

# Programação Imperativa

## Assembly

**Prof. Edson Alves**

Faculdade UnB Gama

2020

# Sumário

1. **Motivação**
2. **Arquitetura Intel x86**
3. **Configuração do ambiente de desenvolvimento**

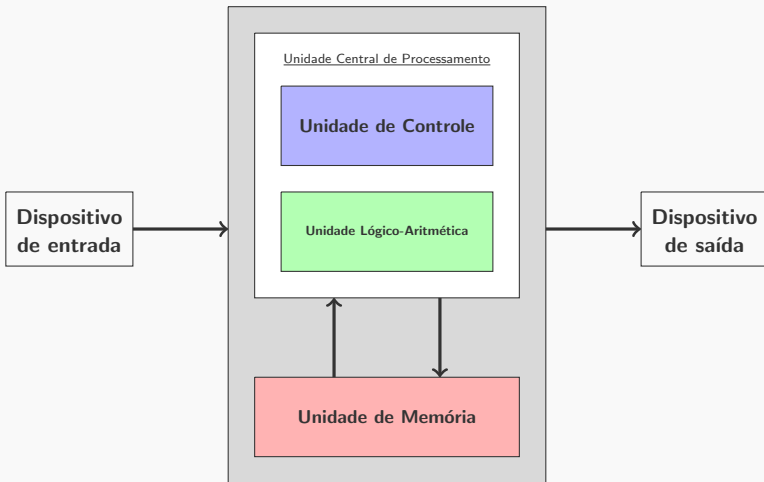
# Máquinas idealizadas e máquinas reais

- ▶ As máquinas de Turing e os ábacos são máquinas idealizadas
- ▶ Algumas de suas características não podem ser traduzidas diretamente para máquinas reais, como o fato da fita ser infinita em um máquina de Turing, ou de um registrador de um ábaco armazenar um inteiro de tamanho arbitrário
- ▶ Além disso, o conjunto de instruções para estas máquinas idealizadas é mínimo, de modo que escrever programas, mesmo simples, se torna uma tarefa árdua e demorada
- ▶ Em 1945, um novo passo foi dado em direção da construção de máquinas reais, pelo matemático e físico John Von Neumann

# Arquitetura de Von Neumann

- ▶ A **arquitetura de Von Neumann** (ou arquitetura de Princeton), descreve a arquitetura básica para um computador digital eletrônico
- ▶ Ela é composta por um unidade lógico-aritmética (*arithmetic logic unit* – ALU) e registradores de processador, uma unidade de controle contendo um registrador de instrução e um contador de programa (*program counter* – PC), uma unidade de memória para armazenar o programa e as instruções, uma memória externa e mecanismos de entrada e saída de dados
- ▶ Esta arquitetura foi o alicerce que deu origem aos computadores modernos que, embora mais elaborados e sofisticados, em grande parte ainda seguem esta arquitetura básica

# Visualização da arquitetura de Von Neumann



Fonte: [Wikipédia](#), com adaptações.

# Linguagem de Máquina

- ▶ As máquinas reais dispõem de dispositivos internos (sejam mecânicos, elétricos ou eletrônicos) que permitem a realização de tarefas básicas (somas, operações lógicas, etc)
- ▶ Nas primeiras máquinas reais, para montar um programa que resolvesse um problema específico era preciso reconfigurar fisicamente tais equipamentos
- ▶ A medida que estas máquinas foram evoluindo, códigos binários foram associadas à estas tarefas (instruções) básicas, de modo que era possível escrever novos programas sem modificar o hardware fisicamente, inserindo tais sequências de zeros e uns por meio de *switches*
- ▶ Estas códigos binários constituem a **linguagem de máquina** do hardware
- ▶ Mesmo com todo este avanço, escrever programas não eram muito mais fácil ou simples do que escrever programas para ábacos ou máquinas de Turing

