Linguagens de Montagem

Instruções Vetoriais SSE

Aula 12

Edmar André Bellorini

Ano letivo 2021

Introdução

Introdução

- Na aula anterior sobre Ponto-flututante
  - 2ª Geração
  - SSE evolução de MMX
    - Instruções Escalares para FP
    - Instruções Vetoriais para FP

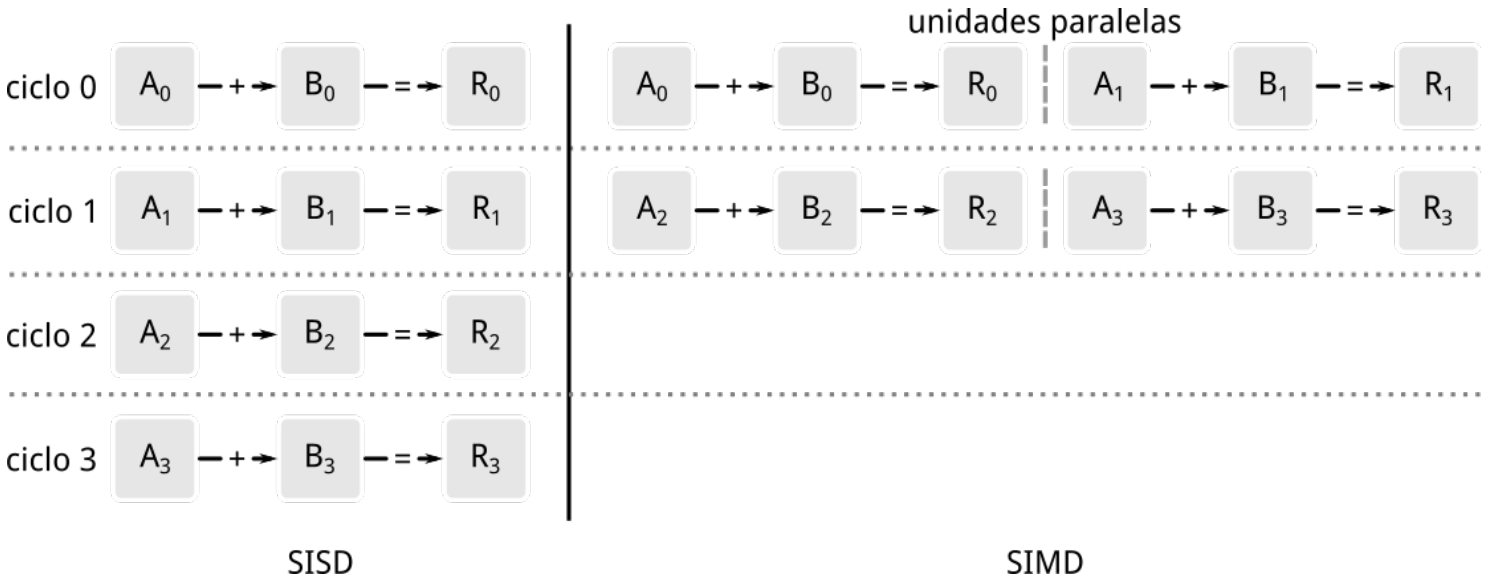
- MMX
  - Instruções SIMD
    Single Instruction, Multiple Data - by Flynn
  - 8 Registradores MMX de 64 bits
  - Packed Integer
    - 1 int 64 bits
    - 2 int 32 bits
    - 4 int 16 bits
    - 8 int 08 bits
  - Não será discutido nesta disciplina.

SSE - Vetorial

SIMD

- Single Instruction, Multiple Data
  - Uma única instrução é executada em vários pares de operandos
  - Exemplo:
    Somar cada par i de elementos de dois vetores A e B

SISD vs SIMD



SSE - Vetorial - Tipos de Dados

SSE

- Packed Single-Precision (32 bits)
  - 4 floats empacotados em 128 bits

SSE2

- Packed Double-Precision (64 bits)
  - 2 doubles empacotados em 128 bits
- Packed Integers-Bytes (08 bits)
  - 16 inteiros empacotados em 128 bits
- Packed Integers-Word (16 bits)
  - 8 inteiros empacotados em 128 bits
- Packed Integers-Doubleword (32 bits)
  - 4 inteiros empacotados em 128 bits
- Packed Integers-Quadword (64 bits)
  - 2 inteiros empacotados em 128 bits

AVX - Vetorial - Tipos de Dados

AVX

- Os mesmos tipos de dados do SSE e SSE2
- Porém, empacotados para:
  - 256 bits em AVX (AVX256)
    - 04 double (quadword/64bits)
    - 08 float (doubleword/32bits)
    - 04 int (quadword/64bits)
    - 08 int (doubleword/32bits)
    - 16 int (word/16bits)
    - 32 int (halfword/08bits)
  - 512 bits em AVX512

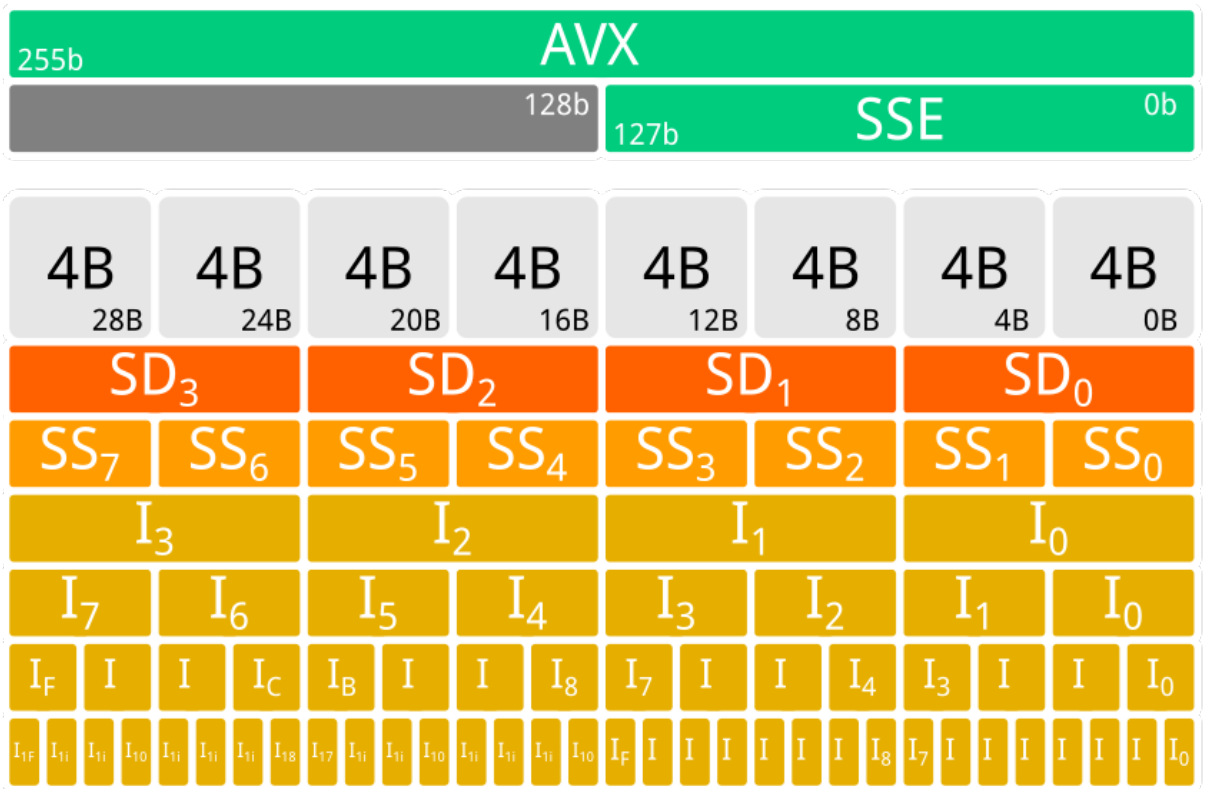
Observação:

- Este material aborda apenas SSE2 e alguns pontos de AVX

Registradores

SSE2 e AVX

08 registradores $[y|x]mm_i$ em 32 bits
16 registradores $[y|x]mm_i$ em 64 bits



Instruções de Transferência

Packed-Integers

MOVD ou MOVQ

- Move Doubleword/Move Quadword

```
MOVD xmmi, reg32/mem32
MOVQ xmmi, reg64/mem64
MOVD reg32/mem32, xmmi
MOVQ reg64/mem64, xmmi
```

Copia 1-Packed ou 2-Packed inteiro(s)$_{32bits}$ de `reg` para xmm_i
Copia 1-Packed ou 2-Packed inteiro(s)$_{32bits}$ de `mem` para xmm_i
Copia 1-Packed ou 2-Packed inteiro(s)$_{32bits}$ de xmm_i para `reg`
Copia 1-Packed ou 2-Packed inteiro(s)$_{32bits}$ de xmm_i para `mem`

- bits altos não utilizados são *zeroed*

More Packed-Integers - Alinhado

MOVDQA

- Move Aligned Packed Integer Values

```
MOVDQA xmmi, xmmj|mem128
```

Copia 4-Packed inteiros$_{32bits}$ de xmm_j para xmm_i
Copia 4-Packed inteiros$_{32bits}$ de `mem128` para xmm_i

- Atenção à palavra mágica **Aligned**
  - `mem128` deve estar alinhado em 16
    isto é: endereço do byte inicial de `mem128` % 16 = 0
    section .data

```
align 16, db 0
```

section .bss

```
alignb 16
```

More Packed-Integers - Alinhado

MOVDQA (again?)

