Linguagem de Montagem

MMX
Aula 13

Edmar André Bellorini

Ano letivo 2016

Introdução

- Taxonomia de Flynn

  "*classificação para computação paralela*"

  - SISD (*Single Instruction, Single Data*)
    Uniprocessadores
  - SIMD (*Single Instruction, Multiple Data*)
    Processadores vetoriais
  - MISD (*Multiple Instruction, Single Data*)
    Não implementado comercialmente
  - MIMD (*Multiple Instruction, Multiple Data*)
    Multiprocessadores
- MMX
  - conjunto de instruções vetoriais para inteiros
    - SIMD
      1996 - Pentium
      Multi**M**edia e**X**tension ou **M**atrix **M**ath e**X**tensions
      (não oficial)

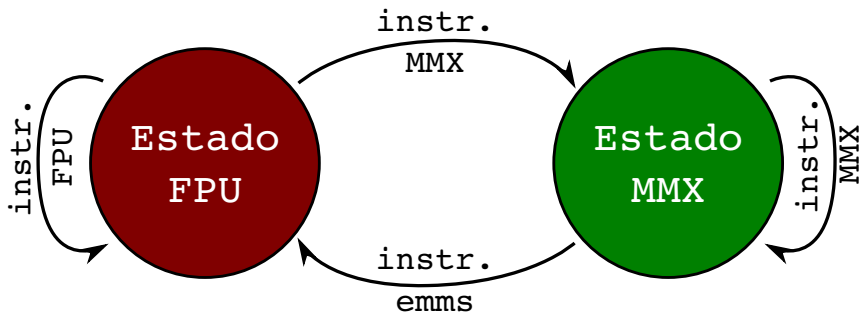
MMX

- SIMD
  - Conjunto de instruções vetoriais para inteiros
  - Todas começam com **p**

    Exceção:

    ```
    emms, movd, movq
    ```

- Registradores:
  - Usa o mesmo espaço de registradores da FPU
    - **Não** é possível usar MMX e FPU ao mesmo tempo sempre deve existir a instrução **emms** ao encerrar uma computação MMX

