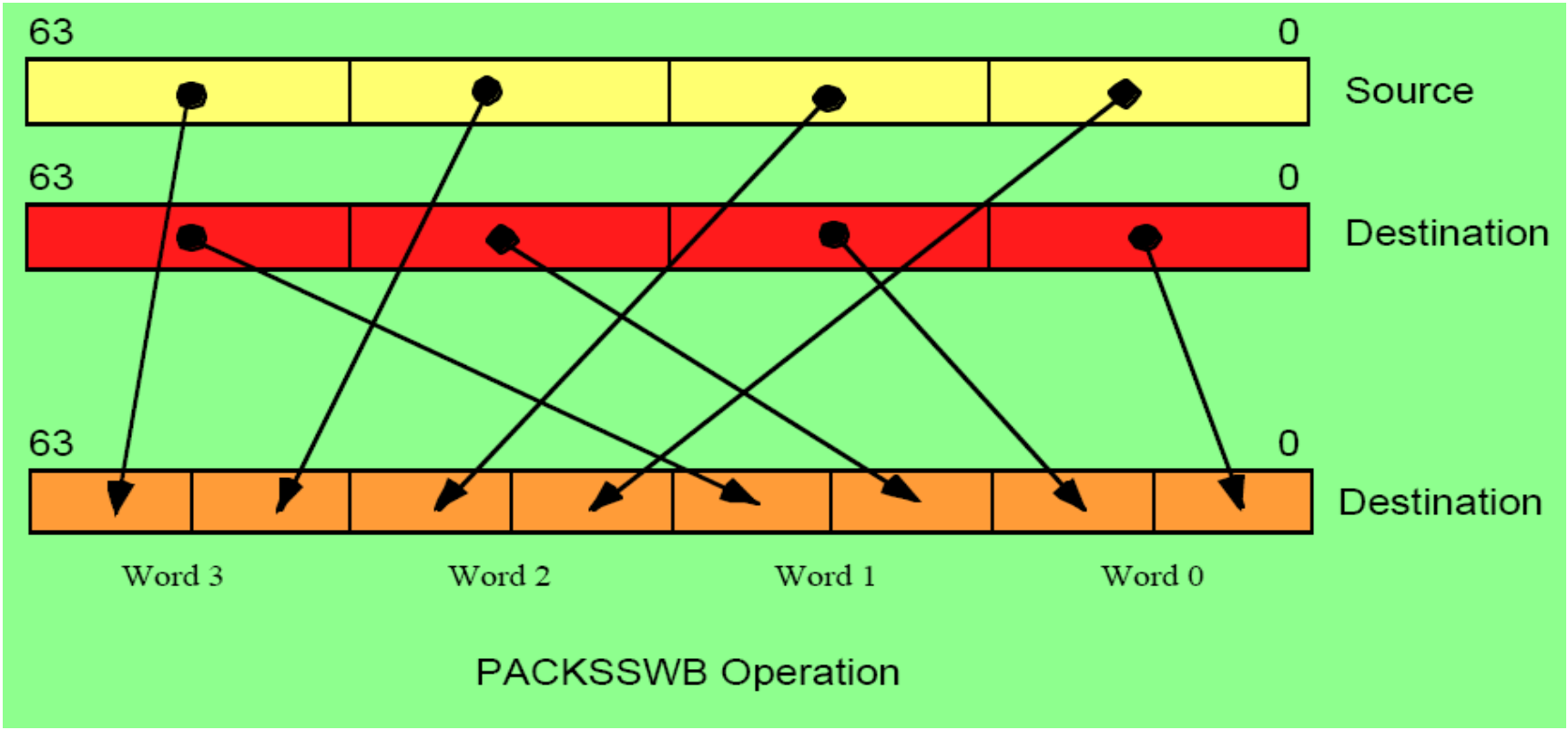
Empacotamento c/ saturação e sinal: PACKSSDW



