Linguagem de Montagem

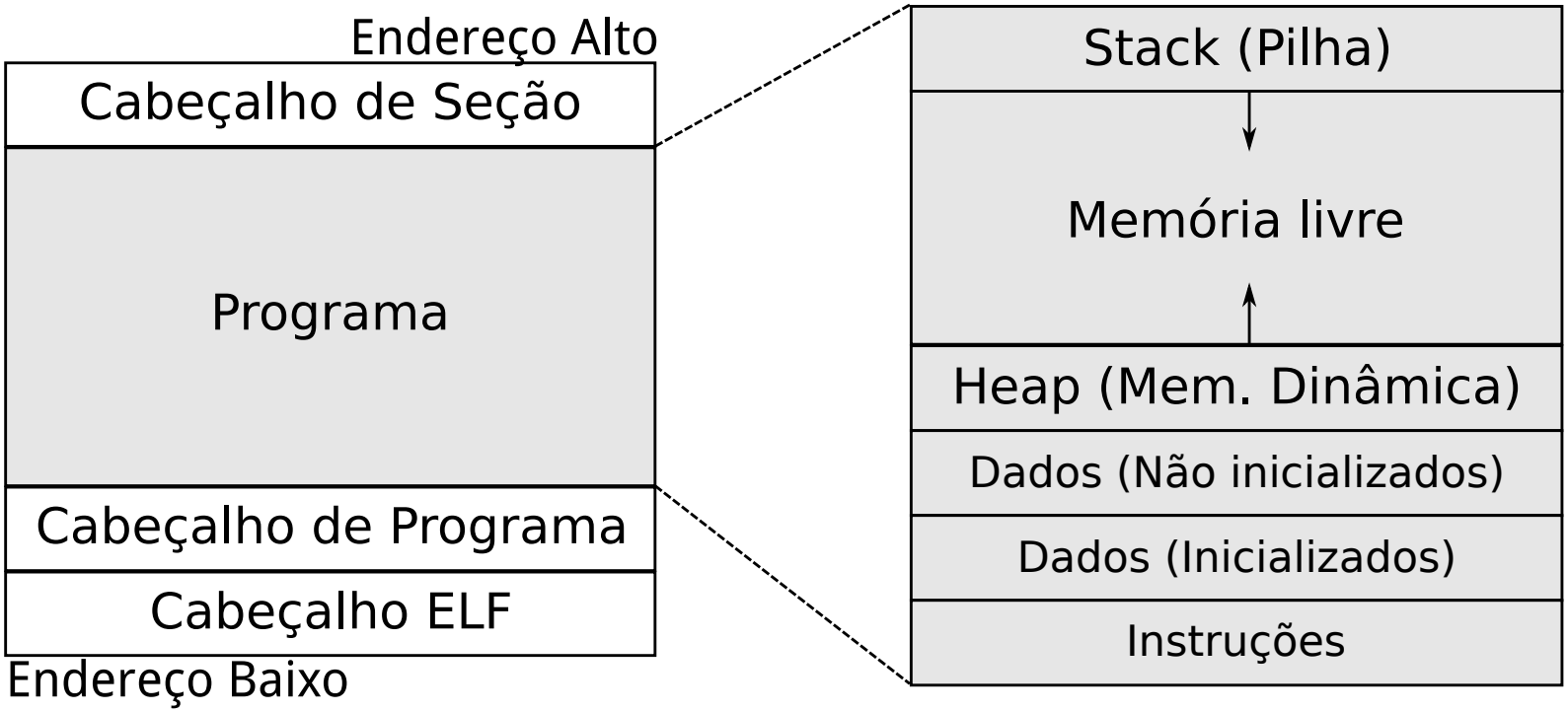
Variáveis Não Inicializadas
e
Estrutura de Programas
Aula 04

Edmar André Bellorini

Ano letivo 2021

Introdução

- Estrutura de um programa `ELF` (Aula 02)



Dados Inicializados (.data)

- Variáveis com valores pré-definidos

```
section .data
  simbolo: tamanho valor
```

- Especificação de tamanho por pseudo-instruções

```
db: byte
dw: word (2B)
dd: dualword (4B)
dq: quadword (8B)
```

Dados Não Inicializados (.bss)

- Variáveis **sem** valores pré-definidos

```
section .bss
  simbolo: tamanho quantidade
```

  - Uma linha é uma variável não inicializada
  - Uma variável é definida por um símbolo (*label*)

```
char c;
```

    - c é o nome da variável, ou seja, o símbolo para referência
  - Uma variável tem um tamanho em memória reservado por uma pseudo-instrução

```
resb: byte
resw: word (2B)
resd: dualword (4B)
resq: quadword (8B)
```

Dados Não Inicializados - Exemplos

```
n1: resb 1   ; 1 byte
n2: resb 2   ; 2 bytes
n3: resb 3   ; 3 bytes
n4: resb 8   ; 8 bytes
n5: resw 1   ; 2 bytes
n6: resw 2   ; 4 bytes
n8: resw 4   ; 8 bytes
n9: resd 1   ; 4 bytes
na: resd 2   ; 8 bytes
```

