

Linguagem de Montagem

Estrutura dos Programas

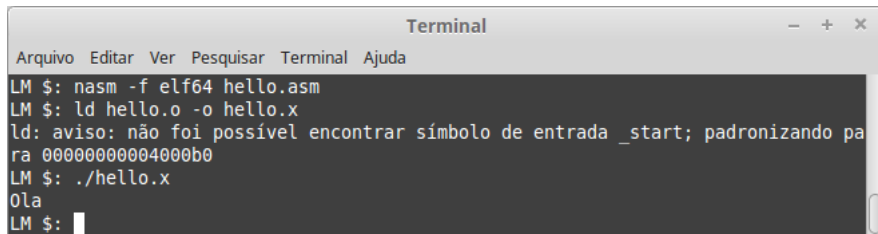
Aula 02

Edmar André Bellorini

Ano letivo 2016

Exercícios da Aula 01

- ▶ EF0102: Com o código EF0101 completo, efetue as seguintes alterações, de forma independente, gere o executável, execute e documente os resultados:
 - ▶ Comente a linha 6 (linha que contém global `_start`)
É possível comentar uma linha utilizando “;” (ponto-e-vírgula)
 - ▶ ...



A terminal window titled "Terminal" with a menu bar (Arquivo, Editar, Ver, Pesquisar, Terminal, Ajuda). The terminal shows the following commands and output:

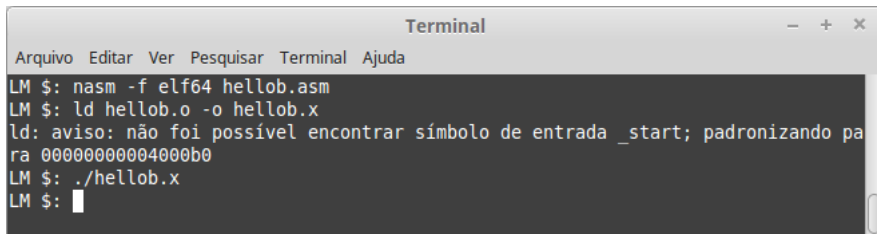
```
LM $: nasm -f elf64 hello.asm
LM $: ld hello.o -o hello.x
ld: aviso: não foi possível encontrar símbolo de entrada _start; padronizando pa
ra 00000000004000b0
LM $: ./hello.x
Ola
LM $: 
```

- ▶ O texto “Ola” ainda é apresentado como saída

Alterando um pouco mais ...

```
1  section .data
2      str0la :  db "0la", 10
3      str0laL:  equ $ - str0la
4
5  section .text
6      ;          global _start
7
8      mov eax, 1
9      mov ebx, 0
10     int 0x80
11
12     _start:
13     mov eax, 4
14     mov ebx, 1
15     mov ecx, str0la
16     mov edx, str0laL
17     int 0x80
18     ...
```

Alterando um pouco mais ...



```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
LM $: nasm -f elf64 hellob.asm
LM $: ld hellob.o -o hellob.x
ld: aviso: não foi possível encontrar símbolo de entrada _start; padronizando pa
ra 00000000004000b0
LM $: ./hellob.x
LM $: 
```

- ▶ O texto “Ola” **não** é apresentado como saída
 - ▶ O trecho de código adicionado finaliza o programa

```
8      mov  eax, 1
9      mov  ebx, 0                      =          return 0;
10     int  0x80
```

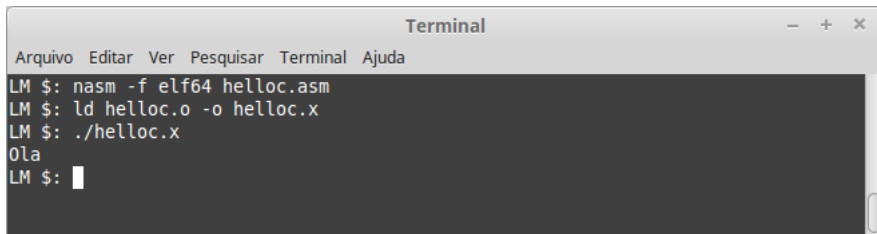
- ▶ Este teste indica que o código começou a ser executado pela instrução que sucede a seção .text

Alterando pela última vez ...