- Move Aligned Packed Integer Values

```
MOVDQA mem128, xmmi
```

Copia 4-Packed inteiros$_{32bits}$ de xmm_i para `mem128`

- Atenção à palavra mágica **Aligned**
  - `mem128` deve estar alinhado em 16

More Packed-Integers - Desalinhado

MOVDQU

- Move Unaligned Packed Integer Values

```
MOVDQU xmmi, xmmj|mem128
MOVDQU mem128, xmmi
```

Copia 4-Packed inteiros$_{32bits}$ de xmm_i para `mem128`

- Atenção à palavra mágica **Unaligned**
  - `mem128` não requer alinhamento em 16
  - impacta negativamente no desempenho da transferência
    - Se está usando SSE, possivelmente quer desempenho, certo?

MORE Packed-Integers - Alinhado

VMOVDQA (AVX)

- Move Aligned Packed Integer Values

```
VMOVDQA ymmi, ymmj|mem256
VMOVDQA mem256, ymmi
```

Copia 8-Packed inteiros$_{32bits}$ de ymm_j para ymm_i
Copia 8-Packed inteiros$_{32bits}$ de `mem256` para ymm_i
Copia 8-Packed inteiros$_{32bits}$ de ymm_i para `mem256`

- Atenção à palavra mágica **Aligned**
  - `mem256` deve estar alinhado em 32
    isto é: endereço do byte inicial de `mem128` % 32 = 0

MORE Packed-Integers - Desalinhado

VMOVDQU (AVX)

- Move Unaligned Packed Integer Values

```
VMOVDQU ymmi, ymmj|mem256
VMOVDQU mem256, ymmi
```

Copia 8-Packed inteiros$_{32bits}$ de ymm_j para ymm_i
Copia 8-Packed inteiros$_{32bits}$ de `mem256` para ymm_i
Copia 8-Packed inteiros$_{32bits}$ de ymm_i para `mem256`

- Atenção à palavra mágica **Unaligned**
  - `mem256` não requer alinhamento em 32
  - impacta negativamente no desempenho da transferência

Packed-Integers - Exemplo

a12e01.asm

```
34      ; 32bits
35      MOVD xmm1, [vetInt1] ; gdb_tip: p $xmm1.v4_int32
36
37      ; 2*32bits -> 64bits
38      MOVQ xmm2, [vetInt1] ; gdb_tip: p $xmm2.v4_int32
39
40      ; 4*32bits -> 128bits alinhado em 16
41      ; -> caso contrário, SIGSEGV!
42      MOVDQA xmm3, [vetInt1] ; gdb_tip: p $xmm3.v4_int32
```

Packed-Integers - Observação

As instruções apresentadas anteriormente também são utilizadas para copiar inteiros de tamanhos inferiores à 32 bits.

- Exemplos

```
MOVD xmmi, mem32
```

Copia 1-Packed inteiro$_{32bits}$ de `reg` para xmm_i
Copia 2-Packed inteiros$_{16bits}$ de `reg` para xmm_i
Copia 4-Packed inteiros$_{08bits}$ de `reg` para xmm_i

```
VMOVDQA ymmi, mem256
```

Copia 4-Packed inteiros$_{64bits}$ de `reg` para ymm_i
Copia 8-Packed inteiros$_{32bits}$ de `reg` para ymm_i
Copia 16-Packed inteiros$_{16bits}$ de `reg` para ymm_i
Copia 32-Packed inteiros$_{08bits}$ de `reg` para ymm_i

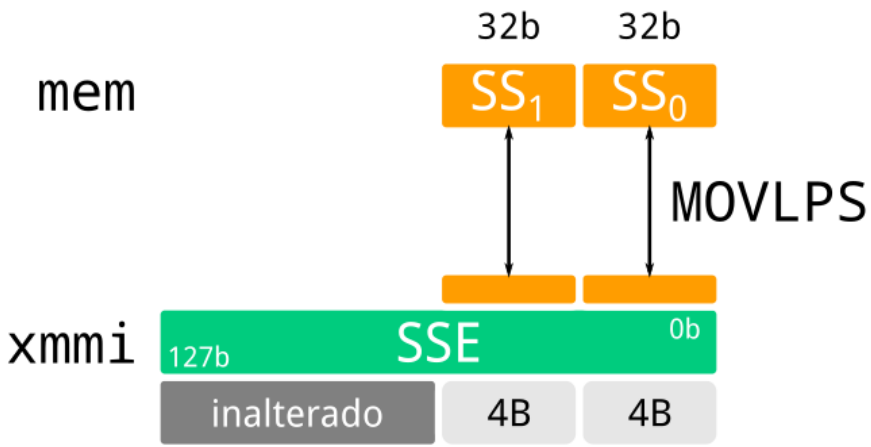
2-Packed-SP

MOVLPS

- Move Low Packed Single-Precision Floating-Point Values

```
MOVLPS xmmi, mem64
MOVLPS mem64, xmmi
```

Copia 2-Packed Single-Precision$_{32bits}$ de `mem64` para xmm_i
Copia 2-Packed Single-Precision$_{32bits}$ de xmm_i para `mem64`



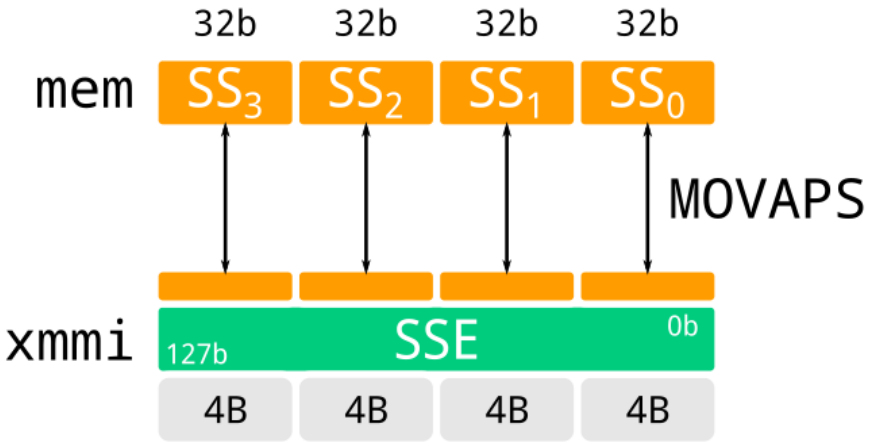
4-Packed-SP - Alinhado$_{16}$

MOVAPS

- Move Aligned Packed Single-Precision Floating-Point Values

```
MOVAPS xmmi, mem128
MOVAPS mem128, xmmi
```

Copia 4-Packed Single-Precision$_{32bits}$ de `mem128` para xmm_i
Copia 4-Packed Single-Precision$_{32bits}$ de xmm_i para `mem128`