# Linguagem Assembly

- ▶ Logo os programadores entenderam que era mais fácil associar os códigos binários das instruções a mnemônicos
- ▶ Estes mnemônicos representavam tanto as instruções em si quanto as principais localizações de memória
- ▶ Assim foram desenvolvidas, a partir dos anos 1950, as linguagens Assembly
- ▶ Para cada linguagem era criado um *assembler* (montador), que consistia em uma ferramenta de software que traduzia estes mnemônicos em linguagem de máquina
- ▶ Cada arquitetura e cada hardware em particular oferecia um conjunto distinto de instruções, de modo que para cada um deles era necessária uma linguagem Assembly distinta
- ▶ Embora elas representem um avanço considerável em relação aos ábacos, as linguagens Assembly carecem da abstração presente na notação matemática convencional

# Registradores x86

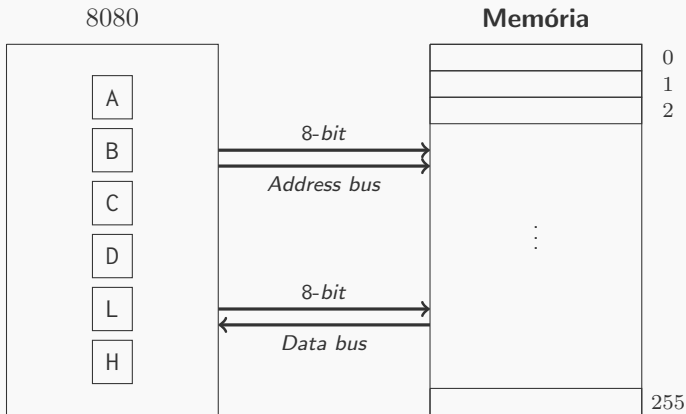
- ▶ Um dos principais motivos do sucesso da linha de processadores x86 da Intel foi o fato de que eles foram desenvolvidos para serem sempre compatíveis com os seus antecessores
- ▶ Deste modo, eram minimizadas as alterações em hardware e software necessárias para que um software desenvolvido para um processador fossem portado e funcionasse em futuras iterações desta arquitetura
- ▶ Os concorrentes que optam por começar do zero em determinadas iterações (como a Motorola e a IBM com os chips PowerPC e a DEC com os processadores Alpha) acabaram não obtendo o mesmo sucesso a longo prazo
- ▶ O estudo dos registradores x86, neste sentido, ganha um aspecto histórico



# Registradores 8080

- ▶ Os processadores 8080 foram os primeiros a serem produzidos em massa
- ▶ O primeiro computador pessoal (Altair, em 1975) utilizava um processador 8080
- ▶ Ele possuía 6 registradores de 8-*bits*: A, B, C, D, H, L
- ▶ Os endereços de memória e a transferência de dados também eram de 8-*bits*
- ▶ O registrador L (*low*) armazenava endereços de memória, e a memória era composta de 256-*bytes*, identificados pelos números de 0 a 255
- ▶ Este registrador poderia ser combinado com o registrador H (*high*) para compôr um endereço de 16-*bits*
- ▶ Quando este par era referenciado por uma instrução, ela utilizava a letra X (*extended*) para sinalizar este uso

# Visualização do 8080



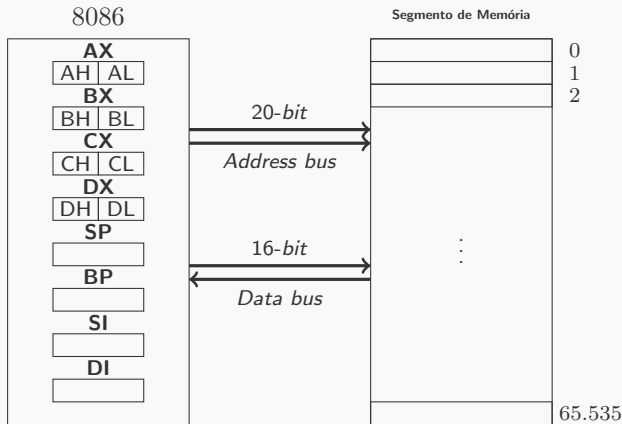
**Figura:** Visualização de um computador com processador 8080.<sup>1</sup>

<sup>1</sup>Fonte: **NEVELN**, Bob. *Linux Assembly Language Programming*, Open Source Technology Series, Prentice-Hall, 2000 (com adaptações).

