



# Programação de Software

## Básico – Arquitetura dos processadores

### Intel x86

MATA49


Prof. Babacar Mane

2021.1 – Aula 2



# Programação de Software Básico

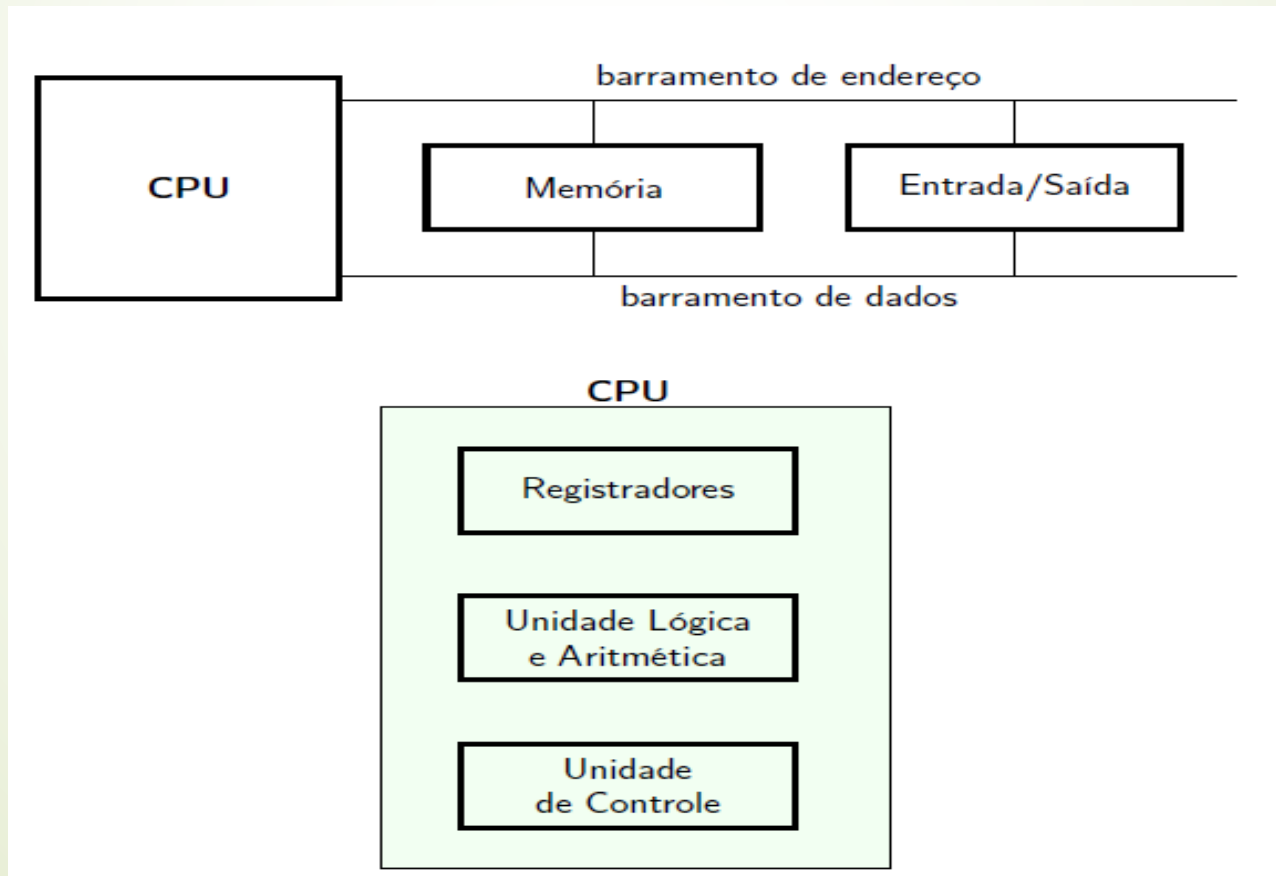
## Arquitetura dos processadores Intel x86

- ❑ Conteúdo
  - ❑ Ciclo de Busca e execução
  - ❑ Arquitetura da família X86
  - ❑ Sistemas de numeração
  - ❑ Linguagem de montagem
- 

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Modelo simplificado de um computador





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

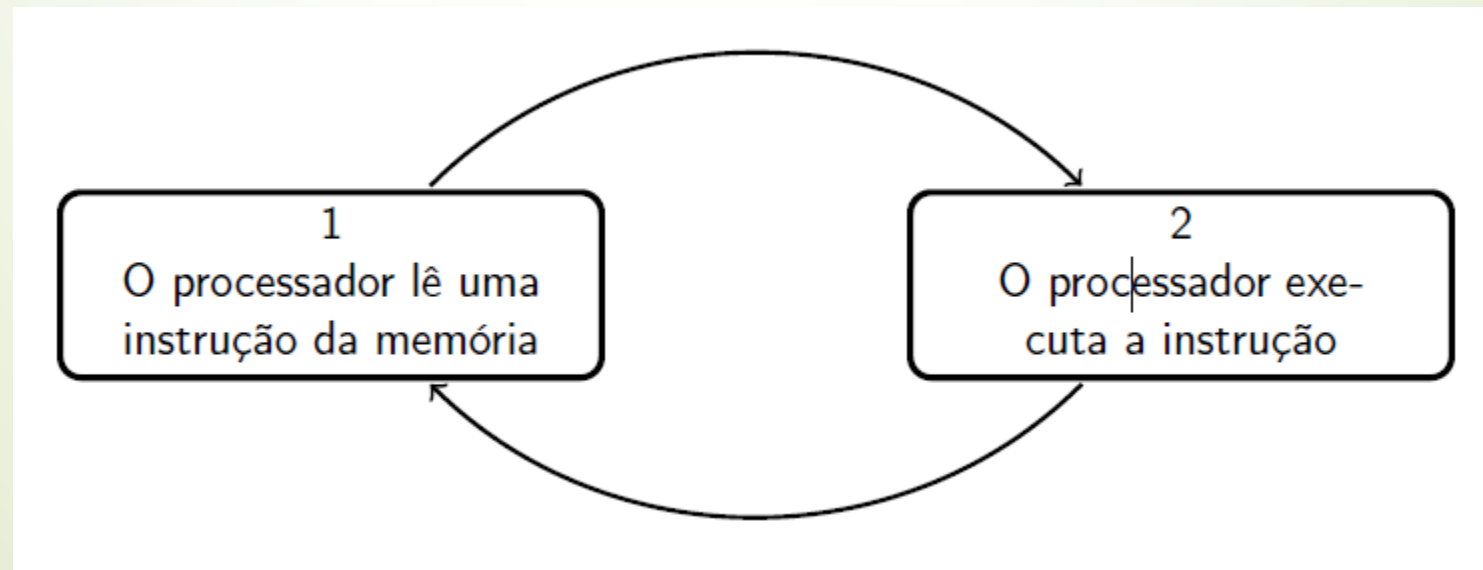
### □ Temos dentro de uma CPU

- Unidade Lógica e Aritmética (ULA)
  - *Realiza todas as tarefas relacionadas a operações lógicas (and, or, not, etc.) e aritméticas (adições, subtrações, etc.)*
- Unidade de Controle (UC)
  - *Controla as ações realizadas pelo computador, comandando todos os demais componentes de sua arquitetura*

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ O ciclo de busca e execução
  - Programa: lista de instruções
  - O Processador efetua uma computação por meio do ciclo de busca e execução:





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ **A UCP executa cada instrução por meio de uma série de pequenos passos:**
  - 1. Lê a próxima instrução na memória e a armazena no registrador de instrução
  - 2. Muda o registrador contador de programa, para que ele aponte para a instrução seguinte
  - 3. Determina o tipo da instrução que acabou de ser lida
  - 4. Se a instrução usa algum dado da memória, determina onde ele está
  - 5. Carrega o dado, se necessário, em um registrador da UCP
  - 6. Executa a instrução
  - 7. Volta para o passo 1, para começar a execução da instrução seguinte





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### □ Temos dentro de uma CPU - Registradores

- Um registrador é uma coleção de circuitos que armazenam bits
- Os registradores de um processador não precisam armazenar uma mesma quantidade de bits (mas é mais fácil de se lidar com eles quando eles são assim)
- A quantidade de bits que se pode armazenar em um registrador típico do processador é um dos atributos que determinam sua classificação (Ex.: processador de 32-bits, ou de 64-bits, etc.)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### □ Temos dentro de uma CPU – Registradores

- Cada registrador possui uma função própria.  
Exemplos:
  - **Contador de programa** (PC - Program Counter ) – aponta para a próxima instrução a executar
  - **Registrador de instrução** (IR - Instruction Register ) – armazena a instrução em execução
  - **Armazenamento de resultados intermediários**





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Processadores – conjunto de instruções

- As instruções são as operações que um processador é capaz de executar; elas são a parte do processador que é “visível” para os programadores
- Cada processador possui o seu próprio conjunto de instruções (**Instruction Set**)
- Por meio de uma linguagem de montagem (assembly), podemos usar o conjunto de instruções diretamente
- Mesmo processadores com arquiteturas internas diferentes podem ter um mesmo conjunto de instruções (ex.: Intel Pentium e AMD Athlon)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Processadores CISC X RISC

- Complex instruction set computer (CISC)
  - Capaz de executar várias centenas de instruções complexas diferentes (= versatilidade)
  - Ex: Power da IBM e Sparc da Oracle
- Reduced instruction set computer (RISC)
  - Capaz de executar apenas algumas poucas instruções simples (= rapidez, menor custo)
  - Ex: Intel



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### **Exemplo**

Comando em linguagem de alto nível:

