Aula 12 - Coprocessamento
Arquitetura de Computadores I

Prof. Wagner Guimarães Al-Alam

Universidade Federal do Ceará
Campus de Quixadá

2020-1

Sumário

- Coprocessamento 80x87
- MMX
- SSE
- AVX

Instruções SIMD provavelmente são o melhor lugar para usar Assembly, pois os compiladores normalmente não fazem um bom trabalho usando essas instruções.

Família de Coprocessadores 80x87

- Família de coprocessadores Intel
  - multiplicação
  - divisão
  - soma
  - subtração
  - raiz quadrada
  - logaritmos
- Tipos de dados:
  - números inteiros com sinal de 16-bit, 32-bit, e 64-bit
  - dados BCD de 18-dígitos
  - números de ponto flutuante de 32-bit, 64-bit e 80-bit.

Formatos de Números de Ponto Flutuante

Precisão Simples	**32 bits:** 1 bit para sinal, 8 bits para expoente, e 23 bits para mantissa. Intervalo normalizado aproximado de 2^{-126} *a* 2^{127}[1]. Também chamado de real curto.
Precisão Dupla	**64 bits:** 1 bit para sinal, 11 bits para expoente, e 52 bits para mantissa. Intervalo normalizado aproximado de 2^{-1022} *a* 2^{1023}. Também chamado de real longo.
Precisão Dupla Extendida	**80 bits:** 1 bit para sinal, 16 bits para expoente, e 63 bits para mantissa. Intervalo normalizado aproximado de 2^{-16382} *a* 2^{16383}. Também chamado de real extendido.

[1] O expoente é armazenado de modo a somar 127 ao expoente que está sendo armazenado, assim os expoentes nunca são negativos. Por exemplo o expoente 5 seria salvo como 132.

Arquitetura do 80x87

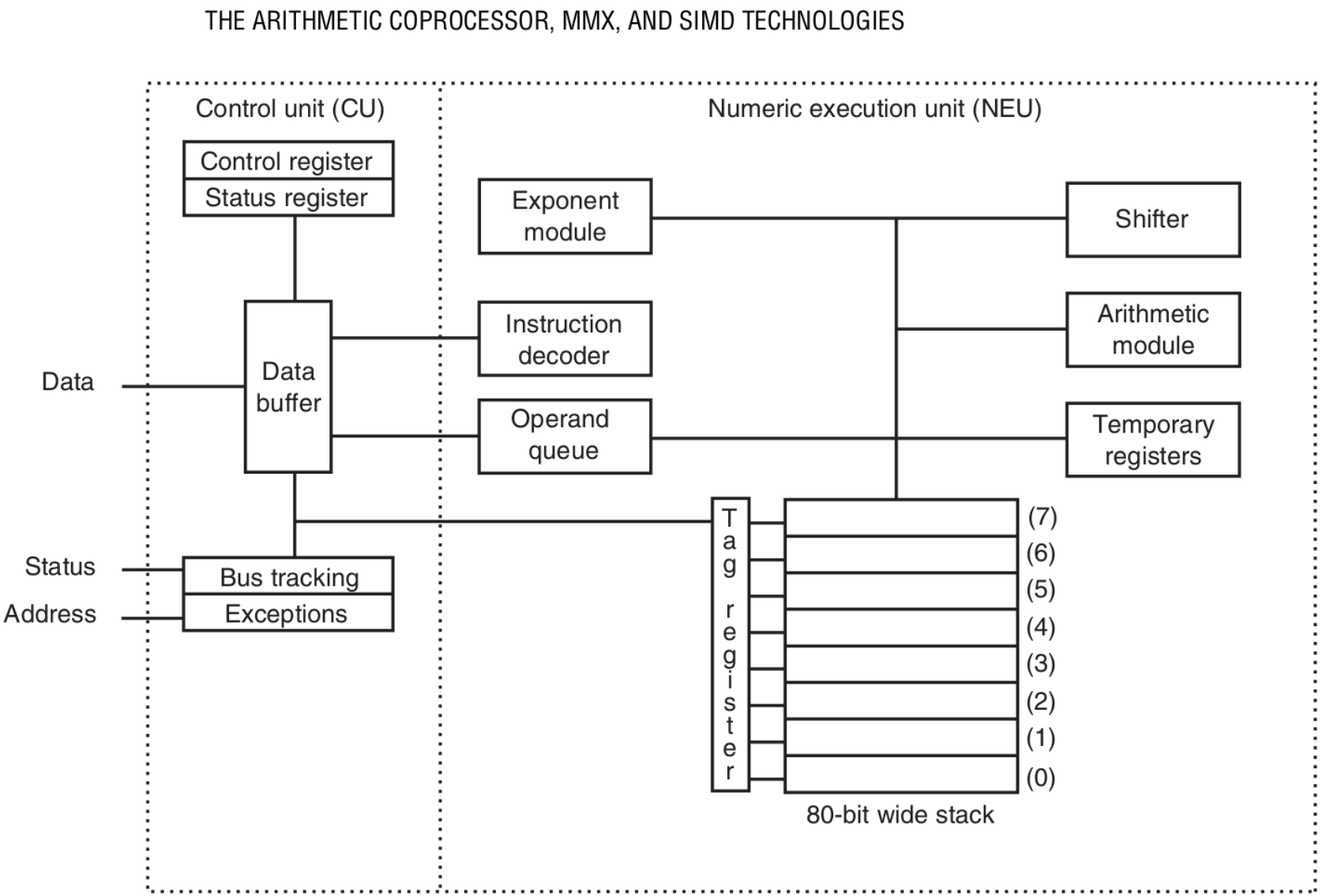
- Conjunto de 68 instruções
- Executa em paralelo ao processador
- Instruções ESC (escape)

Na maioria dos processadores x86 posteriores ao 80486 já possuem internamente as instruções 80x87.