Instrução de movimentação de dados

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

  ```
  MOV destino, fonte
  ```

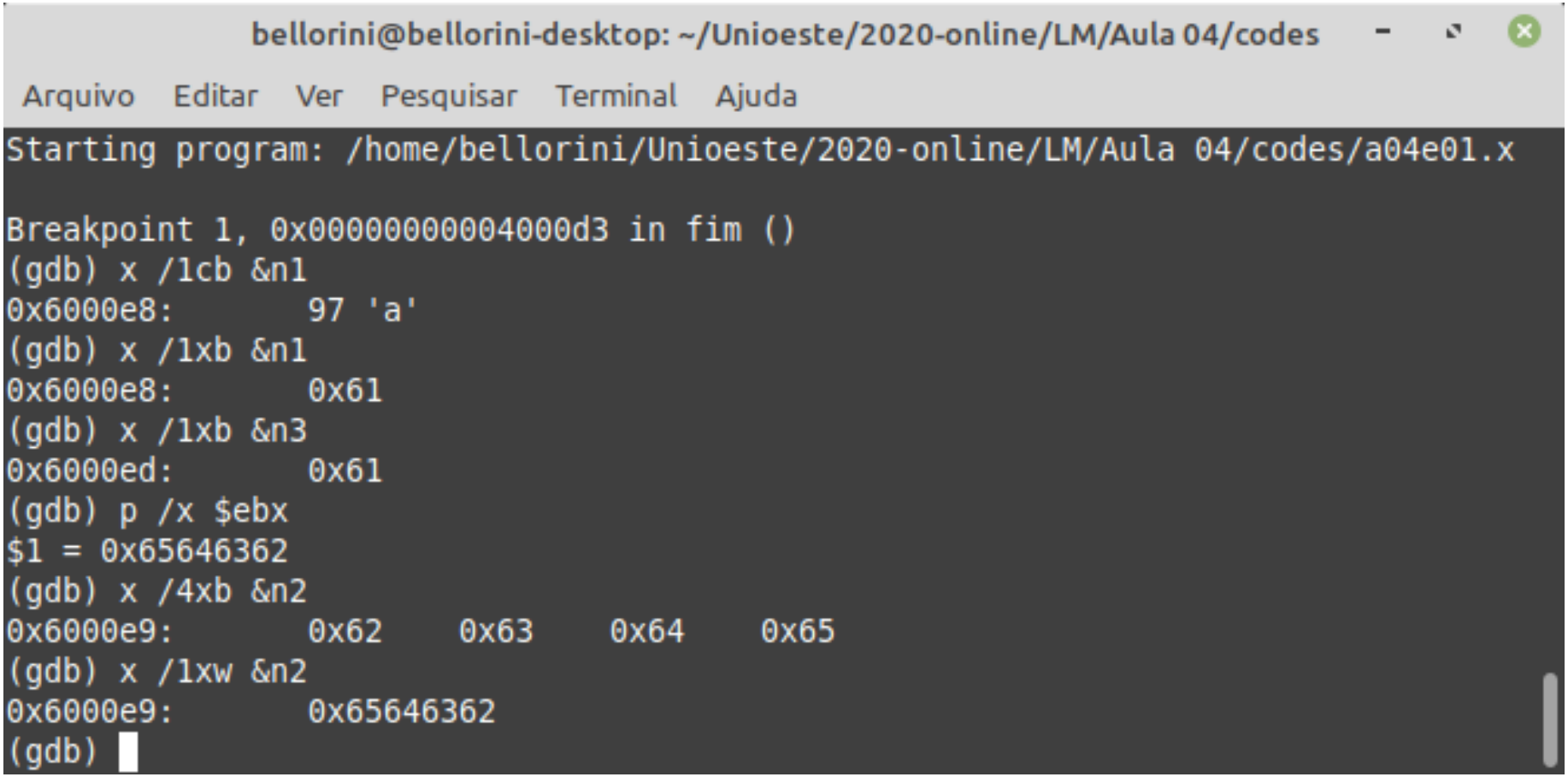
  - Exatamente a mesma instrução para movimentação de dados inicializados