$A = B + C$  /\* considerando que todas as quantidades estão em memória \*/

### **Em linguagem de montagem em um processador CISC:**

add mem(B), mem(C), mem(A)

### **Em linguagem de montagem em um processador RISC:**

load mem(B), reg(1);

load mem(C), reg(2);

add reg(1), reg(2), reg(3);

store reg(3), mem(A);



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

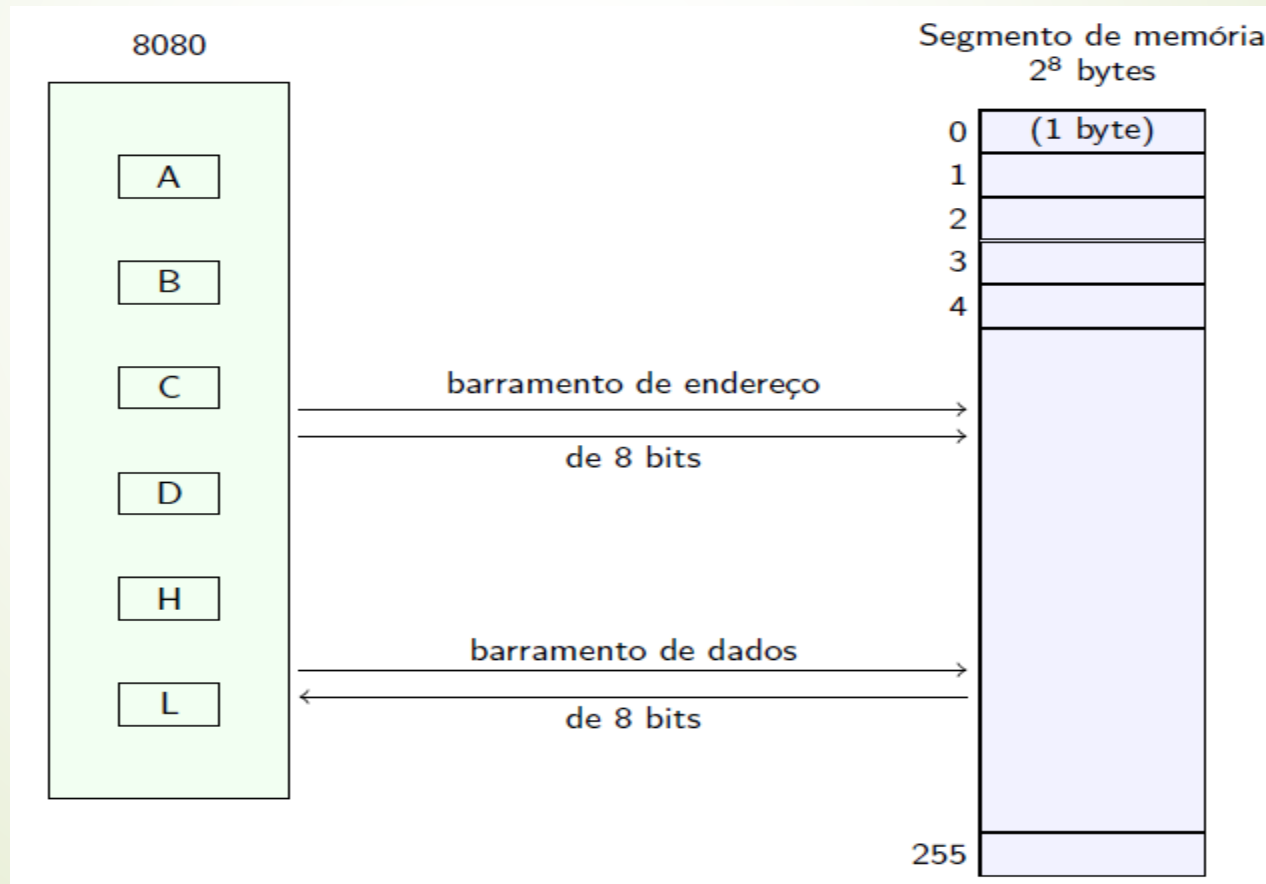
### □ Operações

- As instruções de um processador se relacionam às seguintes funcionalidades:
  - operações matemáticas e lógicas
  - movimentação de dados (transferência de dados da memória para os registradores e vice-versa)
  - operações de entrada/saída (leitura ou escrita de dados em dispositivos de entrada e saída)
  - controle do fluxo de execução (desvios condicionais ou incondicionais)

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Registradores do processador Intel 8080 (8 bits)





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Registradores de uso geral do processador Intel 8080 (8 bits)
  - ❑ A (**Acumulador**) – usado em movimentações de dados, operações aritméticas e entrada/saída
  - ❑ B (**Base**) – usado como ponteiro para acesso a memória; também recebe o valor de retorno de algumas interrupções
  - ❑ C (**Contador**) – usado para controlar o número de vezes que um laço deve ser executado ou o número de shifts a ser realizado; também usado em operações aritméticas





# Programação de Software Básico

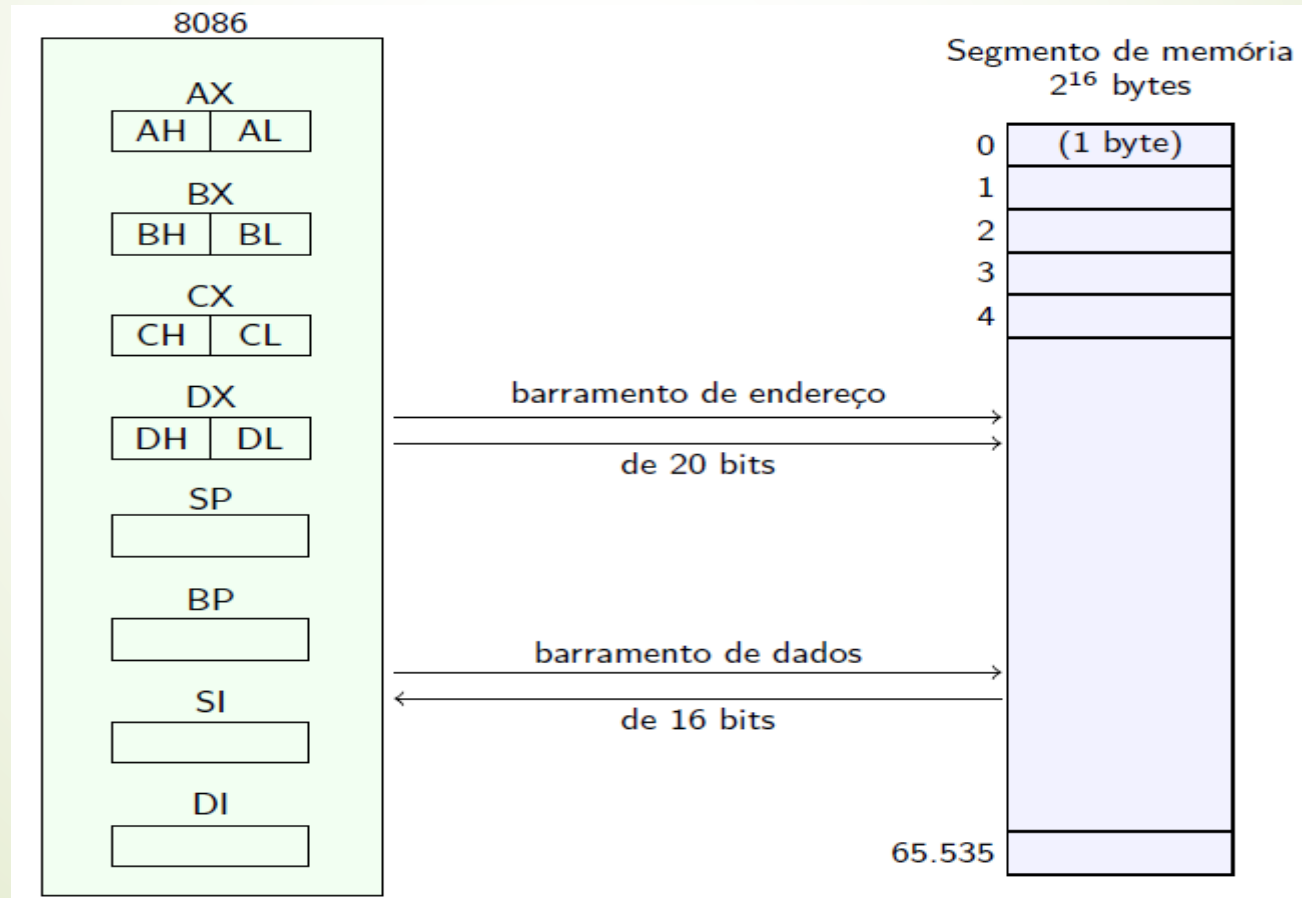
## Arquitetura dos processadores Intel x86

- ❑ Registradores de uso geral do processador Intel 8080 (8 bits)
  - ❑ D (**Dados**) – usado em operações de entrada e saída; também usado em operações de multiplicação ou divisão de números grandes
  - ❑ H (**High**) e L (**Low**) – usados de forma conjunta, como um registrador de 16 bits, que pode ser usado para endereços

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Registradores do processador Intel 8086 (16 bits)





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Registradores do processador Intel 8086 (16 bits)
  - 4 registradores de 16-bits de uso geral: (AX, BX, CX, DX)
    - – São principalmente usados para transferência de dados e operações aritméticas
    - – Pode ser decomposto em dois registradores de 8-bits
      - Ex: AH (higher) e AL (lower)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Registradores para índices e ponteiros do processador Intel 8086 (16 bits)
  - SP (Stack pointer register ) – armazena o endereço do topo da pilha de dados
  - BP (Stack base pointer register ) – armazena o endereço da base da pilha de dados
  - SI (Source index register ) – usado na manipulação de strings e vetores
  - DI (Destination index register ) – usado na manipulação de strings e vetores