Arquitetura do 80x87

- Unidade de Controle
  - Interface entre coprocessador e o barramento de dados.
  - Os dispositivos monitoram as instruções que passam e quando uma instrução de ESCape(instruções da FPU) surge, o coprocessador executa ela.
- Unidade Numérica de Execução
  - Responsável por executar as instruções do coprocessador
  - Pilha de 8 registradores de 80-bit que mantém os operandos para as instruções aritméticas, bem como seus resultados.
  - Outros registradores: status, controle, tag, e ponteiros de exceção
- Registrador de status:
  - Acessado através da instrução FSTSW (salva o conteúdo do status em uma posição da memória)
- Registrador de controle
  - Seleciona precisão, controle de arredondamento e controle de número infinito.
- Registrador TAG
  - Indica o conteúdo de cada localização na pilha do coprocessador

Arquitetura do 80x87 - Visão Geral



Arquitetura do 80x87 - Registrador de Status

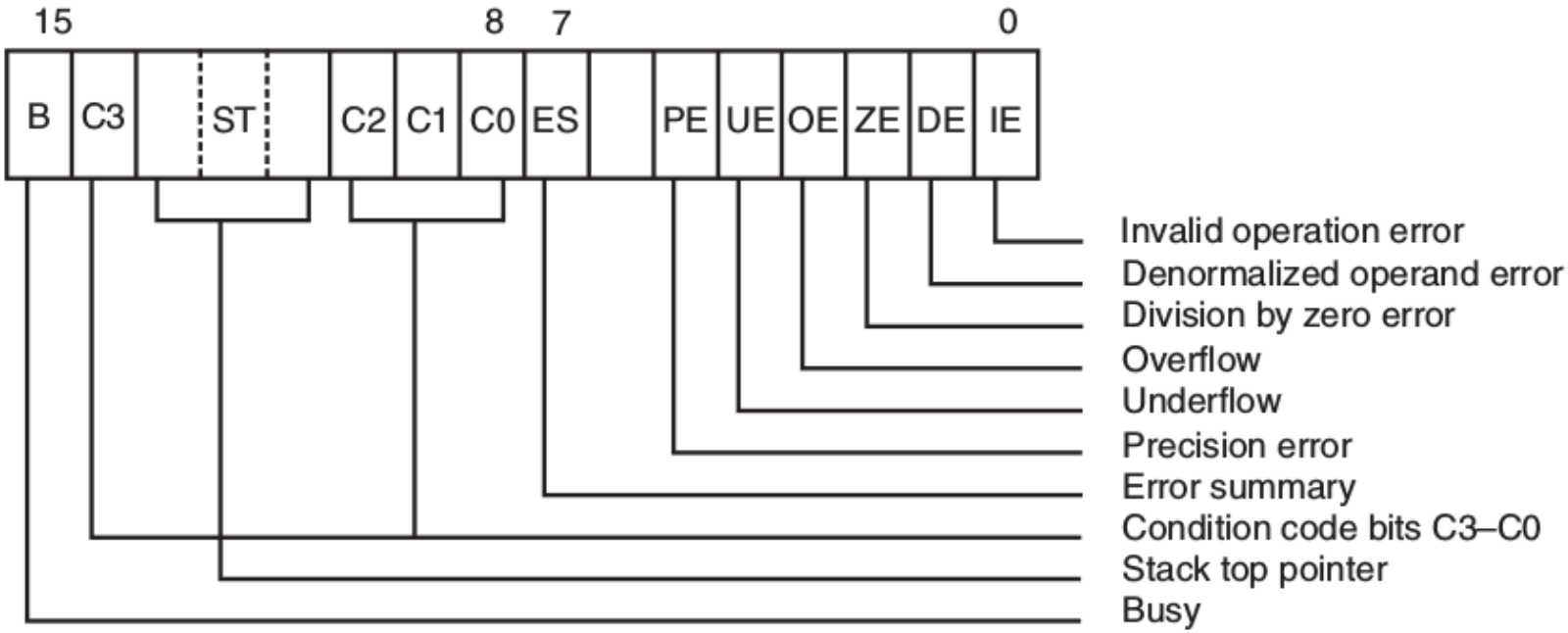


FIGURE 14–5 The 80X87 arithmetic coprocessor status register.