Exemplo a04e01.asm

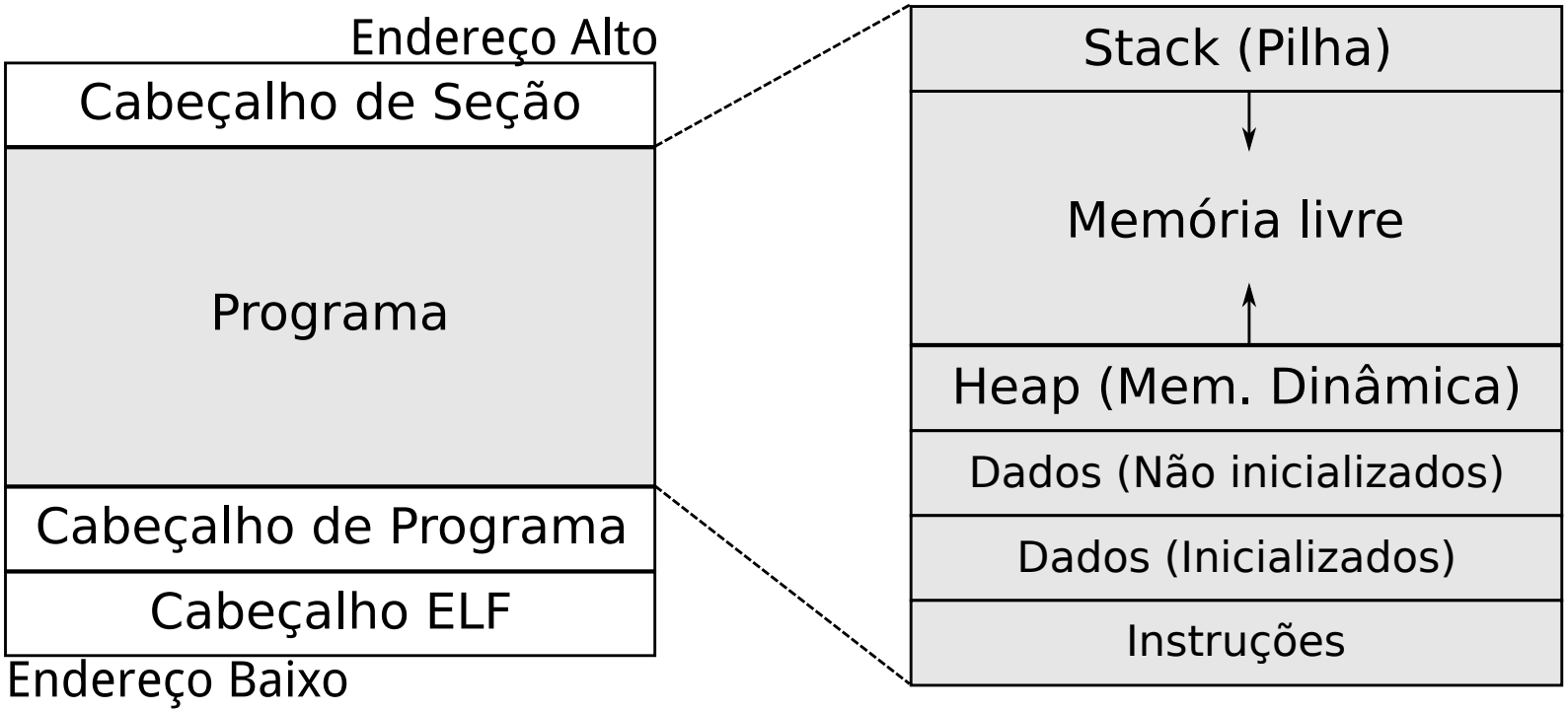
```
6  section .data
7      v1: db 0x61
8      v2: dd 0x65646362
9
10 section .bss
11     n1: resb 1
12     n2: resd 1
13     n3: resb 1
```

```
15 _start:
16     mov al   , [v1]
17     mov [n1] , al
18     mov [n3] , al
19     mov ebx  , [v2]
20     mov [n2] , ebx
21
22 fim:
23     mov rax, 60
24     mov rdi, 0
25     syscall
```

Debugger


bellorini@bellorini-desktop: ~/Unioeste/2020-online/LM/Aula 04/codes
Arquivo Editar Ver Pesquisar Terminal Ajuda
Starting program: /home/bellorini/Unioeste/2020-online/LM/Aula 04/codes/a04e01.x
Breakpoint 1, 0x00000000004000d3 in fim ()
(gdb) x /1cb &n1
0x6000e8: 97 'a'
(gdb) x /1xb &n1
0x6000e8: 0x61
(gdb) x /1xb &n3
0x6000ed: 0x61
(gdb) p /x $ebx
$1 = 0x65646362
(gdb) x /4xb &n2
0x6000e9: 0x62 0x63 0x64 0x65
(gdb) x /1xw &n2
0x6000e9: 0x65646362
(gdb)

Esta imagem de novo?