# Programação de Software Básico

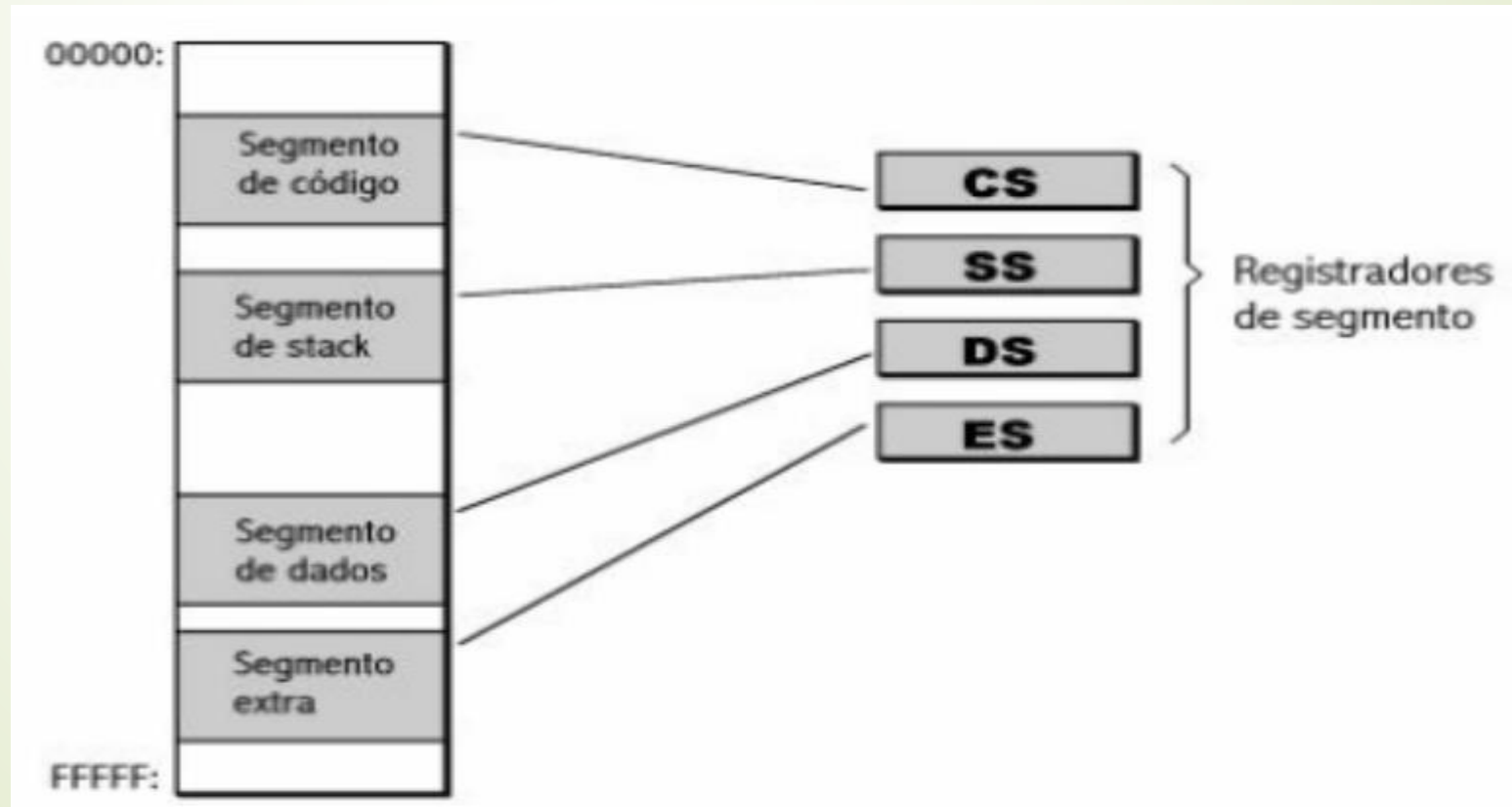
## Arquitetura dos processadores Intel x86

- ❑ Registradores 16-bits de segmento de programa (dentro da memória principal): Dados, variáveis, código etc...
  - Armazenam partes do programa
  - CS (**Code Segment**), DS (**Data Segment**),
  - SS(**Stack Segment**) e ES(**Extra Segment**)
  
- ❑ Cada segmento no 8086 é uma área de memória com no mínimo 64 KB e no máximo 1MB

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Registradores de segmento indicam o endereço inicial do segmento







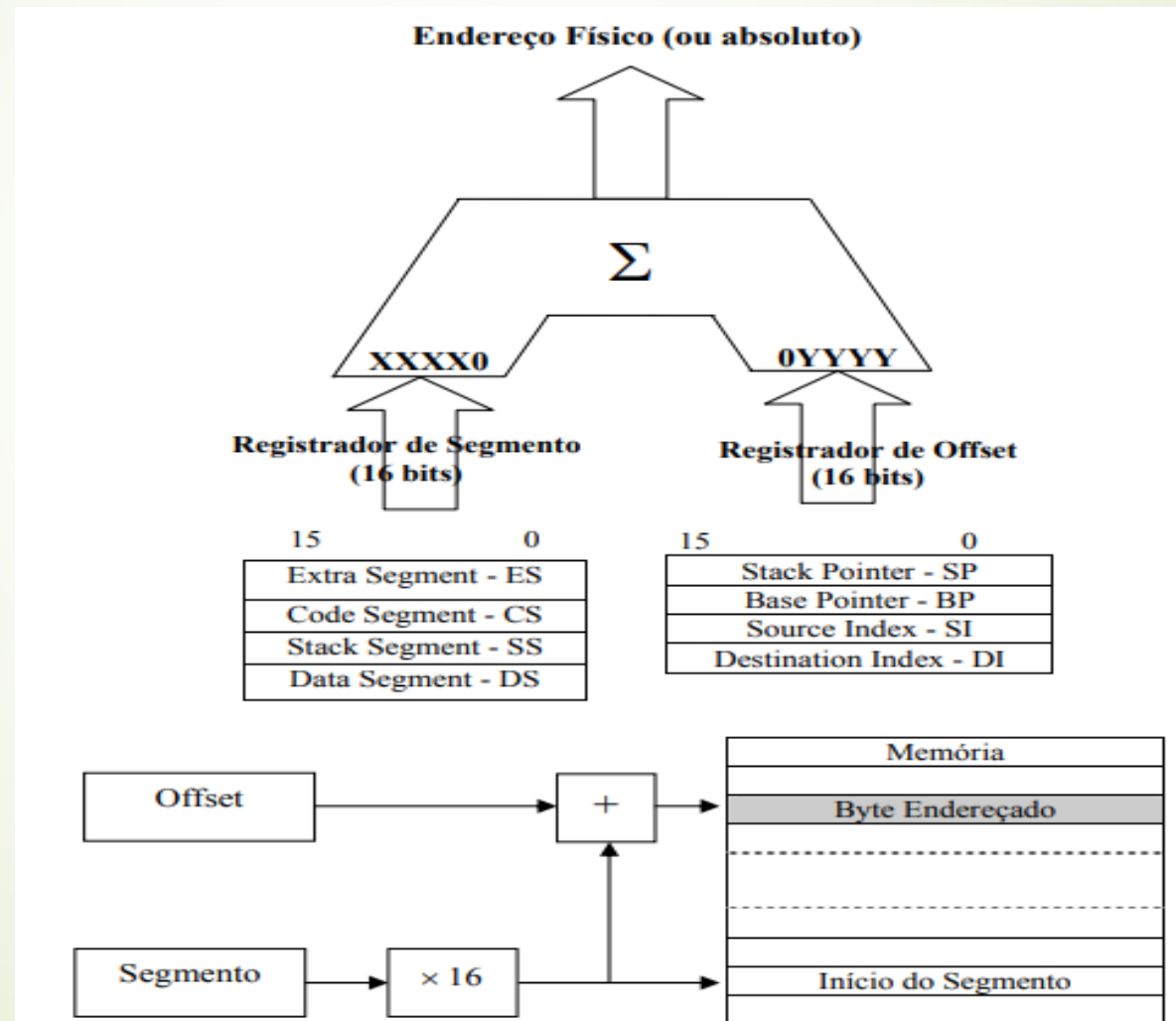
# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Um registrador (**IP Instruction Pointer**) para indicar a próxima instrução a ser executada
- ❑ **Registrador de Segmento**: registrador que armazena o **endereço base**. Deve ser somado ao **endereço de deslocamento** (ou **offset**, ou ainda **endereço lógico**) para obter o **endereço físico** (ou **endereço absoluto**)

