

Linguagem de Montagem

Instruções Básicas Aritméticas e Lógicas Aula 05

Edmar André Bellorini

Ano letivo 2016

Introdução

- ▶ Aula01: Montar, linkar e executar

```
$: nasm -f elf32|elf64 nome.asm
```

```
$: ld nome.o -o nome.x
```

```
$: ./nome.x
```

- ▶ Aula02: Estrutura dos programas e dados inicializados

```
section .data
```

- ▶ Aula03: Registradores e instrução de movimentação

```
MOV reg32, r/m32
```

- ▶ Aula04: Dados não inicializados e estrutura de programas

```
section .bss
```

Instruções Aritméticas e Lógicas

- ▶ Aritméticas
 - ▶ Soma
 - ▶ ADD
 - ▶ ADC
 - ▶ Subtração
 - ▶ SUB
 - ▶ SBB
- ▶ Lógicas
 - ▶ OR
 - ▶ XOR
 - ▶ NOT
 - ▶ AND

Soma

► Instrução ADD

- Acumula em Destino o valor de Fonte

`ADD Destino, Fonte`

Destino = Destino + Fonte

► Sintaxe

`ADD reg, r/m ; 8, 16, 32 e 64 bits`

`ADD mem, reg ; 8, 16, 32 e 64 bits`

`ADD r/m, imm ; 8, 16, 32 e 64 bits`

Subtração

- ▶ Instrução SUB

- ▶ Reduz o Fonte do Destino

`SUB Destino, Fonte`

Destino = Destino - Fonte

- ▶ Sintaxe

`SUB reg, r/m ; 8, 16, 32 e 64 bits`

`SUB mem, reg ; 8, 16, 32 e 64 bits`

`SUB r/m, imm ; 8, 16, 32 e 64 bits`

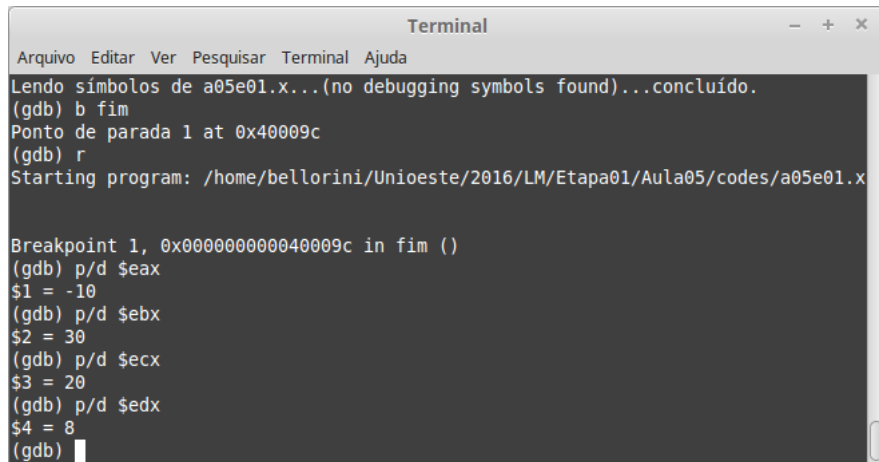
- ▶ Observação

- ▶ Tanto ADD quanto SUB são operações sinalizadas
Geram sinais de overflow (*flag OF*)
Registrador EFLAGS ...

Exemplo a05e01.asm

```
1  section .text
2      global _start
3
4  _start:
5
6      mov eax, 10
7      mov ebx, 20
8      mov ecx, 30
9      mov edx, 0xfffffffffe
10
11     add ebx, eax
12     add edx, eax
13
14     sub ecx, eax
15     sub eax, ecx
16
17     fim:
18     mov eax, 1
19     mov ebx, 0
20     int 0x80
```