# Registradores 8086

- ▶ Quando o processador de 16-*bits* 8086 foi desenvolvido, a letra X foi usada para designar os novos registradores de 16-*bits*: AX, BX, CX e DX
- ▶ Estes novos registradores são pares de registradores de 8-*bits*: AL, AH, BL, BH, CL, CH, DL e DH
- ▶ Os demais registradores de 16-*bits*, a saber: SP, BP, SI e DI, não são pares, e não trazem a letra X em sua nomenclatura
- ▶ A transferência de dados é feita em 16-*bits*
- ▶ Contudo, o endereçamento de memória é feito em 20-*bits*, porém sem uma forma eficiente de usar todos estes *bits* de uma só vez

# Visualização parcial do 8086



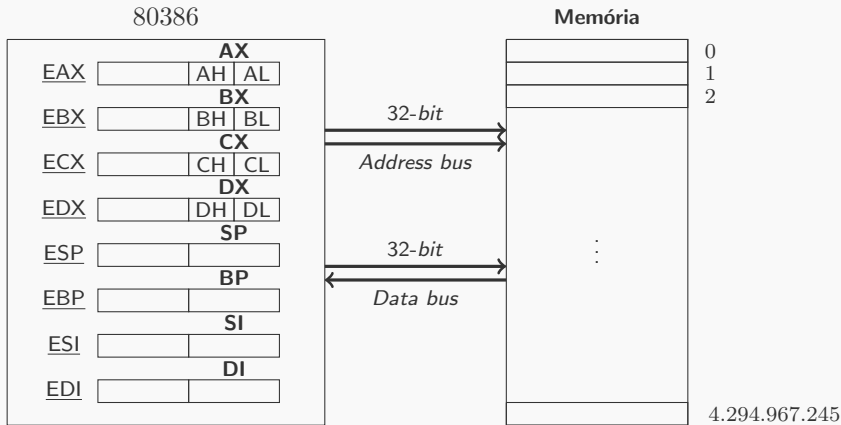
**Figura:** Visualização parcial de um computador com processador 8086.<sup>2</sup>

<sup>2</sup>Fonte: **NEVELN**, Bob. *Linux Assembly Language Programming*, Open Source Technology Series, Prentice-Hall, 2000 (com adaptações).

# Registradores 80386

- ▶ A arquitetura dos processadores 8086 foi aperfeiçoada nos novos processadores 80386
- ▶ Os oito registradores que iniciam com a letra E (*extension*) são extensões dos registradores do 8086
- ▶ Cada registrador tem 32-*bits*, e seus 16-*bits* menos significativos correspondem ao registradores do 8086
- ▶ A transferência de dados e o endereçamento de memória são feitos em 32-*bits*
- ▶ Os processadores 80486 e Pentium também são computadores de 32-*bits*

# Visualização parcial do 80386



**Figura:** Visualização parcial de um computador com processador 80386.<sup>3</sup>

<sup>3</sup>Fonte: **NEVELN**, Bob. *Linux Assembly Language Programming*, Open Source Technology Series, Prentice-Hall, 2000 (com adaptações).

## Registradores de dados (*Data Registers*) do 80386

| Registrador | Nome                        | Uso sugerido  |
|-------------|-----------------------------|---|
| EAX         | <i>Accumulator Register</i> | Operações aritméticas, lógicas e transferência de dados       |
| EBX         | <i>Base Register</i>        | Registro do ponteiro base de dados contíguos                  |
| ECX         | <i>Counter Register</i>     | Contadores  |
| EDX         | <i>Data Register</i>        | Transferência de dados e operações de multiplicação e divisão |

