

Linguagem de Montagem

Registradores e Instrução MOV Aula 03

Edmar André Bellorini

Ano letivo 2021

Dados Inicializados (.data) - Exemplos da Aula 02

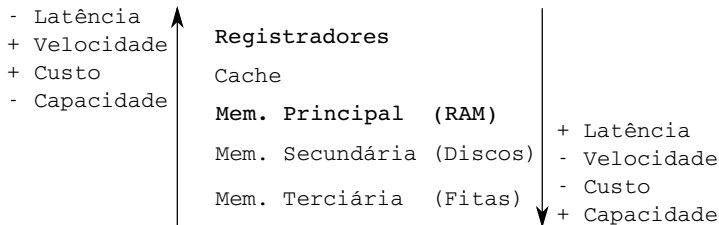
```
1 ; Aula 02 - Estrutura dos Programas
2 ; arquivo: a02e01.asm
3 ; objetivo: dados inicializados
4 ; nasm -f elf64 a02e01.asm ; ld a02e01.o -o a02e01.x
5
6 section .data
7     v1: db    0x55                ; byte 0x55
8     v2: db    0x55,0x56,0x57      ; 3 bytes em sucessao
9     v3: db    'a',0x55            ; caracteres sao aceitos com aspas
10    v4: db    'hello',13,10,'$'   ; strings tambem
11    v5: dw    0x1234              ; 0x34 0x12
12    v6: dw    'a'                 ; 0x61 0x00
13    v7: dw    'ab'                ; 0x61 0x62
14    v8: dw    'abc'               ; 0x61 0x62 0x63 0x00 (string)
15    v9: dd    0x12345678          ; 0x78 0x56 0x34 0x12
```

code: a02e01.asm (parcial)

Dados em memória

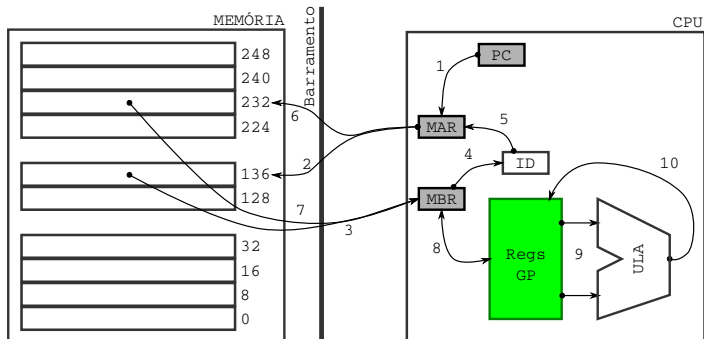
■ Hierarquia das Memórias

- Agrupar memórias de grande capacidade, baixo custo, porém lentas com memórias rápidas, porém de alto custo e baixa capacidade
- Melhor estudado na disciplina de Organização e Arquitetura de Computadores (3ª Série)



Registradores

- É o tipo de memória mais rápida encontrada nos sistemas computacionais
- Faz parte do núcleo dos processadores



Classificação dos Registradores

- Visíveis ao usuário (programador)
 - São usados pelos programadores (ou montadores) para reduzir o acesso à memória e otimizar códigos
 - Ex.: Registradores de Propósito Geral (Reg GP)
- Não Visíveis
 - Controla o fluxo de operações internas e podem ser acessados somente por alguns programas privilegiados do S.O.
 - Ex.:
 - PC (*Program-Counter*),
 - MAR (*Memory Address Register*) e
 - MBR (*Memory Buffer Register*)

Registradores Visíveis

- de Propósito Geral

- Armazenam dados e endereços usados nas operações lógicas e aritméticas
- Usados diretamente pelo usuário

- de Controle

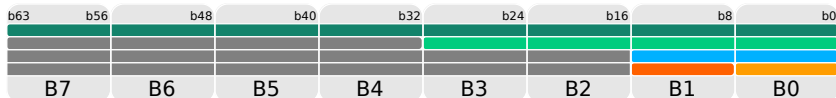
- Flags (condicionais) e ponteiros para segmentos/pilha
- Normalmente não são usados diretamente pelo usuário