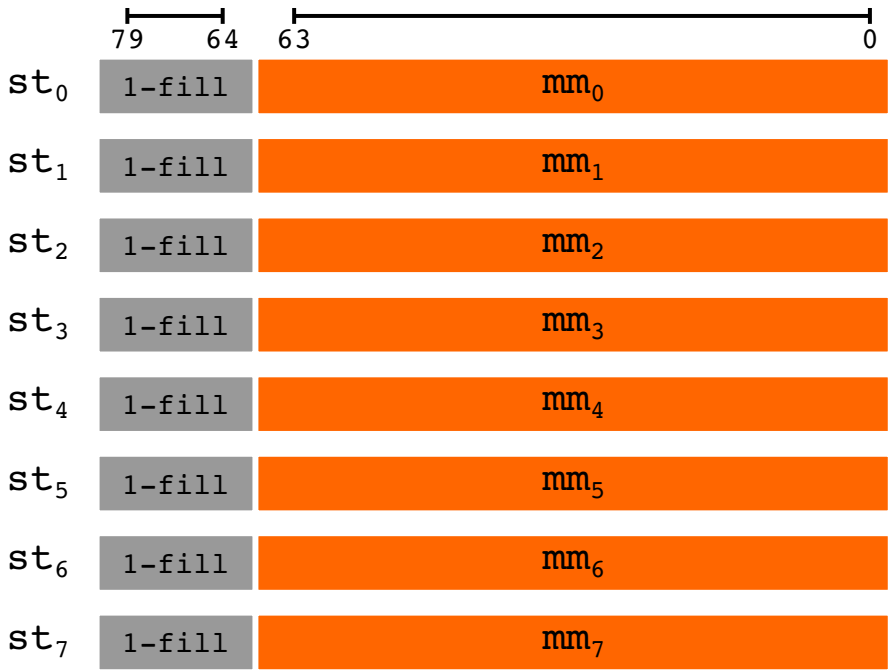


Registradores

- 8 registradores $mm_{[0-7]}$ de 64 bits
  - compartilhado com FPU

    $st_0 = mm_0$

    2 bytes mais significativos de st_0 não são usados por mm_0



Acesso aos Registradores mm_i

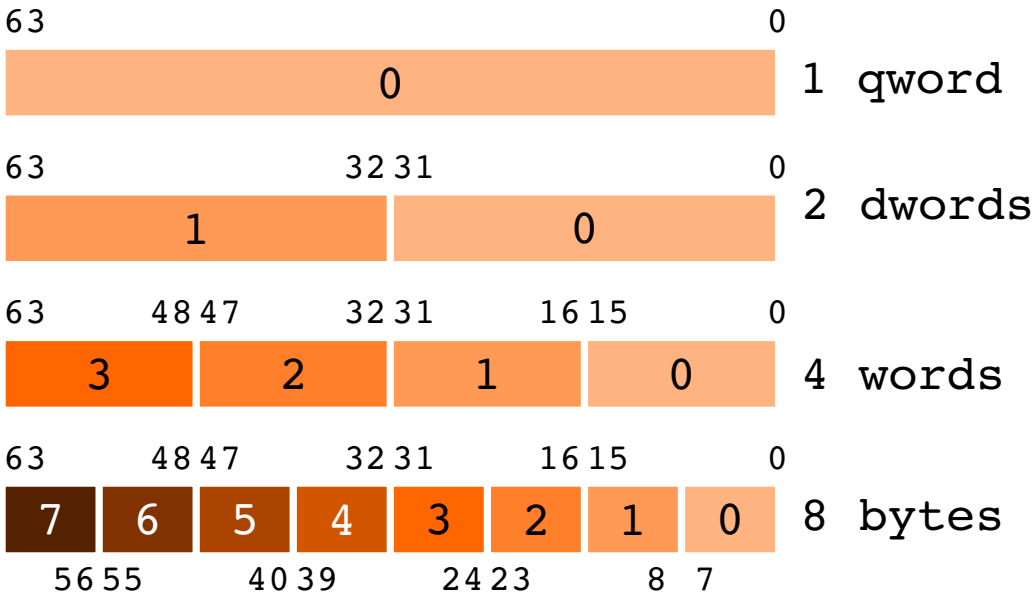
- Acesso é realizado em todas as palavras do reg. mm_i

  1 qword $= \{0_{63..0}\}$

  2 dwords $= \{1_{63..32}, 0_{31..0}\}$

  4 words $= \{3_{63..48}, 2_{47..32}, 1_{31..16}, 0_{15..0}\}$

  8 bytes $= \{7_{63..56}, 6_{55..48}, 5_{47..40}, 4_{39..32}, 3_{31..24}, 2_{23..16}, 1_{15..8}, 0_{7..0}\}$



Movimentação de Dados - mov[q,d]

- Carregar palavras de *mem* ou *reg* para reg. mm_i

```
movd  mmxreg64,r/m32 ; '0-fill'
movq  mmxreg64,r/m64
```

  - instrução 32 bits *0-fill* parte alta do registrador *destino*

- Armazenar palavras de um reg. mm_i para *mem* ou *reg*

```
movd  r/m32, mmxreg64 ; 'truncamento'
movq  r/m64, mmxreg64
```

  - instrução de 32 bits *trunca* parte alta do registrador *fonte*

Exemplo a13e01.asm

- GDB não apresenta registradores mmx_i
  - Abstração e verificação usando

```
$ info float
```

```
30  ...
31      ; mm0 = vetorChar[0-7]
32    movq mm0, [vetorChar]
33
34    ; mm1 = vetorChar[0-3]
35    movd mm1, [vetorChar]
36  ...
```

Operações Lógicas MMX - pt 1 / 3

- Op. Lógica **AND**

```
pand mmxreg64, mmxreg64/m64
```

- Op. Lógica **OR**

```
por  mmxreg64, mmxreg64/m64
```

- Op. Lógica **XOR**

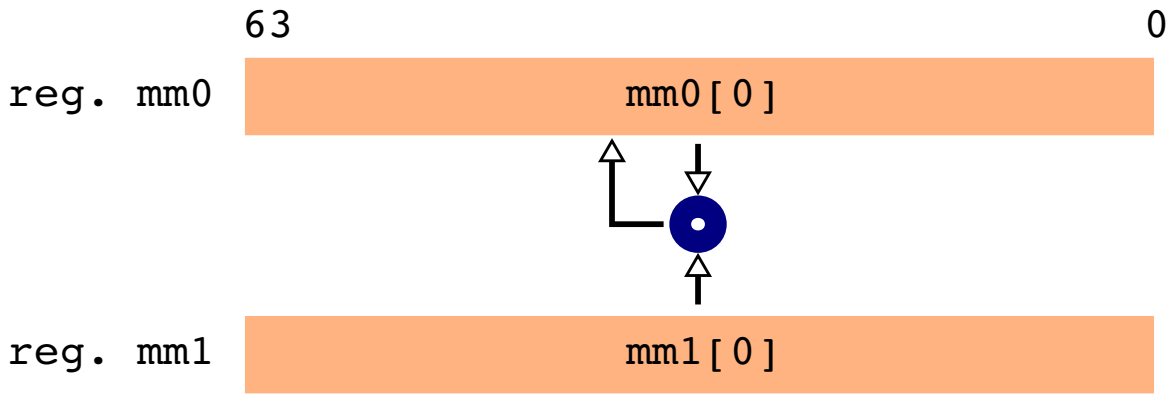
```
pxor mmxreg64, mmxreg64/m64
```

- **pand**, **por** e **pxor** são operações bit-a-bit
  - operação é aplicada em todos os 64 bits das palavras

