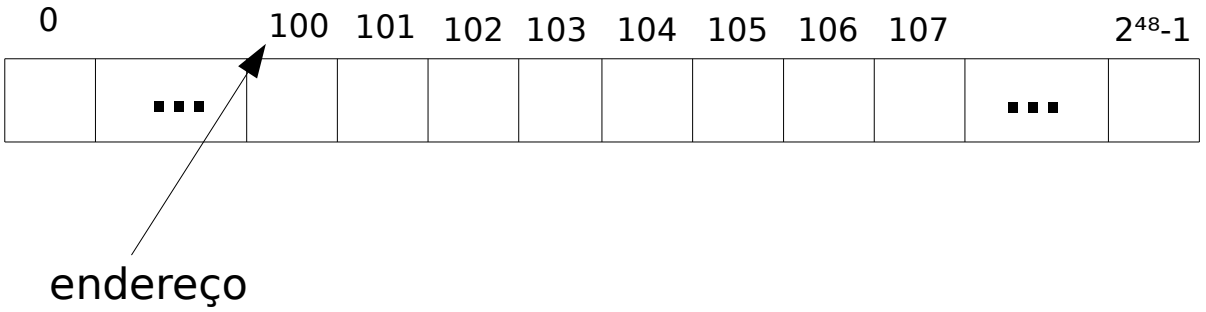
# Representação de Dados (inteiros não negativos)

Noemi Rodriguez
Ana Lúcia de Moura

http://www.inf.puc-rio.br/~inf1018

# Memória

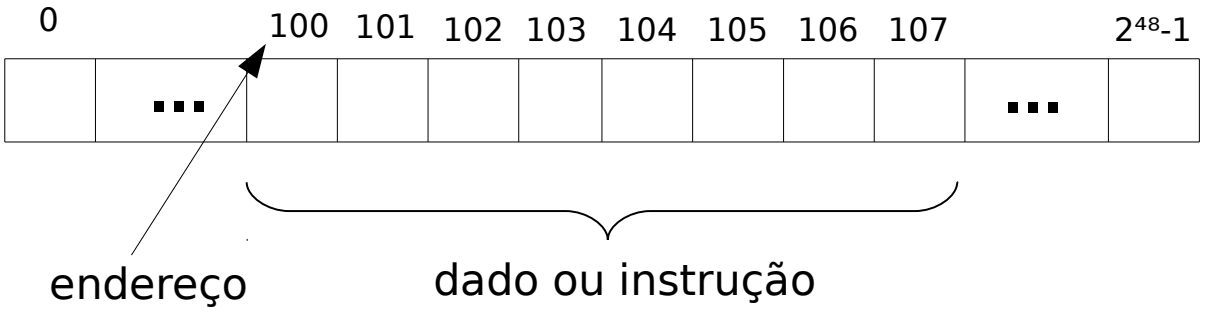
Pode ser vista como um *array* de bytes, identificados por seus "índices" (**endereços**)



# Memória

Pode ser vista como um *array* de bytes, identificados por seus "índices" (**endereços**)

Armazena dados e instruções

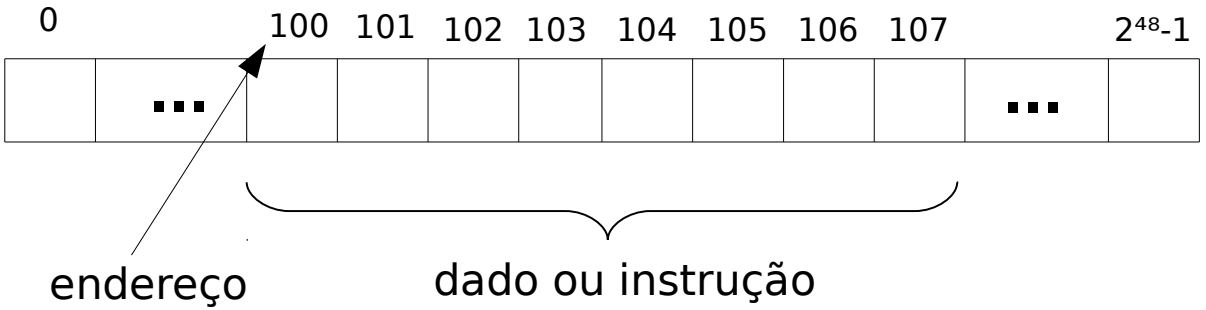


# Memória

Pode ser vista como um *array* de bytes, identificados por seus "índices" (**endereços**)

Armazena dados e instruções

- dados ocupam um número de bytes que depende de seu tipo

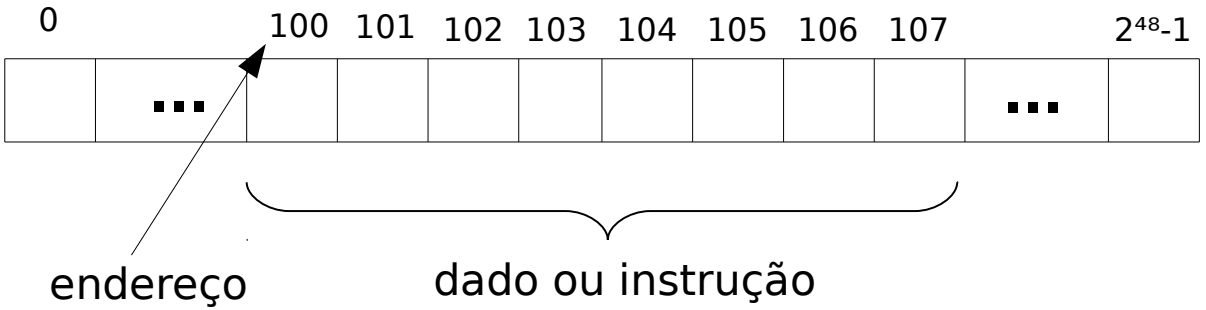


# Memória

Pode ser vista como um *array* de bytes, identificados por seus "índices" (**endereços**)

Armazena dados e instruções

- dados ocupam um número de bytes que depende de seu tipo
- instruções ocupam um número variável de bytes



# Representação da Informação

Computadores armazenam "sinais" de dois valores: 0 e 1

- binary digits ou "bits"

Agrupando sequências de bits podemos representar valores numéricos

- representação em notação posicional (base 2)