Registradores de Propósito Geral

- Máquinas de 32 bits (x86)
 - Gerais: EAX, EBX, ECX, EDX
 - Segmentos: ESI, EDI, EBP, ESP
- Máquinas 64 bits (x86_64)
 - Gerais: RAX, RBX, RCX, RDX, R8 até R15
 - Segmentos: RSI, RDI, RBP, RSP
- Acesso aos dados de um registrador
 - Importante: o Registrador é um único espaço físico 4B (x32) ou 8Bytes (x86_64)
 - Acesso realizado em palavras de 1B, 2B, 4B ou 8Bytes (x86_64)

Exemplo de Registrador: RAX

Registrador #A#: ■ rax ■ eax ■ ax ■ ah ■ al ■ não acessado



- O acesso ao registrador pode ser realizado de diversas formas
 - rax: acessa os 64 bits
 - eax: 32 bits menos significativos de RAX (0 até 31)
 - ax: 16 bits menos significativos de RAX (0 até 15)
 - al: 8 bits menos significativos de RAX (0 até 7)
 - ah: 8 bits mais significativos de AX (8 até 15)
- Essa forma de acesso também vale para RBX, RCX e RDX

Exemplo de acesso ao registrador RAX

■ Programa de 64 bits

```
6 section .data
7     num: dq 0x7766554433221101
8
9 section .text
10    global _start
11
12    _start:
13        mov rax, [num]
14
15    fim:
16        mov rax, 60
17        mov rdi, 0
18        syscall
```

code: a03e01.asm

Exemplo de acesso ao registrador RAX

```
mov rax, [num] ; rax = 0x7766554433221101
```

Registrador #A#: ■ rax ■ eax ■ ax ■ ah ■ al ■ não acessado

b63	b56	b48	b40	b32	b24	b16	b8	b0
B7	B6	B5	B4	B3	B2	B1	B0	
0x77	66	55	44	33	22	11	01	

gdb:

p/x \$registrador ; hexadecimal
p/d \$registrador ; decimal

■ al
0x01 ou 1_d

■ ah
0x11 ou 17_d

■ ax
0x1101 ou 4353_d

■ eax
0x33221101 ou 857.870.593_d

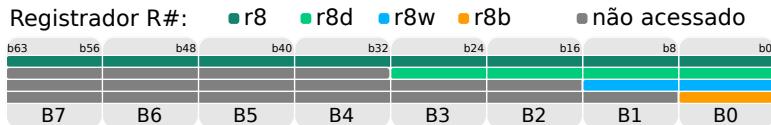
■ rax
0x7766554433221101 ou 8.603.657.889.541.918.977_d

Debugger

```
bellorini@SS-Note-Mint-EAB: ~/Unioeste/2021/LM/Etapa 01/Aula 03/codes
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Starting program: /home/bellorini/Unioeste/2021/LM/Etapa 01/Aula 03/codes/a03e01.x

Breakpoint 1, 0x000000000401008 in fim ()
(gdb) p/x $al
$1 = 0x1
(gdb) p/c $al
$2 = 1 '\001'
(gdb) p/x $ah
$3 = 0x11
(gdb) p/d $ah
$4 = 17
(gdb) p/x $ax
$5 = 0x1101
(gdb) p/d $ax
$6 = 4353
(gdb) p/x $eax
$7 = 0x33221101
(gdb) p/d $eax
$8 = 857870593
(gdb) Quit
(gdb) p/x $rax
$9 = 0x7766554433221101
(gdb) p/d $rax
$10 = 8603657889541918977
(gdb) █
```