Arquitetura do 80x87 - Registrador de Controle

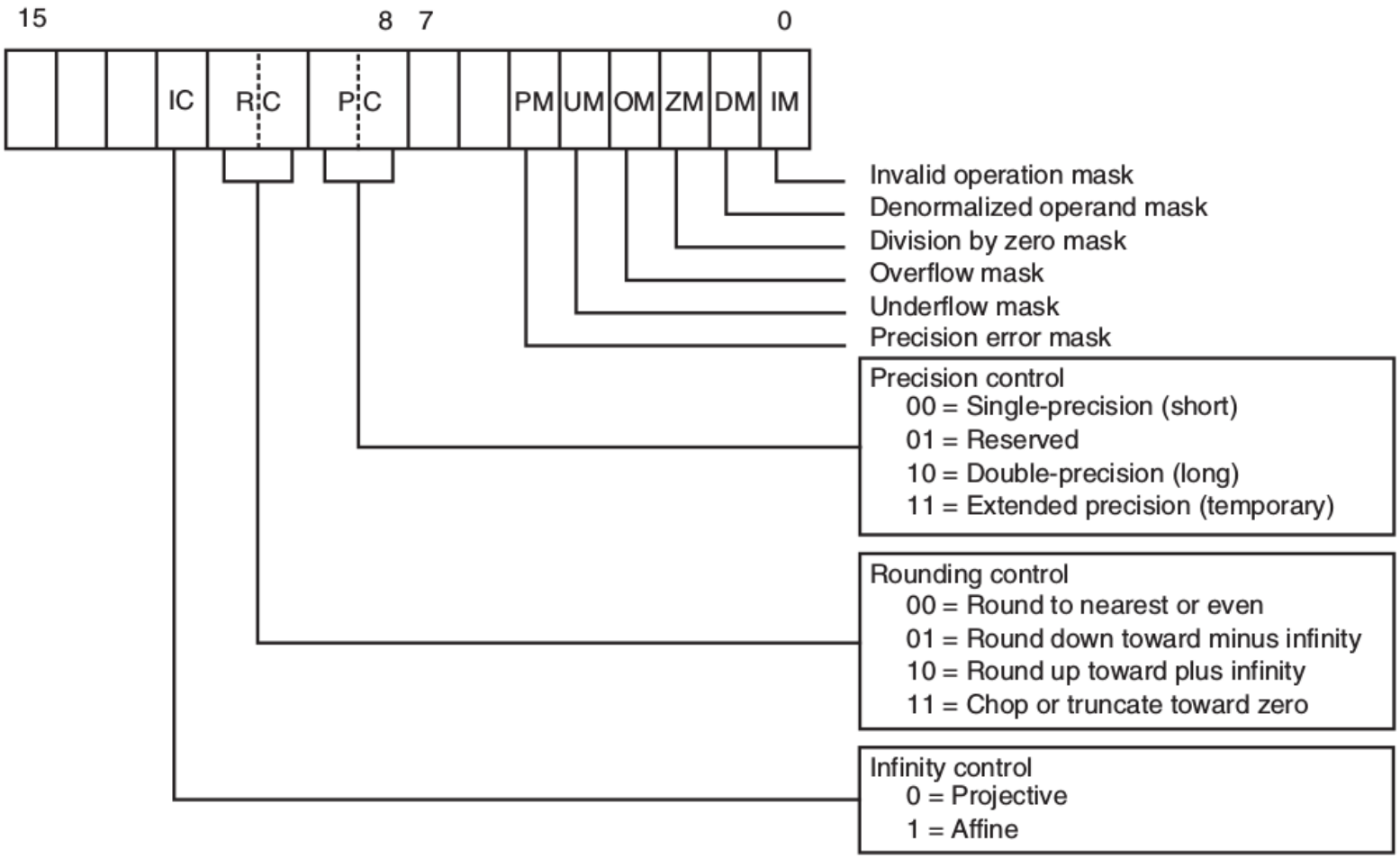


FIGURE 14–6 The 80X87 arithmetic coprocessor control register.

Arquitetura do 80x87 - Registrador TAG

FIGURE 14–7 The 80X87 arithmetic coprocessor tag register.

TAG VALUES:
00 = VALID
01 = ZERO
10 = INVALID or INFINITY
11 = EMPTY

Instruções

- FINIT: Inicialização da FPU
- Transferência de Dados
  - FLD: (Load Floating-Point Value) Carrega um valor da memória para a pilha de registradores da FPU
  - FILD: Mesma coisa de FLD porém converte um número inteiro em um número de precisão dupla no registrador da FPU.
  - FST: (Store Floating-Point Value) Retira um valor de um registrador da FPU e salva na memória preservando o valor no registrador.
  - FSTP: (Store Floating-Point Value and Pop) Mesma coisa do FST removendo ST0

Instruções Aritméticas

- FCHS: Mudar sinal
- FABS: Limpa o sinal, transformando no valor absoluto
- FADD: Soma origem ao destino (Variações: FADDP(Soma e desempilha), FIADD(Soma Inteiro))
- FSUB: Subtrai a origem do destino (Variações:FSUBP, FISUB)
- FSUBR: Subtrai o destino da origem
- FMUL: Multiplica a origem pelo destino (Variações: FMULP, FIMUL)
- FDIV: Divide o destino pela origem (Variações: FDIVP, FIDIV)
- FDIVR: Divide a origem pelo destino

Exemplo

```
1  %include "io.inc"
2  extern printf
3  section .data
4  msg: db "area is %f ",0xA,0      ;msg for printf statement
5  raio dd 2.34
6
7  section .bss
8  area RESQ 1
9
10 section .text
11 global CMAIN
12 CMAIN:
13 FLD DWORD[raio]                  ;push on ndp stack(register) 80bit-radius value
14 FMUL st0, st0                    ;multiply top of stack with top of stack
15 FLDPI                            ;load pi(3.14) on tos hence tos is pi
16 FMUL st1                         ;st1 is 1 below top |||ly st2 is 2 below stack
17 FSTP qword[area]                 ;pops and store the tos(top of stack) to memory
18
19 ;printing results
20 PUSH dword[area+4]               ;result is 64bit so push the upper dword
21 PUSH dword[area]                 ;then lower bcoz u cannot use push qword[]
22 PUSH msg                         ;push message
23 CALL printf                      ;call C-printf
24 ADD esp,12                       ;add e;sp=no of push*4
25
26 XOR eax, eax
27 RET
```