- Endereço das variáveis
  - Dados Inicializados é mais "baixo"
  - Dados Não Inicializados é mais "alto"

Endereços das Variáveis

- Cada símbolo para uma variável é uma referência

```
section. data
    v1: db 0x61
```

  - v1 é uma referência para o conteúdo 0x61
- Cada referência é um endereço de memória
  - v1 é na verdade um endereço de memória
  - É válido tanto para .data quanto para .bss
- Como descobrir os endereços em tempo de execução?
  - Instrução LEA

```
LEA rax, [v1]
```

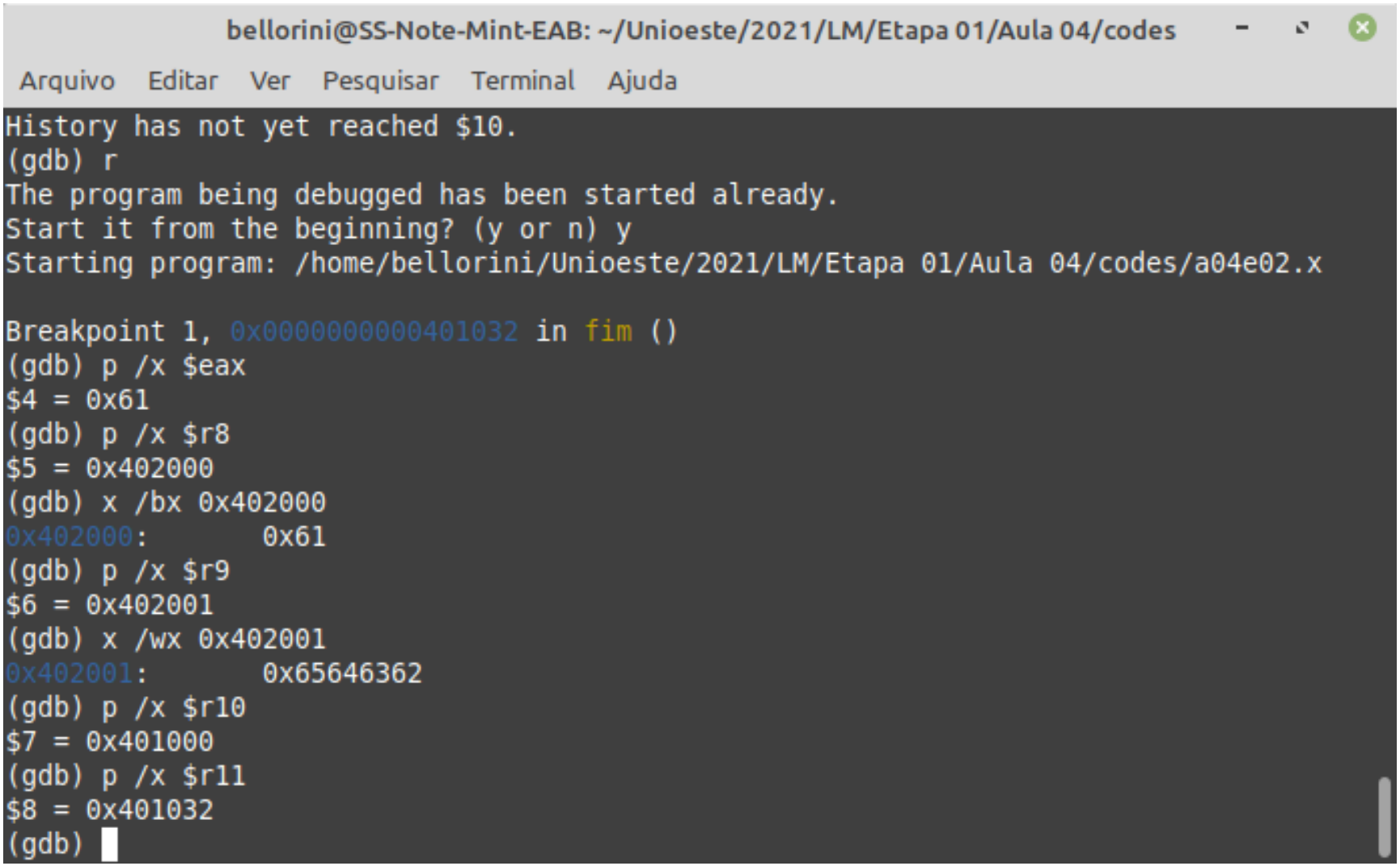
    - Registrador RAX contém o endereço do primeiro byte de v1