- ▶ Apenas descomentar a linha 6

```
1  section .data
2      str0la :  db "0la", 10
3      str0laL:  equ $ - str0la
4
5  section .text
6      global _start
7
8      mov eax, 1
9      mov ebx, 0
10     int 0x80
11
12     _start:
13         mov eax, 4
14         mov ebx, 1
15         mov ecx, str0la
16         mov edx, str0laL
17         int 0x80
18         ...
```

Alterando pela última vez ...



```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
LM $: nasm -f elf64 helloc.asm
LM $: ld helloc.o -o helloc.x
LM $: ./helloc.x
Ola
LM $: 
```

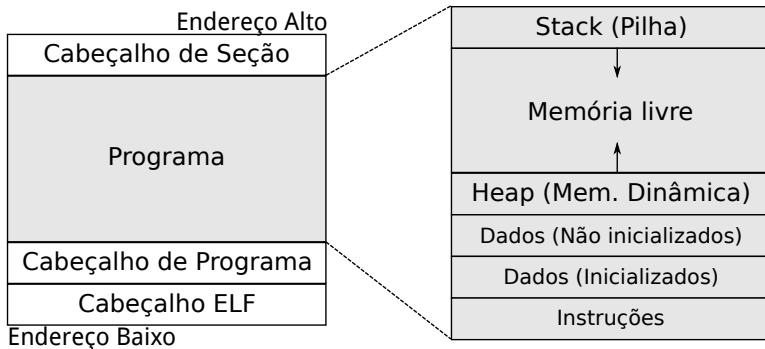
- ▶ O programa voltou a funcionar corretamente
 - ▶ As linhas 8, 9 e 10 não foram executadas
 - ▶ Este teste indica que o ponto de entrada de execução de um programa é definido pela linha 6

```
4   ...
5   section .text
6       global _start
7   ...
```

Estrutura básica de um programa



- Formato utilizado para exemplo é o ELF



Formato ELF - Geral

- ▶ Cabeçalho ELF
 - ▶ Assinatura 0x7F E L F
 - ▶ Arquitetura (x86 ou x64)
 - ▶ Ponto de Entrada
 - ▶ Tamanho dos cabeçalhos
 - ▶ entre outros
- ▶ Cabeçalho de Programa
 - ▶ Informações para o Sistema Operacional
 - ▶ Como é a “imagem do programa” em memória
- ▶ Programa
- ▶ Cabeçalho de Seção
 - ▶ Tamanho, deslocamento e nomes das seções do programa
- ▶ Curiosidade: `readelf -e hello.x`

Formato ELF - Programa

- ▶ Instruções

- ▶ Área que contém as instruções do programa

```
section .text
```

- ▶ A execução é realizada a partir deste ponto (*top-down*)

- ▶ Dados Inicializados

- ▶ Variáveis com valores pré-definidos

```
section .data
```

```
str01a : db "01a", 10
```

- ▶ Normalmente utilizada para definição de constantes
 - ▶ Pode ser alterado

Formato ELF - Programa

- ▶ Dados Não Inicializados
 - ▶ Variáveis
 - ▶ É realizada uma reserva de bytes em memória
 - ▶ Discutido na próxima aula
- ▶ Memória Dinâmica
 - ▶ Variáveis Dinâmicas
 - ▶ Calloc/Malloc
 - ▶ Será discutido no futuro
- ▶ Área Livre de Memória
 - ▶ Área entre Memória Dinâmica e Pilha
- ▶ Pilha
 - ▶ Passagem de Parâmetros, armazenamento temporário
 - ▶ Será discutido no futuro

Dados

- ▶ Comportamento
 - ▶ Não existe “tipos de dados” em memória
 - ▶ É a instrução que determina o comportamento
- ▶ Normalmente
 - ▶ É feito a definição do valor em memória (inicializadas)
ou
 - ▶ É feito a reserva de espaço em memória (não inicializadas)

Dados Inicializados (.data)

- ▶ Variáveis com valores pré-definidos

```
section .data
```

```
    simbolo: tamanho valor
```

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

```
    char c = 10;
```

- ▶ c é o nome da variável, ou seja, o símbolo para referência
 - ▶ 10 é o seu valor pré-definido
- ▶ Uma variável tem um tamanho em memória especificado por uma pseudo-instrução

```
    db: byte
```

```
    dw: word (2B)
```

```
    dd: dualword (4B)
```

```
    dq: quadword (8B)
```

Dados Inicializados (.data) - Exemplos

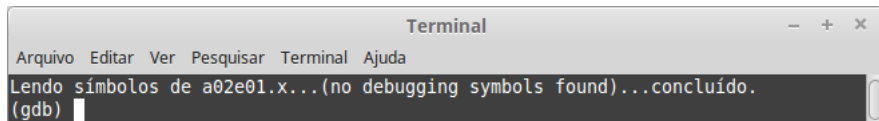
```
v1: db      0x55                ; byte 0x55
v2: db      0x55,0x56,0x57      ; 3 bytes em sucessao
v3: db      'a',0x55           ; caracteres com aspas
v4: db      'hello',13,10,'$' ; strings tambem
v5: dw      0x1234              ; 0x34 0x12
v6: dw      'a'                 ; 0x61 0x00
v7: dw      'ab'                ; 0x61 0x62
v8: dw      'abc'               ; 0x61 0x62 0x63 0x00
v9: dd      0x12345678          ; 0x78 0x56 0x34 0x12 0x00..
```

Debugger?