PACKSSDW Operation

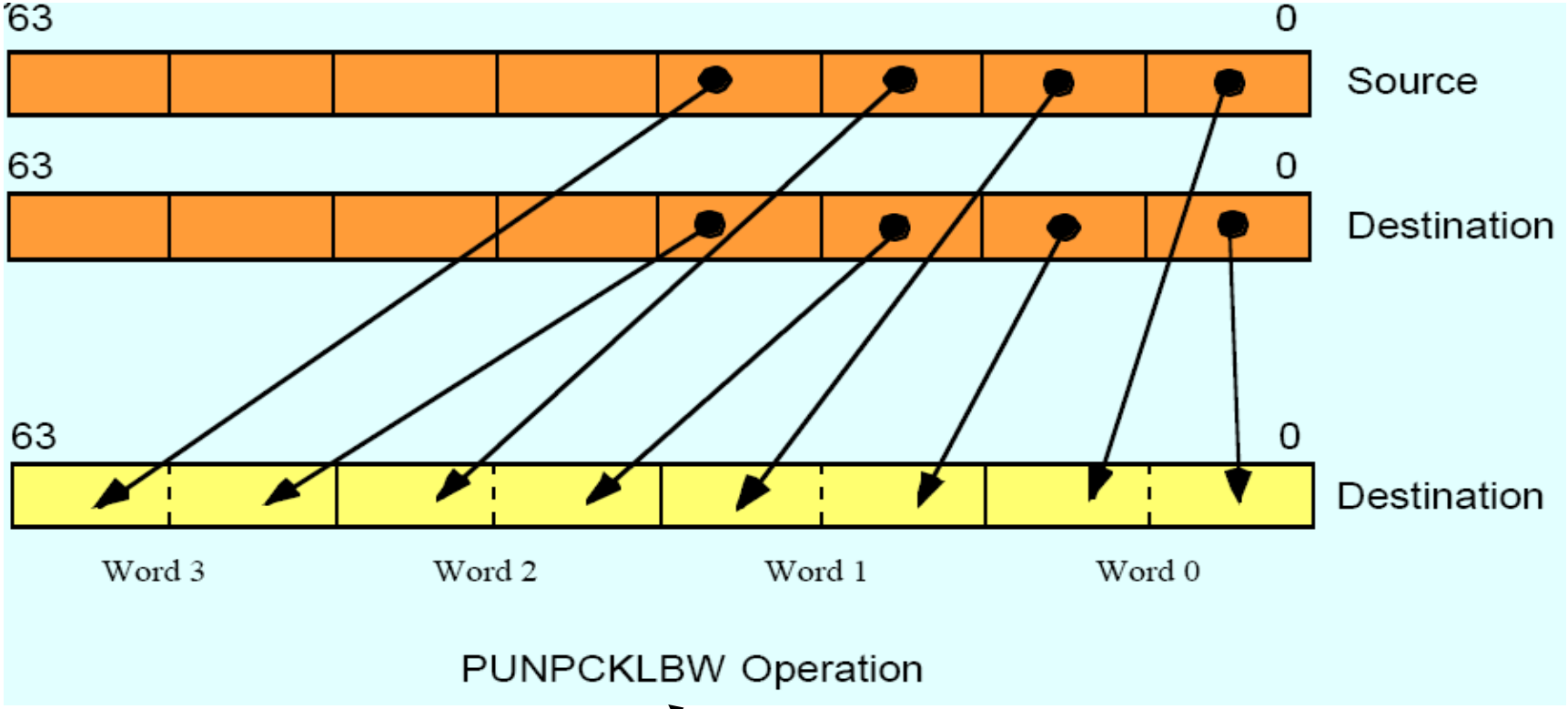
Empacotamento c/ saturação e sinal

Empacotamento c/ saturação e sinal: PACKSSWW



Desempacotando "Low"

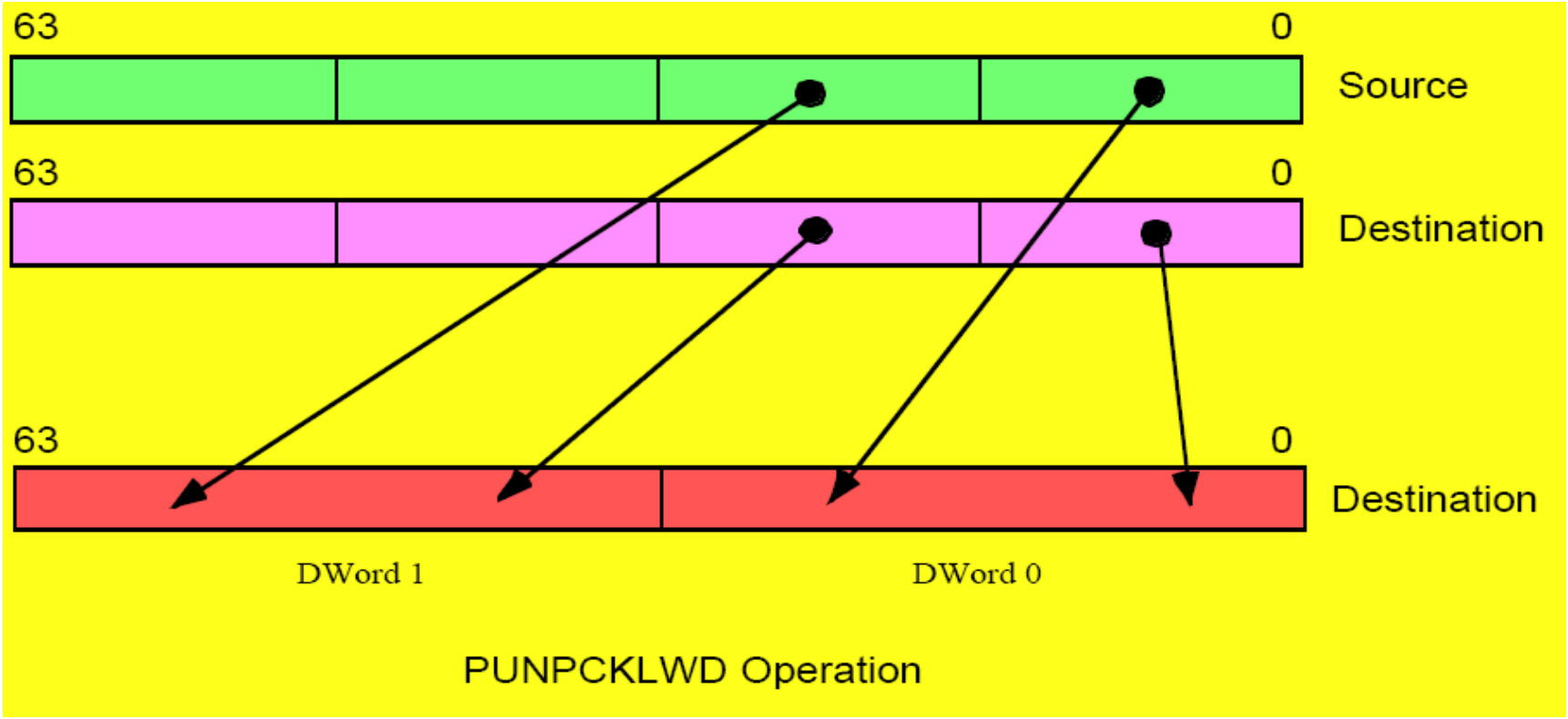
Desmpacotando 4 bytes → 4 words: PUNPCKLBW



Parte "low" (bits de ordem mais baixa)