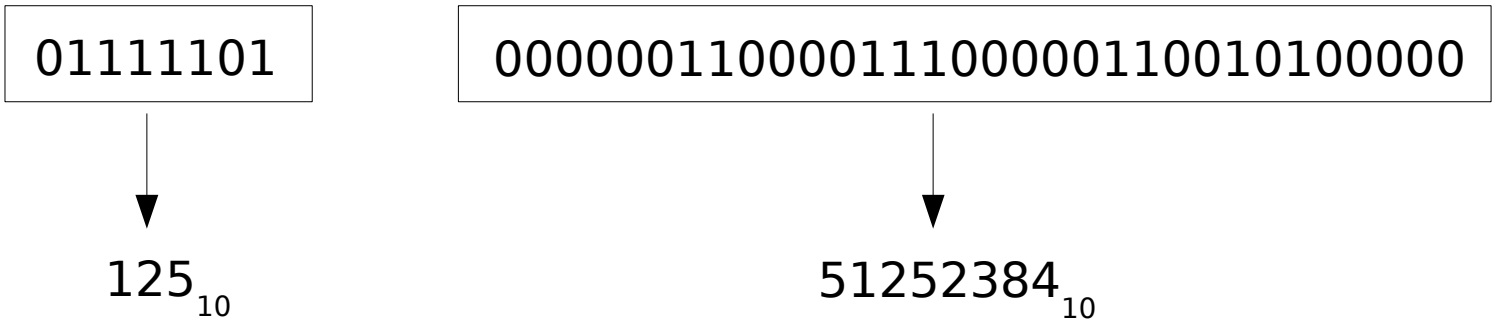
# Representação da Informação

Computadores armazenam "sinais" de dois valores: 0 e 1

- binary digits ou "bits"

Agrupando sequências de bits podemos representar valores numéricos

- representação em notação posicional (base 2)



# Notação Posicional

A base determina o número de dígitos

- sistema decimal: base 10 e dígitos de 0 a 9

Multiplicamos o "valor" de cada dígito pela base elevada à posição deste dígito e somamos os produtos

# Notação Posicional

A base determina o número de dígitos

- sistema decimal: base 10 e dígitos de 0 a 9

Multiplicamos o "valor" de cada dígito pela base elevada à posição deste dígito e somamos os produtos

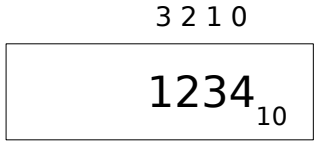
$1234_{10}$

# Notação Posicional

A base determina o número de dígitos

- sistema decimal: base 10 e dígitos de 0 a 9

Multiplicamos o "valor" de cada dígito pela base elevada à posição deste dígito e somamos os produtos

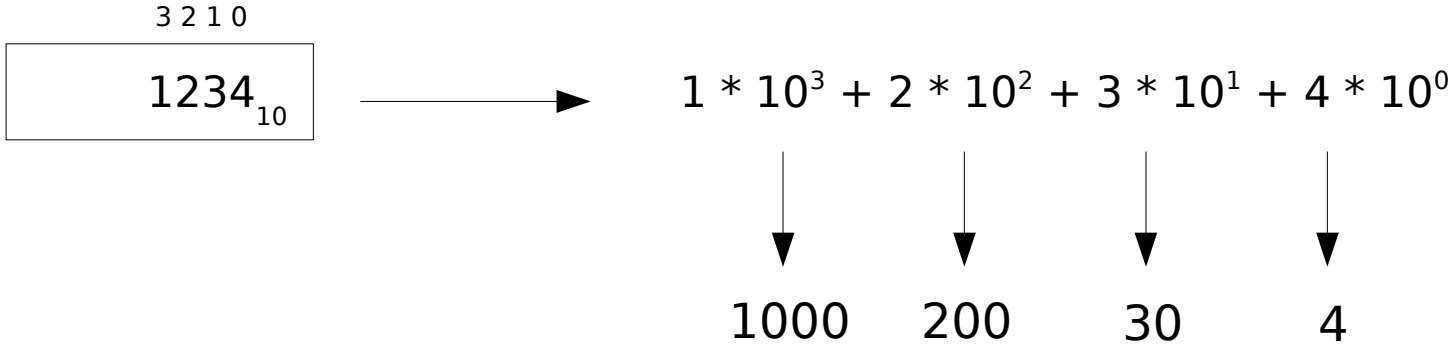


# Notação Posicional

A base determina o número de dígitos

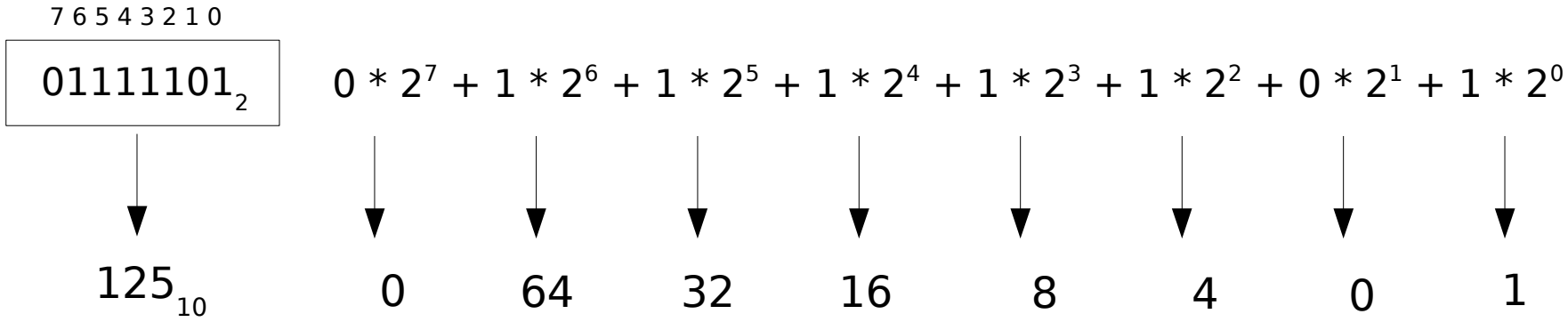
- sistema decimal: base 10 e dígitos de 0 a 9

Multiplicamos o "valor" de cada dígito pela base elevada à posição deste dígito e somamos os produtos