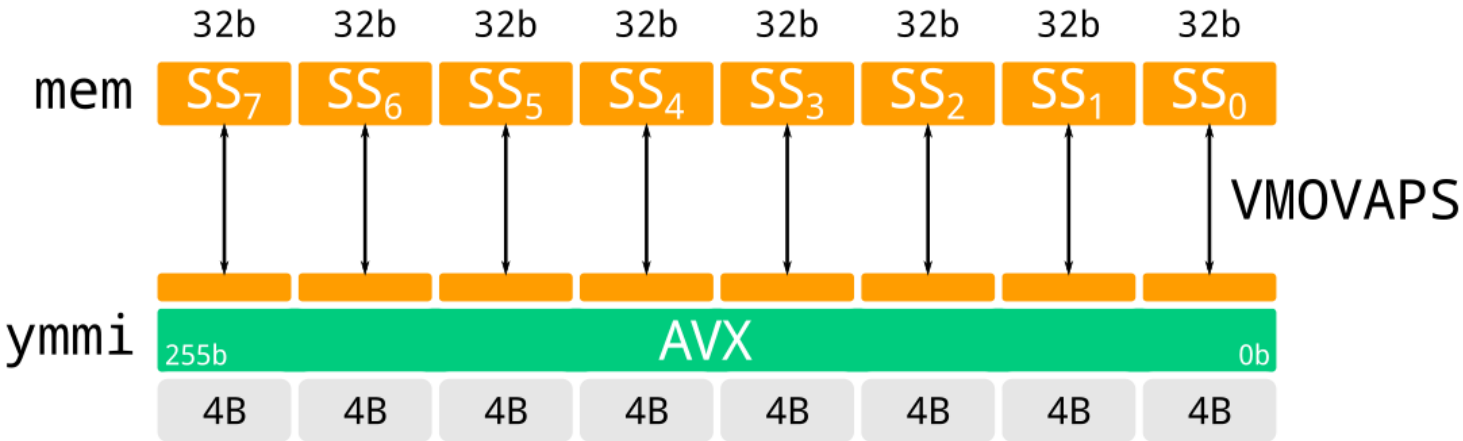
8-Packed-SP - Alinhado$_{32}$

VMOVAPS

- Move Aligned Packed Single-Precision Floating-Point Values

```
VMOVAPS ymmi, mem256
VMOVAPS mem256, ymmi
```

Copia 8-Packed Single-Precision$_{32bits}$ de `mem256` para ymm_i
Copia 8-Packed Single-Precision$_{32bits}$ de ymm_i para `mem256`



4,8-Packed-SP - Desalinhado

MOVUPS

- Move Unaligned Packed Single-Precision Floating-Point Values

```
MOVUPS xmmi, mem128
MOVUPS mem128, xmmi
```

Copia 4-Packed Single-Precision$_{32bits}$ de `mem128` para xmm_i
Copia 4-Packed Single-Precision$_{32bits}$ de xmm_i para `mem128`

VMOVUPS

- Move Unaligned Packed Single-Precision Floating-Point Values

```
VMOVUPS ymmi, mem256
VMOVUPS mem256, ymmi
```

Copia 8-Packed Single-Precision$_{32bits}$ de `mem256` para ymm_i
Copia 8-Packed Single-Precision$_{32bits}$ de ymm_i para `mem256`

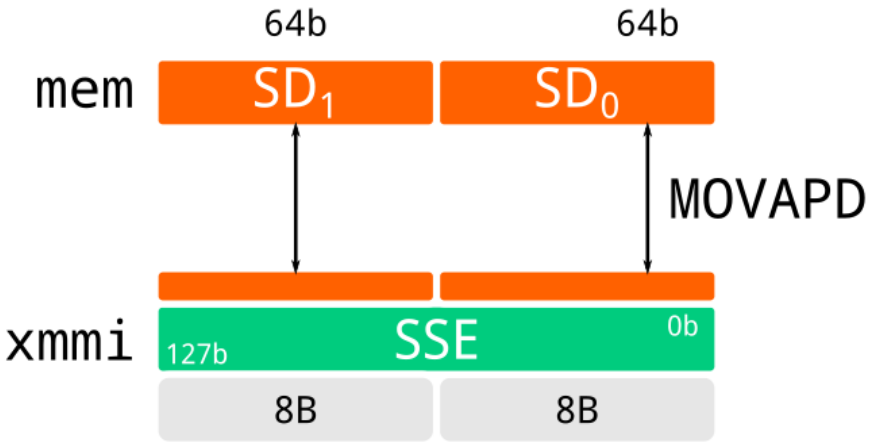
2-Packed-DP - Alinhado$_{16}$

MOVAPD

- Move Aligned Packed Double-Precision Floating-Point Values

```
MOVAPD xmmi, mem128
MOVAPD mem128, xmmi
```

Copia 2-Packed Double-Precision$_{64bits}$ de `mem128` para xmm_i
Copia 2-Packed Double-Precision$_{64bits}$ de xmm_i para `mem128`



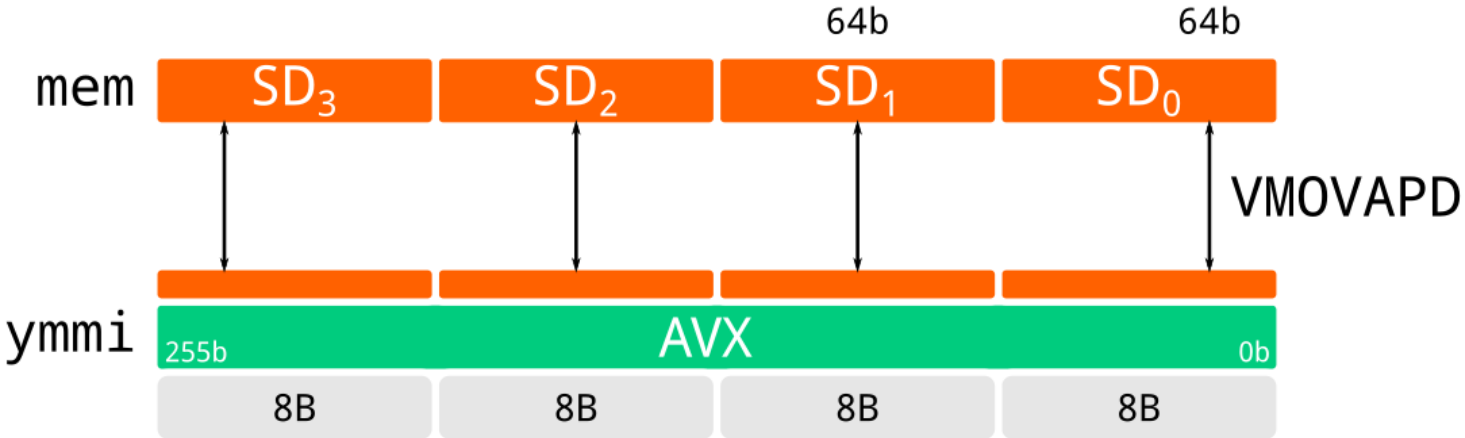
4-Packed-DP - Alinhado$_{32}$

VMOVAPD

- Move Aligned Packed Double-Precision Floating-Point Values

```
VMOVAPD ymmi, mem256
VMOVAPD mem256, ymmi
```

Copia 4-Packed Double-Precision$_{64bits}$ de `mem256` para ymm_i
Copia 4-Packed Double-Precision$_{64bits}$ de ymm_i para `mem256`



2,4-Packed-DP - Desalinhado

MOVUPD

- Move Unaligned Packed Double-Precision Floating-Point Values

```
MOVUPD xmmi, mem128
MOVUPD mem128, xmmi
```

Copia 2-Packed Double-Precision$_{64bits}$ de `mem128` para xmm_i
Copia 2-Packed Double-Precision$_{64bits}$ de xmm_i para `mem128`

VMOVUPD

- Move Unaligned Packed Double-Precision Floating-Point Values

```
VMOVUPD ymmi, mem256
VMOVUPD mem256, ymmi
```

Copia 4-Packed Double-Precision$_{64bits}$ de `mem256` para ymm_i
Copia 4-Packed Double-Precision$_{64bits}$ de ymm_i para `mem256`

Packed-SP e DP - Exemplo

a12e02.asm

```
34      ; ============= FLOAT ================================
35      ; Transferência FLOAT mem -> xmm ====================
36      ; 2*32bits -> 64bits
37      MOVLPS xmm1, [vetFloat1] ; gdb_tip: p $xmm1.v4_float
38
39      ; 4*32bits -> 128bits
40      MOVAPS xmm2, [vetFloat1] ; gdb_tip: p $xmm2.v4_float
41
42      ; 8*32bits -> 256bits
43      VMOVAPS ymm3, [vetFloat1] ; gdb_tip: p $ymm3.v8_float
44
45      ; Transferência FLOAT xmm -> mem ====================
46      ; 2*32bits -> 64bits
47      MOVLPS [vetRFloat2], xmm1 ; gdb_tip: x /2f &vetRFloat2
```