Debugger



```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Lendo símbolos de a05e01.x...(no debugging symbols found)...concluído.
(gdb) b fim
Ponto de parada 1 at 0x40009c
(gdb) r
Starting program: /home/bellorini/Unioeste/2016/LM/Etapa01/Aula05/codes/a05e01.x

Breakpoint 1, 0x000000000040009c in fim ()
(gdb) p/d $eax
$1 = -10
(gdb) p/d $ebx
$2 = 30
(gdb) p/d $ecx
$3 = 20
(gdb) p/d $edx
$4 = 8
(gdb) 
```

Registrador FLAGS

- ▶ É um registrador especial de estado do processador
- ▶ É alterado indiretamente por instruções aritméticas e lógicas
- ▶ FLAG
 - ▶ Registrador de estado de 16 bits
 - ▶ Somente para máquinas de 16 bits (antigas)
- ▶ EFLAG
 - ▶ Registrador de estado de 32 bits
 - ▶ Mantém retrocompatibilidade

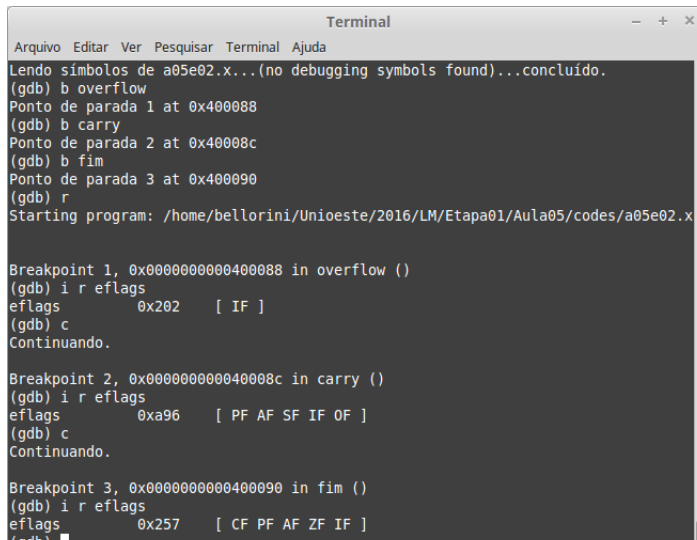
Duas Flags de interesse

- ▶ Overflow
 - ▶ $OF = 1 \rightarrow$ operação aritmética gerou overflow
 - ▶ Importante para operações sinalizadas
- ▶ Carry
 - ▶ $CF = 1 \rightarrow$ operação aritmética gerou bit *carry*
 - ▶ Importante para operações não sinalizadas
- ▶ GDB
 - (gdb) info register eflags
 - ou
 - (gdb) i r eflags
 - ▶ Será mostrado as flags ativas (valor = '1')

Exemplo a05e02.asm

```
1  section .text
2      global _start
3
4  _start:
5      mov ax, 0x7fff ; 32767
6      mov bx, 0xffff ; 65535
7
8  overflow:
9      add ax, 1 ; -32768
10
11      carry:
12          add bx, 1 ; 0
13
14      fim:
15          mov rax, 1
16          mov rbx, 0
17          int 80h
```

Debugger



```
Terminal
Arquivo Editar Ver Pesquisar Terminal Ajuda
Lendo símbolos de a05e02.x...(no debugging symbols found)...concluído.
(gdb) b overflow
Ponto de parada 1 at 0x400088
(gdb) b carry
Ponto de parada 2 at 0x40008c
(gdb) b fim
Ponto de parada 3 at 0x400090
(gdb) r
Starting program: /home/bellorini/Unioeste/2016/LM/Etapa01/Aula05/codes/a05e02.x

Breakpoint 1, 0x000000000400088 in overflow ()
(gdb) i r eflags
eflags      0x202    [ IF ]
(gdb) c
Continuando.

Breakpoint 2, 0x00000000040008c in carry ()
(gdb) i r eflags
eflags      0xa96    [ PF AF SF IF OF ]
(gdb) c
Continuando.

Breakpoint 3, 0x000000000400090 in fim ()
(gdb) i r eflags
eflags      0x257    [ CF PF AF ZF IF ]
(gdb)
```

