Linguagem de Montagem

Introdução
Aula 01

Edmar André Bellorini

Ano letivo 2021

Introdução

- Linguagem de Máquina (Baixo Nível)
  - É a linguagem que o computador realmente entende e executa
  - Composta por *strings* binárias
- Linguagem de Montagem (Código Intermediário)
  - Transcrição de primeiro nível da Linguagem de Máquina
    - Normalmente 1-para-1
  - Usa-se textos e números para compreensão humana
- Linguagem de Alto Nível
  - Linguagem com alto nível de abstração da Linguagem de Montagem

Linguagens (Alto Nível vs Montagem vs Máquina)

L. de Alto Nível (C)

```c
int main(){
   puts("Ola");
   return 0;
}
```

(43 bytes)

L. de Montagem (as)

```asm
; Aula 01 - Introdução
; hello.asm
; Meu primeiro assembly!
; nasm -f elf64 hello.asm ; ld hello.o -o hello.e

section .data
   strOla :  db "Ola", 10
   strOlaL:  equ $ - strOla

section .text
        global _start

_start:
   mov rax, 1
   mov rdi, 1
   lea rsi, [strOla]
   mov edx, strOlaL
   syscall

   mov rax, 60
   mov rdi, 0
   syscall
```

(341 bytes)

L. de Máquina (parcial)

```
7f454c46020101000000
00000000000002003e00
01000000b00040000000
00040000000000000000
00010000000000000000
...
000000bb00000000cd80
4f6cc3a10a0000002e73
796d746162002e737472
746162002e7368737472
746162002e7465787400
2e646174610000000000
00000000000000000000
...
2e6f007374724f6c6100
7374724f6c614c005f73
74617274005f5f627373
5f7374617274005f6564
617461005f656e6400
```

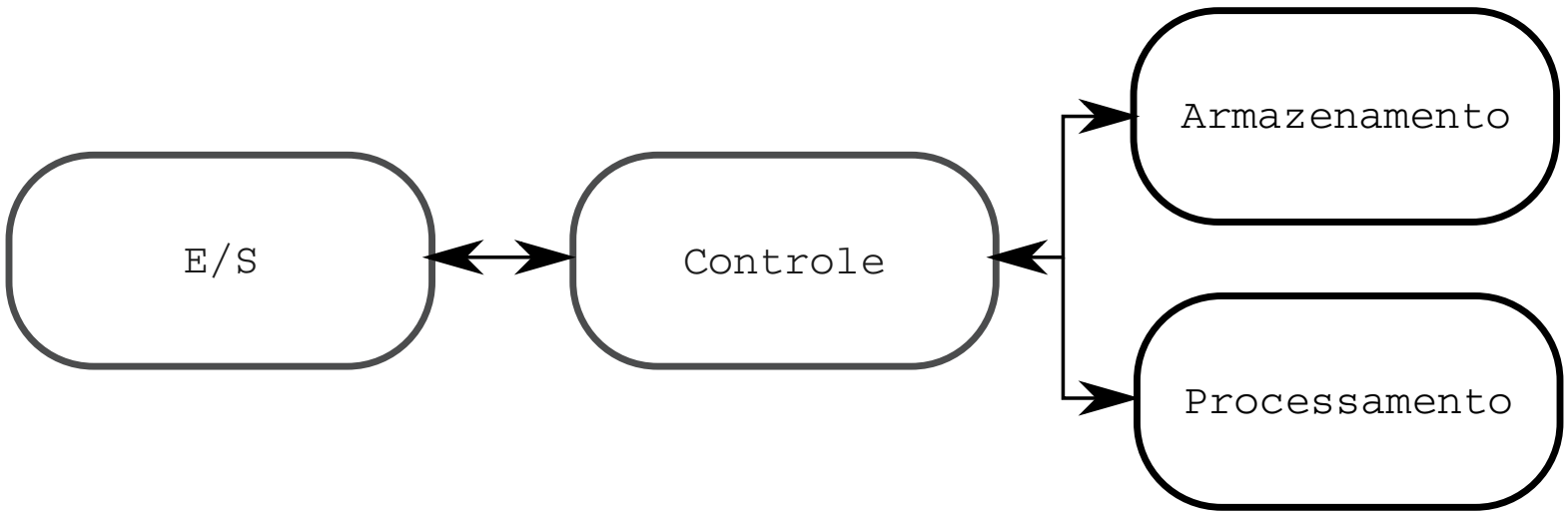
(8,9 KiB)

Linguagem de Montagem

- Por que estudar LM?
  - Como as instruções são executadas?
  - Como os dados estão representados em memória?
  - Como um programa interage com o S.O. e outros programas?
  - Como as instruções de Alto Nível são traduzidas para Baixo Nível?
    - Deseja tornar-se um melhor programador de Linguagens de Alto Nível?
  - Como é a arquitetura de um computador x86_64?