Instruções de Conversão
Inteiro para FP

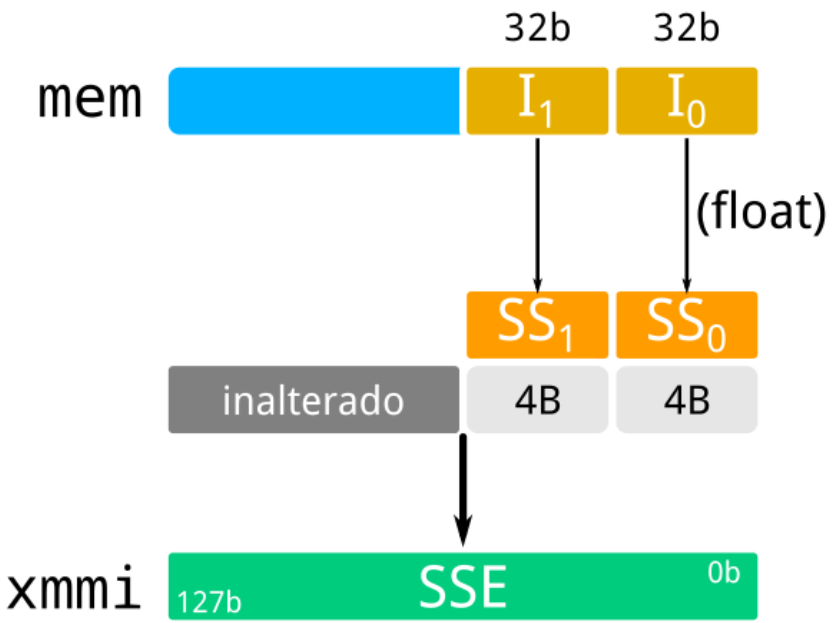
Conversão de Packed-Integer para FP-Single

CVTPI2PS

- Convert Packed-DWord Integers to Packed Single-Precision

```
CVTPI2PS xmmi, mem64
```

Converte 2 inteiros sinalizado (32 bits) para 2 FP (32 bits)
$hxmm_i$ inalterado



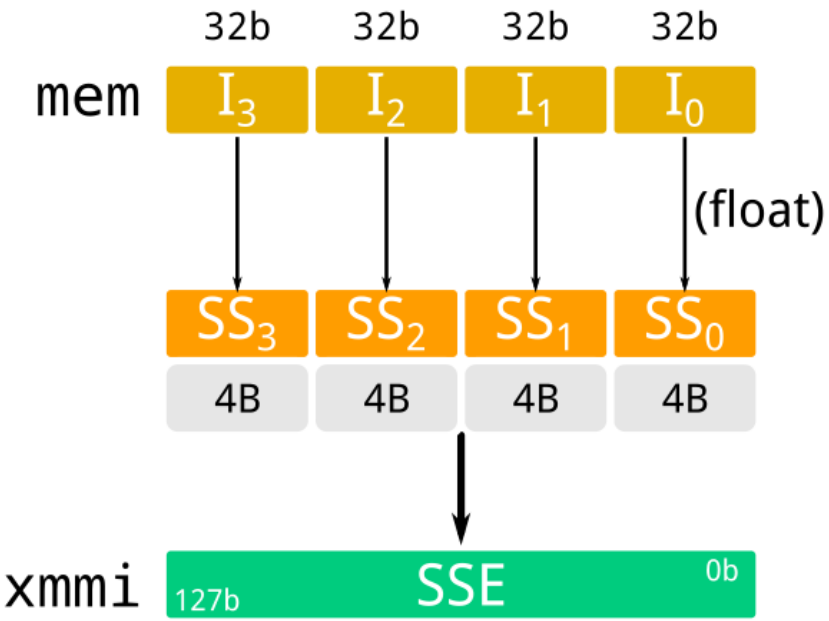
Conversão de Packed-Integer para FP-Single

VCVTDQ2PS (SSE)

- Convert Packed-DWord Integers to Packed Single-Precision

```
VCVTDQ2PS xmmi, mem128
```

Converte 4 inteiros sinalizado (32 bits) para 4 FP (32 bits)



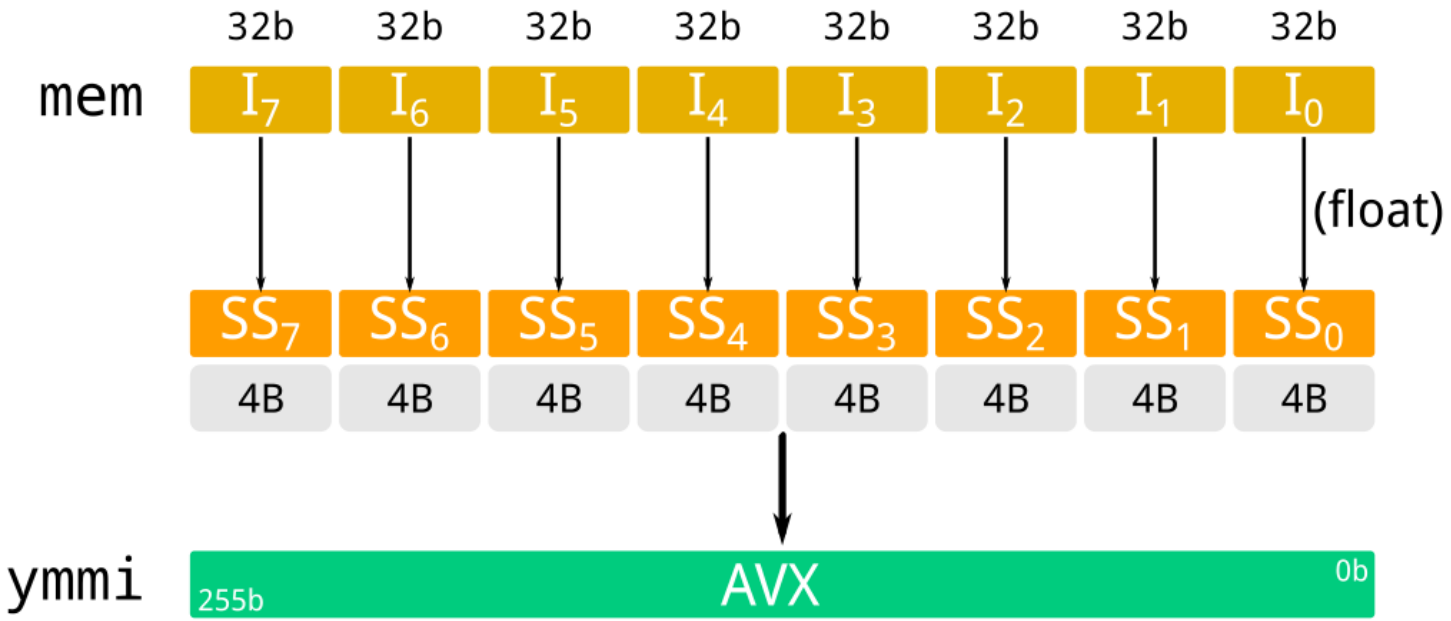
Conversão de Packed-Integer para FP-Single

VCVTDQ2PS (AVX)

- Convert Packed-DWord Integers to Packed Single-Precision

```
VCVTDQ2PS ymmi, mem256
```

Converte 8 inteiros sinalizado (32 bits) para 8 FP (32 bits)