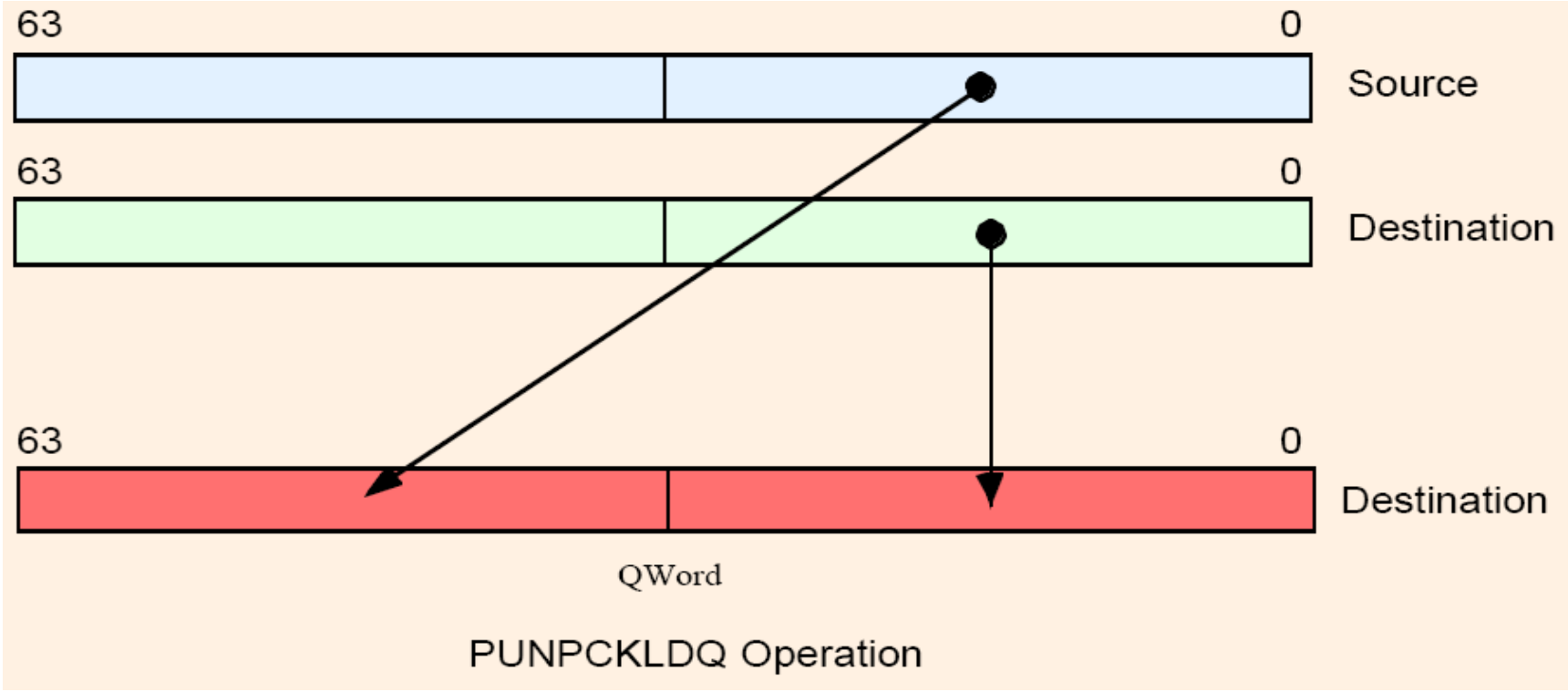
Desempacotando "Low"

Desmpacotando 2 words → 2 dwords: PUNPCKLWD



Desempacotando "Low"

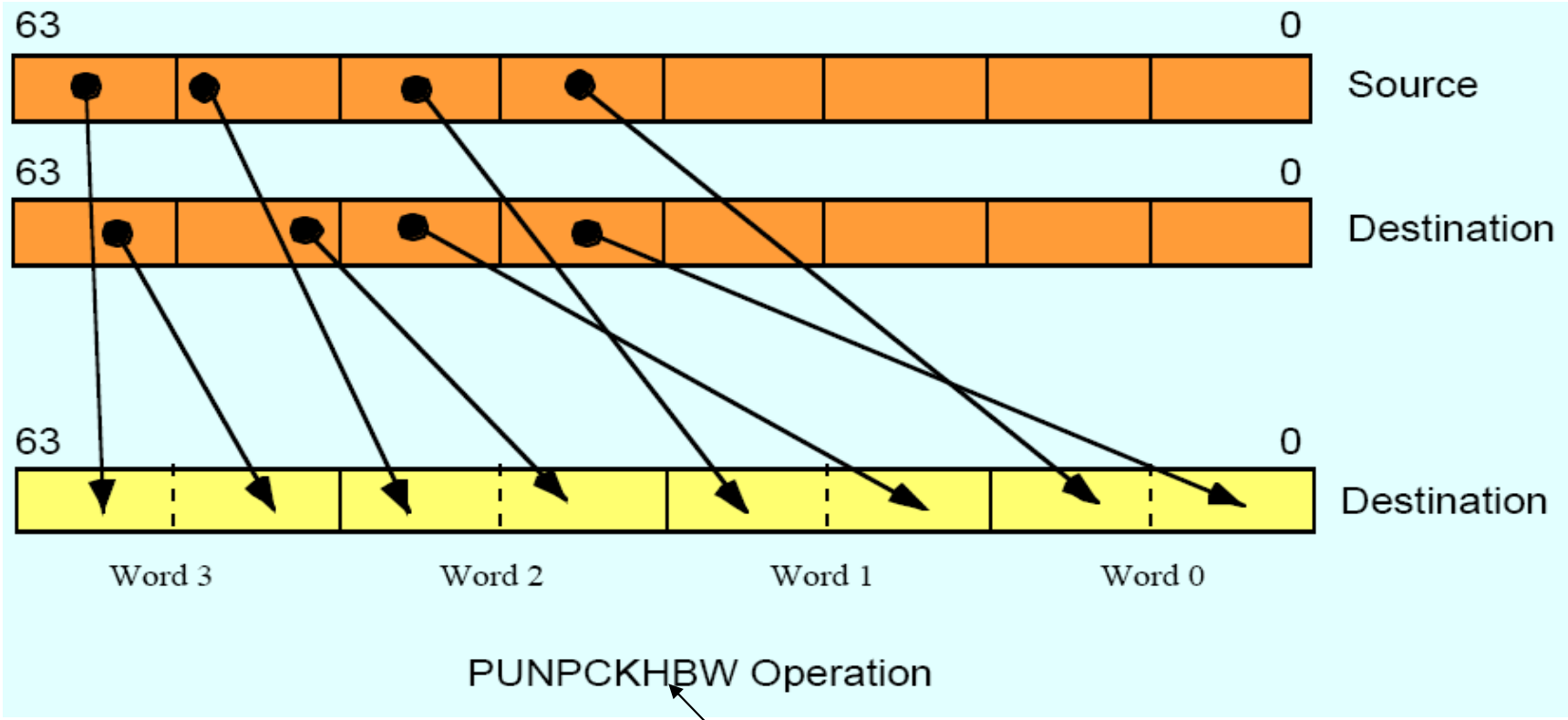
Desmpacotando 1 dword → 1 qword: PUNPCKLDQ