Arquitetura

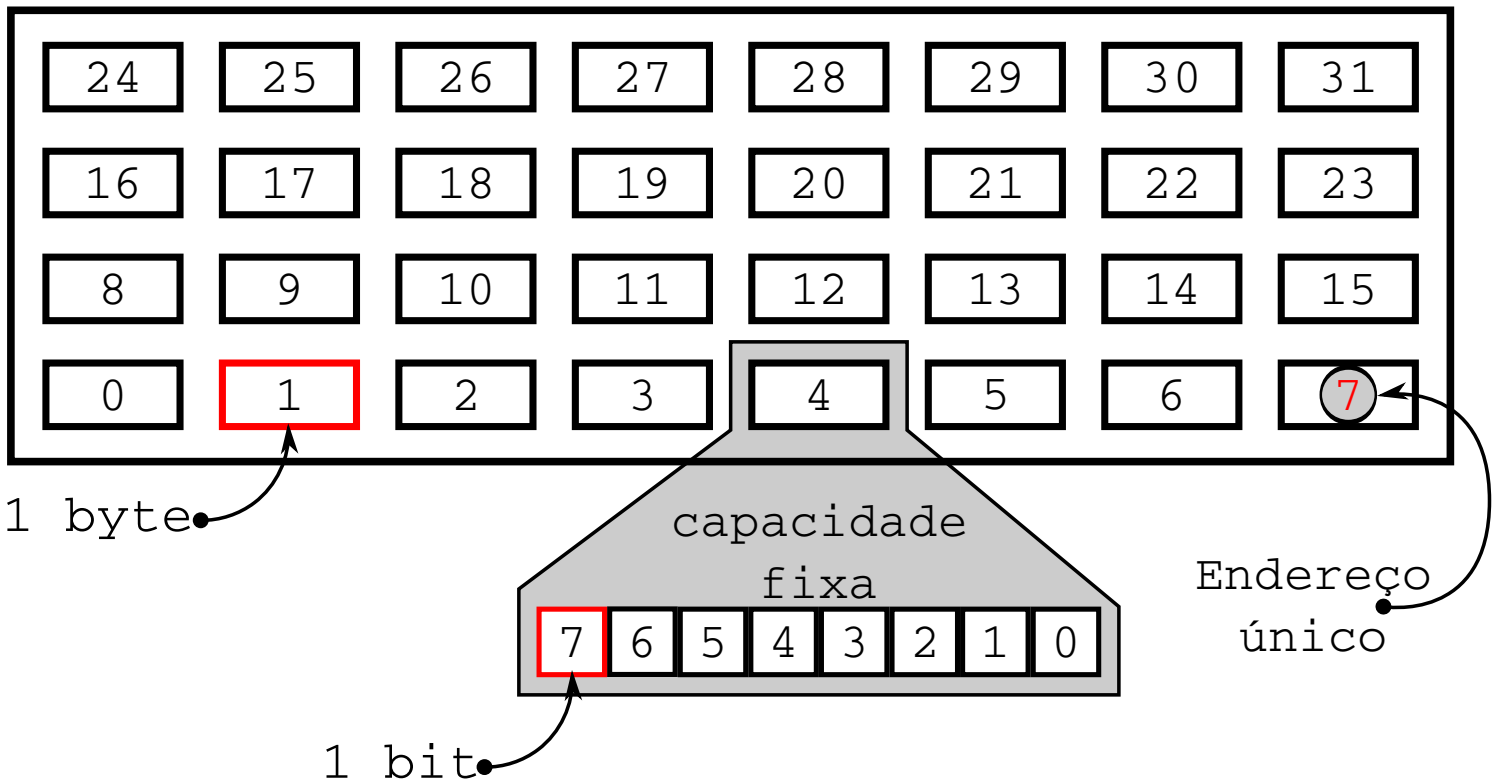
- Von Neumann
  - Armazenamento
  - Unidade de Processamento
  - Movimentação de Dados (E/S)
  - Controle do Fluxo de Dados (Conexões)



Armazenamento (Memória)