# Programação de Software Básico

## Arquitetura dos processadores Intel x86





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

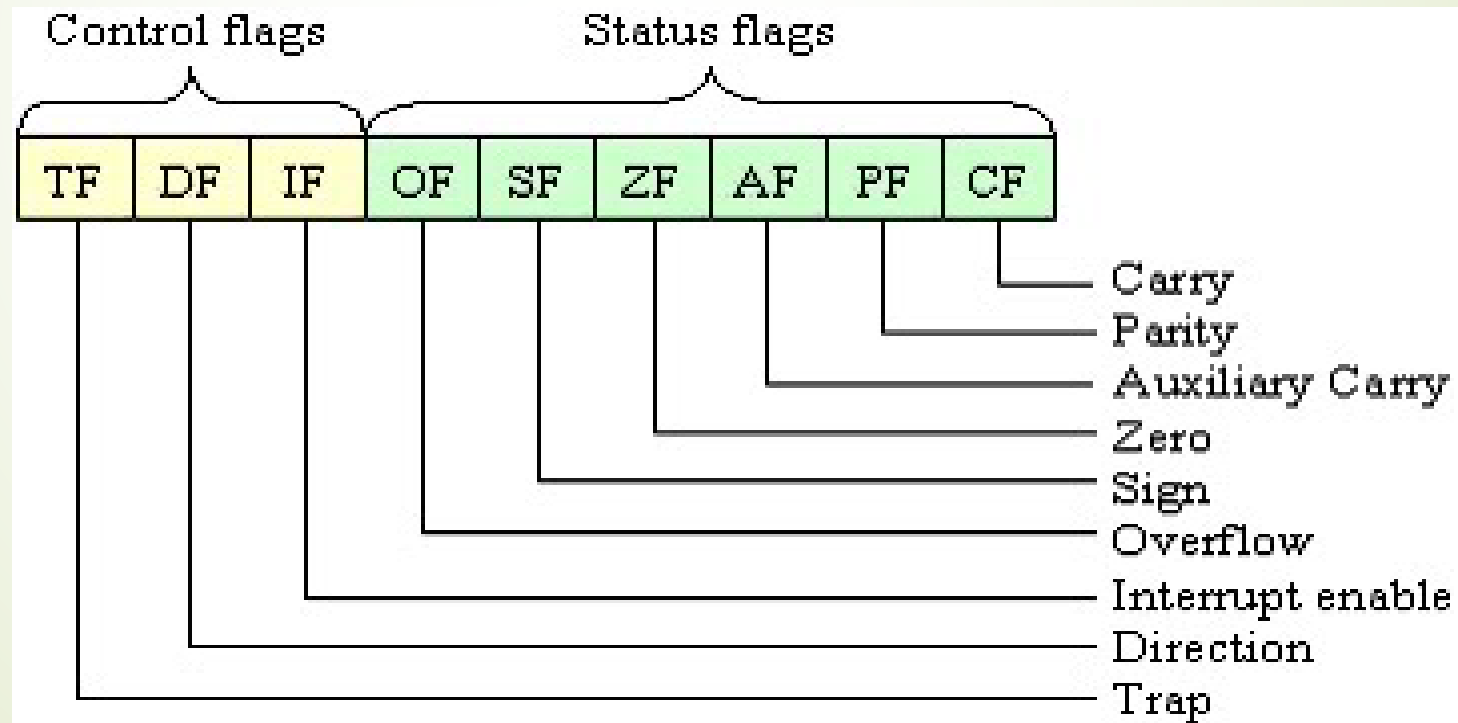
### ❑ Registrador FLAGS:

- É também conhecido como registrador F (de Flags), ou registrador PSW (Program Status Word)
- É um registrador de 8 bits (mas somente 5 bits são utilizados) que armazena o estado da última operação realizada na ULA.

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ FLAGS: Cada bit possui uma semântica específica





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ FLAGS:

- **Carry flag (CF)**: indica se o resultado de uma operação aritmética (sem sinal) extrapolou o tamanho do destino
- **Overflow flag (OF)**: O mesmo que o carry flag, mas para operações com sinal
- **Sign flag (SF)**: Indica se uma operação lógica ou aritmética gera um resultado negativo
- **Zero flag (ZF)**: Indica se uma operação lógica ou aritmética gera um resultado zero

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ FLAGS:

- **Auxiliary Carry flag (AC)**: Quando uma operação aritmética provoca um transporte de bit de 3 bits para 4 num operando de 8 bits

Carry Flag

$$\begin{array}{r} \text{AC} \\ + \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array} \\ \hline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}$$

$$\begin{array}{r} \begin{array}{cccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\ \underbrace{\hspace{10em}}_{P=1 \text{ e } Z=1} \end{array}$$





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

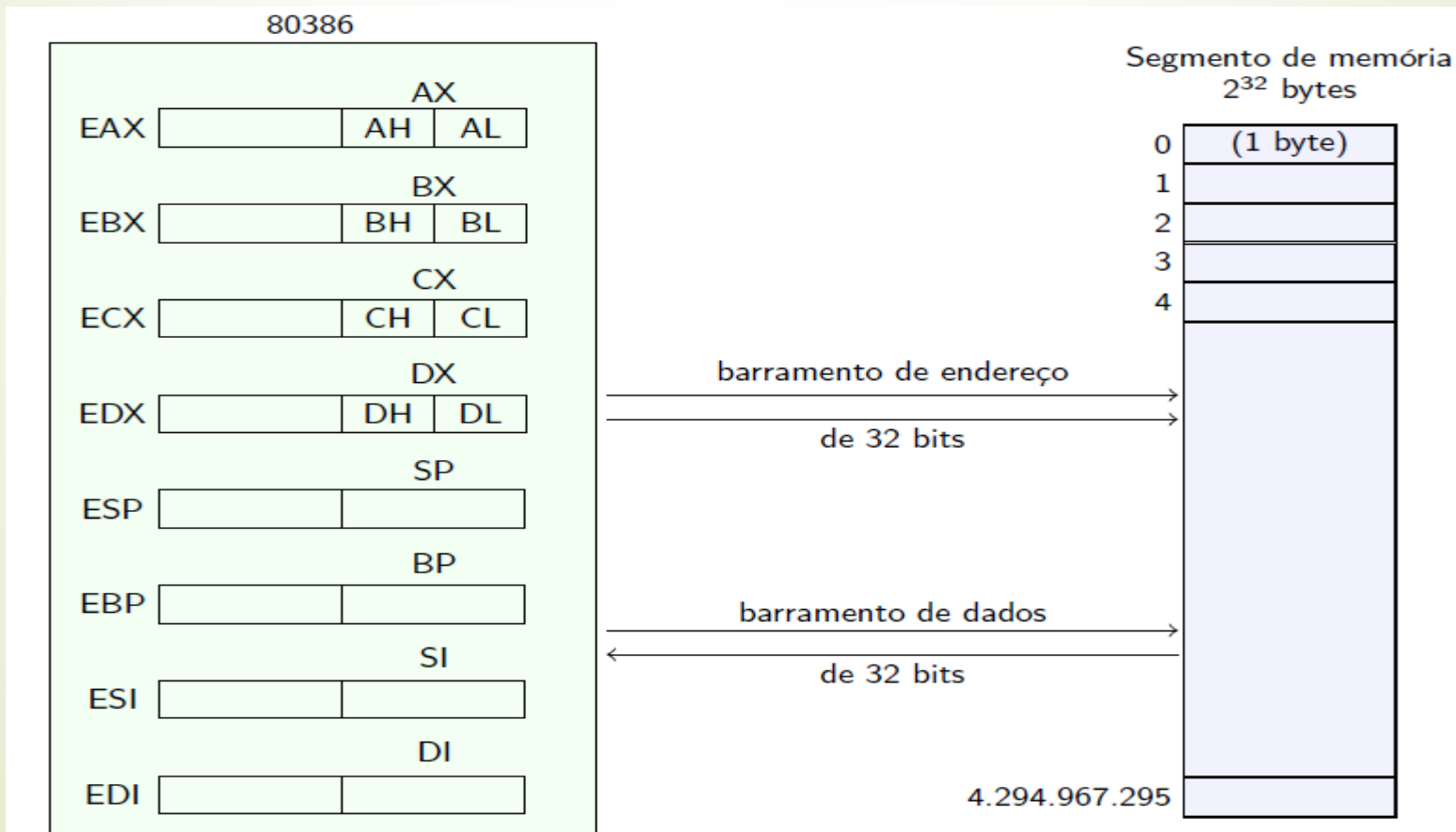
### ❑ FLAGS:

- **Parity flag (PF)**: Indica a paridade do número resultante. Em geral, é utilizado para a verificação de erro, quando existe uma possibilidade de que os dados podem ser alterados ou corrompidos
- **Interrupt enable Flag (IF)**: flag de interrupção (bit 1) de E/S
- **Direction Flag (DF)** - usado por algumas instruções para processar cadeias de dados.