Operações Lógicas MMX - pt 2 / 3

- Exemplo:

```
pand mm0, mm1
```



Operações Lógicas MMX - pt 3 / 3 - Exemplo

- *upperCase* e *lowerCase* para palavras de 8 caracteres

```
49  ...
50  L0:
51    ; charMisto para Maiuscula (charMisto2Maius)
52    movq mm1, [charMisto]
53    pand mm1, [mask2Maius]
54    movq [charMisto2Maius], mm1
55    mov byte [charMisto2Maius+maxChar], 0
56  ...
```

Operações de Deslocamento - pt 1/3

- Op. de Deslocamento à **esquerda**
  - para **quatro** palavras de 16 bits

```
psllw mmxreg, r/m64 ; w = 4 words (16 bits)
psllw mmxreg, imm8  ; w = 4 words (16 bits)
```

  - para **duas** palavras de 32 bits

```
pslld mmxreg, r/m64 ; d = 2 dwords (32 bits)
pslld mmxreg, imm8  ; d = 2 dwords (32 bits)
```

  - para **uma** palavra de 64 bits

```
psllq mmxreg, r/m64 ; q = 1 qwords (64 bits)
psllq mmxreg, imm8  ; q = 1 qwords (64 bits)
```

Operações de Deslocamento - pt 2/3

- Op. de Deslocamento à **direita**
  - para **quatro** palavras de 16 bits

```
psrlw mmxreg, r/m64 ; w = 4 words (16 bits)
psrlw mmxreg, imm8  ; w = 4 words (16 bits)
```

  - para **duas** palavras de 32 bits

```
psrld mmxreg, r/m64 ; d = 2 dwords (32 bits)
psrld mmxreg, imm8  ; d = 2 dwords (32 bits)
```

  - para **uma** palavra de 64 bits

```
psrlq mmxreg, r/m64 ; q = 1 qwords (64 bits)
psrlq mmxreg, imm8  ; q = 1 qwords (64 bits)
```

Operações de Deslocamento - pt 3/3

- Op. de Deslocamento à **direita**
  - para **quatro** palavras de 16 bits

```
psraw mmxreg, r/m64 ; w = 4 words (16 bits)
psraw mmxreg, imm8  ; w = 4 words (16 bits)
```

  - para **duas** palavras de 32 bits

```
psrad mmxreg, r/m64 ; d = 2 dwords (32 bits)
psrad mmxreg, imm8  ; d = 2 dwords (32 bits)
```

Exemplo a13e03.asm

- Dado um vetor de inteiros:

```
vetInt1 = {1, 2, 3, 4, 5, 6, 7, -8}
```

  - multiplicação dos elementos de *vetInt*1 por 4

```
vetResult1[] = 4 * vetInt[] ; des. esq em 2
```

  - divisão dos elementos de *vetResult*1 por 2

```
vetResult2[] = vetResult1[]/2 ; des. dir em 1
```

```
22  ...
23  multi:
24    movq mm0, [vetInt1+rcx]
25    pslld mm0, 2
26    movq [vetResult1+rcx], mm0
27    add rcx, 8
28    cmp rcx, 32
29    jl multi
30  ...
```

Adição MMX - pt 1 / 3

- Não-Saturadas
  - para **8** palavras de 8 bits

```
paddb mmxreg64, mmxreg64/m64
```