RExemplo de Registrador: R8



- O acesso ao registrador pode ser realizado de diversas formas
 - r8: acessa 64 bits
 - r8d: 32 bits menos significativos de R8 (0 até 31)
 - r8w: 16 bits menos significativos de R8 (0 até 15)
 - r8b: 8 bits menos significativos de R8 (0 até 7)
use r8l quando for debuggar no gdb
- Esta forma de acesso também vale para R9 até R15

Exemplo para registradores de 64bits R8 até R15

■ Programa de 64 bits

```
6 section .data
7     num: dq 0x7766554433221101
8
9 section .text
10    global _start
11
12    _start:
13        mov r8, [num]
14        mov r9d, [num]
15
16    fim:
17        mov rax, 60
18        mov rdi, 0
19        syscall
```

code: a03e02.asm

Debugger

```
bellorini@SS-Note-Mint-EAB: ~/Unioeste/2021/LM/Etapa 01/Aula 03/codes
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Starting program: /home/bellorini/Unioeste/2021/LM/Etapa 01/Aula 03/codes/a03e02.x

Breakpoint 1, 0x0000000000401010 in fim ()
(gdb) p /x $r8
$1 = 0x7766554433221101
(gdb) p /d $r8
$2 = 8603657889541918977
(gdb) p /x $r8d
$3 = 0x33221101
(gdb) p /d $r8d
$4 = 857870593
(gdb) p /x $r9
$5 = 0x33221101
(gdb) p /d $r9
$6 = 857870593
(gdb) p /x $r9d
$7 = 0x33221101
(gdb) p /d $r9d
$8 = 857870593
(gdb) █
```

Registradores de segmentos

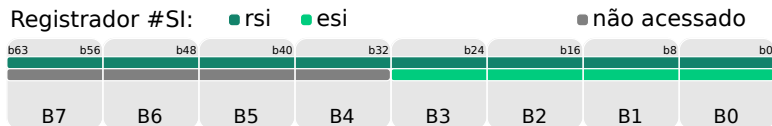
■ RSI e RDI

- Registradores de fonte e destino para algumas instruções
- A princípio, podem ser usados como propósito geral para instruções básicas

■ RBP e RSP

- São registradores especiais (ponteiros) para estrutura do programa
- RBP pode ser usado para propósito geral, porém é altamente não recomendado
- RSP pode ser usado para propósito geral, porém normalmente é fatal para a aplicação

Exemplo para registradores de segmento: RSI



- O acesso ao registrador pode ser realizado de duas forms
 - RSI: acessa 64 bits
 - ESI: 32 bits menos significativos (0 até 31)
- Vale também para RDI, RBP e RSP
 - 8 bits: SIL, DIL, BPL e SPL
 - 16 bits: SI, DI, BP e SP
- A função destes registradores será melhor estudada em aulas futuras

Instrução de movimentação de dados: MOV

- MOV: movimento (cópia) de dados da fonte para destino

`MOV destino, fonte`

- Para os exemplos, considere:

```
6  section .data
7      v1: dq 0x1111111122334455
8      v2: dq 0x0000000000000000
9      v3: dq 0x0000000000000000
10
```

code: a03e03.asm (parcial)

Sintaxe MOV - pt1

■ Sintaxe

- Cópia de dados da memória para registrador

```
MOV reg8 , r/m8  
MOV reg16, r/m16  
MOV reg32, r/m32  
MOV reg64, r/m64
```

- Exemplo

```
MOV al , [v1] ; 8 bits de conteudo de v1 para al  
MOV ebx, [v1] ; 32 bits de conteudo de v1 para EBX  
MOV rcx, [v1] ; 64 bits de conteudo de v1 para RCX
```

Sintaxe MOV - pt2

■ Sintaxe

- Cópia de dados de registrador para memória

```
MOV r/m8 , reg8  
MOV r/m16, reg16  
MOV r/m32, reg32  
MOV r/m64, reg64
```