Instruções Comparação

- FCOM: Compara ST0 e ST1
- FCOMP: Compara ST0 e ST1 e desempilha ST0
- FCOMPP: Compara ST0 e ST1 e desempilha ST0 e ST1

Para usar as instuções de salto condicional, use as seguintes instruções:

- Use a instrução FNSTSW para copiar a palavra de status da FPU em AX.
- Use a instrução SAHF para copiar AH no registrador EFLAGS.

A partir o Intel Pentium foi incluída a instrução FCOMI que já realiza automaticamente a cópia das flags da FPU para as EFLAGS da CPU

Comparação de Igualdade

- Números de ponto flutuante não podem ser comparados por igualdade diretamente devido aos arredondamentos.
- Define-se uma constante *epsilon* que irá designar a diferença máxima aceita como uma igualdade.

```
1  section .data
2  epsilon DD 1.0E-12
3  val2 DD 0.0
4  val3 DD 1.001E-13
5  mWrite DB "Os valores sao iguais",0dh,0ah
6  ; valores a comparar
7  ; considerado igual ao val2
8
9  SECTION .text                    ; secao de codigo
10 global main        ; ponto de entrada padrao do GCC
11 main:
12 mov ebp, esp; for correct debugging
13 ; if( val2 == val3 ), mostrar "Valores sao iguais".
14 fld DWORD [epsilon]
15 fld DWORD[val2]
16 fsub DWORD[val3]
17 fabs
18 fcomi ST0,ST1
19 ja end     ; nao imprime mensagem
20 ...
```

Melhorando o Desempenho

- Melhorias na arquitetura como pipeline, cache e SIMD são muito significantes
- A intel analisou aplicações multimídia e constatou que elas compartilham as seguintes características:
  - Pequenos tipos de dados nativos (pixel 8-bit e áudio 16-bit).
  - Operações recorrentes
  - Paralelismo implícito

Tecnologia MMX

- Tecnologia MMX (multimedia extensions)
  - Especialmente para aplicações como movimento em vídeos, gráficos combinados com vídeo, processamento de áudio, síntese de áudio, síntese de fala e compressão, telefonia, video conferência, e ainda gráficos em 2D e 3D.
  - Operam em paralelo com outras instruções como as instruções do coprocessador aritmético.

Arquitetura MMX

- Adiciona 87 novas instruções
  - aritméticas, comparação, conversão, lógicas, deslocamentos e transferências de dados.
- Novo pacote de tipo de dados
  - oito pacotes consecutivos de bytes 8-bit;
  - quatro pacotes consecutivos de palavras de 16-bit;
  - dois pacotes consecutivos de palavras duplas de 32-bit.
- Pode executar em dois modos 32-bit ou 64-bit
- Registradores MM0 a MM7