Exemplo a04e02.asm

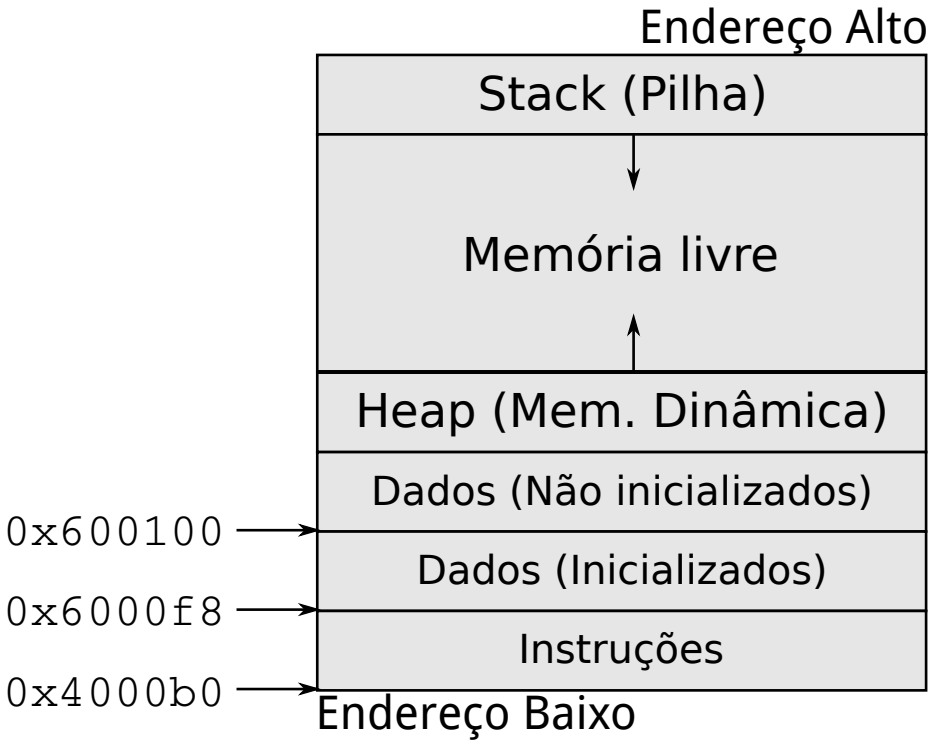
```
6  section .data
7      v1: db 0x61
8      v2: dd 0x65646362
9
10 section .bss
11     n1: resb 1
12     n2: resd 1
13     n3: resb 1
14
15 section .text
16     global _start
```

```
18 _start:
19     mov al, [v1]
20     lea r8, [v1]
21
22     lea r9, [v2]
23     mov rbx, [v2]
24     mov rcx, [r9]
25
26     lea r10, [_start] ; ?
27     lea r11, [fim]
28
29 fim:
30     mov rax, 60
31     mov rdi, 0
32     syscall
```