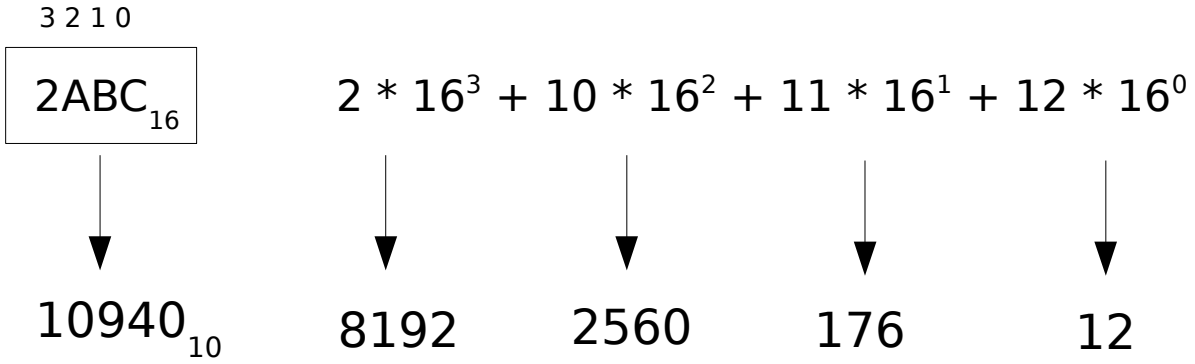
# Notação Binária

Base 2, dígitos 0 e 1



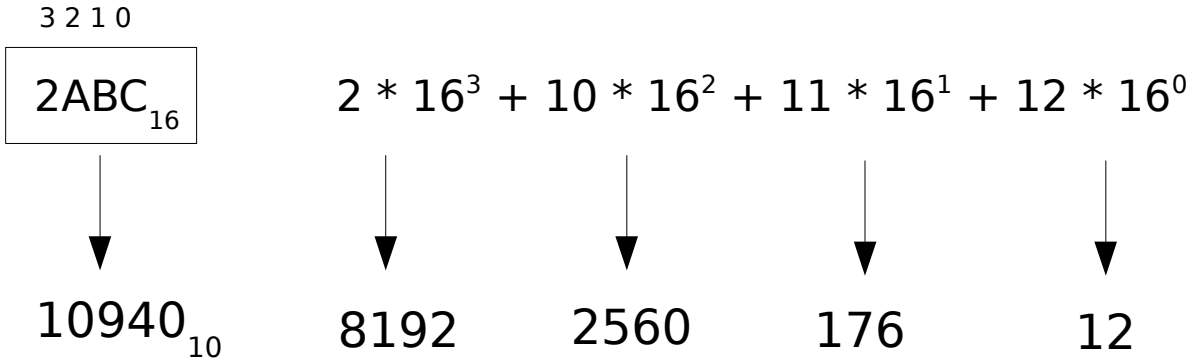
# Notação Hexadecimal

Base 16, dígitos de 0 a 9 e letras de A a F



# Notação Hexadecimal

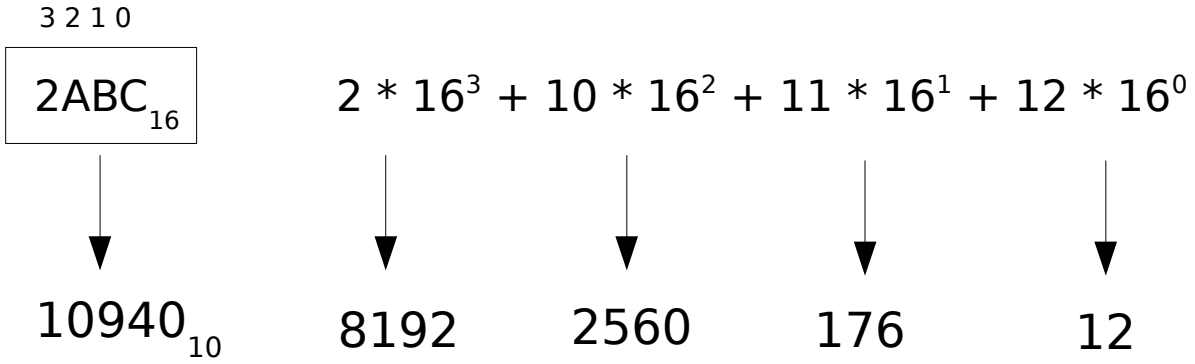
Base 16, dígitos de 0 a 9 e letras de A a F




Notações decimal e binária são inconvenientes para descrever padrões de bits e representar endereços

# Notação Hexadecimal

Base 16, dígitos de 0 a 9 e letras de A a F




Notações decimal e binária são inconvenientes para descrever padrões de bits e representar endereços

Em C (e *assembly*) constantes que começam com **0x** estão em notação hexadecimal: **0x10**, **0xFF**, **0x55aa**

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| | | | | |
|---|---|---|---|---|
| hexa | 3 | A | 4 | C |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| | | | | |
|---|---|---|---|---|
| hexa | 3 | A | 4 | C |
| binário | 0011 | | | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| | | | | |
|---|---|---|---|---|
| hexa | 3 | A | 4 | C |
| binário | 0011 | 1010 | | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| | | | | |
|---|---|---|---|---|
| hexa | 3 | A | 4 | C |
| binário | 0011 | 1010 | 0100 | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

Binário para hexa: substituimos cada grupo de 4 bits pelo dígito hexadecimal equivalente:

| binário | (00)11 | 1100 | 1010 | 1101 |
|---|---|---|---|---|

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

Binário para hexa: substituimos cada grupo de 4 bits pelo dígito hexadecimal equivalente:

| binário | (00)11 | 1100 | 1010 | 1101 |
|---|---|---|---|---|
| hexa | 3 | | | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

Binário para hexa: substituimos cada grupo de 4 bits pelo dígito hexadecimal equivalente:

| binário | (00)11 | 1100 | 1010 | 1101 |
|---|---|---|---|---|
| hexa | 3 | C | | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

Binário para hexa: substituimos cada grupo de 4 bits pelo dígito hexadecimal equivalente:

| binário | (00)11 | 1100 | 1010 | 1101 |
|---|---|---|---|---|
| hexa | 3 | C | A | |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Conversão Binário X Hexadecimal

Hexa para binário: "expandimos" cada dígito hexadecimal:

| hexa | 3 | A | 4 | C |
|---|---|---|---|---|
| binário | 0011 | 1010 | 0100 | 1100 |