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

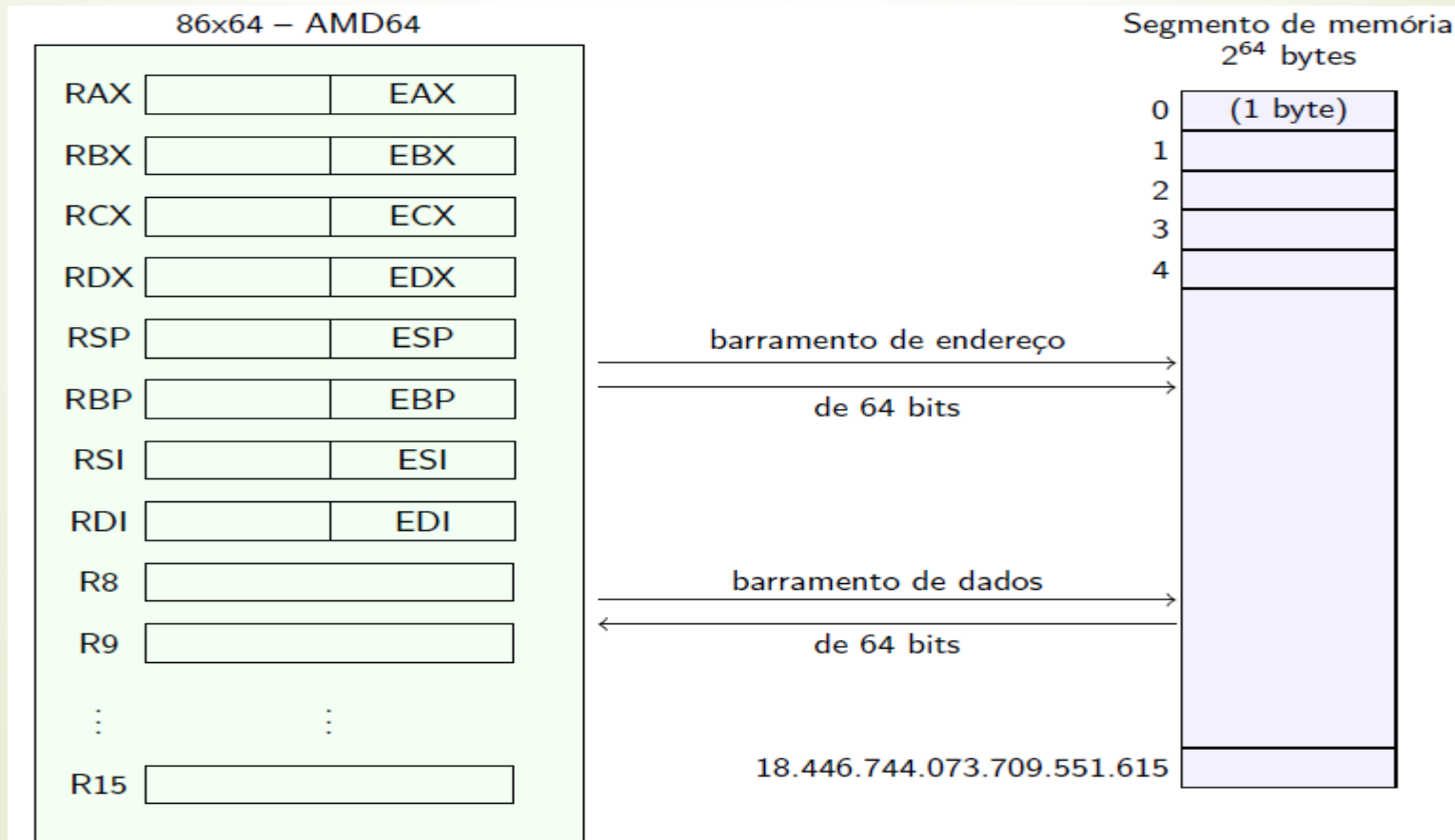
### ❑ Registradores do processador Intel 80386 (32 bits)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

- ❑ Registradores do processador Intel x86-64 ou AMD64 (64 bits)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Sistemas de numeração

O ENIAC usava o sistema de numeração decimal. Depois dele, todos os computadores eletrônicos usam em seus cálculos aritméticos **o sistema de numeração binário**.

#### ▪ Sistema decimal (base 10)

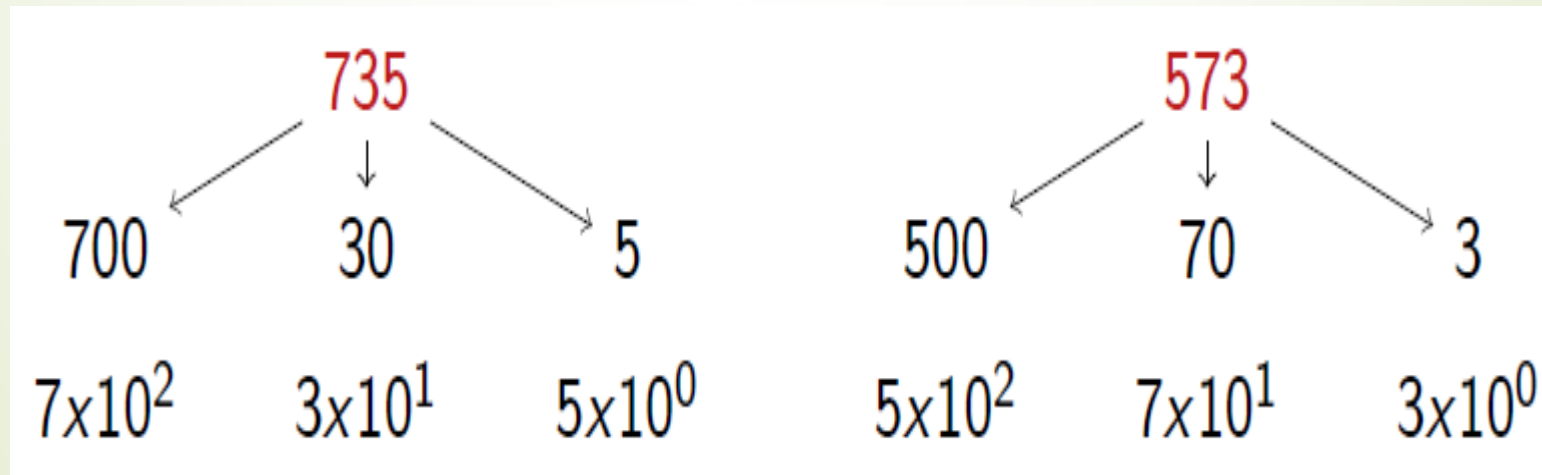
- Usa dez dígitos distintos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- É um sistema posicional
  - *Valor de um dígito depende da posição em que ele se encontra no conjunto de dígitos que representa uma quantidade*
  - *O valor total do número é a soma dos valores relativos de cada dígito*

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Sistemas de numeração

- Sistema decimal (base 10)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Sistema binário (base 2)

- Usa dois dígitos distintos (0, 1)
- Estrutura de pesos dos números binários:

$\dots 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0, \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \ 2^{-5} \dots$

- Conversão de binário para decimal
- Exemplo:  $(111001, 1)_2$

$$\begin{aligned} &= (1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1})_{10} \\ &= (32 + 16 + 8 + 1 + 0, 5)_{10} \\ &= (57, 5)_{10} \end{aligned}$$



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Conversão de decimal para binário

Exemplo:  $(57,3125)_{10} = (111001,0101)_2$

Parte inteira – Método das divisões sucessivas

$57 / 2 = 28$  com resto 1 -> bit menos significativo

$28 / 2 = 14$  com resto 0

$14 / 2 = 7$  com resto 0

$7 / 2 = 3$  com resto 1

$3 / 2 = 1$  com resto 1

$1 / 2 = 0$  com resto 1 -> bit mais significativo



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### Conversão de decimal para binário

Temos o número 111001.

Logo, temos que  $(57)_{10} = (111001)_2$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Conversão de decimal para binário

Exemplo:  $(57,3125)_{10} = (111001,0101)_2$

#### Parte fracionária – Método das multiplicações sucessivas

$0,3125 \times 2 = 0,625 \rightarrow$  bit mais significativo

$0,625 \times 2 = 1,25$

$0,25 \times 2 = 0,5$

$0,5 \times 2 = 1,0 \rightarrow$  bit menos significativo

Tomando-se os restos na ordem em que foram gerados, temos o número 0101

Logo, temos o resultado  $(0,3125)_{10} = (0,0101)_2$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ☐ Aritmética binário

#### Soma

$$0 + 0 = 0$$

$$0 + 1 = 1 + 0 = 1$$

$$1 + 1 = 10 = 0 \text{ vai } 1 \text{ para próxima posição}$$

$$1 + 1 + 1 = 11 = 1 \text{ vai } 1 \text{ para próxima posição}$$

$$\text{Exemplo: } 1111 + 11100 = 101011$$

$$\begin{array}{r} \phantom{+} 11 \\ \phantom{+} 1111 \\ + 11100 \\ \hline 101011 \end{array}$$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ☐ Aritmética binário

#### Subtração

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$10 - 1 = 1$$

$$\text{Exemplo: } 10001 - 1110 = 00011$$

		1	1		
	0	<del>10</del>	<del>10</del>	10	
	<del>1</del>	0	0	0	1
-		1	1	1	0
<hr/>					
	0	0	0	1	1



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Organização da memória de um computador

- A memória é organizada como “retângulos” de bits
- Cada retângulo é chamado de **palavra**
- Transferências de dados de/para a memória são feitas de 1 (ou mais) palavra(s) por vez
- Palavras na memória de um computador são numeradas consecutivamente, iniciando em 0; dizemos que esses números são os **endereços** das palavras
- Os endereços das palavras são usados pelos processadores, nas operações de transferência de dados de/para a memória
- Capacidade de uma memória = número de palavras x tamanho da palavra
- Computadores com processadores Intel usam palavras de 8 bits





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### Organização da memória de um computador

- 1 byte = 8 bits
- O número de palavras na memória de um computador geralmente é uma potência grande de 2, ou um múltiplo menor de uma dessas potências
- É conveniente o uso de símbolos/prefixos especiais para denotar essas potências:

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Organização da memória de um computador

	Valor Exato	Símbolo	Prefixo	Valor Aprox.
$2^{10}$	1 024	k	kilo	mil
$2^{20}$	1 048 576	M	mega	milhão
$2^{30}$	1 073 741 824	G	giga	bilhão
$2^{40}$	1 099 511 627 776	T	tera	trilhão

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Sistema hexadecimal (base 16)

- Usa 16 dígitos distintos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Estrutura de pesos dos números binários:

$\dots 16^5 \ 16^4 \ 16^3 \ 16^2 \ 16^1 \ 16^0, \ 16^{-1} \ 16^{-2} \ 16^{-3} \ 16^{-4} \ 16^{-5} \dots$

### Razões para aprendê-lo

- Endereços de memória são números muito grandes -> representação hexadecimal é mais “curta”
- Depuradores de código geralmente exibem os valores contidos nos registradores em hexadecimal; é útil sabermos verificar a aritmética de valores em hexadecimal sem a necessidade de convertê-los para a base 10