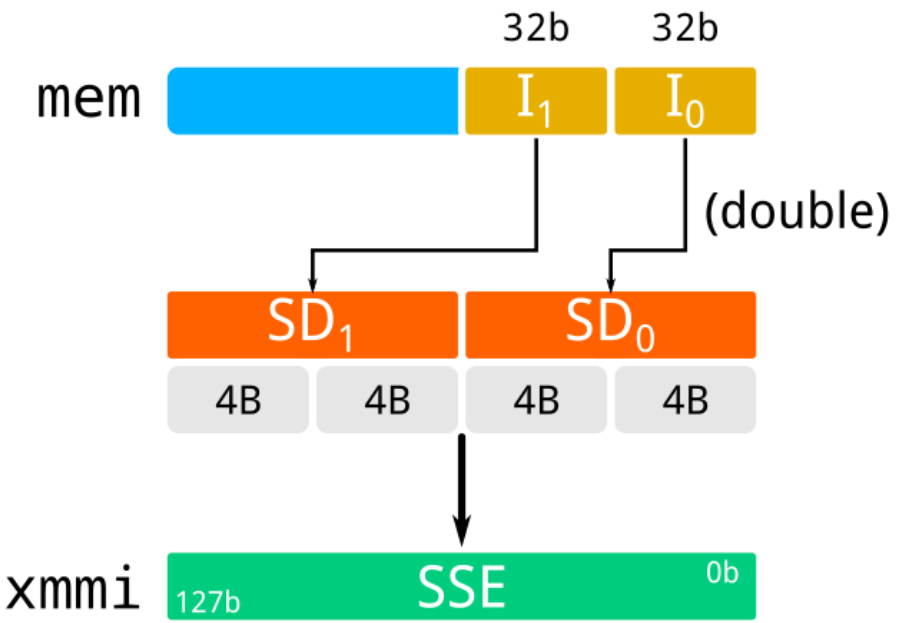
Conversão de Packed-Integer para FP-Double

CVTPI2PD

- Convert Packed-DWord Integers to Packed Double-Precision

```
CVTPI2PD xmmi, mem64
```

Converte 2 inteiros sinalizado (32 bits) para 2 FP (64 bits)



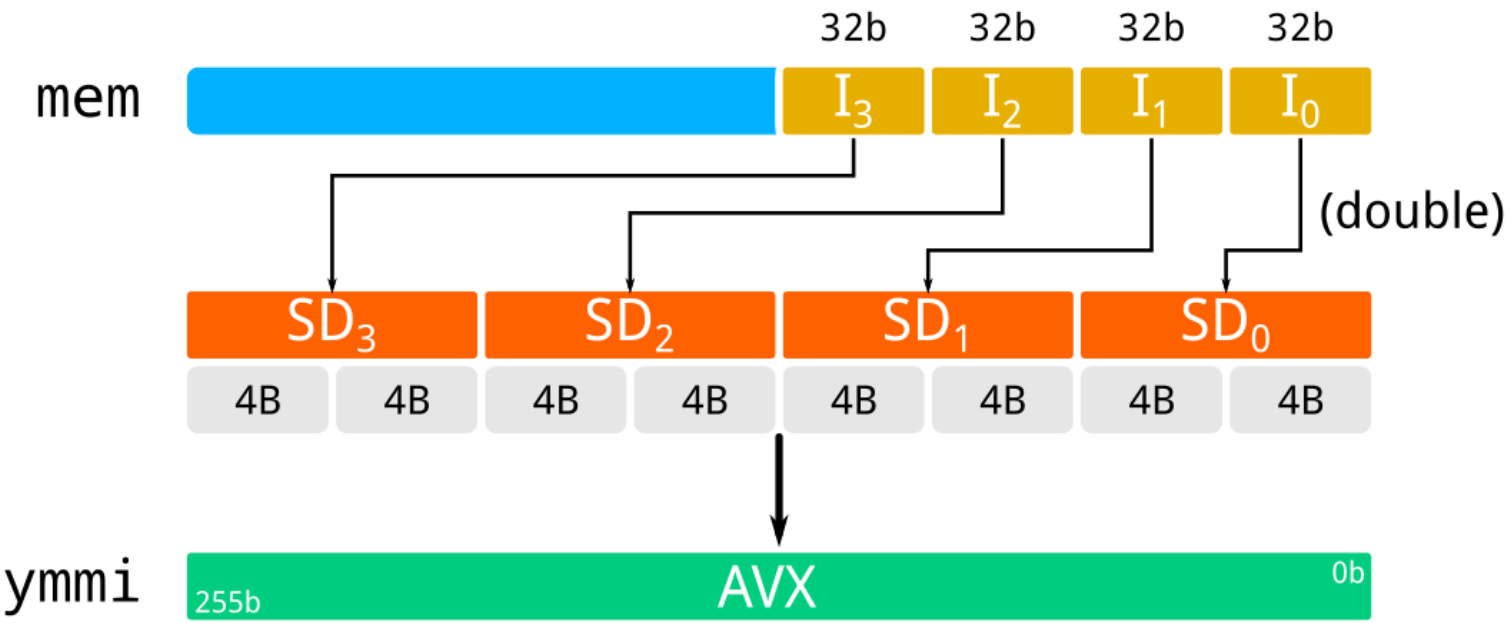
Conversão de Packed-Integer para FP-Double

CVTPI2PD (AVX)

- Convert Packed-DWord Integers to Packed Double-Precision

```
CVTPI2PD ymmi, mem128
```

Converte 4 inteiros sinalizado (32 bits) para 4 FP (64 bits)



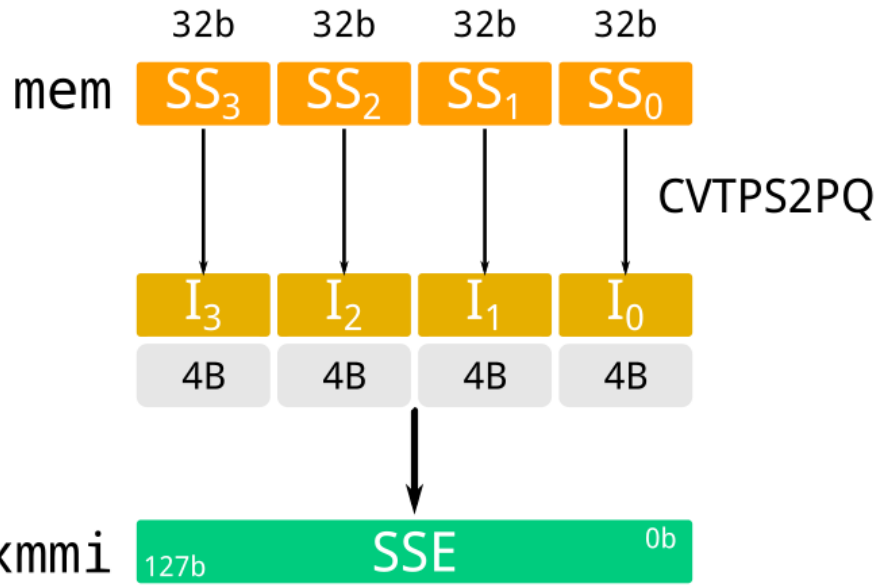
Conversão de FP-Single para Packed-Integer

CVTPS2DQ / VCVTPS2DQ

- Convert Packed Single-Precision Floating-Point Values to Packed Signed Doubleword Integer

```
CVTPS2DQ  xmmi, xmmj|mem128 ; SSE 4 FP
VCVTPS2DQ ymmi, ymmj|mem256 ; AVX 8 FP
```

Converte 4/8 FP (32 bits) para 4/8 inteiros sinalizado (32 bits)



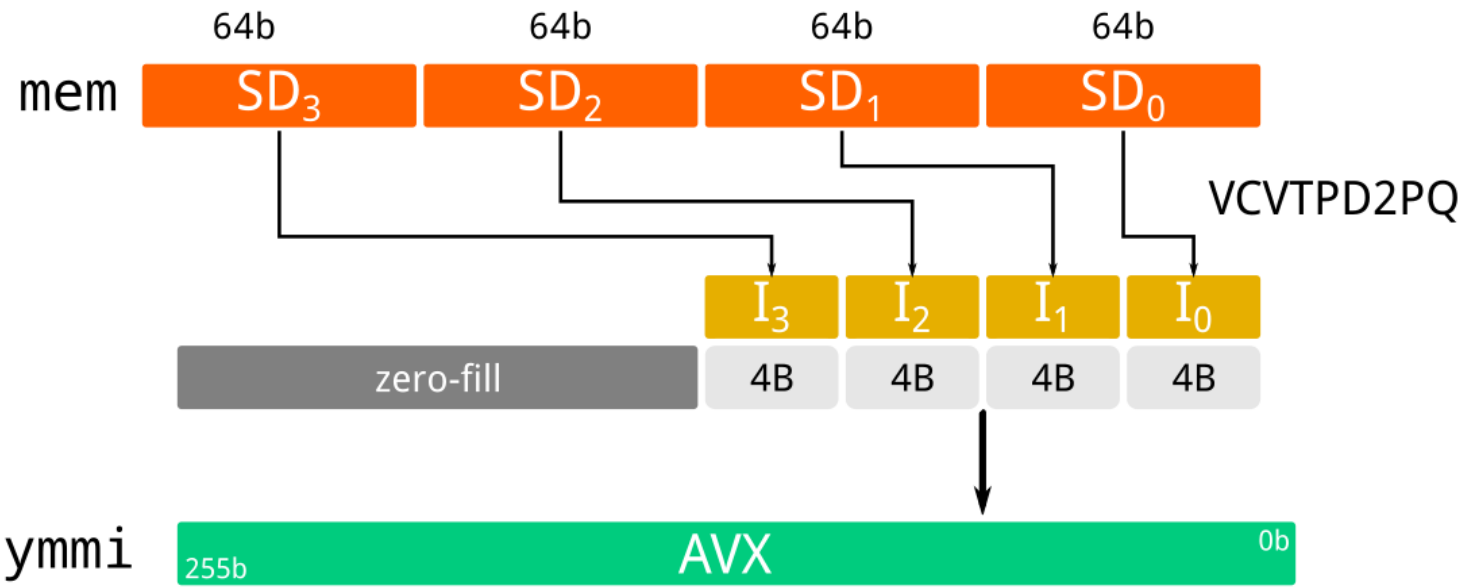
Conversão de FP-Double para Packed-Integer