Acessos aos Registradores

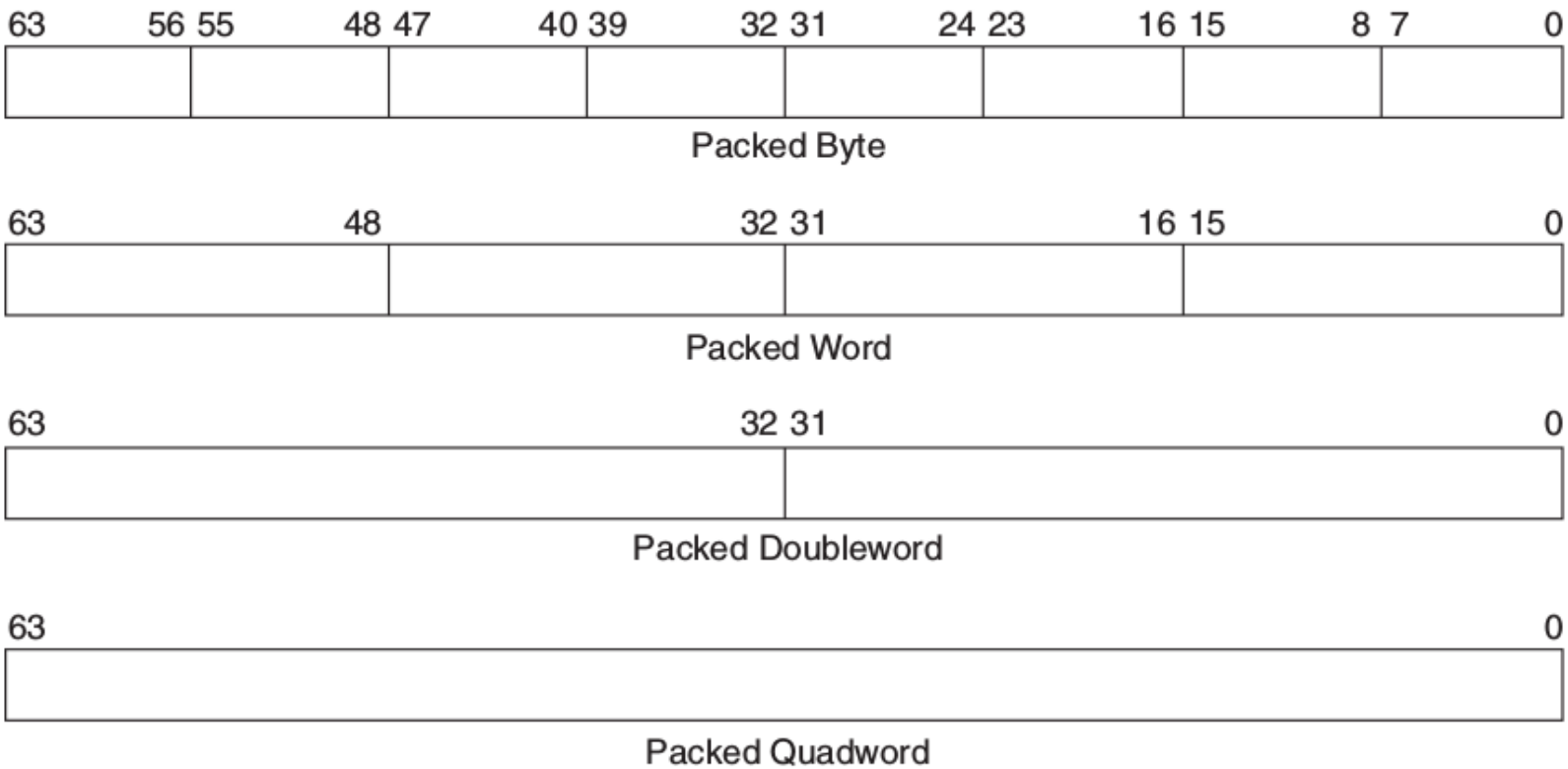


FIGURE 14–11 The structure of data stored in the MMX registers.

Exemplo SomaArrayBytes

```
1  %include "io.inc"
2
3  section .bss
4  BLOCKA RESB 1000
5  BLOCKB RESB 1000
6  BLOCKC RESB 1000
7
8  section .text
9  global CMAIN
10 CMAIN:
11     mov ebp, esp; for correct debugging
12     call SUM
13     xor eax, eax
14     ret
15
16 SUM:
17     MOV ECX, 1000
18 REPEAT:
19     MOV AL, [BLOCKA+ECX-1]
20     ADD AL, [BLOCKB+ECX-1]
21     MOV [BLOCKC+ECX-1], AL
22     loop REPEAT
23     ret
```

Procedimento que soma BLOCKA0 a BLOCKB e armazena o resultado em BLOCKC

Exemplo SomaArrayBytes - MMX

```
1  %include "io.inc"
2
3  section .bss
4  BLOCKA RESB 1000
5  BLOCKB RESB 1000
6  BLOCKC RESB 1000
7
8  section .text
9  global CMAIN
10 CMAIN:
11     mov ebp, esp; for correct debugging
12     call SUM
13     xor eax, eax
14     ret
15
16 SUM:
17     MOV ECX, 125
18 REPEAT:
19     MOVQ MM0, QWORD [BLOCKA+ECX-8]
20     PADDD MM0, QWORD [BLOCKB+ECX-8]
21     MOVQ QWORD [BLOCKC+ECX-8],MM0
22     loop REPEAT
23     ret
```

Procedimento que soma BLOCKA0 a BLOCKB e armazena o resultado em BLOCKC

Exemplo SomaArrayBytes - MMX

- Se você comparar as duas versões, irá reparar que a versão MMX irá iterar 125 vezes enquanto a versão tradicional irá iterar 1000 vezes.