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Sistema hexadecimal (base 16)

Exemplo de conversão de hexadecimal para decimal

$$\begin{aligned}(14D)_{16} &= (1 \times 16^2 + 4 \times 16^1 + 13 \times 16^0)_{10} \\ &= (256 + 64 + 13)_{10} \\ &= (333)_{10}\end{aligned}$$

Exemplo de conversão de decimal para hexadecimal

$$1000 / 16 = 62 \text{ com resto } 8$$

$$62 / 16 = 3 \text{ com resto } 14 = E$$

$$3 / 16 = 0 \text{ com resto } 3$$

Tomando-se os restos na ordem inversa da que foram gerados, temos o número 3E8. Temos:  $(1000)_{10} = (3E8)_{16}$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Relação entre a base binária e a base hexadecimal

- Dividir por 2 quatro vezes equivale a dividir por 16 uma vez
- Se agruparmos os dígitos do número binário quatro a quatro, veremos a seguinte relação:

Número na base decimal: 1000

Número na base binária: 11 1110 1000

Número na base hexadecimal: 3 E 8

- Assim, podemos usar o sistema hexadecimal como uma forma “mais legível” do binário
- Com dois dígitos em hexadecimal representamos 1 byte
- Outro exemplo:

Binário : 1011 0010 1001 0101 0000 0111 1010 1000 1000

Hexadecimal : B 2 9 5 0 7 A 8 8

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Aritmética no hexadecimal

Exemplo:  $47BC + A78 = 5234$

	1	1	1	
	4	7	B	C
+		A	7	8
<hr/>				
	5	2	3	4

“Colinha”:

$$(C + 8)_{16} = (12 + 8)_{10} = (20)_{10} = (14)_{16}$$

$$(1 + B + 7)_{16} = (1 + 11 + 7)_{10} = (19)_{10} = (13)_{16}$$

$$(1 + 7 + A)_{16} = (1 + 7 + 10)_{10} = (18)_{10} = (12)_{16}$$

$$(1 + 4)_{16} = (5)_{16}$$



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Aritmética hexadecimal

Exemplo:  $47BC - A4E = 3D6E$

	3	17	A	1C
	<del>4</del>	<del>7</del>	<del>B</del>	<del>C</del>
-		A	4	E
<hr/>				
	3	D	6	E

“Colinha”:

$$\begin{aligned}(1C - E)_{16} &= (28 - 14)_{10} = (14)_{10} = (E)_{16} \\(A - 4)_{16} &= (10 - 4)_{10} = (6)_{10} = (6)_{16} \\(17 - A)_{16} &= (23 - 10)_{10} = (13)_{10} = (D)_{16} \\(3 - 0)_{16} &= (3)_{16}\end{aligned}$$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Números negativos em binário

#### Representação sinal-e-magnitude

- **Bit mais significativo** representa o sinal do número
- 0 – número positivo
- 1 – número negativo

Exemplo:  $(0101)_2 = (5)_{10}$  e  $(1101)_2 = (-5)_{10}$

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Números negativos em binário

Problema: a soma fica complicada para o computador

“Algoritmo” para a soma:

- Caso 1 – os dois números são positivos: basta somá-los
- Caso 2 – os dois números são negativos: remova os sinais dos números, some-os e depois coloque o sinal de menos no resultado
- Caso 3 – um número é positivo e outro negativo: subtraia o de menor magnitude do de maior; se o de maior magnitude tem um sinal de menos, então coloque o sinal de menos no resultado

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Números negativos em binário

Complemento de 2

- Usada nos computadores
- Facilita a soma: não é preciso se preocupar se o número é positivo ou negativo... basta somá-los
- Funcionamento “análogo” ao do odômetro

Exemplo:  $(4 + (-7))_{10} = (0100 + 1001)_2 = (1101)_2 = (-3)_{10}$

Decimal	Binário (4 bits) em Complemento de 2
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Números negativos em binário

#### Conversão de binário “puro” para Complemento de 2

- Passo 1: inverter os bits (= trocar zeros por uns e uns por zeros)
- Passo 2: somar 1 ao número resultante da inversão

Obs.: os mesmos passos valem para converter de complemento de 2 para binário puro.

Exemplos:

Decimal	Binário puro	Complemento de 2 (8 bits)
-108	-01101100	10010100

$$(108)_{10} = (01101100)_2$$

Depois da inversão: 10010011

Depois de + 1 : 10010100

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Números negativos em binário

Exemplo: soma/subtração com complemento de dois

$$(109 - 108)_{10} = (109 + (-108))_{10}$$

$$(109)_{10} = 01101101$$

$$(-108)_{10} = 10010100$$

$$\begin{array}{r} \phantom{01101101} \\ + \phantom{01101101} 01101101 \\ \phantom{01101101} 10010100 \\ \hline \cancel{1} \phantom{0000000} 00000001 \end{array}$$

bit de  
"carry"



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

❑ **Lembrete: arquitetura da família x86**

### Registradores de propósito geral

- A (acumulador)
- B (base)
- C (contador)
- D (dados)
- processador 8086 (16 bits): AX (AH,AL), BX (BH,BL), CX (CH,CL), DX (DH,DL), SP, BP, SI, DI
- processador 80386 (32 bits): EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI
- processador Intel x86-64 e AMD64 (64 bits): RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8–15



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Estrutura geral das instruções

Cada linha de um programa em linguagem de montagem é composto por 4 campos:

- **rótulo (label )**: “nomeia” os blocos do programa. São usados nos saltos. Devem ser alfanuméricos começando por letras
- **mnemônico**: especifica uma instrução (ex.: MOV, ADD, ...)



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Estrutura geral das instruções

- **operando(s)**: objeto(s) sobre o qual(is) a instrução opera. Quando uma instrução possui mais de um operando, eles devem vir separados por vírgulas. Nem toda instrução tem um operando
- **comentário**: documenta o código. É iniciado por um ponto-e-vírgula. É permitido que uma linha tenha somente o campo de comentário. (Obs.: comentários são particularmente importantes em linguagem de montagem!)

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Linguagem de montagem

Exemplo de programa

[Rótulo: ]	[Mnemônico]	[Operando]	[;Comentário]
	MOV	CX, 5	; inicializa contador com 5
inicio:	MOV	AX, 25h	; inicializa AX com 25h
	ADD	AX,AX	; $AX \leftarrow AX + AX$
	DEC	CX	; contador $\leftarrow$ contador - 1
	JNZ	inicio	

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Comando para transferência de dados: MOV

Copia o valor do segundo operando no primeiro operando. O conteúdo do segundo operando permanece inalterado.

#### Formatos

- **MOV reg, reg |mem| const**
- **MOV mem, reg |const**

#### Operandos

- **reg** – um registrador de propósito geral
- **mem** – posição de memória (pode ser indicada por meio de uma constante, como [1000], ou por meio de um registrador, como [EBX])
- **const** – valor constante

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Comando para transferência de dados: MOV

#### Exemplos

Correto	Incorreto	Problema
MOV AH,-14	MOV AL,999	; 999 não cabe em 8 bits
MOV AX,36H		
MOV AL,'A'	MOV EBX,DX	; não possuem o mesmo ; tamanho
MOV EAX,EBX		
MOV BX,1000		
MOV AX,[EBX]		
MOV AX,[1000]		
MOV AX,[1000+EBX]		
MOV [1000],AX	MOV [1000],[EBX]	; não há MOV direto ; entre memórias
MOV [1000],36H		





# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Considerações sobre o uso de memória como operando

#### Casos de não ambiguidade no tamanho

Acontecem quando a instrução envolve um operando do tipo **mem** e outro do tipo **reg**.

Neste caso, o número de palavras manipuladas na memória é determinado pelo tamanho de reg.

**Exemplo:** a instrução

MOV AX, [1000]

copia 2 palavras da memória (posições 1000 e 1001) porque o registrador AX é de 16 bits



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Considerações sobre o uso de memória como operando

#### Casos de ambiguidade no tamanho

Acontecem quando a instrução envolve um operando do tipo **mem** e outro do tipo **const**.

Exemplo: `MOV [EBX], 5`

Neste caso, o número de palavras manipuladas na memória pode ser determinado de duas maneiras:

- A arquitetura do processador determina a quantidade de bits default (16 bits, 32 bits, 64 bits)

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Considerações sobre o uso de memória como operando

#### Casos de ambiguidade no tamanho

- Uso de notação para determinar o quantidade de bytes manipulados.

#### Exemplo:

MOV BYTE [EBX],5 ; BYTE para designar 8 bits

MOV WORD [EBX],5 ; WORD para designar 16 bits

MOV DWORD [EBX],5 ; DWORD para designar 32 bits

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Um “parênteses”: Convenções de notação

#### Soluções para problemas de ambiguidade

- Problema-exemplo 1: **50** pode ser um número em notação decimal ou hexadecimal
- Solução: usar sufixos que determinam o sistema de numeração. Por exemplo, **50D** designa um número decimal, enquanto **50H** é hexadecimal (**10B** é binário)
- Problema-exemplo 2 (consequência da solução anterior): **AH**, **BH**, **CH** e **DH** designam números hexadecimais, mas também são nomes de registradores
- Solução: na linguagem de montagem, fazer com que todos os números hexadecimais sejam também iniciados por um dígito em 0; 1; : : : ; 9<sup>1</sup>. Por exemplo, **0AH** designa o número hexadecimal **A** e não o registrador **AH**

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Comando para troca de dados: XCHG

Troca os valores dos operandos (ou seja, faz o primeiro receber o valor do segundo e o segundo receber o valor do primeiro). Os operandos precisam ser do mesmo tamanho.