PUNPCKLDQ Operation

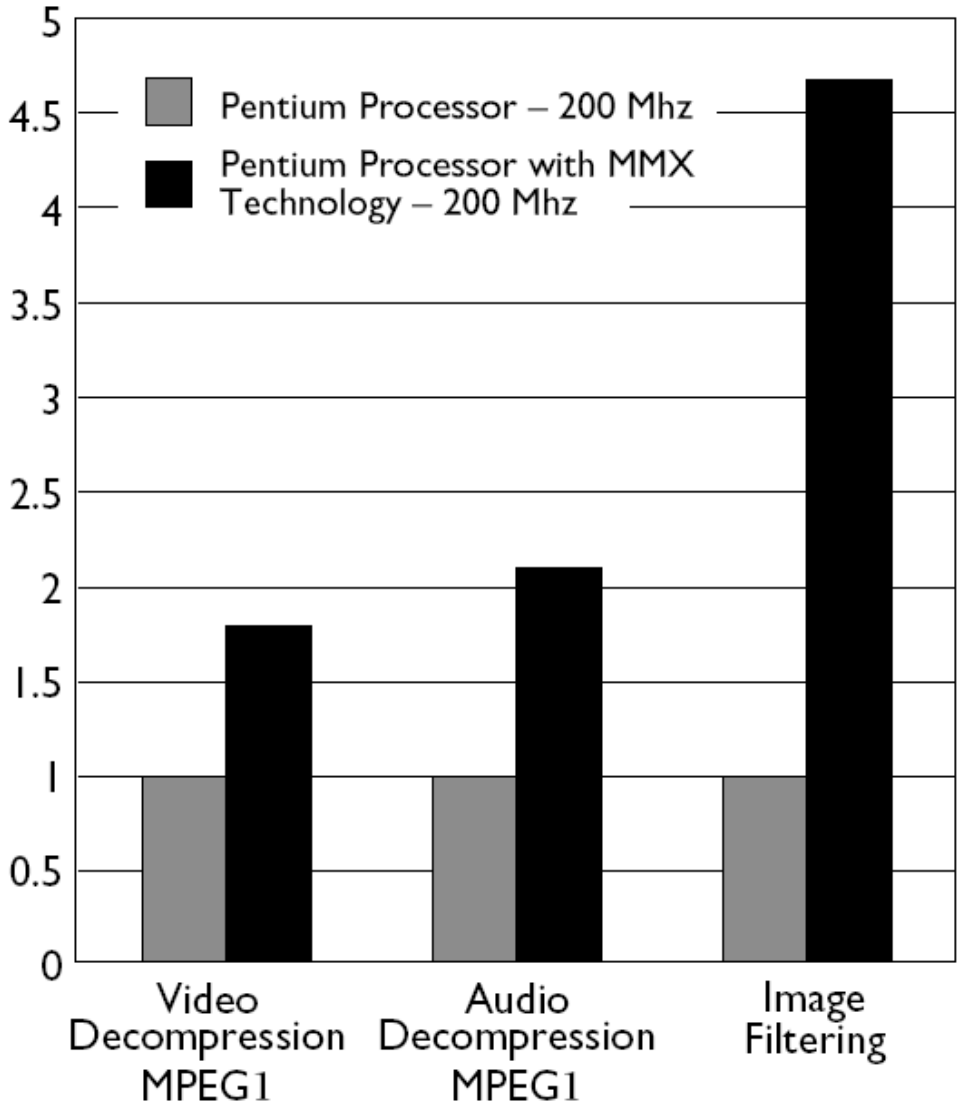
Desempacotando "High"

Desmpacotando 4 bytes → 4 words: PUNPCKHBW



Parte "high" (bits de ordem mais alta)

Aumento de desempenho (dados de 1996)



Programação SIMD

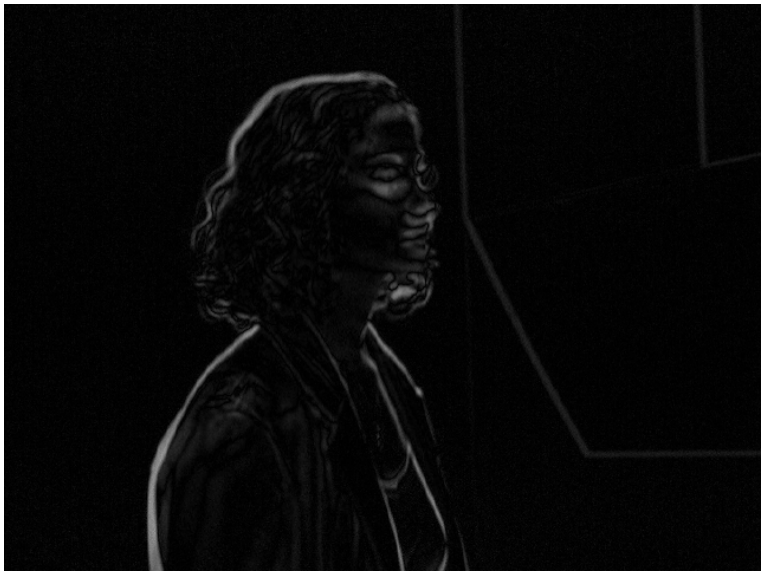
Pontos chave:

- Organização de dados eficiente (data layout)
- Eliminação de instruções de desvio (if-then-else)