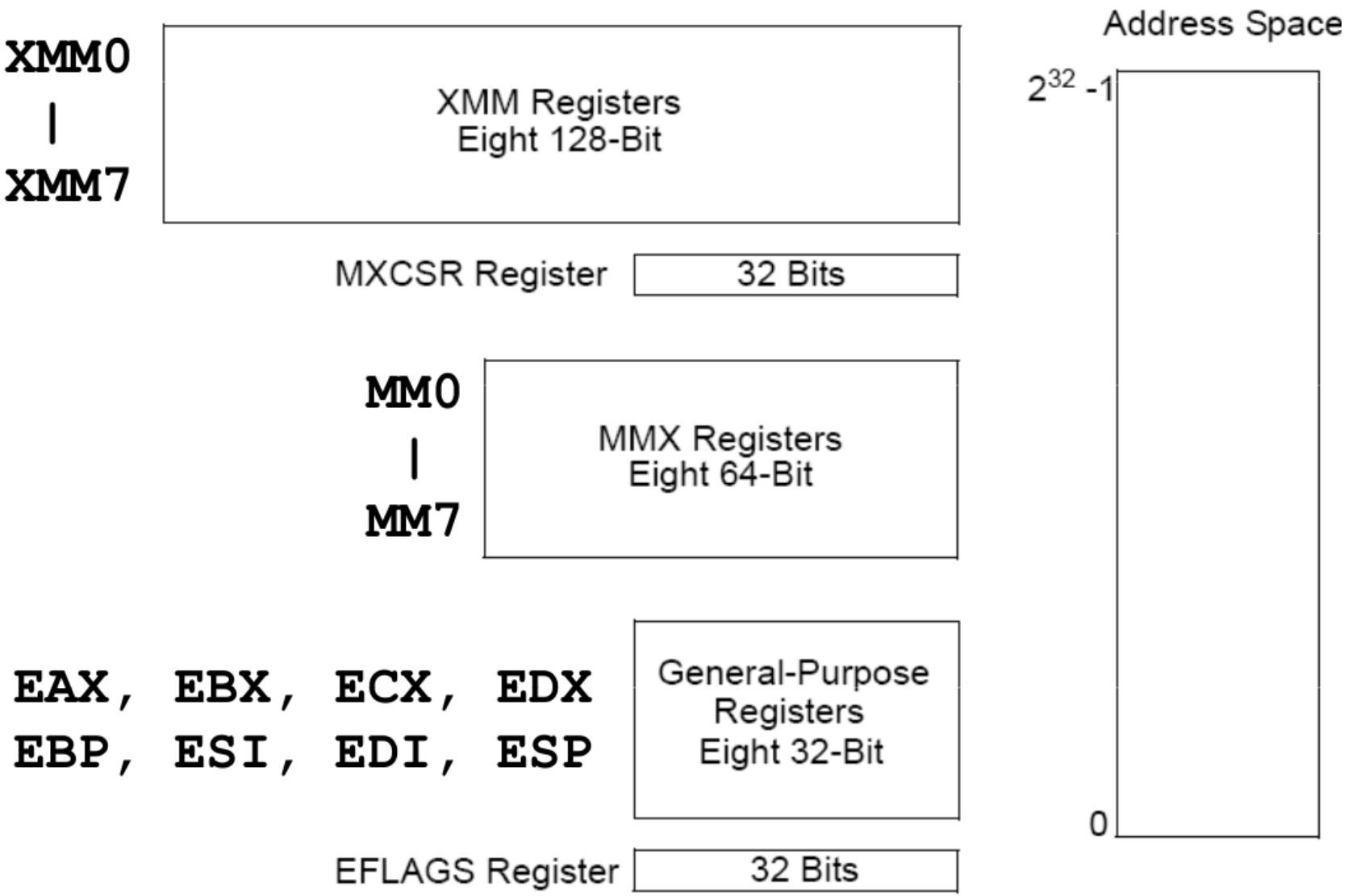
Instrução PADDB

```
1   case PADDB:
2   if(OperandSize == 64) {
3       //PADDB instruction with 64-bit operands:
4       Destination[0..7] = Destination[0..7] + Source[0..7];
5       Destination[8..15] = Destination[8..15] + Source[8..15];
6       Destination[16..23] = Destination[16..23] + Source[16..23];
7       Destination[24..31] = Destination[24..31] + Source[24..31];
8       Destination[32..39] = Destination[32..39] + Source[32..39];
9       Destination[40..47] = Destination[40..47] + Source[40..47];
10      Destination[48..55] = Destination[48..55] + Source[48..55];
11      Destination[56..63] = Destination[56..63] + Source[56..63];
12  }else {
13      //PADDB instruction with 128-bit operands:
14      Destination[0..7] = Destination[0..7] + Source[0..7];
15      Destination[8..15] = Destination[8..15] + Source[8..15];
16      Destination[16..23] = Destination[16..23] + Source[16..23];
17      Destination[24..31] = Destination[24..31] + Source[24..31];
18      Destination[32..39] = Destination[32..39] + Source[32..39];
19      Destination[40..47] = Destination[40..47] + Source[40..47];
20      Destination[48..55] = Destination[48..55] + Source[48..55];
21      Destination[56..63] = Destination[56..63] + Source[56..63];
22      Destination[64..71] = Destination[64..71] + Source[64..71];
23      Destination[72..79] = Destination[72..79] + Source[]72..79;
24      Destination[80..87] = Destination[80..87] + Source[80..87];
25      Destination[88..95] = Destination[88..95] + Source[88..95];
26      Destination[96..103] = Destination[96..103] + Source[96..103];
27      Destination[104..111] = Destination[104..111] + Source[104..111];
28      Destination[112..119] = Destination[112..119] + Source[112..119];
29      Destination[120..127] = Destination[120..111] + Source[120..127];
30  }
```