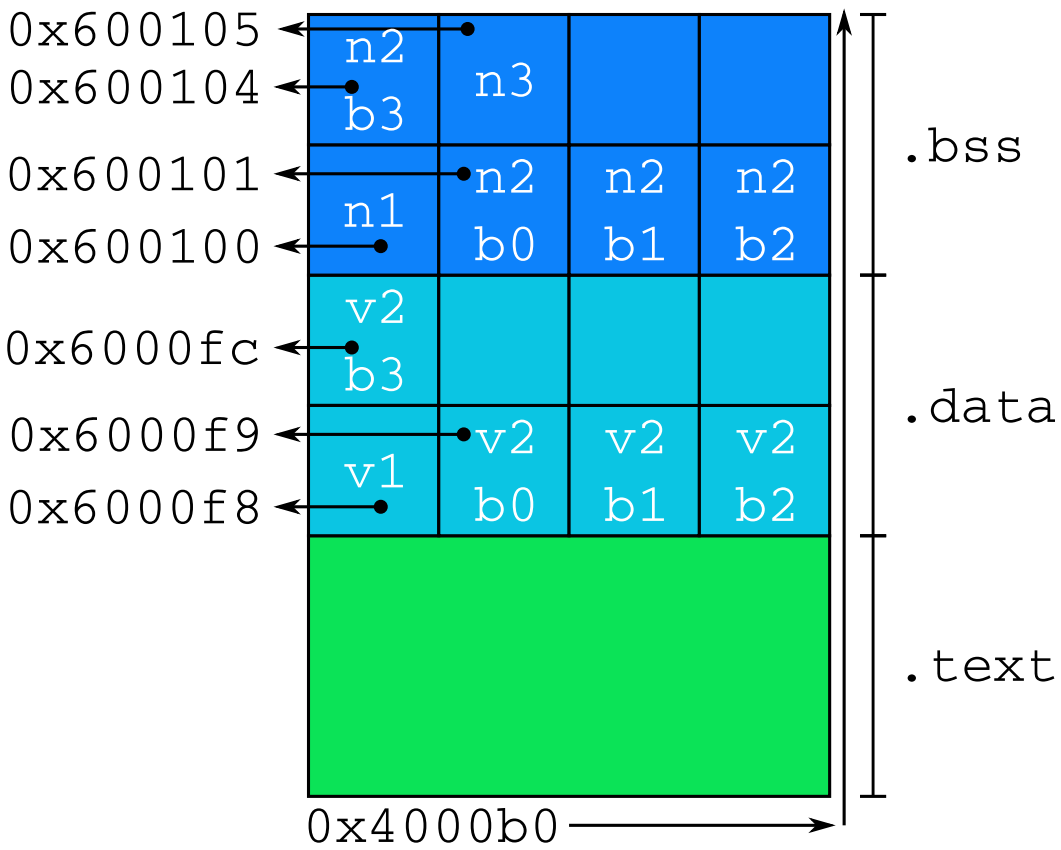
Debugger


bellorini@SS-Note-Mint-EAB: ~/Unioeste/2021/LM/Etapa 01/Aula 04/codes
Arquivo Editar Ver Pesquisar Terminal Ajuda
History has not yet reached $10.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/bellorini/Unioeste/2021/LM/Etapa 01/Aula 04/codes/a04e02.x
Breakpoint 1, 0x0000000000401032 in fim ()
(gdb) p /x $eax
$4 = 0x61
(gdb) p /x $r8
$5 = 0x402000
(gdb) x /bx 0x402000
0x402000: 0x61
(gdb) p /x $r9
$6 = 0x402001
(gdb) x /wx 0x402001
0x402001: 0x65646362
(gdb) p /x $r10
$7 = 0x401000
(gdb) p /x $r11
$8 = 0x401032
(gdb)

Estrutura do exemplo (Visão Abstrata)



Estrutura do exemplo (Visão Detalhada)



- Obs.: Os valores podem ser diferentes de acordo com a máquina/montador para um mesmo código

Comportamento

- Aula 02
  - Não existe "tipo de dados" em memória
  - É a instrução que determina o comportamento
- É possível indicar um possível tipo de dado
  - Já fazemos isso quando definimos uma variável inicializada

```
section .data
    v1: db 0x61
```

    - v1 é uma variável de 1 byte
    - o valor 0x61 é armazenado como um número hexadecimal
    - é possível usar outras bases numéricas

Números

- Bases numéricas aceitas
  - Hexadecimal

```
v1a: dq 0x61 ; 0x ou h
v1b: dq 61h
```

  - Decimal

```
v2a: dq 97d ; d ou sem definicao
v2b: dq 97  ; nasm assume base decimal
```