# Registradores de ponteiros e de índices do 80386

| Tipo                    | Registrador | Nome                     | Uso sugerido                        |
|-------------------------|-------------|--------------------------|-------------------------------------|
| <i>Pointer Register</i> | ESP         | <i>Stack Pointer</i>     | Contém o endereço do topo da pilha  |
|                         | SBP         | <i>Base Pointer</i>      | Contém o endereço da base da pilha  |
| <i>Index Register</i>   | ESI         | <i>Source Index</i>      | Ponteiro/índice inicial da contagem |
|                         | EDI         | <i>Destination Index</i> | Ponteiro/índice final da contagem   |



# NASM

- ▶ The Netwide Assembler, ou NASM, é um montador (*assembler*) de código livre, que roda em plataforma DOS e Linux
- ▶ Desde a versão 2.07, o NASM está sobre a licença BSD (*Simplified (2-clause)*)
- ▶ Ele foi desenvolvido inicialmente por Simon Tatham e Julian Hall, sendo mantido atualmente por um time liderado por H. Peter Anvin
- ▶ O NASM tem suporte para todas as arquiteturas x86
- ▶ Em distribuições Linux com suporte ao apt, ele pode ser instalado com o comando

```
$ sudo apt-get install nasm
```

# Teste do ambiente

- ▶ Como os programas serão escritos para rodar em ambiente Linux, estes códigos devem levar em conta as características deste ambiente operacional
- ▶ Isto porque será o sistema operacional o responsável pela interface com os periféricos
- ▶ Assim, o código *assembly* não tem acesso direto ao hardware
- ▶ As interações com os hardwares serão feitas por meio dos *drivers* do sistema Linux, através de chamadas de sistema (*syscalls*)
- ▶ O menor código possível simples retorna a execução para o sistema operacional, com código 0 (zero), indicando sucesso

# Menor código *assembly*

```
1 ; Menor programa assembly possível com interação com ambiente Linux
2
3 SECTION .text
4 global _start
5
6 _start:
7     mov ebx, 0 ; Move o código de retorno (zero) para EBX
8     mov eax, 1 ; Move o código de SYS_EXIT (opcode 1) para EAX
9     int 80h    ; Executa a syscall, encerrando o programa com sucesso
```

# Compilação, linkedição e execução

- ▶ Para compilar (montar) um código *assembly* (extensões `.asm` ou `.s`), é preciso invocar o NASM, indicando o formato do objeto a ser criado:

```
$ nasm -f elf main.s
```

- ▶ No processo de linkedição é preciso também indicar, por meio da opção `-m`, que a arquitetura alvo é de 64 *bits*:

```
$ ld -m elf_i386 main.o -o main
```

- ▶ A opção `-o` indica o nome do executável a ser gerado
- ▶ Para rodar o executável criado, basta usar os mesmo mecanismos disponíveis em Linux para invocar um programa como, por exemplo, indicar seu caminho:

```
$ ./main
```

# Script para compilação, linkedição e execução

```
1 #!/bin/bash
2
3 #####
4 # Monta o código assembly indicado no parâmetro de entrada e executa #
5 # o programa de mesmo nome.                                         #
6 #####
7 if [ "$#" -lt 1 ]; then
8     echo "Usage: $0 {source.s}"
9     exit
10 fi
11
12 source=$1
13 output=${source%. *}
14
15 # Compila (monta) o código com o NASM
16 nasm -f elf $source
17
18 # Linka o código para sistemas de 64 bits
19 ld -m elf_i386 $output.o -o $output
20
21 # Executa o programa gerado
22 ./ $output
```

# Referências

1. asmtutor.com. [Learn Assembly Language](#), acesso em 16/01/2020.
2. AssemblyLT. [x86 Assembly Registers \[All Types Explained\]](#), acesso em 24/01/2020.
3. NASM. [Site Oficial](#), acesso em 16/01/2020.
4. **NEVELN**, Bob. *Linux Assembly Language Programming*, Open Source Technology Series, Prentice-Hall, 2000.
5. ScienceDirect. [Security in embedded systems](#), acesso em 16/01/2020.
6. **SHALOM**, Elad. *A Review of Programming Paradigms Throughout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.