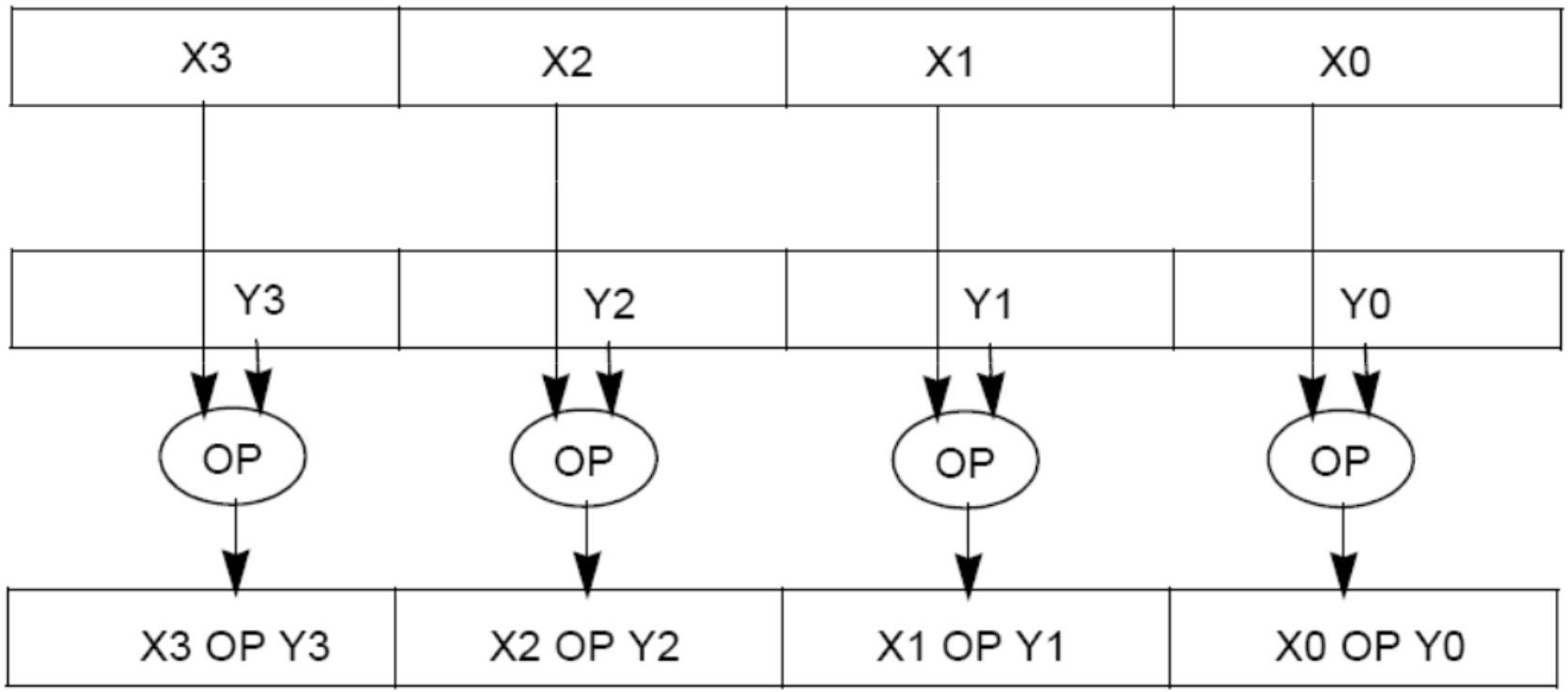
Tecnologia SSE

- Adiciona 8 registradores de dados de 128-bit (registradores XMM) em modos menores que 64-bit ou 16 registradores XMM em modo 64-bit.
- Possui um registrador de 32-bit para controle e status.
- Adiciona um novo tipo de dado: pacote de números de ponto flutuante de precisão simples 128-bit (4 números de ponto flutuante)
- Inclui operações SIMD sobre pacotes de números de ponto flutuante de precisão simples.
- Muitas instruções SSE requerem alinhamento de endereços de 16-bit.