  - Octal

```
v3: dq 141o
```

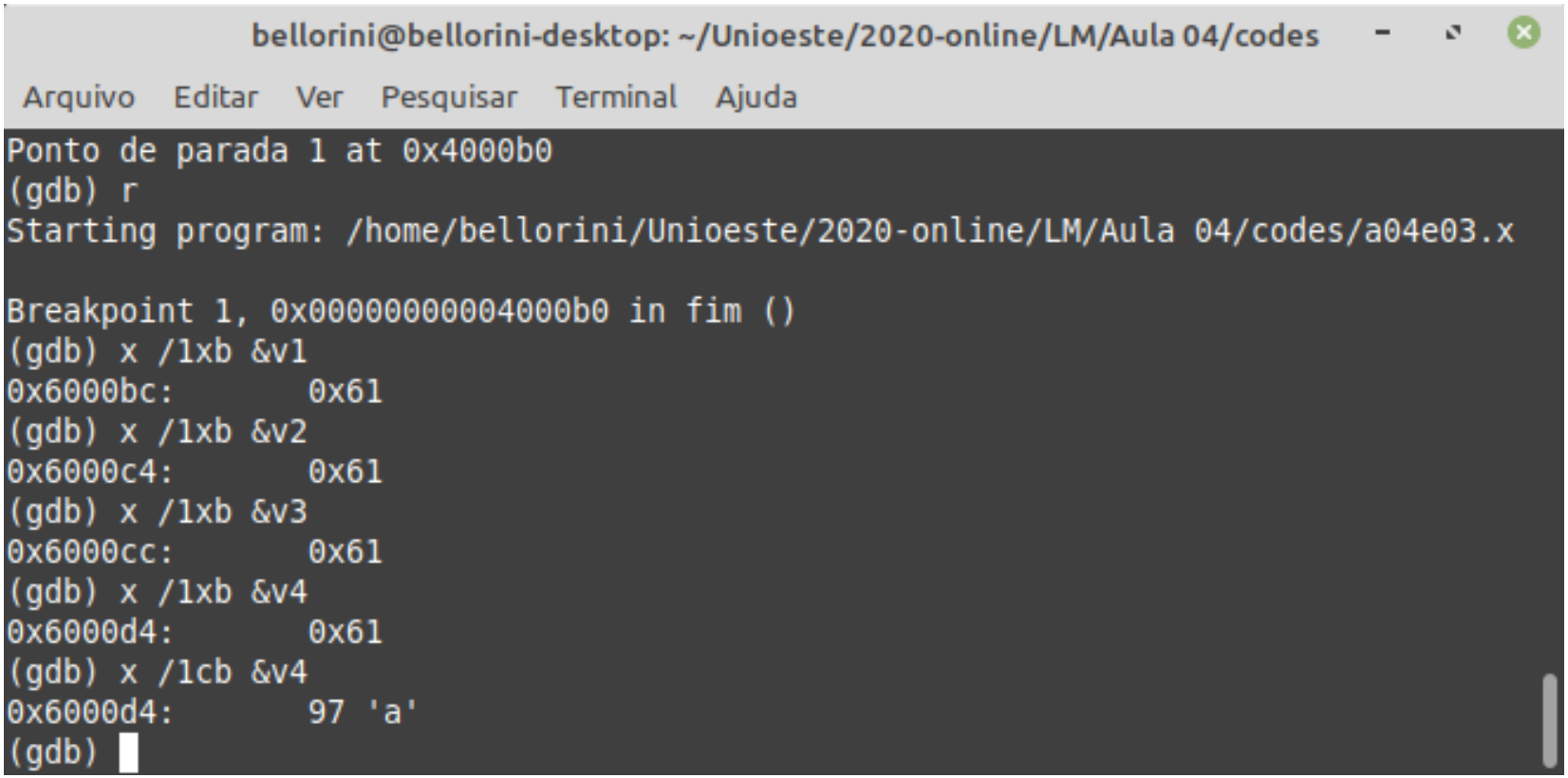
  - Binário

```
v4: dq 1100001b
```

Exemplo a04e03.asm

```
6  section .data
7     v1: dq 0x61
8     v2: dq 97d
9     v3: dq 141o
10    v4: dq 1100001b
11
12 section .text
13    global _start
14
15 _start:
16       ; gdb x /[bhwg][cduxto] &<nomeVar>
17 fim:
18    mov rax, 60
19    mov rdi, 0
20    syscall
```

Debugger



- v1 = v2 = v3 = v4
- 0x00000061 = 'a'

Números negativos

- Números são representados sempre em complemento de dois
  - Exemplo a04e04.asm

```
6   section .data
7       v1: dq  100d
8       v2: dq -100d
9
10  section .text
11      global _start
12
13  _start:
14          ; gdb x /[bhwg][cduxto]
15  fim:
16      mov rax, 60
17      mov rdi, 0
18      syscall
```

Debugger


bellorini@bellorini-desktop: ~/Unioeste/2020-online/LM/Aula 04/codes
Arquivo Editar Ver Pesquisar Terminal Ajuda
Starting program: /home/bellorini/Unioeste/2020-online/LM/Aula 04/codes/a04e04.x
Breakpoint 1, 0x00000000004000b0 in fim ()
(gdb) x /1dw &v1
0x6000bc: 100
(gdb) x /1dw &v2
0x6000c4: -100
(gdb) x /1xw &v1
0x6000bc: 0x00000064
(gdb) x /1xw &v2
0x6000c4: 0xffffff9c
(gdb) x /1uw &v1
0x6000bc: 100
(gdb) x /1uw &v2
0x6000c4: 4294967196
(gdb)

Movimentação de dados numéricos COM extensão de sinal

- Movimentar (copiar) n bytes para p bytes
  - $n < p$
- Instrução movsx (*move sign-extends*)
  - Copia fonte para destino e mantém magnitude
  - Valores sinalizados (*signed*)
  - Sintaxe

```
MOVSX reg16,r/m8  ; 1b para 2b mantendo sinal
MOVSX reg32,r/m8  ; 1b para 4b
MOVSX reg64,r/m8  ; 1b para 4b
MOVSX reg32,r/m16 ; 2b para 4b
MOVSX reg64,r/m16 ; 2b para 4b
MOVSX reg64,r/m32 ; 4b para 8b
```

Movimentação de dados numéricos SEM extensão de sinal

- Movimentar (copiar) n bytes para p bytes
  - $n < p$
- Instrução movzx (*move zero-extends*)
  - Copia fonte para destino e estende zero
  - Valores não-sinalizados (*unsigned*)
  - Sintaxe

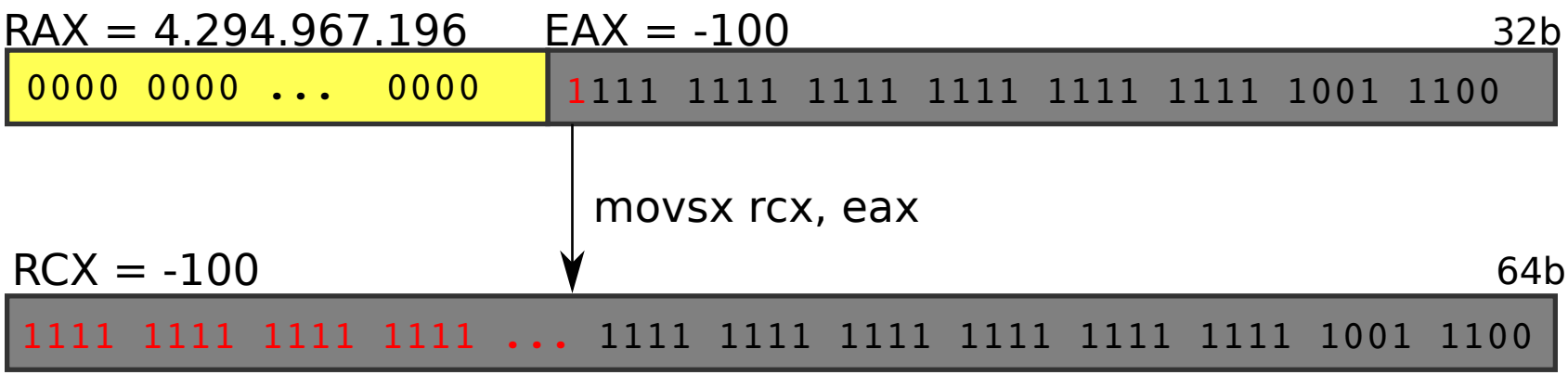
```
MOVZX reg16,r/m8  ; 1b para 2b
MOVZX reg32,r/m8  ; 1b para 4b
MOVZX reg64,r/m8  ; 1b para 8b
MOVZX reg32,r/m16 ; 2b para 4b
MOVZX reg64,r/m16 ; 2b para 8b
```