CVTPD2DQ / VCVTPD2DQ

- Convert Packed Double-Precision Floating-Point Values to Packed Signed Doubleword Integer

```
CVTPD2DQ  xmmi, xmmj|mem128 ; SSE 2 FP
VCVTPD2DQ xmmi, ymmj|mem256 ; AVX 4 FP
```

Converte 2/4 FP (64 bits) para 2/4 inteiros sinalizado (32 bits)



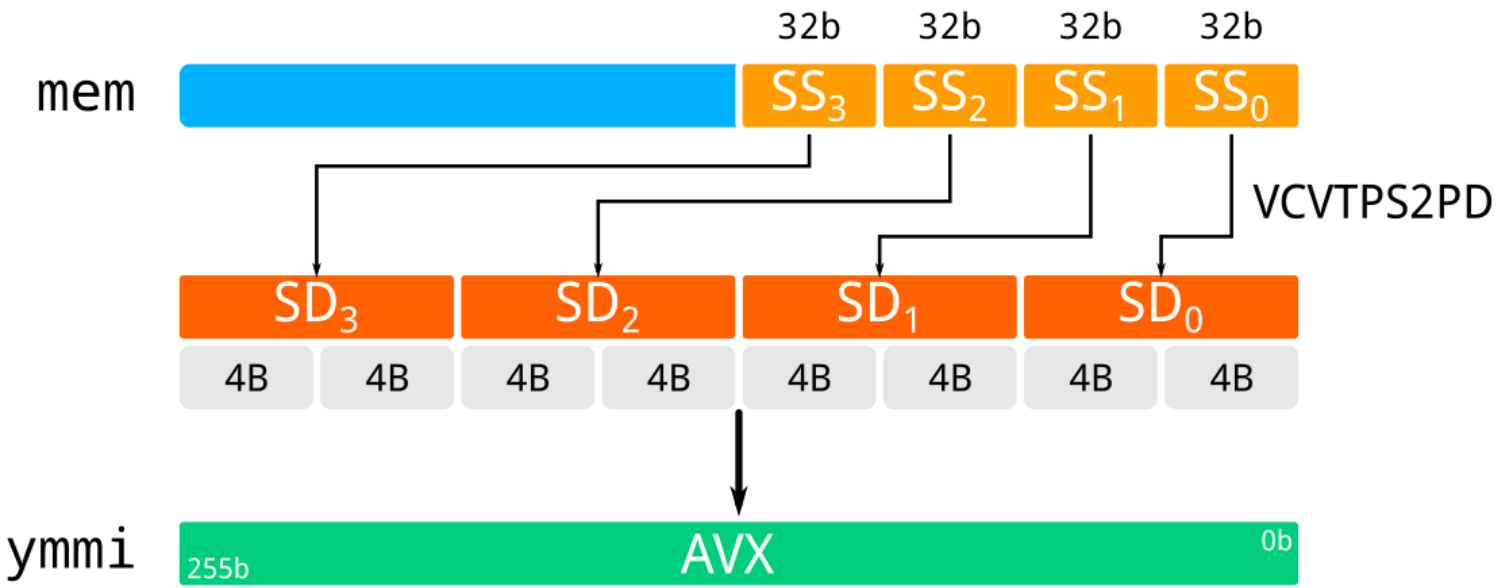
Conversão de FP-Single para FP-Double

CVTPS2PD

- Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point

```
CVTPS2PD  xmmi, xmmj|mem64  ; SSE 2 FP
VCVTPS2PD ymmi, xmmj|mem128 ; AVX 4 FP
```

Converte 2/4 FP (32 bits) para 2/4 FP (64 bits)



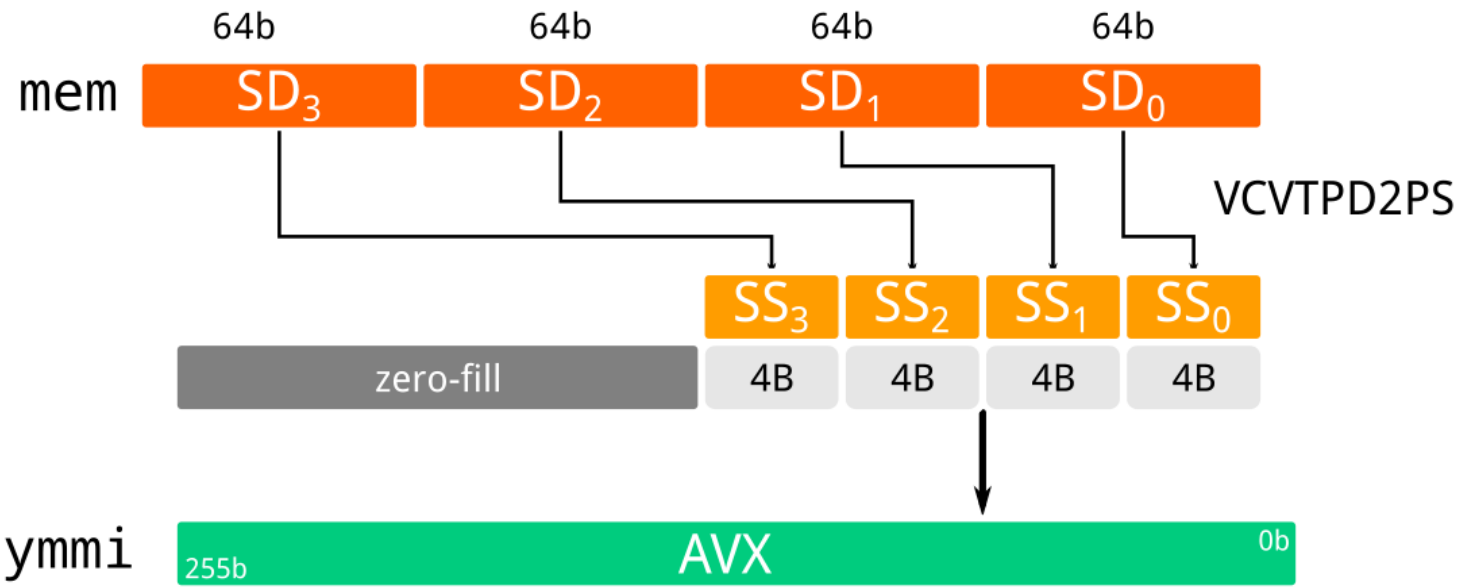
Conversão de FP-Double para FP-Single

CVTPD2PS

- Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point

```
CVTPD2PS  xmmi, xmmj|mem128  ; SSE 2 FP
VCVTPD2PS xmmi, ymmj|mem256 ; AVX 4 FP
```

Converte 2/4 FP (64 bits) para 2/4 FP (32 bits)