Tecnologia SSE

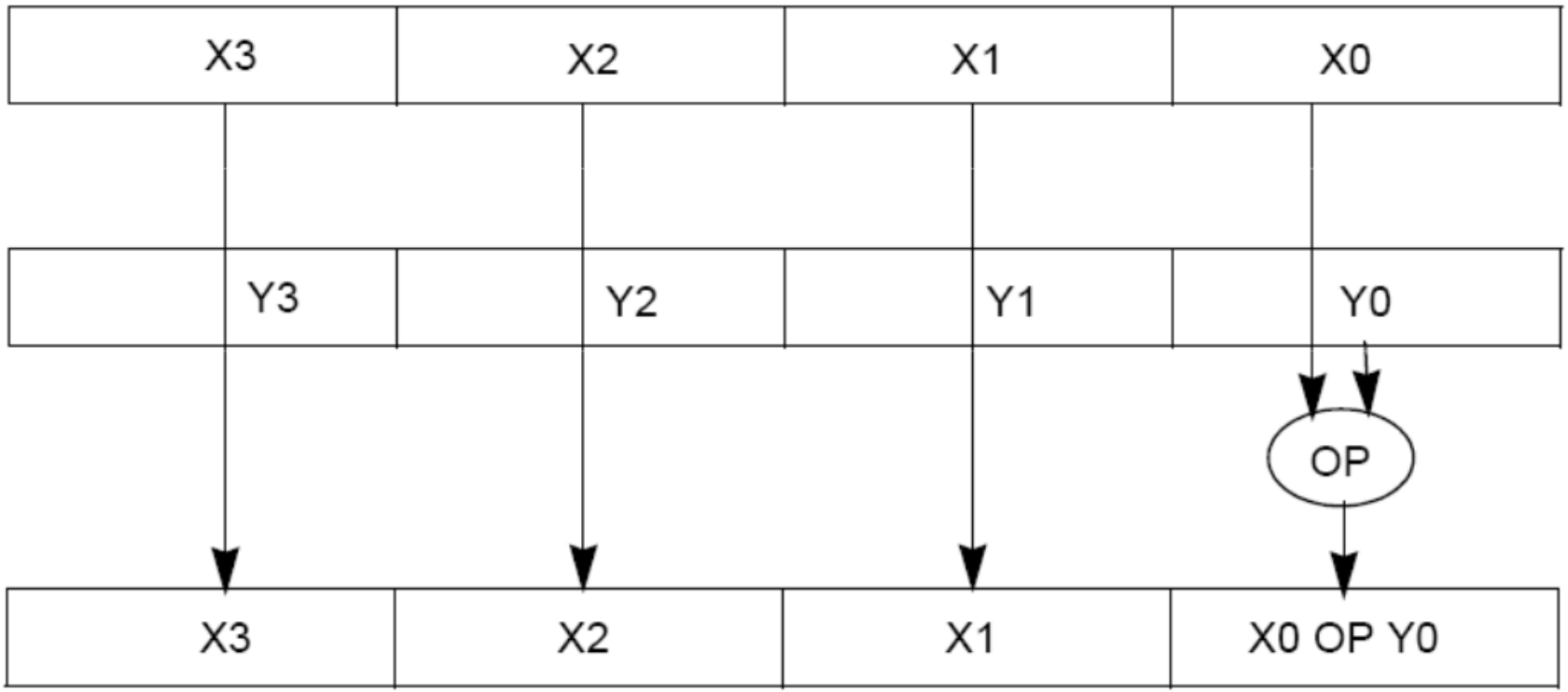


Operação de Pacote de Números de Ponto Flutuante



ADDPS/SUBPS: packed single-precision FP

Operação Escalar de Números de Ponto Flutuante

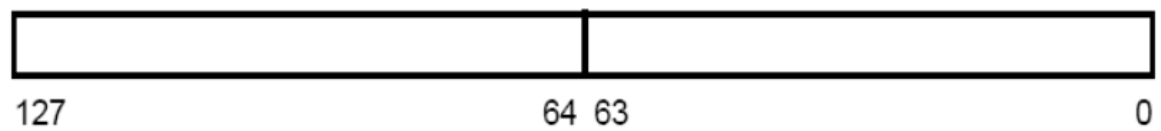


ADDSS/SUBSS: scalar single-precision FP used as FPU

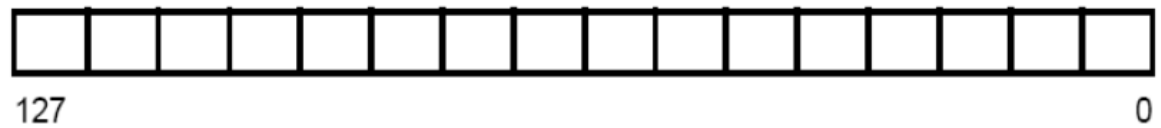
SSE2

- Provê a habilidade de executar operações SIMD em números de ponto flutuante de precisão dupla, permitindo técnicas gráficas avançadas como ray tracing (método de renderização 3D pela simulação dos raios de luz que percorrem o mundo)
- Provê throughput melhor operando com inteiros empacotados em 128-bit, útil para criptografias como RSA e RC5

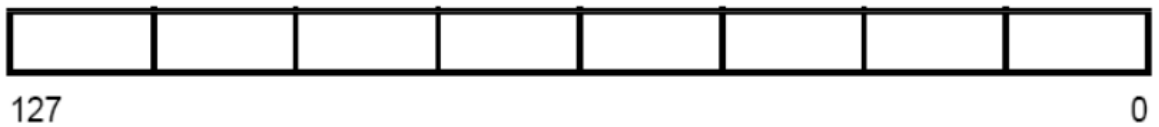
SSE2



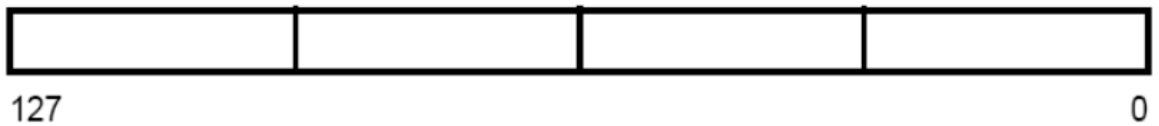
128-Bit Packed Double-Precision Floating-Point



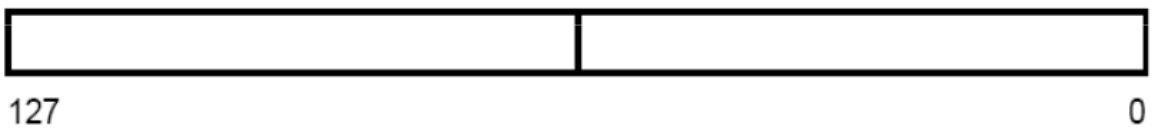
128-Bit Packed Byte Integers



128-Bit Packed Word Integers



128-Bit Packed Doubleword Integers



128-Bit Packed Quadword Integers

SSE3

- Extensão do SSE e SSE2 que permite operações assimétricas, diferentes do modelo simétrico vertical permitido nos outros modos.

SSE4

- Incorporada nos processadores da Intel a partir de 2008.
- Novas instruções para processamento de dados empacotados que incluíram, entre outras:
  - pesquisa de strings
  - produtos vetoriais
  - verificação de integridade (CRC)

AVX

- AVX usa 8 registradores YMM em modo 32-bit (YMM0–YMM7) ou 16 registradores em modo 64-bit (YMM0–YMM15).
- Cada registrador YMM contém:
  - oito números de ponto flutuante de precisão simples de 32-bit, ou
  - quatro números de ponto flutuante de precisão dupla de 64-bit.
- Permite instruções com 3 operandos (c = a + b)

Evolução das Tecnologias de Coprocessamento

MMX:
Multimedia extension

SSE:
Streaming SIMD extension

AVX:
Advanced vector extensions