  - Observação:
    - Não existe extensão em zero para destino de 64bits

Exemplo a04e05.asm

```
6  section .text
7      global _start
8
9  _start:
10     mov    eax, -100
11     movsx rcx, eax
12
13 fim:
14     mov rax, 60
15     mov rdi, 0
16     syscall
```

Exemplo a04e05.asm



Debugger


bellorini@bellorini-desktop: ~/Unioeste/2020-online/LM/Aula 04/codes
Arquivo Editar Ver Pesquisar Terminal Ajuda
Ponto de parada 1 at 0x400088
(gdb) r
Starting program: /home/bellorini/Unioeste/2020-online/LM/Aula 04/codes/a04e05.x
Breakpoint 1, 0x0000000000400088 in fim ()
(gdb) p/d $eax
$1 = -100
(gdb) p/d $rax
$2 = 4294967196
(gdb) p/d $rcx
$3 = -100
(gdb) p/u $rax
$4 = 4294967196
(gdb) p/u $rcx
$5 = 18446744073709551516
(gdb)

Atividade - slide 1

- a04a01: considere o código incompleto em anexo a04a01.asm:

```
6  section .data
7      ; vetor largura, altura e profundidade
8      dimensoes : dw 50, 65, -75
9  section .bss
10     volume : resq 1
11
12 section .text
13     global _start
14 _start:
15     ; aluno deve:
16         ; mover largura para registrador r8?
17         ; mover altura para registrador r9?
18         ; mover profundidade para registrador r10?
19         ; ter cuidado com os tamanhos dos registradores
```

Atividade - slide 2

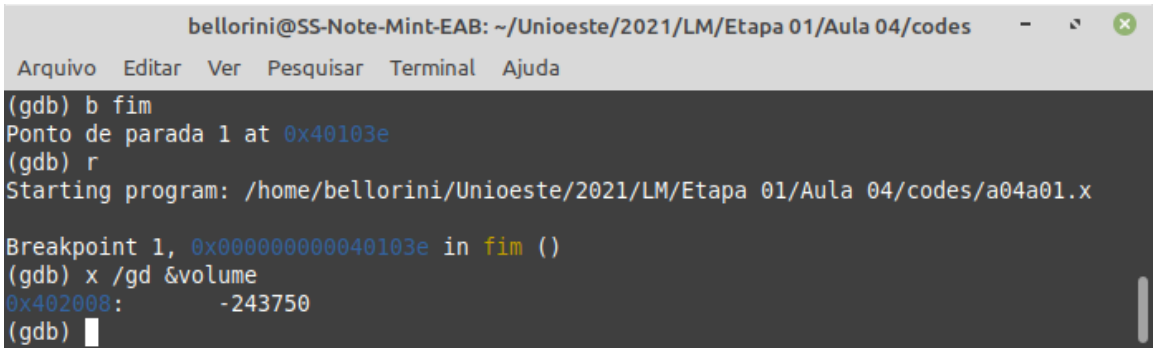
- O programa calcula o volume de um hexaedro:
  - $volume = largura * altura * profundidade$
    - para trabalhar com sinais, a profundidade é negativa.
  - O cálculo do volume está presente nas linhas 28 até 35.
    - O aluno, neste momento, não precisa se preocupar com isso.
  - O que o aluno deve realizar:
    - mover corretamente os valores do vetor dimensoes para os seguintes registradores:
      r8 $\leftarrow$ dimensoes[0]
      r9 $\leftarrow$ dimensoes[1]
      r10 $\leftarrow$ dimensoes[2]
      código deverá ser criado nas linhas 23, 24 e 25
    - armazenar o resultado do cálculo na variável `volume`
      volume $\leftarrow$ ecx
      código deve ser criado na linha 41

Atividade - slide 3

- Para verificar se as movimentações foram feitas corretamente:
  $volume = largura * altura * profundidade$
  $-243750 = 50 * 65 * -75$

  - Use o Debugger gdb

```
b fim
r
x /gd &volume
```



Fim do Documento

Dúvidas?

Próxima aula:

- Aula 05: Operações Aritmética e Lógicas