Conversão Integer, FP-Single e FP-Double - Exemplo

a12e03.asm

```
26      ; converter 2 inteiros 32 bits cada para 2 FP single
27      CVTPI2PS xmm1, [vetInt1] ; gdb_tip: p $xmm1.v4_float
28
29      ; converter 4 inteiros 32 bits cada para 4 FP single
30      VCVTDQ2PS xmm2, [vetInt1]
31
32      ; converter 8 inteiros 32 bits cada para 8 FP single
33      VCVTDQ2PS ymm3, [vetInt1] ; gdb_tip: p $ymm3.v8_float
34
35      ; converter 2 inteiros 32 bits cada para 2 FP double
36      CVTPI2PD xmm4, [vetInt1] ; gdb_tip: p $xmm4.v2_double
```

Instruções Aritméticas

Aritmética de Inteiros

PADDB / PADDW / PADDD / PADDQ

- Add Packed Integers

```
PADDB xmmi, xmmj|mem128 ; 16 inteiros de 08 bits
PADDW xmmi, xmmj|mem128 ; 08 inteiros de 16 bits
PADDD xmmi, xmmj|mem128 ; 04 inteiros de 32 bits
PADDQ xmmi, xmmj|mem128 ; 02 inteiros de 64 bits
```

Executa operação SIMD de adição sobre os operandos

Aritmética de Inteiros

PSUBB / PSUBW / PSUBD / PSUBQ

- Subtract Packed Integers

```
PSUBB xmmi, xmmj|mem128 ; 16 inteiros de 08 bits
PSUBW xmmi, xmmj|mem128 ; 08 inteiros de 16 bits
PSUBD xmmi, xmmj|mem128 ; 04 inteiros de 32 bits
PSUBQ xmmi, xmmj|mem128 ; 02 inteiros de 64 bits
```

Executa operação SIMD de subtração sobre os operandos

Aritmética de Inteiros

PSUBB / PSUBW / PSUBD / PSUBQ

- Subtract Packed Integers

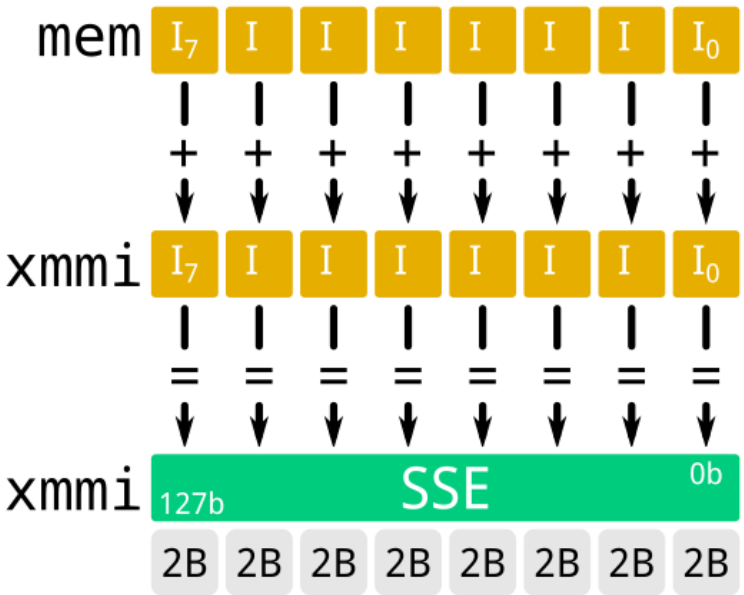
```
PSUBB xmmi, xmmj|mem128 ; 16 inteiros de 08 bits
PSUBW xmmi, xmmj|mem128 ; 08 inteiros de 16 bits
PSUBD xmmi, xmmj|mem128 ; 04 inteiros de 32 bits
PSUBQ xmmi, xmmj|mem128 ; 02 inteiros de 64 bits
```

Executa operação SIMD de subtração sobre os operandos

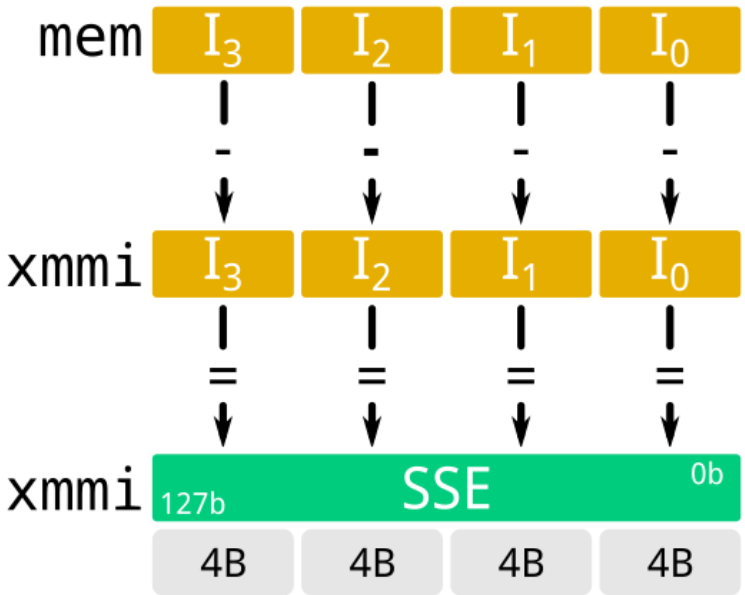
Aritmética de Inteiros - Exemplos ADD/SUB

```
PADDW xmmi, mem128 ; 08 inteiros de 16 bits
PSUBD xmmi, mem128 ; 04 inteiros de 32 bits
```

PADDW:



PSUBD:



Aritmética de Inteiros

PMULDQ / PMULUDQ

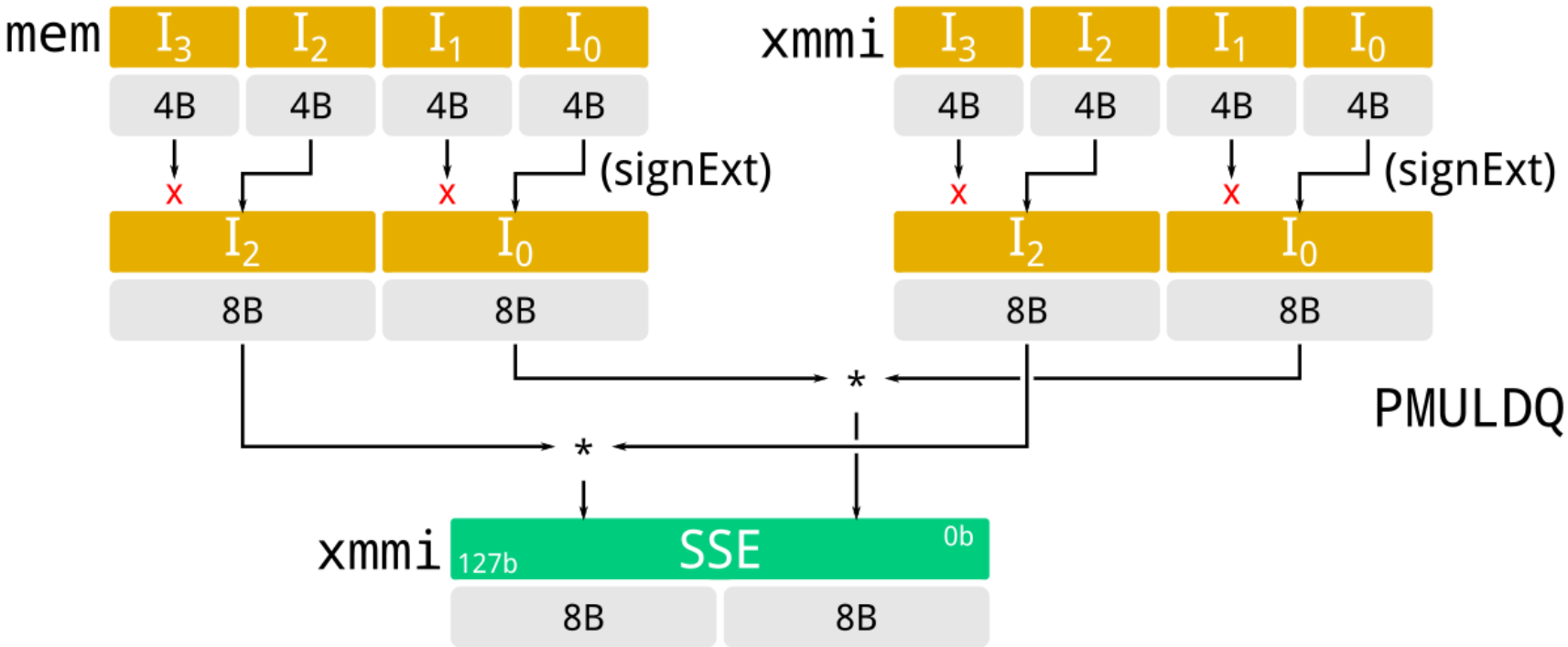
- Multiply Packed Doubleword Integers

```
    ; 2 ints sinalizados de 32 bits
    PMULDQ  xmmi, xmmj|mem128
    ; 2 ints não-sinalizados de 32 bits
    PMULUDQ xmmi, xmmj|mem128
```