Exemplo: Diferença entre Frames

Exemplo: Diferença entre Frames

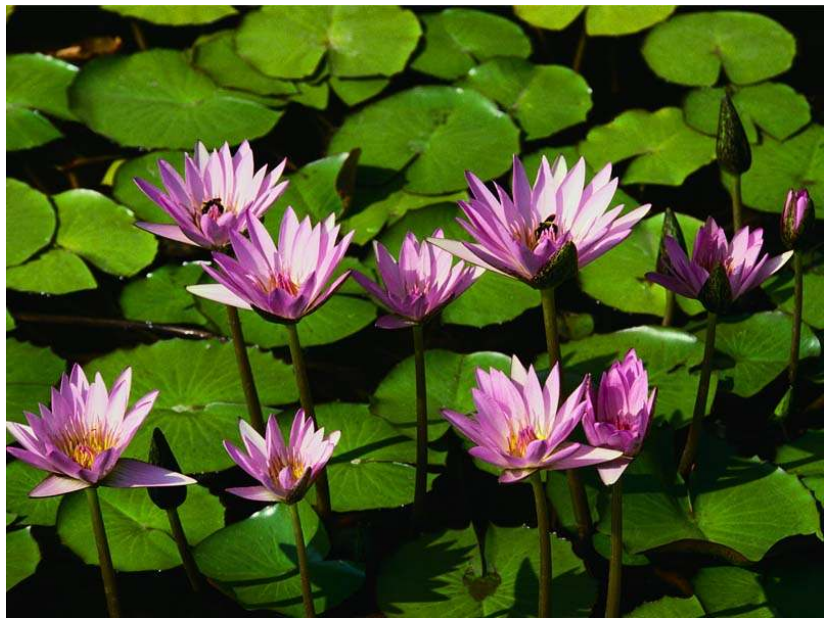
(A-B) or (B-A)