Soma com Carry-In

► Instrução ADC

- Acumula em Destino o valor de Fonte e o valor Carry
- Valor Carry é o bit carry-out da última operação aritmética

ADC Destino, Fonte

$$\textit{Destino} = (\textit{Destino} + \textit{Fonte}) + CF$$

► Sintaxe

ADC reg, r/m ; 8, 16, 32 e 64 bits

ADC mem, reg ; 8, 16, 32 e 64 bits

ADC r/m, imm ; 8, 16, 32 e 64 bits

Subtração com Borrow-In

► Instrução SBB

- Reduz Fonte e Borrow de Destino
- Valor Borrow é o bit carry-out da última operação aritmética

SBB Destino, Fonte

$$\text{Destino} = (\text{Destino} - \text{Fonte}) - CF$$

► Sintaxe

SBB reg, r/m ; 8, 16, 32 e 64 bits

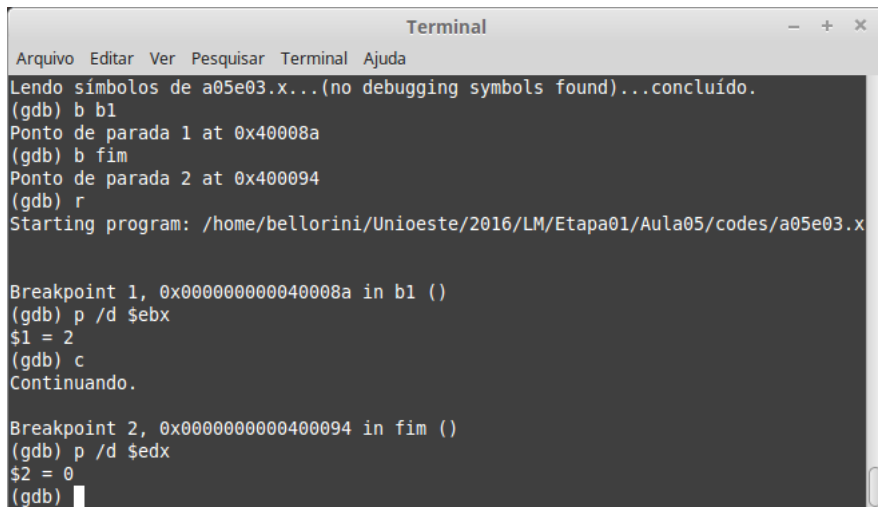
SBB mem, reg ; 8, 16, 32 e 64 bits

SBB r/m, imm ; 8, 16, 32 e 64 bits

Exemplo a05e03.asm

```
1  section .text
2      global _start
3
4  _start:
5
6  c1:
7      mov eax, -1
8      add eax, 2
9  c2:
10     adc ebx, eax
11     b1:
12         mov ecx, 2
13         sub ecx, 3
14     b2:
15         sbb edx, ecx
16
17     fim:
18         mov rax, 1
19         mov rbx, 0
20         int 0x80
```

Debugger



The image shows a terminal window titled "Terminal" with a menu bar containing "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The terminal output shows a GDB session for a program named "a05e03.x". The user sets two breakpoints: one at address 0x40008a in function "b1" and another at address 0x400094 in function "fim". The first breakpoint is hit, and the user inspects the value of register \$ebx, which is 2. The user then continues execution. The second breakpoint is hit, and the user inspects the value of register \$edx, which is 0. The session ends with the user entering a blank line in the GDB prompt.

```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Lendo símbolos de a05e03.x...(no debugging symbols found)...concluído.
(gdb) b b1
Ponto de parada 1 at 0x40008a
(gdb) b fim
Ponto de parada 2 at 0x400094
(gdb) r
Starting program: /home/bellorini/Unioeste/2016/LM/Etapa01/Aula05/codes/a05e03.x

Breakpoint 1, 0x000000000040008a in b1 ()
(gdb) p /d $ebx
$1 = 2
(gdb) c
Continuando.

Breakpoint 2, 0x0000000000400094 in fim ()
(gdb) p /d $edx
$2 = 0
(gdb) 
```