- Exemplo

```
MOV [v2], al ; 8 bits de al para conteudo de v2  
MOV [v2], ebx ; 32 bits de EBX para conteudo de v2  
MOV [v2], rcx ; 64 bits de RCX para conteudo de v2
```

Sintaxe MOV - pt3

■ Sintaxe

- Cópia de dados de imediato para registrador

```
MOV reg8 , imm8  
MOV reg16, imm16  
MOV reg32, imm32  
MOV reg64, imm64
```

- Exemplo

```
MOV al , 0x10 ; 8 bits para al  
MOV ebx, 0x20202020 ; 32 bits para EBX  
MOV rcx, 0x3030303030303030 ; 64 bits para RCX
```

Sintaxe MOV - pt4

■ Sintaxe

- Cópia de dados de imediato para memória

`MOV r/m8 , imm8`

`MOV r/m16, imm16`

`MOV r/m32, imm32`

- Exemplo

`MOV byte [v3], 0x10`

; 8 bits para conteudo de v3

`MOV word [v3], 0x1515`

; 16 bits para conteudo de v3

`MOV dword [v3], 0x20202020`

; 32 bits para conteudo de v3

- Identificador de tamanho de palavra é requerido
- NASM não suporta movimentação de imediato de 64 bits para memória

MOV vs LEA

■ Aula 01 - hello.asm

- na linha 16 foi usado a instrução `lea rsi, [strOla]` .
- e depois alterado para `mov rsi, strOla`
Qual foi o resultado?
- uso de `[strOla]` indica conteúdo `strOla`
- uso de `strOla` indica endereço de `strOla`

■ Instrução LEA - Load Effective Address

- Realiza cálculos de endereçamento com base em um operando, sem movimentar efetivamente o operando
- Utilizada para encontrar endereços de operandos em memória antes de buscá-los na memória
 - Alguns operandos, como vetores, matrizes e estruturas, requerem algum tipo de cálculo antes de seu acesso
- Muito utilizada pelo compilador para otimizar códigos (tópico avançado)

Introdução à instrução LEA - exemplo

```
6  section .data
7      ; int vetorInt[10] = {42, 1, 2, 3, 4, 5, 6, 7, 8, 9};
8      vetorInt : dd 42, 1, 2, 3, 4, 96, 6, 7, 8, 9
9
10 section .text
11     global _start
12
13     _start:
14         ; ponteiro para o vetorInt
15         ; *vetorInt
16         ; cuidado: x86_64 contem endereços de 8 bytes
17         lea r8, [vetorInt]
```

code: a03e04.asm (parcial)

Atividades

- Escreva um código funcional (montável e linkável) que realize todas as 15 formas de movimentação de dados.
 - Considere a seguinte seção para seu código

```
1  section .data
2      pt1r8  : db 0x10                ; parte 1
3      pt1r16 : dw 0x2020
4      pt1r32 : dd 0x30303030
5      pt1r64 : dq 0x4040404040404040
6
7      pt2m8   : db 0x00                ; parte 2
8      pt2m16  : dw 0x0000
9      pt2m32  : dd 0x00000000
10     pt2m64  : dq 0x0000000000000000
11
12     ; parte 3 nao contem variaveis
13
14     pt4m8    : db 0x00                ; parte 4
15     pt4m16   : dw 0x0000
16     pt4m32   : dd 0x00000000
```


Atividades

- É necessário debuggar para confirmar os valores em memória e registradores
- Use breakpoints
 - `p /d $reg` para conteúdo de registrador em decimal
 - `p /x $reg` para conteúdo de registrador em hexadecimal
 - `x /d &end` para conteúdo de memória em decimal
 - `x /[bhwg]d &end` para alterar o tamanho do dado
- Uso de LEA é opcional

Fim do Documento

Dúvidas?

Próxima aula:

- Aula 04: Variáveis Não-Inicializadas e Estrutura de Programas
 - Nova Seção para declaração de espaços de memória
 - Novas palavras mágicas de reserva de memória