Exemplo: Diferença entre Frames

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

Exemplo: fade-in / fade-out de imagens



A

B

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

$\alpha=0.75$

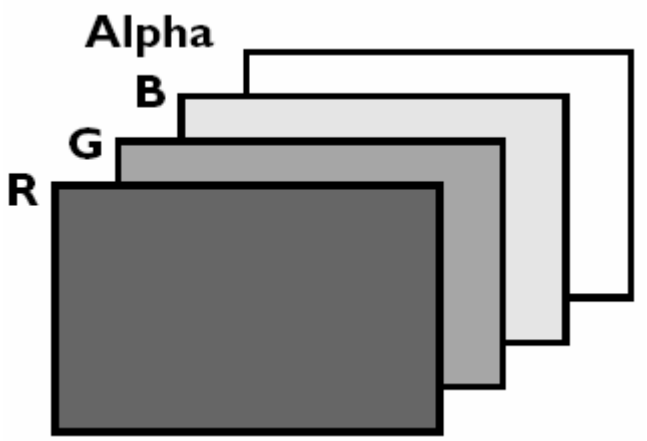
$\alpha=0.5$

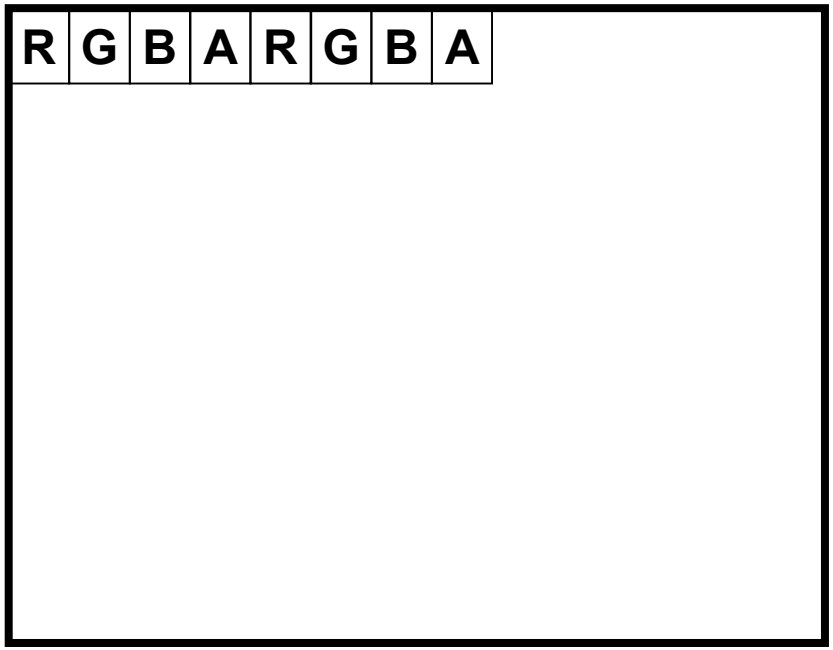
$\alpha=0.25$

Exemplo: fade-in / fade-out de imagens

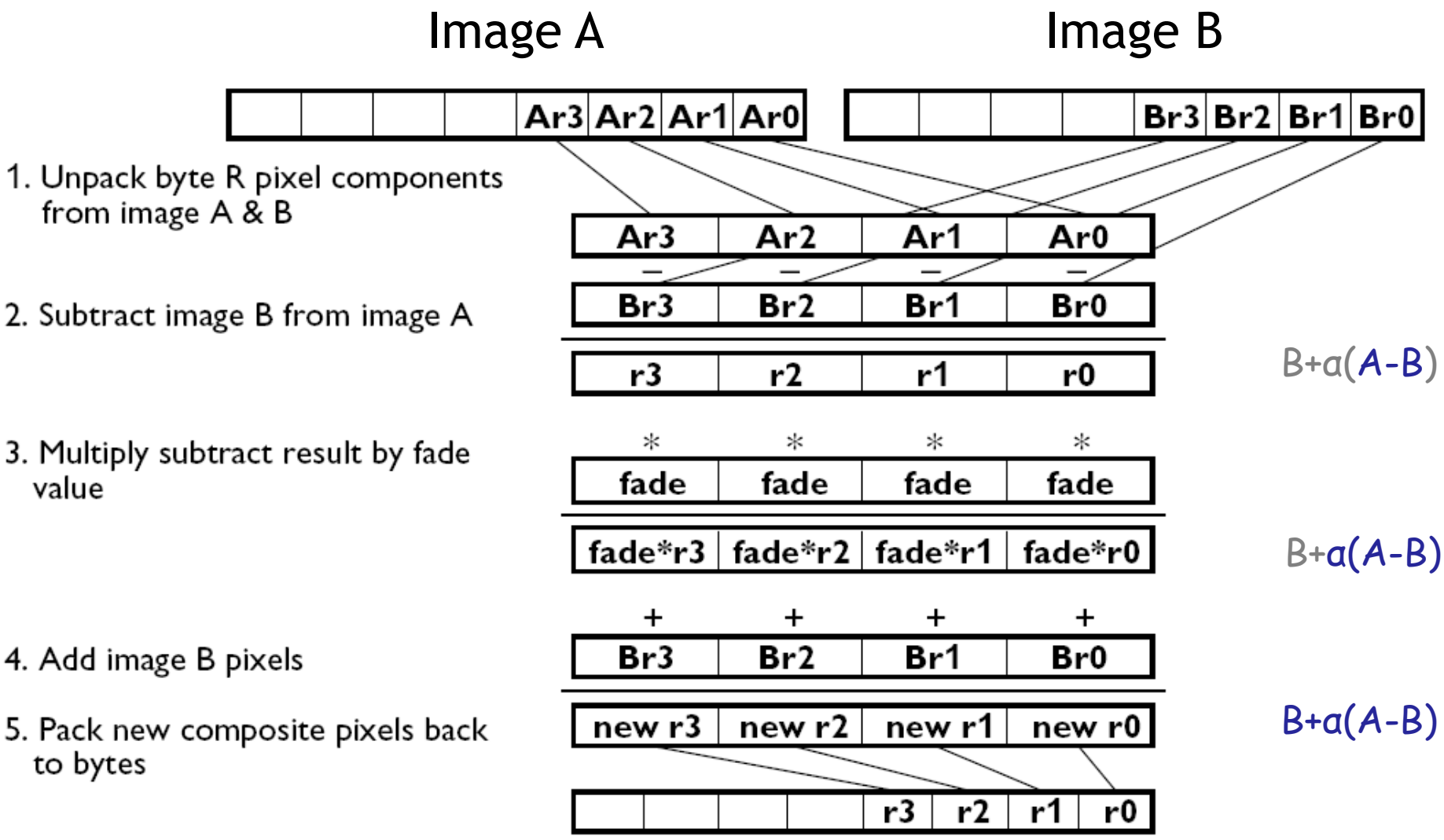
Cada pixel da imagem é representado por 4 bytes:

- R (red)
- G (green)
- B (blue)
- Alpha (transparência)





Exemplo: fade-in / fade-out de imagens

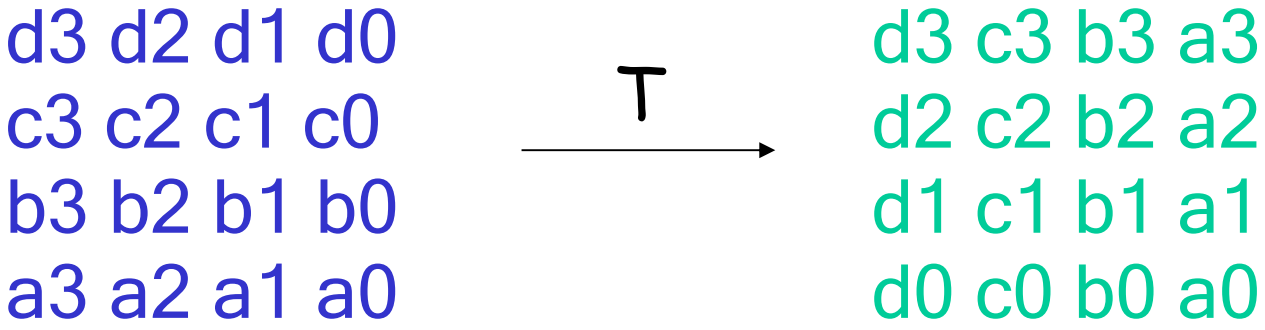


Exemplo: fade-in / fade-out de imagens

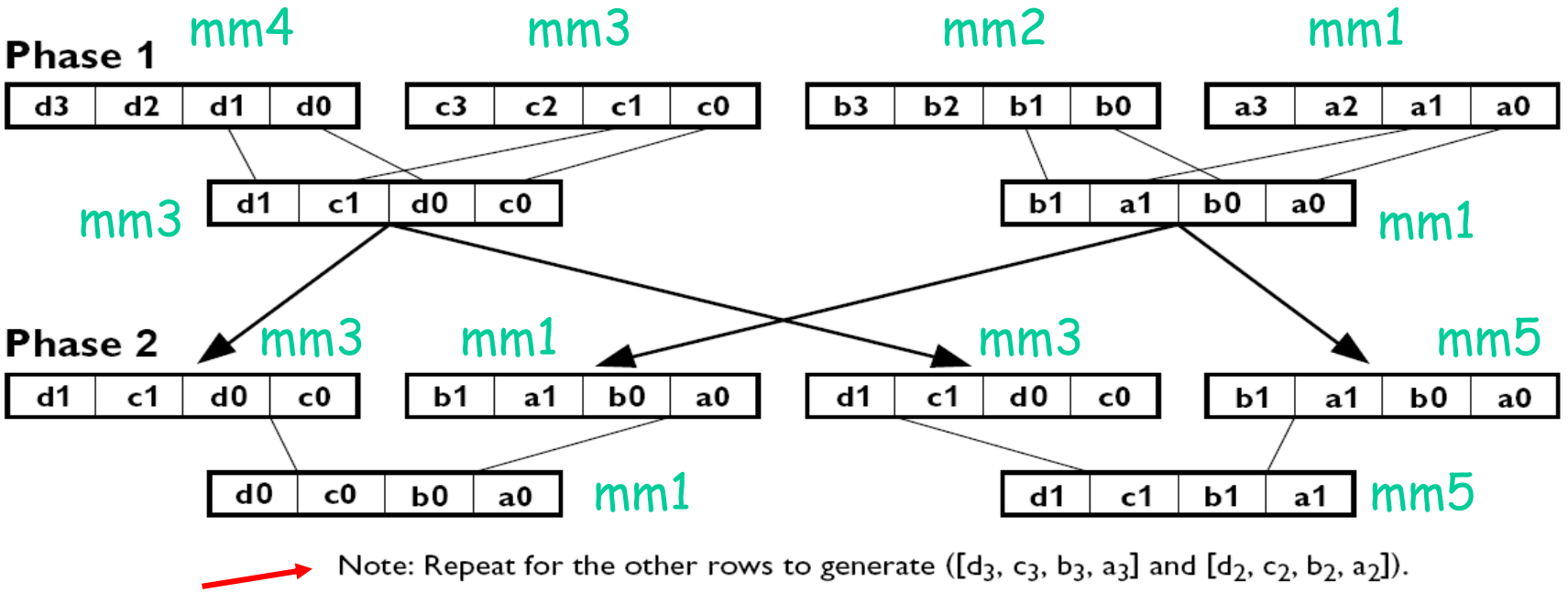
```
MOVQ       mm0, alpha//4 16-b zero-padding α
MOVD       mm1, A //move 4 pixels of image A
MOVD       mm2, B //move 4 pixels of image B
PXOR       mm3, mm3 //clear mm3 to all zeroes
//unpack 4 pixels to 4 words
PUNPCKLBW mm1, mm3 // Because B-A could be
PUNPCKLBW mm2, mm3 // negative, need 16 bits
PSUBW      mm1, mm2 //(B-A)
PMULHW     mm1, mm0 //(B-A)*fade/256
PADDW      mm1, mm2 //(B-A)*fade + B
//pack four words back to four bytes
PACKUSWB   mm1, mm3
```