Executa operação SIMD de multiplicação sobre os operandos
Contém *caveat*

Aritmética de Inteiros - caveat da Multiplicação

- I_0 e I_2
  - $\text{Operandos}_{32bits}$ são estendidos para $\text{Operandos}_{64bits}$
  - I_1 e I_3 são destruídos em registrador *xmmi*



Aritmética de Inteiros - Exemplo

a12e04.asm

```
32      ; 4-Packed ADD / SUB ==================================
33      MOVDQA xmm1, [vetInt1]
34      MOVDQA xmm2, [vetInt2]
35
36      ; vetIntADD432 = vetInt1 + vetInt2
37      PADDD xmm1, xmm2
38      MOVDQA [vetIntADD432], xmm1 ; gdb-tip: x /4d &vetIntADD432
39
40      ; vetIntSUB432 = vetInt1 - vetInt2
41      MOVDQA xmm1, [vetInt1]
42      PSUBD xmm1, xmm2
43      MOVDQA [vetIntSUB432], xmm1 ; gdb-tip: x /4d &vetIntSUB432
```

Aritmética de SP

ADDPS / VADDPS

- Add Packed Single-Precision Floating-Point

```
ADDPS  xmmi, xmmj|mem128 ; 04 SP
VADDPS ymmi, ymmj|mem256 ; 08 SP
```

Executa operação SIMD de adição sobre os operandos SP

SUBPS / VSUBPS

- Subtract Packed Single-Precision Floating-Point

```
SUBPS  xmmi, xmmj|mem128 ; 04 SP
VSUBPS ymmi, ymmj|mem256 ; 08 SP
```

Executa operação SIMD de subtração sobre os operandos SP

Aritmética de SP

DIVPS / VDIVPS

- Divide Packed Single-Precision Floating-Point

```
DIVPS  xmmi, xmmj|mem128 ; 04 SP
VDIVPS ymmi, ymmj|mem256 ; 08 SP
```

Executa operação SIMD de adição sobre os operandos SP

MULPS / VMULPS

- Multiply Packed Single-Precision Floating-Point

```
MULPS  xmmi, xmmj|mem128 ; 04 SP
VMULPS ymmi, ymmj|mem256 ; 08 SP
```

Executa operação SIMD de multiplicação sobre os operandos SP

Aritmética de DP

ADDPD / VADDPD

- Add Packed Double-Precision Floating-Point

```
ADDPD  xmmi, xmmj|mem128 ; 02 DP
VADDPD ymmi, ymmj|mem256 ; 04 DP
```

Executa operação SIMD de adição sobre os operandos DP

SUBPD / VSUBPD

- Subtract Packed Double-Precision Floating-Point

```
SUBPD  xmmi, xmmj|mem128 ; 02 DP
VSUBPD ymmi, ymmj|mem256 ; 04 DP
```

Executa operação SIMD de subtração sobre os operandos DP

Aritmética de DP

DIVPD / VDIVPD

- Divide Packed Double-Precision Floating-Point

```
DIVPD  xmmi, xmmj|mem128 ; 02 DP
VDIVPD ymmi, ymmj|mem256 ; 04 DP
```

Executa operação SIMD de adição sobre os operandos DP

MULPD / VMULPD

- Multiply Packed Double-Precision Floating-Point

```
MULPD  xmmi, xmmj|mem128 ; 02 DP
VMULPD ymmi, ymmj|mem256 ; 04 DP
```

Executa operação SIMD de multiplicação sobre os operandos DP

Aritmética de SP e DP - Exemplo

A12e05.asm

```
32      ; 4-Packed ADD / MUL SP ================================
33      MOVAPS xmm0, [vetSP1]
34      ADDPS xmm0, [vetSP2] ; gdb-tip: p $xmm0.v4_float
35
36      MOVAPS xmm1, [vetSP1]
37      MULPS xmm1, [vetSP2] ; gdb-tip: p $xmm1.v4_float
38
39
40      ; 4-Packed SUB / DIV DP ================================
41      VMOVAPS ymm2, [vetDP1]
42      VSUBPD ymm2, [vetDP2] ; gdb-tip: p $ymm2.v4_double
43
44      VMOVAPS ymm3, [vetDP1]
45      VDIVPD ymm3, [vetDP2] ; gdb-tip: p $ymm3.v4_double
```

Outras Instruções

Algumas instruções que podem ser uteis no futuro - I

VZEROALL / VZEROUPPER (AVX)

- Zero XMM, YMM and ZMM Registers
- Zero Upper Bits of YMM and ZMM Registers

```
    VZEROALL    ; zero-fill registradores
    VZEROUPPER  ; zero-fill (i>=128) bits de registradore
```

Algumas instruções que podem ser uteis no futuro - II

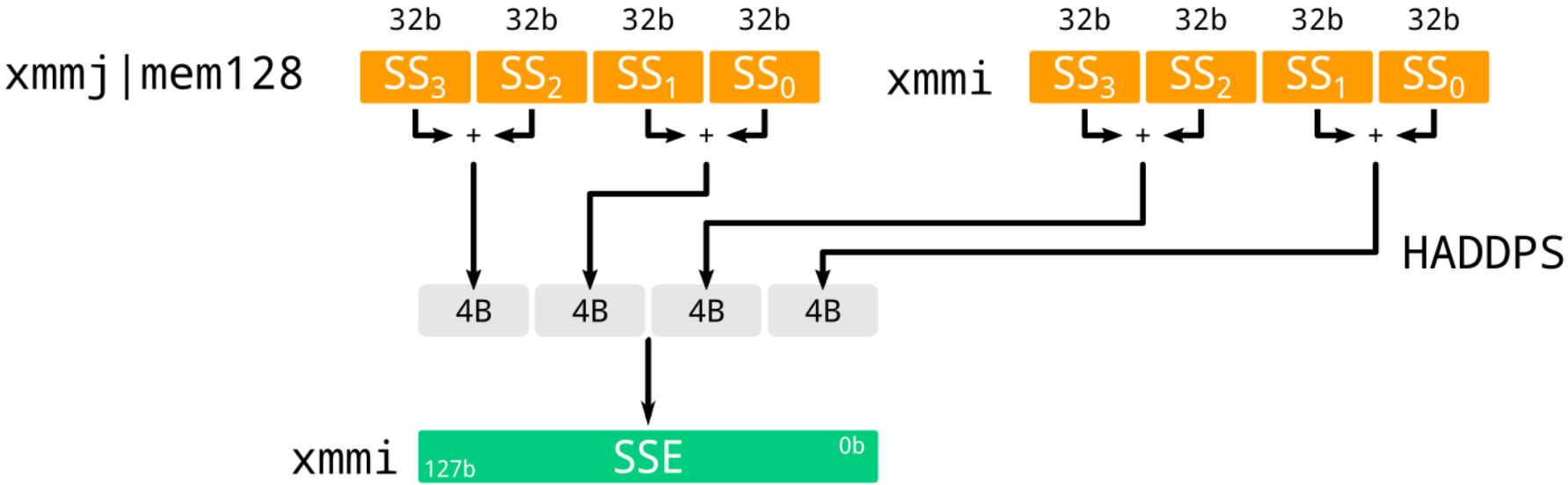
HADDPS / HADDPD (SSE3)

- Packed Single-FP Horizontal Add
- Packed Double-FP Horizontal Add

```
HADDPS xmm1, xmm2/m128 ; 4 SP
HADDPD xmm1, xmm2/m128 ; 2 DP
```

Não é tão simples quanto uma redução!

HADDPS (SSE3)


xmmj|mem128
32b
32b
32b
32b
SS3
SS2
SS1
SS0
xmmi
32b
32b
32b
32b
SS3
SS2
SS1
SS0
HADDPS
4B
4B
4B
4B
xmmi
127b
SSE
0b

Atividades

AT1201

TO-DO!