#### Formatos

- XCHG reg,reg|mem
- XCHG mem,reg

#### Exemplos

XCHG	AH,BL
XCHG	AH,[BL]
XCHG	[EBX], AH

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Instruções aritméticas – soma: ADD

Soma o valor do segundo operando ao valor do primeiro, armazenando o resultado no primeiro operando. O valor do segundo operando permanece inalterado.

#### Formato

- ADD reg,reg|mem|const

#### Exemplos

ADD BL,10 ; BL  $\leftarrow$  BL + 10

ADD BL,AL ; BL  $\leftarrow$  BL + AL

ADD BL,[1000] ; BL  $\leftarrow$  BL + [1000]



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Instruções aritméticas – subtração: **SUB**

Subtrai o valor do segundo operando do valor do primeiro, armazenando o resultado no primeiro operando. O valor do segundo operando permanece inalterado

#### Formato

- **SUB** reg,reg|mem|const

#### Exemplos

SUB BL,10 ; BL ← BL - 10

SUB BL,AL ; BL ← BL - AL

SUB BL,[1000] ; BL ← BL - [1000]

# Programação de Software Básico

## Arquitetura dos processadores Intel x86

### ❑ Instruções aritméticas – Incremento e Decremento: INC e DEC

Incrementa ou decrementa o valor do operando em 1

#### Formato

- INC reg|mem
- DEC reg|mem

#### Exemplos

INC CX ! ADD CX, 1

DEC CX ! SUB CX, 1

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – multiplicação: MUL

#### Formato

Não tem o mesmo formato que as operações aritméticas anteriores porque a multiplicação pode gerar um número que tem até o dobro de bits que os operandos.

**MUL é válida apenas para a multiplicação de números sem sinal.**

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – multiplicação: MUL

Formato

#### ■ MUL reg|mem

Se o operando tem 8 bits, por exemplo,

MUL BH

então o comando equivale a

$AX \leftarrow AL \times BH$

Ou seja, o operando é sempre multiplicado pelo valor em AL e o resultado é armazenado em AX.

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – multiplicação: MUL

#### Formato

#### ■ MUL reg|mem

Se o operando tem 16 bits, por exemplo,

MUL BX

então o comando equivale a

$DX:AX \leftarrow AX \times BX$

Ou seja, o operando é sempre multiplicado pelo valor em AX e o resultado de 32 bits é armazenado em 2 registradores de 16 bits: os 16 primeiros bits em AX e os 16 últimos em DX.

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – multiplicação: MUL

#### Formato

- MUL reg|mem

Se o operando tem 32 bits, por exemplo,

MUL EBX

então o comando equivale a

$EDX:EAX \leftarrow EAX \cdot EBX$

Ou seja, o operando é sempre multiplicado pelo valor em EAX e o resultado de 64 bits é armazenado em 2 registradores de 32 bits: os 32 primeiros bits em EAX e os 32 últimos em EDX.





# Programação de Software Básico

## Linguagem assembly

### Instruções aritméticas – multiplicação: MUL

**Obs.:** O MUL não pode ser usado com um valor constante. Por exemplo, o comando a seguir é inválido: MOV 7

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – divisão inteira: DIV

#### Formato

Funciona de forma inversa ao MUL.

**DIV é válida apenas para a divisão de números inteiros sem sinal.**

- DIV reg|mem

Por exemplo,

- DIV BH

divide o valor em AX pelo valor em BH, armazenando o quociente em AL e o resto em AH

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – divisão inteira: DIV

#### Formato

Divisor	Dividendo	Resto	Quociente
32 bits	EDX:EAX	EDX	EAX
16 bits	DX:AX	DX	AX
8 bits	AX	AH	AL

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções aritméticas – divisão inteira: DIV

Situações que geram exceção:

- Divisão por zero
- Transbordamento (**overflow**) – ocorre quando o resto gerado na divisão não cabe no registrador.

Exemplo: MOV AX,1024

MOV BH,2

DIV BH

Quociente deveria ser armazenado em AL, mas 512 ocupa no mínimo 10 bits!



# Programação de Software Básico

## Linguagem assembly

- ❑ Instruções aritméticas – divisão e multiplicação envolvendo números com sinal: IMUL e IDIV

Funcionam de modo análogo aos comandos DIV e MUL, mas podem ser **aplicados a números com sinal**.

### Formato

- IMUL reg|mem
- IDIV reg|mem

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções lógicas: AND, OR, NOT

O resultado é armazenado no primeiro operando

#### Formato

- AND **reg,reg|mem|const** ou AND mem,reg|const
- OR **reg,reg|mem|const** ou OR mem,reg|const
- XOR **reg,reg|mem|const** ou XOR mem,reg|const
- NOT **reg|mem**; inverte os bits



# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções lógicas: AND, OR, NOT

O resultado é armazenado no primeiro operando

#### Formato

AND	0	1	OR	0	1	XOR	0	1	NOT	0	1
0	0	0	0	0	1	0	0	1		1	0
1	0	1	1	1	1	1	1	0			

#### Exemplos

AND AX,BX    |    OR CX,5Fh    |    NOT AX



# Programação de Software Básico

## Linguagem assembly

### ❑ “Truques” com números binários

As operações lógicas podem ser usadas para:

- “resetar”/limpar (= atribuir zero a) bits
- “setar” (= atribuir 1 a) bits
- inverter bits
- examinar bits

# Programação de Software Básico

## Linguagem assembly

### ❑ “Truques” com números binários

Para “setar” um bit

**Exemplo:** setar o 3o bit menos significativo do AH.

OR AH, 00000100B

Para “resetar” um bit

**Exemplo:** resetar o 3o bit menos significativo do AH.

AND AH, 11111011B

# Programação de Software Básico

## Linguagem assembly

### ❑ “Truques” com números binários

Para inverter bits específicos

**Exemplo:** Inverter o quarto bit mais significativo do AX.

```
XOR AX,1000H
```

Para examinar bits específicos

**Exemplo:** determinar o valor do quarto bit mais significativo do AX.

```
AND AX,1000H
```

Se o resultado da operação for zero, o bit desejado vale 0. Senão, o bit vale 1.

# Programação de Software Básico

## Linguagem assembly

### ❑ “Truques” com números binários

Para zerar um registrador

**Exemplo:** zerar o registrador ECX.

```
XOR ECX,ECX
```

Para verificar se um registrador é nulo

**Exemplo:** verificar se ECX é nulo.

```
OR ECX,ECX
```

Obs.: se o registrador for nulo, então a flag zero é setada

# Programação de Software Básico

## Linguagem assembly

### ❑ Instrução para trocar sinal – NEG

Gera o Complemento 2 do operando e armazena-o no próprio operando (ou seja, troca o sinal do operando).

#### Formato

- **NEG reg|mem**

#### Exemplo

NEG EAX <-> NOT EAX, ADD EAX,1



# Programação de Software Básico

## Linguagem assembly

### ❑ Instrução para trocar sinal – NEG

Gera o Complemento 2 do operando e armazena-o no próprio operando (ou seja, troca o sinal do operando).

#### Formato

- **NEG reg|mem**

#### Exemplo

NEG EAX <-> NOT EAX, ADD EAX,1

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para a transferência de controle Salto incondicional – **JMP**

Transfere a execução para o endereço especificado pelo rótulo

**Formato:** **JMP** *rot*

Exemplo de programa

...

*inicio*: MOV AX,5

ADD AX,AX

...

**JMP** *inicio*

# Programação de Software Básico

## Linguagem assembly

### ❑ Instrução para comparação – **CMP**

Compara o valor do primeiro operando com o valor do segundo.

Formato:

**CMP** reg,reg | mem | const

Resultado da comparação é armazenado em uma **flag**

### Exemplos

**CMP** AX,5

**CMP** CX,[EBX]



# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais

#### Variações:

- JE – jump if equal (salta se é igual)
- JNE – jump if not equal (salta se não é igual)
- JG – jump if greater (salta se é maior)
- JGE – jump if greater or equal (salta se é maior ou igual)
- JNG – jump if not greater (salta se não é maior)
- JNGE – jump if not greater or equal (salta se não é maior ou igual)

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais

#### Variações:

- JL – jump if less (salta se é menor)
- JLE – jump if less or equal (salta se é menor ou igual)
- JNL – jump if not less (salta se não é menor)
- JNLE – jump if less or equal (salta se não é menor ou igual)