Matriz Transposta -Estratégia

1. Empacotar linhas em registradores mmx
2. Desempacotar combinando registradores de linhas diferentes



Exemplo: Matriz Tranposta



Note: Repeat for the other rows to generate ([d_3, c_3, b_3, a_3] and [d_2, c_2, b_2, a_2]).

MMX code sequence operation:

```
movq        mm1, row1    ; load pixels from first row of matrix
movq        mm2, row2    ; load pixels from second row of matrix
movq        mm3, row3    ; load pixels from third row of matrix
movq        mm4, row4    ; load pixels from fourth row of matrix
punpcklwd   mm1, mm2     ; unpack low order words of rows 1 & 2, mm 1 = [b1, a1, b0, a0]
punpcklwd   mm3, mm4     ; unpack low order words of rows 3 & 4, mm3 = [d1, c1, d0, c0]
movq        mm5, mm1     ; copy mm1 to mm5
punpckldq   mm1, mm3     ; unpack low order doublewords -> mm2 = [d0, c0, b0, a0]
punpckhdq   mm5, mm3     ; unpack high order doublewords -> mm5 = [d1, c1, b1, a1]
```

Desempenho: Versão C x Versão SIMD ?

Exemplo: Matriz Transposta - 4x8 → 8x4

```
char M1[4][8]; // matrix to be transposed
char M2[8][4]; // transposed matrix
for (int i=0;i<4;i++)
  for (int j=0;j<8;j++)
    M2[j][i]= M1[i][j];
```

Exemplo: Matriz Transposta (1/3)

```
char M1[4][8];// matrix to be transposed
char M2[8][4];// transposed matrix
int n=0;
for (int i=0;i<4;i++)
  for (int j=0;j<8;j++)
    { M1[i][j]=n; n++; }
__asm{
//move the 4 rows of M1 into MMX registers
movq mm1,M1
movq mm2,M1+8
movq mm3,M1+16
movq mm4,M1+24
```

Exemplo: Matriz Tranposta (2/3)

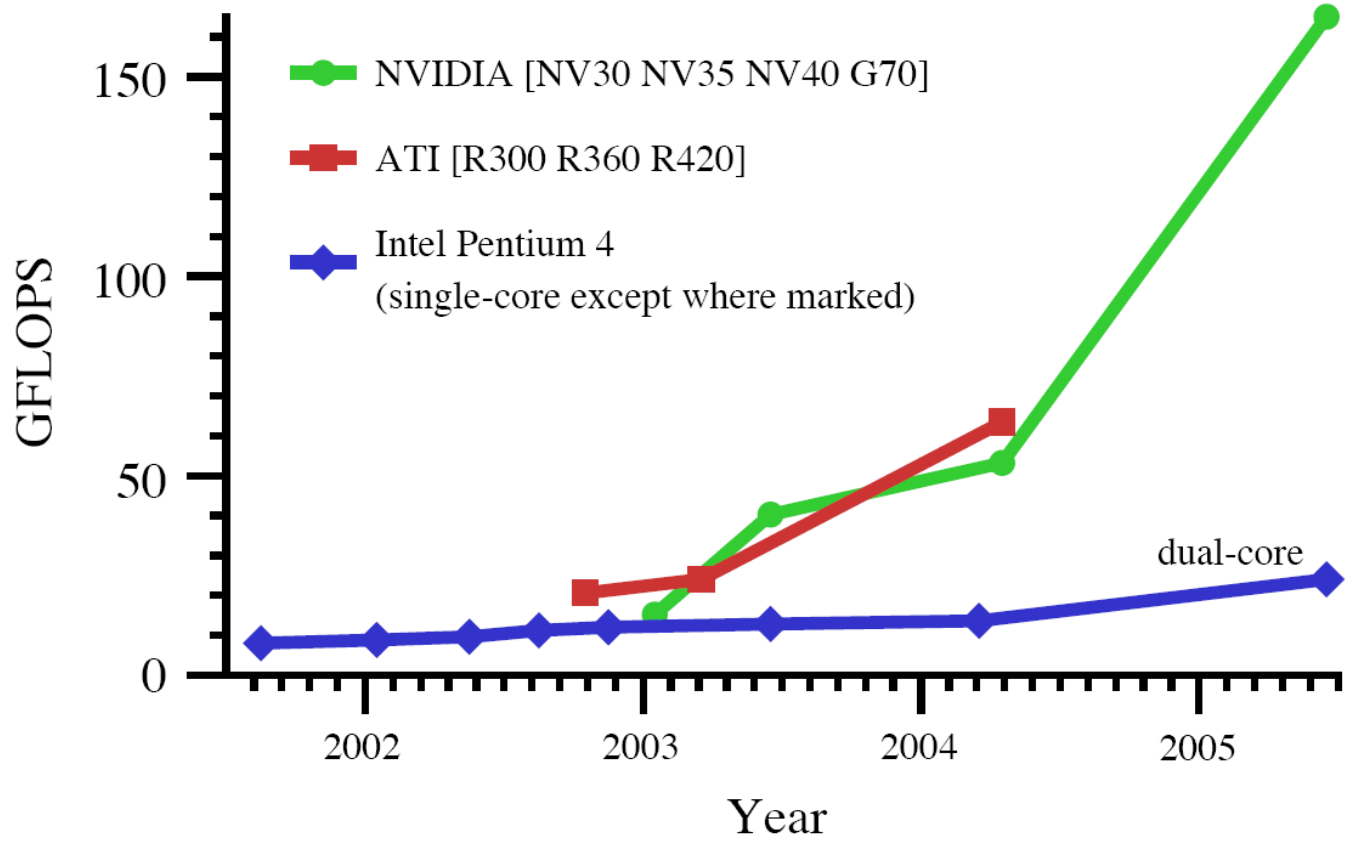
```
//generate rows 1 to 4 of M2
punpcklbw mm1, mm2
punpcklbw mm3, mm4
movq mm0, mm1
punpcklwd mm1, mm3 //mm1 has row 2 & row 1
punpckhwd mm0, mm3 //mm0 has row 4 & row 3
movq M2, mm1
movq M2+8, mm0
```