- ▶ Almirante Grace Hoper em 1947
 - ▶ Marinha dos EUA
 - ▶ Popularizou o termo na computação quando seus colegas encontraram uma mariposa impedindo o correto funcionamento do MARK II
 - ▶ Os termos *bug* e *debug* são antigos (registros em 1878 - Thomas Edison)
 - ▶ †1992
- ▶ gdb - GNU Debugger
 - ▶ Nativo na maioria das distros Linux
 - ▶ Permite depurar outros programas para encontrar *bugs* ou validá-los
 - ▶ GDB: The GNU Project Debugger

GDB

- ▶ Com o arquivo executável que se deseja verificar, faça:
`gdb nome.x`



```
Terminal
Arquivo Editar Ver Pesquisar Terminal Ajuda
Lendo símbolos de a02e01.x...(no debugging symbols found)...concluído.
(gdb) █
```

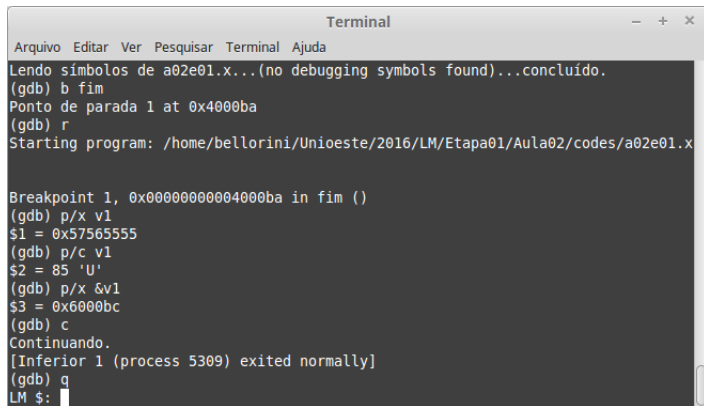
- ▶ Comandos:
 - ▶ Sair: `quit (q)`
 - ▶ BreakPoint: `break (b) nomeLabel`
 - ▶ Executar: `run (r) args`
 - ▶ Continuar: `continue (c)` - quando a execução encontra um BreakPoint
 - ▶ Mostrar valor de variável: `print/x (p/x) nomeVar`
 - ▶ Será mostrado o valor de 4 bytes em hexadecimal
 - ▶ Mostrar valor de variável: `print/c (p/c) nomeVar`
 - ▶ Será mostrado o valor de 1 bytes em hexadecimal

GDB - Passos para exemplo: a02e01.x

- ▶ Chamar GDB para o exemplo
`$ gdb a02e01.x`
- ▶ Configurar breakPoint
`(gdb) b fim`
- ▶ Executar programa exemplo
`(gdb) r`
- ▶ Mostrar o valor (4 bytes) da variável V1
`(gdb) p/x v1`
- ▶ Mostrar o valor (1 byte) da variável V1
`(gdb) p/c v1`
- ▶ Mostrar endereço da variável V1
`(gdb) p/x &v1`
- ▶ ...

GDB - Passos para exemplo: a02e01.x

- ▶ Continuar a execução
(gdb) c



```
Terminal
Arquivo Editar Ver Pesquisar Terminal Ajuda
Lendo símbolos de a02e01.x...(no debugging symbols found)...concluído.
(gdb) b fim
Ponto de parada 1 at 0x4000ba
(gdb) r
Starting program: /home/bellorini/Unioeste/2016/LM/Etapa01/Aula02/codes/a02e01.x

Breakpoint 1, 0x0000000004000ba in fim ()
(gdb) p/x v1
$1 = 0x57565555
(gdb) p/c v1
$2 = 85 'U'
(gdb) p/x &v1
$3 = 0x6000bc
(gdb) c
Continuando.
[Inferior 1 (process 5309) exited normally]
(gdb) q
LM $:
```

Exercício de Fixação

- ▶ EF0201: Debuggar as outras variáveis do código exemplo a02e01.asm
 - ▶ Crie um diagrama da memória com seu conteúdo
 - ▶ Posicionar cada variável em seu devido endereço a nível de byte
 - ▶ Ao todo são 28 bytes para o exemplo
 - ▶ Exemplo para v1 e v2:

v1	0x6000bc		
	0x55		
v2	0x6000bd	0x6000be	0x6000bf
	0x55	0x56	0x57
??	0x6000c0		
	??		

Relatório

- ▶ Somente relatório
 - ▶ O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
Arquivo **modeloRelatorioLM.odt**
 - ▶ Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão **PXX.nome.sobrenome.pdf**
Onde PXX é o número da prática, neste caso: P02