*Esses saltos consideram o resultado da última comparação realizada.*

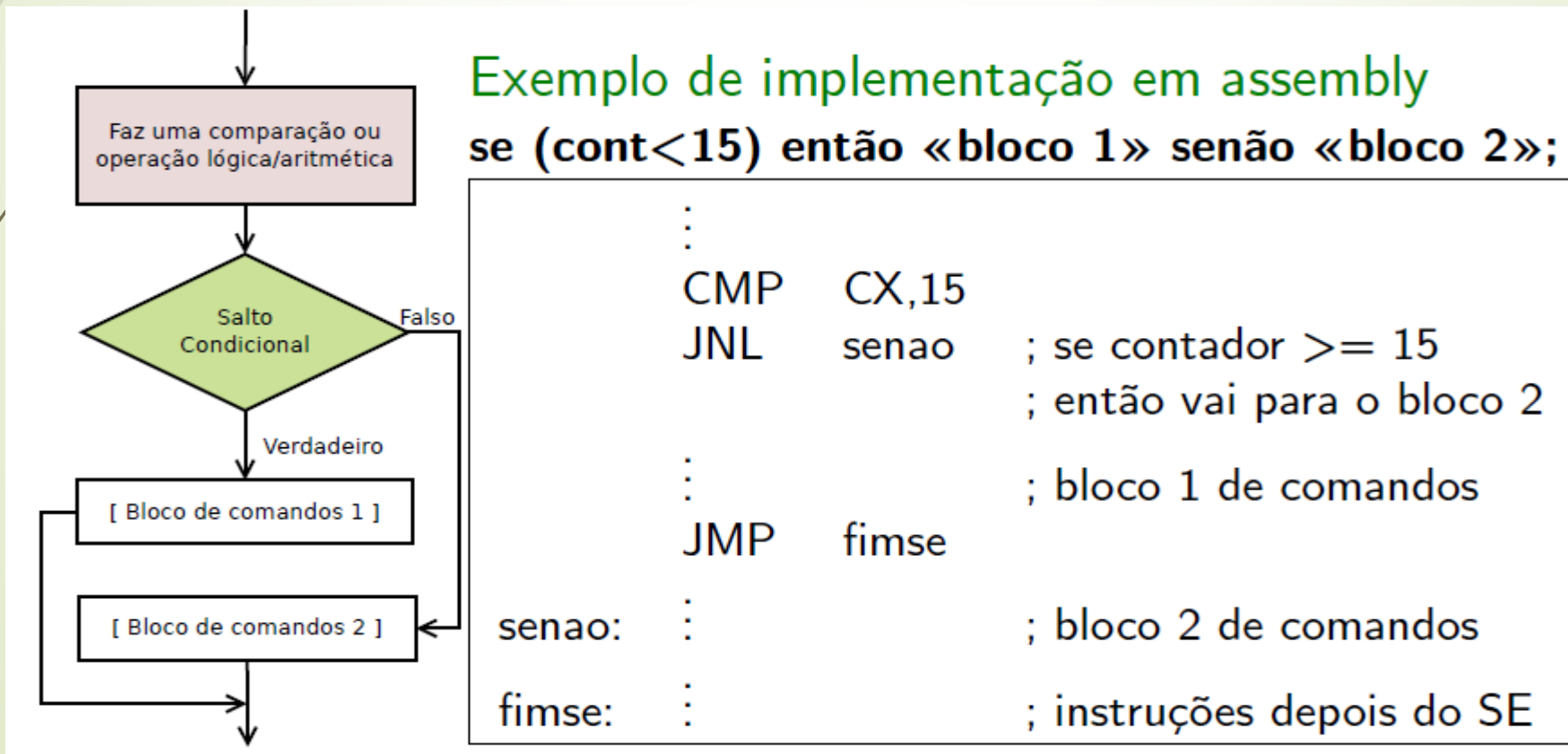
**Importante:** *esses saltos consideram que a comparação envolveu números com sinal (signed)*

# Programação de Software Básico

## Linguagem assembly

### ❑ Estrutura de um comando “if-else”

se ( expressão ) então «bloco 1» senão «bloco 2» fim;



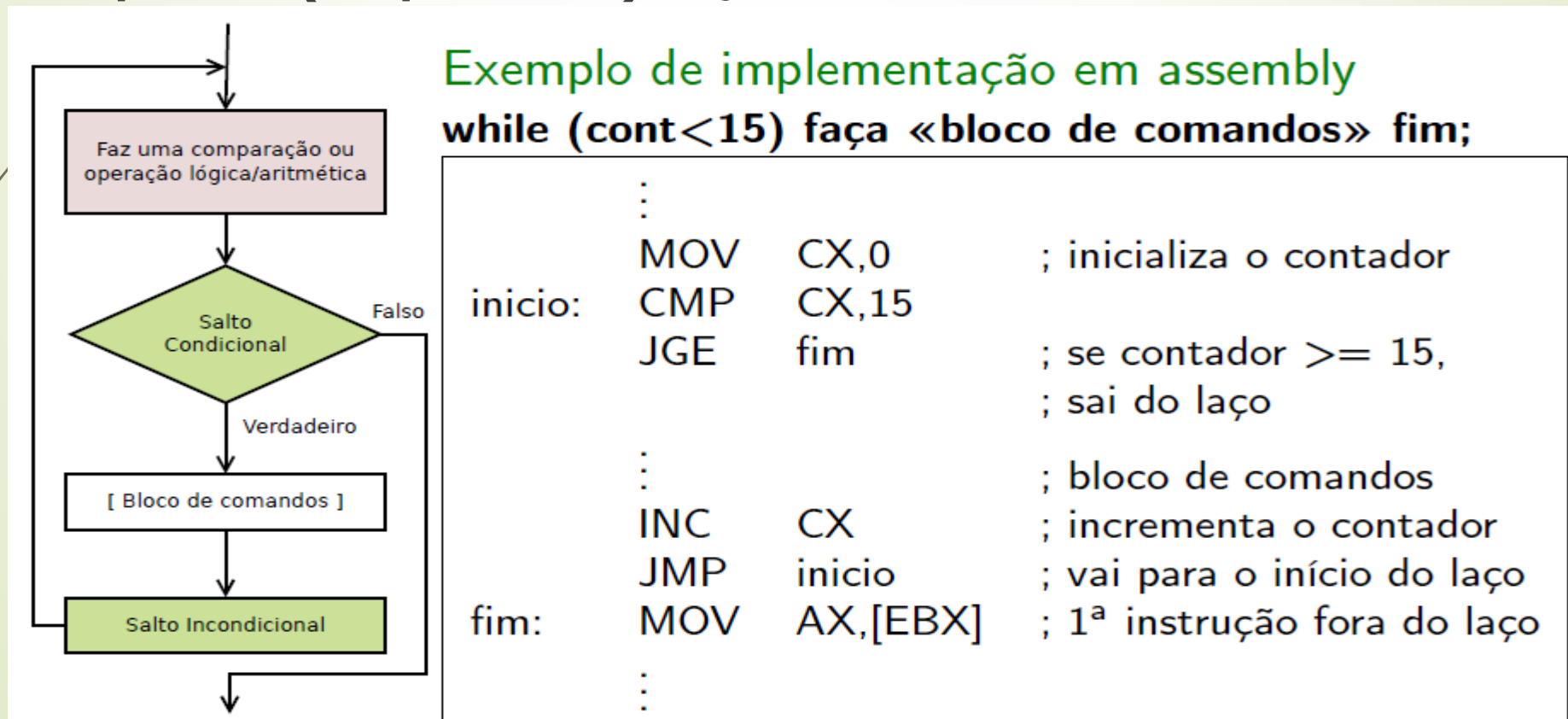


# Programação de Software Básico

## Linguagem assembly

### ❑ Estrutura de um comando “while”

enquanto ( expressão ) faça «bloco de comandos» fim;



# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais – JZ e JNZ

#### Variações:

- JZ – jump if zero (salta se é nulo)
- JNZ – jump if not zero (salta se não é nulo)

Esses saltos consideram o resultado da última operação aritmética ou lógica realizada

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais – JZ e JNZ

#### Exemplo de programa

**MOV CX, 5** ; laço será executado 5 vezes

início:

.....

; bloco de comandos do laço

**DEC CX** ; contador  $\leftarrow$  contador - 1

**JNZ início**

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais (versão unsigned)

Esses saltos consideram o resultado da última comparação realizada. Consideram também que a comparação envolveu números sem sinal (unsigned).

Variações:

I JA – jump if above (salta se é maior)

I JAE – jump if above or equal (salta se é maior ou igual)

I JNA – jump if not above (salta se não é maior)

I JNAE – jump if not above or equal (salta se não é maior ou igual)

# Programação de Software Básico

## Linguagem assembly

### ❑ Instruções para saltos condicionais (versão unsigned)

#### Variações:

- JB – jump if below (salta se é menor)
- JBE – jump if below or equal (salta se é menor ou igual)
- JNB – jump if not below (salta se não é menor)
- JNBE – jump if below or equal (salta se não é menor ou igual)



# Programação de Software Básico

## Linguagem assembly

### Chamadas ao sistema operacional

Chamadas ao sistema (= system calls, ou somente **syscalls**)

- Forma por meio da qual programas solicitam serviços ao núcleo do SO
- **Exemplos de serviços:** operações para leitura e escrita em arquivos, criação e execução de novos processos, etc.



# Programação de Software Básico

## Linguagem assembly

### ❑ Chamadas ao sistema operacional

Chamadas ao sistema – como fazê-las em assembly?

- colocar número da chamada ao sistema em EAX
- colocar 3 primeiros argumentos em EBX, ECX, EDX (mais ESI e EDI se necessário)
- gerar a interrupção de chamada ao sistema (instrução INT 0x80)
- quando há valor de retorno, ele é colocado em EAX



# Programação de Software Básico

## Linguagem assembly

### Montadores

#### GCC Inline Assembly

- Suporte à arquitetura x86 bastante satisfatório
- Possibilita que código em linguagem de máquina seja inserido em programas em C

# Programação de Software Básico

## Linguagem assembly

### ❑ Montadores

Usa o GAS

GAS – GNU Assembler

- Por padrão, segue a sintaxe da AT&T (e não a da Intel, usada pela maioria dos montadores). Mas, em suas versões mais novas, aceita também a sintaxe da Intel
- Plataformas: Unix-like, Windows, DOS, OS/2
- Parte do pacote binutils do Linux
- Nome do executável: gas ou simplesmente as

# Programação de Software Básico

## Linguagem assembly

### ❑ Montadores

#### NASM – Netwide Assembler

- Bastante usado (confiável para o desenvolvimento de aplicações de grande porte, de uso comercial e industrial)
- Plataformas: Windows, Linux, Mac OS X, DOS, OS/2
- Instalação: pacote nasm do Linux:

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



# Programação de Software Básico

## Arquitetura dos processadores Intel x86

Dúvidas!!!