  - para **4** palavras de 16 bits

```
paddw mmxreg64, mmxreg64/m64
```

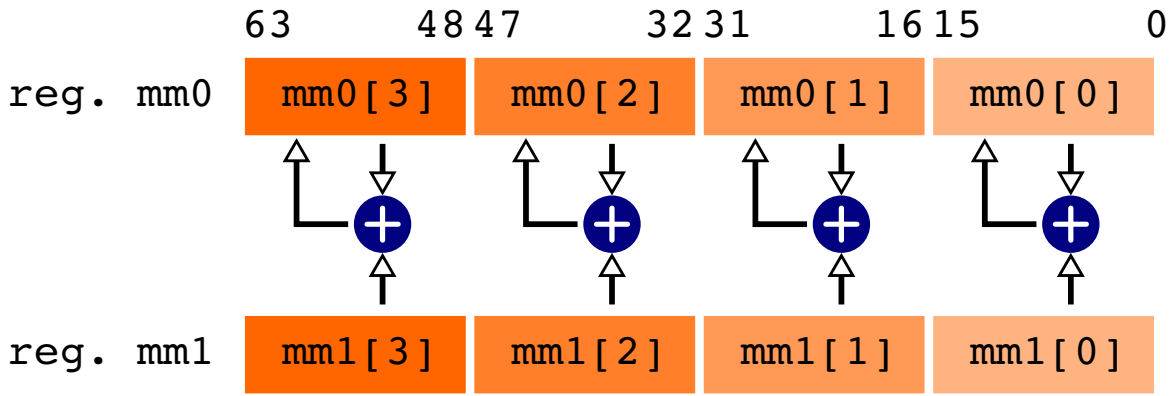
  - para **2** palavras de 32 bits

```
paddd mmxreg64, mmxreg64/m64
```

Adição MMX - pt 2 / 3

- Exemplo:

```
paddw mm0, mm1
```



Adição MMX - pt 3 / 3

- Saturadas
  - para **8** palavras **sinalizadas** de 8 bits

    ```
    paddsb mmxreg64, mmxreg64/m64
    ```

    soma **satura** em -128 ou +127
  - para **8** palavras **não sinalizadas** de 8 bits

    ```
    paddusb mmxreg64, mmxreg64/m64
    ```

    soma **satura** em 255
  - para **4** palavras **sinalizadas** de 16 bits

    ```
    paddsw mmxreg64, mmxreg64/m64
    ```

    soma **satura** em -32.768 ou +32.767
  - para **4** palavras **não sinalizadas** de 16 bits

    ```
    paddusw mmxreg64, mmxreg64/m64
    ```

    soma **satura** em 65.535

Subtração MMX - pt 1 / 2

- Não-Saturadas
  - para **8** palavras de 8 bits

```
psubb mmxreg64, mmxreg64/m64
```

  - para **4** palavras de 16 bits

```
psubw mmxreg64, mmxreg64/m64
```

  - para **2** palavras de 32 bits

```
psubd mmxreg64, mmxreg64/m64
```

Subtração MMX - pt 2 / 2

- Saturadas
  - para **8** palavras **sinalizadas** de 8 bits

    ```
    psubsb mmxreg64, mmxreg64/m64
    ```

    subtração **satura** em -128 ou +127
  - para **8** palavras **não sinalizadas** de 8 bits

    ```
    psubusb mmxreg64, mmxreg64/m64
    ```

    subtração **satura** em 0
  - para **4** palavras **sinalizadas** de 16 bits

    ```
    psubsw mmxreg64, mmxreg64/m64
    ```

    subtração **satura** em -32.768 ou +32.767
  - para **4** palavras **não sinalizadas** de 16 bits

    ```
    psubusw mmxreg64, mmxreg64/m64
    ```

    subtração **satura** em 0

Exercício de Fixação

- UEF1301 (**Ú**ltimo **E**xercício de **F**ixação)
  - Teste **ao menos 2** instruções lógicas MMX
  - Teste **ao menos 3** instruções MMX de soma
  - Teste **ao menos 3** instruções MMX de subtração
- Seja criativo
  - use vetores **inicializados** e **não inicializados**
  - use o **gdb** para comprovar os resultados
- Entrega de pacote .zip contendo:
  - Relatório .pdf com os nomes **da dupla**
  - Códigos fontes usados para os testes

Obs.: Exercício pensado originalmente: construa um programa para aumentar ou diminuir o brilho de uma imagem .bmp (32bits). Onde os parâmetros são passados por argumentos (S.O.) e a saída em arquivo.

Dica: obtenham aprovação esse ano caso não queiram fazê-lo!

Relatório

- Somente relatório
  - O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
    - Arquivo **modeloRelatorioLM.odt**
    - A data do relatório é a data de entrega (ver moodle)
  - Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão: **TY.PXX.nome.sobrenome.pdf**
    - TY é o número da turma prática (1, 2, 3 ou 4)
    - PXX é o número da prática, neste caso: P13
  - Ex.: alunos LinChao e Millind Mittal da turma prática 7 (de 1992):
    - T7.P13.Lin.Chao.Millind.Mittal.pdf