Binário para hexa: substituimos cada grupo de 4 bits pelo dígito hexadecimal equivalente:

| binário | (00)11 | 1100 | 1010 | 1101 |
|---|---|---|---|---|
| hexa | 3 | C | A | D |

| Hex | Decimal | Binário |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

$11_{10}$

| | |
|---|---|
| 11 | 2 |
| 1 | 5 |

# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

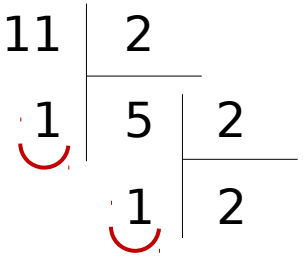
$11_{10} = 5 * 2^1 + 1 * 2^0$

# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

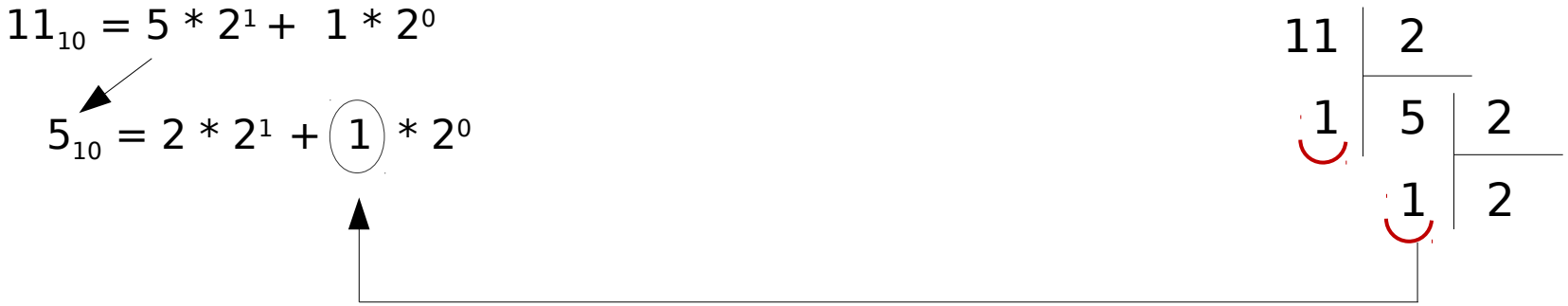
$11_{10} = 5 * 2^1 + 1 * 2^0$



# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição



# Decimal para Binário

## Divisões sucessivas por 2 (base)

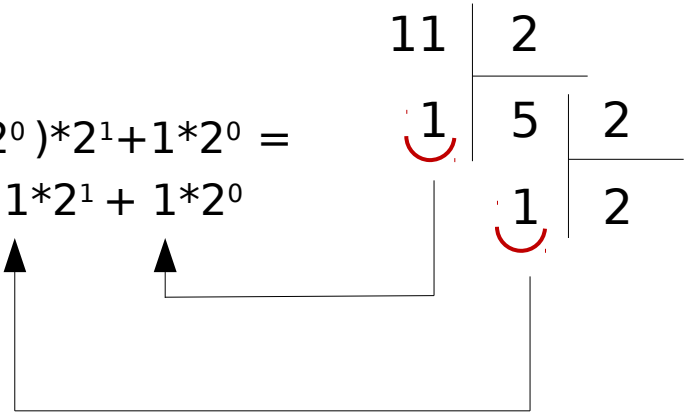
relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

$11_{10} = 5 * 2^1 + 1 * 2^0$

$5_{10} = 2 * 2^1 + 1 * 2^0$

$11_{10} = (2*2^1+1*2^0)*2^1+1*2^0 =$

$2*2^2 + 1*2^1 + 1*2^0$



# Decimal para Binário

## Divisões sucessivas por 2 (base)

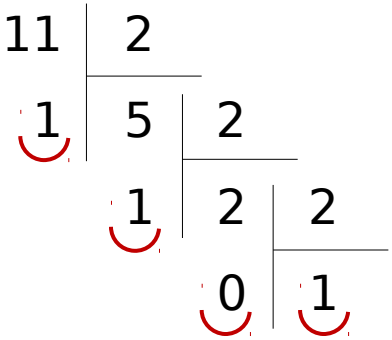
relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

$11_{10} = 5 * 2^1 + 1 * 2^0$

$5_{10} = 2 * 2^1 + 1 * 2^0$

$11_{10} = (2*2^1+1*2^0)*2^1+1*2^0 =$

$2*2^2 + 1*2^1 + 1*2^0$



# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

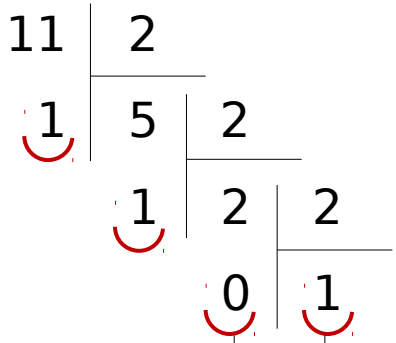
$11_{10} = 5 * 2^1 + 1 * 2^0$

$5_{10} = 2 * 2^1 + 1 * 2^0$

$2_{10} = (1) * 2^1 + (0) * 2^0$

$11_{10} = (2*2^1+1*2^0)*2^1+1*2^0 =$

$2*2^2 + 1*2^1 + 1*2^0$



# Decimal para Binário

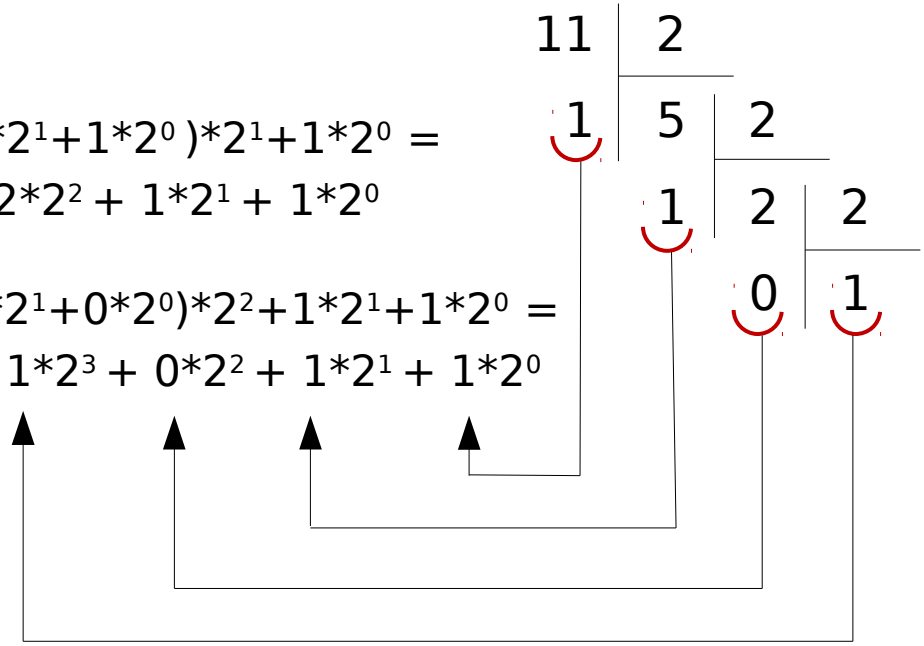
## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