- Função
  - Local onde residem as informações
    - Sistema Operacional
    - Programas e Dados do usuário
- Estrutura
  - Local constituído de “espaços” para armazenamento
    - Cada espaço tem uma capacidade fixa (8 bits)
    - Cada espaço tem um endereço único

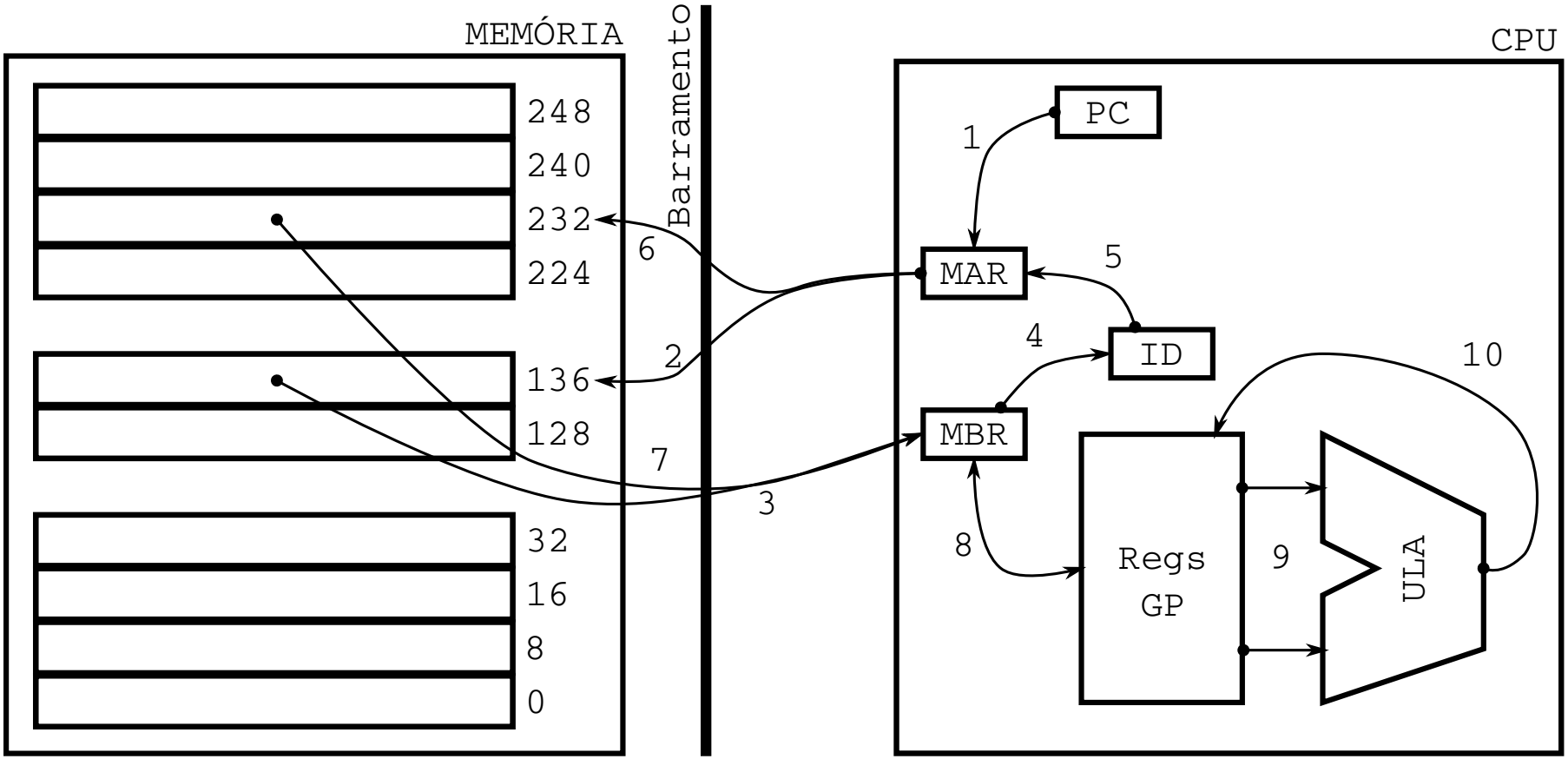
Armazenamento (Memória)



Unidade de Processamento

- Função
  - Ler instruções da memória (uma-a-uma)
  - Executar instruções
- Estrutura
  - Contador de Programa (PC - `Program-Counter`)
  - Decodificador de Instruções (ID - `Instruction Decoder`)
  - Registradores de Propósito Geral (GP - `General-Purpose`)
  - Registradores de Propósito Específico (MAR e MBR)
  - Unidade Lógica e Aritmética