Exemplo: Matriz Tranposta (3/3)

```
//generate rows 5 to 8 of M2
movq mm1, M1 //get row 1 of M1
movq mm3, M1+16 //get row 3 of M1
punpckhbw mm1, mm2
punpckhbw mm3, mm4
movq mm0, mm1
punpcklwd mm1, mm3 //mm1 has row 6 & row 5
punpckhwd mm0, mm3 //mm0 has row 8 & row 7
//save results to M2
movq M2+16, mm1
movq M2+24, mm0
emms
} //end
```

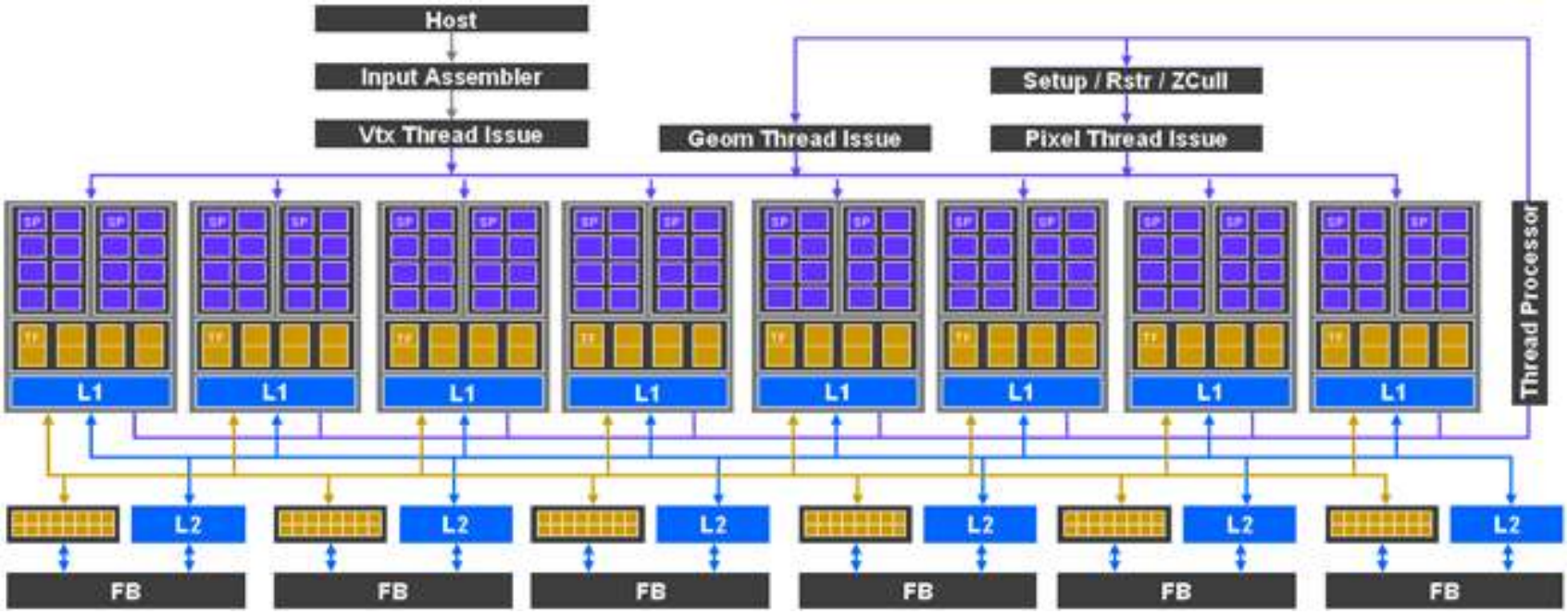
Outras Arquiteturas SIMD

Graphics Processing Unit (GPU): nVidia 7800, 24 pipelines



NVidia GeForce 8800, 2006

Possui até 128 processadores de fluxo (stream processors)



Cell processor

- Cell Processor (IBM/Toshiba/Sony):
  - 1 PPE (Power Processing Unit) + 8 SPEs (Synergistic Processing Unit)
  - SPE é um processador RISC com instruções SIMD de 128-bits, 128 registradores de 128-bits, e cache local de 256K.
  - Usado no Play Station 3 (PS3)

Cell processor