$11_{10} = 5 * 2^1 + 1 * 2^0$

$5_{10} = 2 * 2^1 + 1 * 2^0$ $\quad 11_{10} = (2*2^1+1*2^0)*2^1+1*2^0 = 2*2^2 + 1*2^1 + 1*2^0$

$2_{10} = 1 * 2^1 + 0 * 2^0$ $\quad 11_{10} = (1*2^1+0*2^0)*2^2+1*2^1+1*2^0 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$



# Decimal para Binário

## Divisões sucessivas por 2 (base)

relação com a notação posicional: "parcelas" são multiplicações de 0 ou 1 (restos) pela base elevada à posição

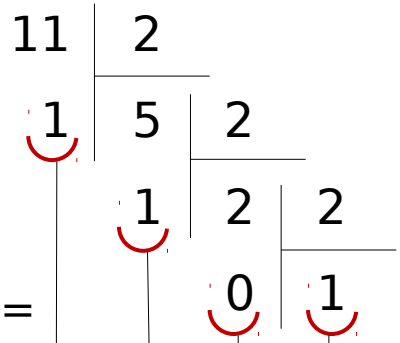
$11_{10} = 5 * 2^1 + 1 * 2^0$

$5_{10} = 2 * 2^1 + 1 * 2^0$ $\quad 11_{10} = (2*2^1+1*2^0)*2^1+1*2^0 =$

$2*2^2 + 1*2^1 + 1*2^0$

$2_{10} = 1 * 2^1 + 0 * 2^0$ $\quad 11_{10} = (1*2^1+0*2^0)*2^2+1*2^1+1*2^0 =$

$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

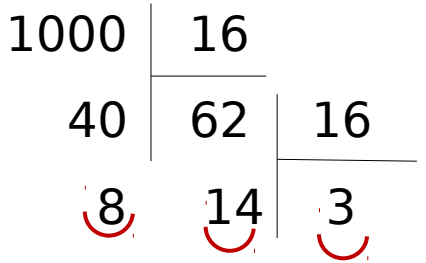



**$1011_2$**

# Decimal para Hexadecimal

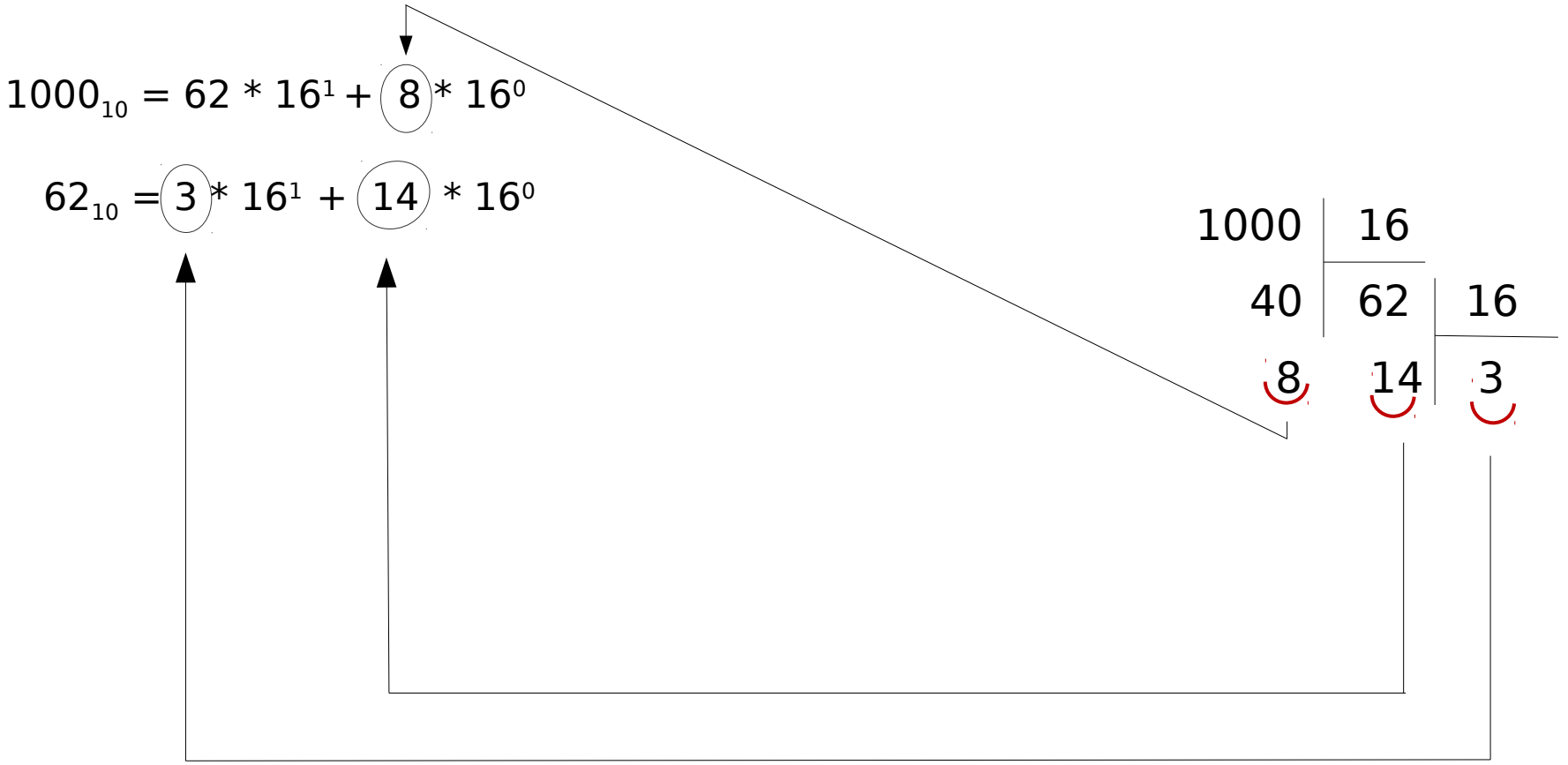
Divisões sucessivas por 16 (base)