Unidade de Processamento



Montador NASM

- **N**etwide **Ass**e**m**bler
  - Sintaxe simplificada
    Sintaxe Intel / alta legibilidade
  - Compatível com Windows e **Linux**
  - Programas x86 e x64
    no terminal execute: `uname -m`
    i686 ou i386 = máquina 32 bits (x86)
    x86_64 = máquina 64 bits (x64)
  - Licensa `BSD - 2 Clause`
  - Constante desenvolvimento
    versão 2.15.05 (Ubuntu rep) disponibilizada em 28/08/2020
    ▸ www.nasm.us

Sintaxe Intel vs AT&T

- Sintaxe Intel
  - Instrução:
    mnemônico destino, fonte
  - Exemplo:
    mov eax, 4
    copia para o local chamado eax o valor 4
  - Montadores: NASM, TASM (Turbo), MASM (Microsoft)
- Sintaxe AT&T
  - Instrução:
    mnemônico fonte, destino
  - Exemplo:
    mov $4, %eax
    copia o valor 4 para o local chamado eax
  - Montadores: GAS(GNU), nativo em Linux (as)

Sintaxe

Intel

```
section .data
   strOla :  db "Ola", 10
   strOlaL:  equ $ - strOla

section .text
        global _start

_start:
   mov rax, 1
   mov rdi, 1
   lea rsi, [strOla]
   mov edx, strOlaL
   syscall

   mov rax, 60
   mov rdi, 0
   syscall
```

AT&T

```
.section .data
   strOla:  .string "Ola\n\0"
   .equ strOlaL, .-strOla

.section .text
.globl _start

_start:
   movl $1, %eax
   movq $1, %rdi
   lea (strOla), %rsi
   movl $strOlaL, %edx
   syscall

   movq $60, %rax
   movq $0, %rsi
   syscall
```

HelloWorld

hello.c

```c
int main(){
    puts("Ola");
    return 0;
}
```

```
section .data
    strOla :  db "Ola", 10
    strOlaL:  equ $ - strOla

section .text
        global _start

_start:
    mov rax, 1
    mov rdi, 1
    lea rsi, [strOla]
    mov edx, strOlaL
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Compilar? Montar? Linkar?

- Compilar o código `hello.c` com compilador `gcc`

```
gcc hello.c -o helloC.x
```

- Montar e `linkar` o código `hello.asm` com:
  - montador `nasm` para máquinas 64 bits

```
nasm -f elf64 hello.asm
```

  - linkador `ld`

```
ld hello.o -o helloS.x
```

- Qual é o tamanho dos arquivos executáveis (linguagem de máquina) resultantes?
  - *Fast, Powerful and Small*

Executar

- Para executar arquivo (em Linguagem de Máquina):
  - O arquivo `helloC.x` compilado:

    `./helloC.x`
  - O arquivo `helloS.x` montado e linkado:

    `./helloS.x`
- Observações
  - O ./ indica para o S.O. Linux que o programa a ser executado encontra-se no diretório atual.
  - O retorno da execução do programa é avaliado com:

    `echo $?`

Atividades para Praticar

- AP0101: Elaborar um executável que imprima seu nome no terminal.
  - Utilize o código exemplo com sintaxe Intel para auxiliá-lo
  - É necessário montar, linkar e executar o código
  - Se algum erro for apresentado durante a produção do executável: anote, avalie e corrija.

Atividades para Praticar

- AP0102-Exploração: Com o código AP0101 completo, efetue as seguintes alterações de forma independente
  - para cada alteração: gere o executável, execute e documente os resultados
  - desfaça a alteração atual antes de passar para a próxima alteração

1. Comente a linha 11 (linha que contém global _start)
   É possível comentar uma linha utilizando ";" (ponto-e-vírgula)
2. Altere a linha 16
   de lea rsi, strOla para mov rsi, strOla
3. Comente a linha 17 (mov edx, strOlaL)
4. Altere a linha 17
   de mov edx, strOlaL para mov edx, 2
5. Comente a linha 18 (syscall)
6. Altere a linha 21
   de mov rdi, 0 para mov rdi, 5
   Após executar digite "echo $?" no terminal

Fim do Documento

Dúvidas?

Próxima aula:

- Aula 02: Estrutura dos Programas
  - Conceitos gerais de como os Programas são armazenados em disco e carregados para memória
    - Sistema ELF
  - Dados inicializados
  - Montar, Ligar e Executar
  - Debugger (GDB)