OR, XOR, AND

- ▶ Instruções

- ▶ OR

- ▶ Operação lógica OR bit-a-bit

- `OR reg , r/m/i`

- `OR mem , r/i`

- ▶ XOR

- ▶ Operação lógica XOR bit-a-bit

- `XOR reg , r/m/i`

- `XOR mem , r/i`

- ▶ AND

- ▶ Operação lógica AND bit-a-bit

- `AND reg , r/m/i`

- `AND mem , r/i`

- ▶ NOT

- ▶ Operação lógica NOT bit-a-bit (**não** carrega EFLAGS)

- `NOT r/m`

Exemplo - a05e04.asm

```
10      ...
11      section .text
12          global _start
13
14      _start:
15          mov ax, [v1]
16          mov bx, [v2]
17          mov cx, [v2]
18
19          ; or reg x reg
20          or cx, ax
21          ...
```

```
44      ...
45      xorlabel:
46          ; xor
47          mov ax, [v1]
48          mov bx, [v2]
49          mov cx, [v2]
50
51          ; or reg x reg
52          xor cx, ax
53
54          ; or mem x reg
55          mov [v1xorv2], ax
56          xor [v1xorv2], bx
57          ...
```

Exercícios de Fixação

- ▶ EF0501: Complete o código do arquivo ef0501.asm em anexo
 - ▶ Simulação de computação ADD/SUB de 64 bits utilizando somente registradores de 32 bits
 - ▶ Soma
 - ▶ Linha 19:
$$; i3 = i1 + i2$$
 - ▶ Subtração
 - ▶ Linha 21:
$$; i4 = i2 - i3$$
 - ▶ Recomendação
 - ▶ ebx:eax = i1 (bytes mais significativos em ebx)
 - ▶ edx:ecx = i2
 - ▶ Qual é a faixa de representação em complemento de dois para 32 bits e 64 bits?

Exercício EF0501 - Continuação

- ▶ Os registradores R14 e R15 contém os endereços das variáveis i3 e i4
 - ▶ No debugger GDB use

```
(gdb) b fim
(gdb) r
(gdb) p /x $r14
```

 - ▶ Será retornado um valor hexadecimal
É o endereço da variável i3 (ex.: 0x600110)
 - ▶ Para imprimir o valor desta posição de memória faça

```
(gdb) x /x 0x600110
```

Será impresso os primeiros 4 bytes
 - ▶ Para imprimir os próximos 4 bytes

```
(gdb) x /x 0x600114
```
 - ▶ Resultados devem ser gravados em i3 e i4
 - ▶ Confira no gdb

Exercícios de Fixação

- ▶ EF0502 - Escreva um código funcional (montável, *linkável* e executável) que:
 - ▶ Contenha 2 variáveis inicializadas:
`maiuscula: db 'A'`
`minusculta: db 'b'`
 - ▶ Contenha 2 variáveis não inicializadas:
`lowercase: resb 1`
`uppercase: resb 1`
 - ▶ Usando apenas as instruções ADD e/ou SUB converta:
 - ▶ *maiuscula* para minusculta e grave em *lowercase*
 - ▶ *minusculta* para maiúscula e grave em *uppercase*
 - ▶ A tabela ASCII é sua amiga e extremamente necessária para completude deste exercício

Exercícios de Fixação

- ▶ EF0503 - O exercício EF0502 contém um *bug*
 - ▶ O que acontece se os caracteres inicializados forem colocados de maneira incorreta? Por exemplo:
`maiuscula: db 'a'`
`minuscule: db 'B'`
 - ▶ Usando somente as operações lógicas OR, XOR e/ou AND, reescreva o código do exercício EF0502 de modo a eliminar o *bug*

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: P05