$1000_{10}$



# Decimal para Hexadecimal

## Divisões sucessivas por 16 (base)



# Decimal para Hexadecimal

## Divisões sucessivas por 16 (base)

$1000_{10} = 62 * 16^1 + 8 * 16^0$

$62_{10} = 3 * 16^1 + 14 * 16^0$

$1000_{10} = (3 * 16^1 + 14 * 16^0) * 16^1 + 8 * 16^0$

$= 3 * 16^2 + 14 * 16^1 + 8 * 16^0$

| 1000 | 16 | |
|---|---|---|
| 40 | 62 | 16 |
| 8 | 14 | 3 |

# Decimal para Hexadecimal

## Divisões sucessivas por 16 (base)

$1000_{10} = 62 * 16^1 + 8 * 16^0$

$62_{10} = 3 * 16^1 + 14 * 16^0$

$1000_{10} = (3 * 16^1 + 14 * 16^0) * 16^1 + 8 * 16^0$

$= 3 * 16^2 + 14 * 16^1 + 8 * 16^0$

E

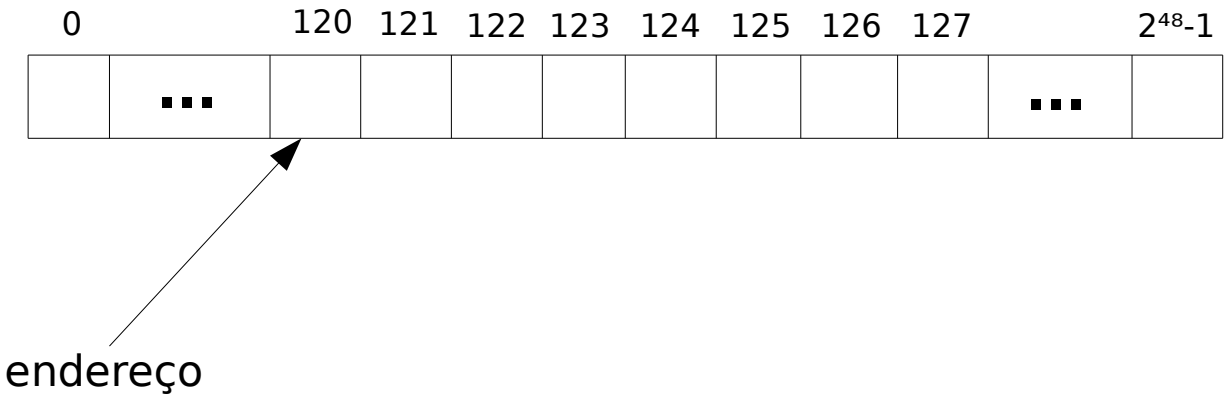
| 1000 | 16 | |
|---|---|---|
| 40 | 62 | 16 |
| 8 | 14 | 3 |

**$3E8_{16}$**

# Palavras (words)

Cada computador tem seu **tamanho de palavra**

- número de bits transferidos em um *chunk* entre memória e CPU
- número de bits de endereços (tamanho de um ponteiro)

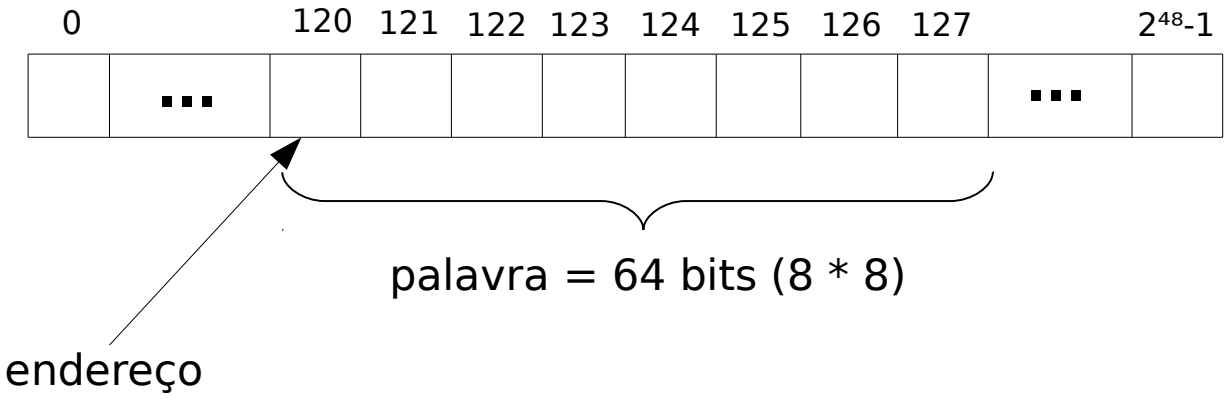


# Palavras (words)

Cada computador tem seu **tamanho de palavra**

- número de bits transferidos em um *chunk* entre memória e CPU
- número de bits de endereços (tamanho de um ponteiro)

Trabalharemos com uma plataforma de **64 bits** (8 bytes)



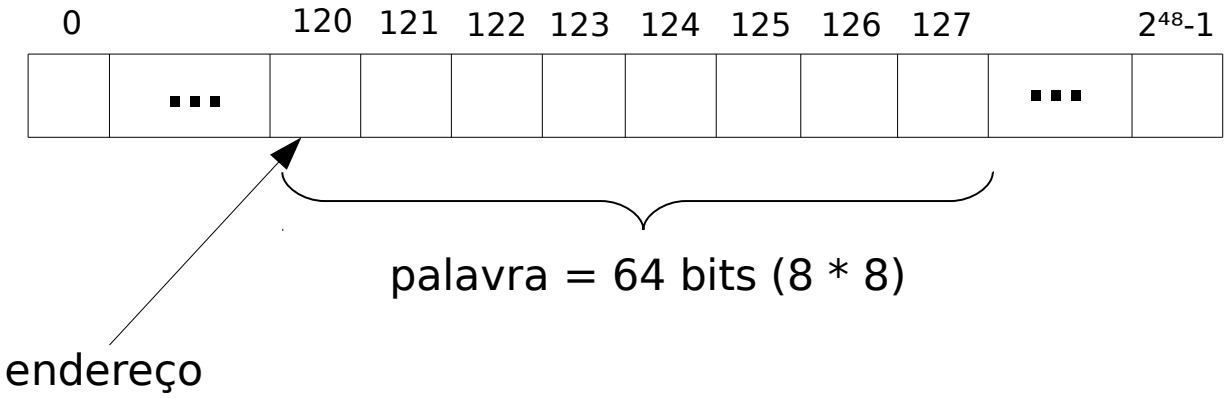
# Palavras (words)

Cada computador tem seu **tamanho de palavra**

- número de bits transferidos em um *chunk* entre memória e CPU
- número de bits de endereços (tamanho de um ponteiro)

Trabalharemos com uma plataforma de **64 bits** (8 bytes)

Alguns tipos de dados podem ocupar apenas parte de uma palavra, mas sempre **um número inteiro de bytes**



# Tamanhos de Tipos Numéricos de C

O tamanho de cada tipo depende da máquina e do compilador

**sizeof(T)**: número de bytes usado pelo tipo T

inteiros sem sinal (**unsigned**): ocupam o mesmo tamanho que os tipos com sinal, mas representam um intervalo diferente de valores

| Tipo C | 32-bit | 64-bit |
|---|---|---|
| char | 1 | 1 |
| short int (short) | 2 | 2 |
| int | 4 | 4 |
| long int (long) | **4** | **8** |
| | | |
| (T *) | **4** | **8** |

# Intervalos de Valores

Valores inteiros em diferentes tamanhos (número de bytes)

- com 1 byte (8 bits) podemos representar inteiros de 0 a 255 ($2^8$-1)
- com 2 bytes (16 bits), de 0 a 65535 ($2^{16}$-1)
- com 4 bytes (32 bits), de 0 a 4294967295 ($2^{32}$-1)
- com 8 bytes (64 bits), de 0 a $2^{64}$-1

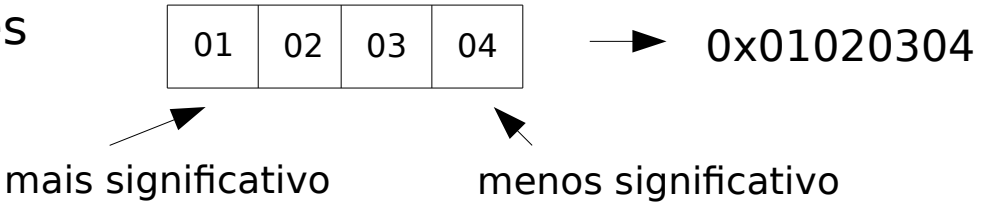
A mesma limitação vale para endereços

- com 4 bytes podemos endereçar 4GB de memória
- com 8 bytes podemos endereçar (teoricamente) $2^{64}$ bytes

# Ordenação de Bytes

Dados representados na memória como **sequência de bytes**

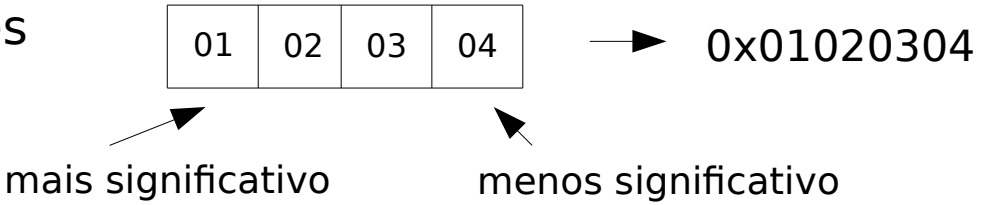
inteiro (32 bits): 4 bytes



# Ordenação de Bytes

Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

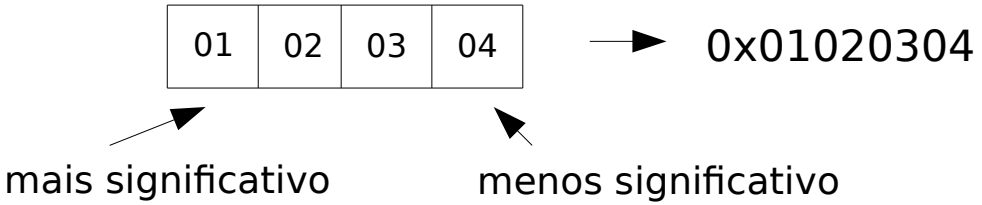



Duas convenções para **ordem de armazenamento dos bytes na memória**

# Ordenação de Bytes

Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes



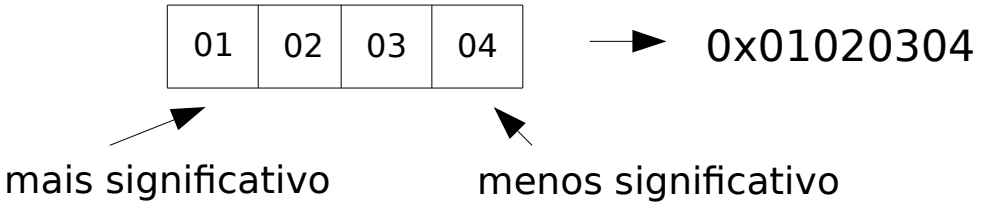

Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC, MIPS): do byte mais significativo para o menos significativo

# Ordenação de Bytes

Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes



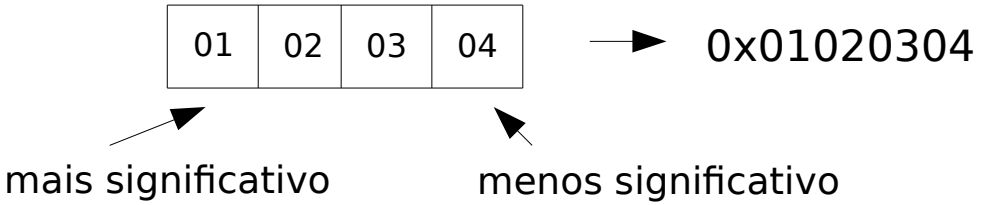

Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC, MIPS): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

# Ordenação de Bytes

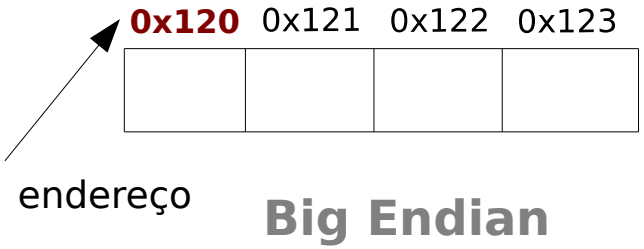
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes




Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

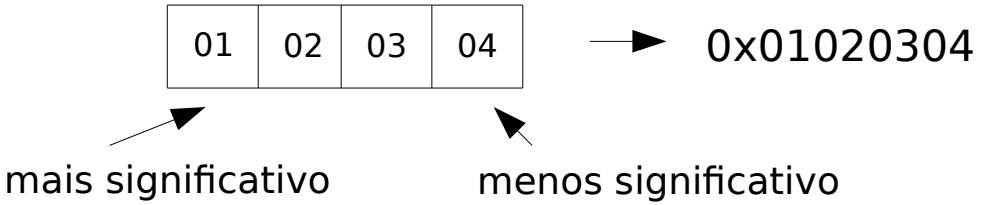



**Big Endian**

# Ordenação de Bytes

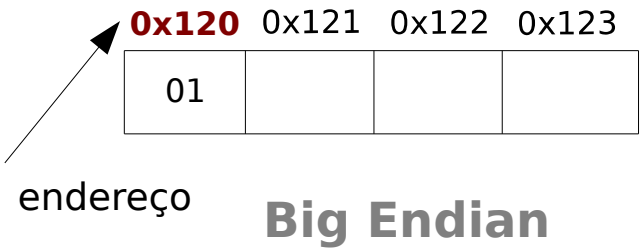
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes




Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

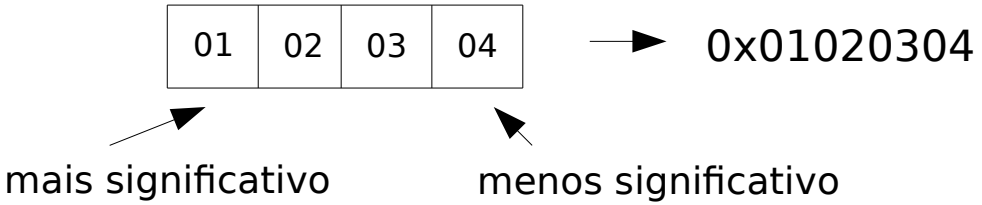



**Big Endian**

# Ordenação de Bytes

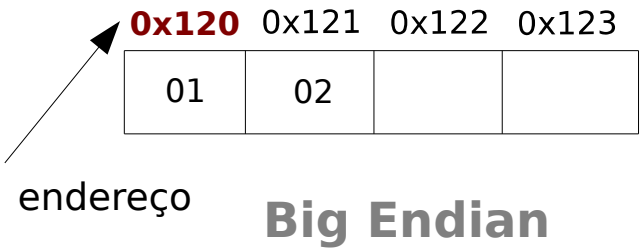
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes




Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

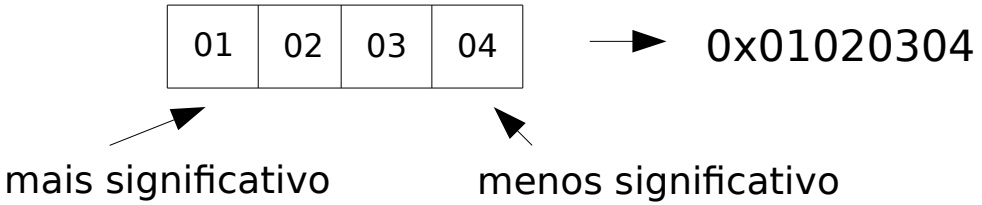



**Big Endian**

# Ordenação de Bytes

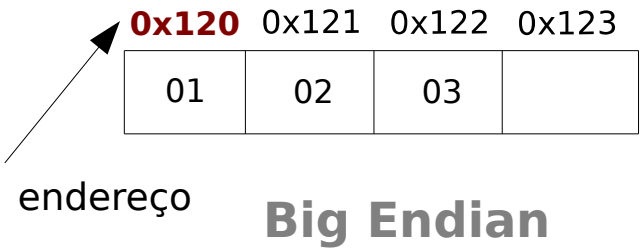
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes




Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

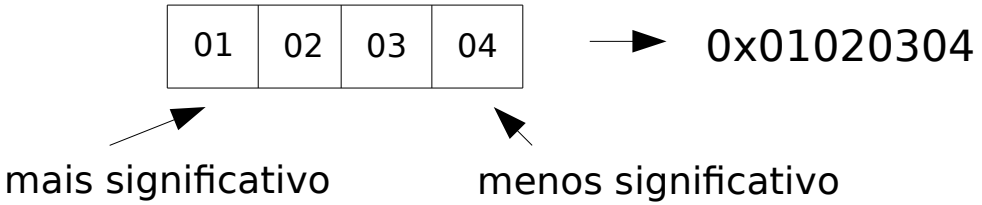



**Big Endian**

# Ordenação de Bytes

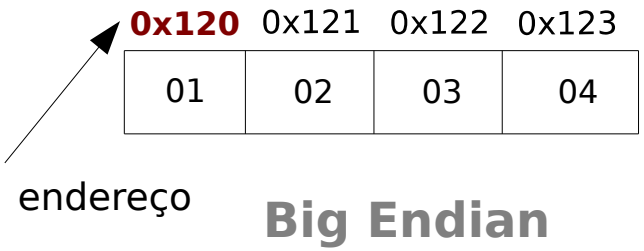
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes




Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo

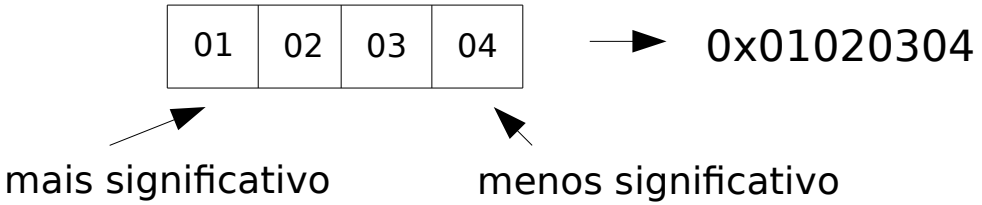



**Big Endian**

# Ordenação de Bytes

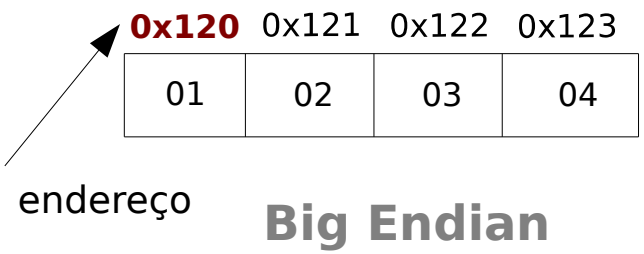
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

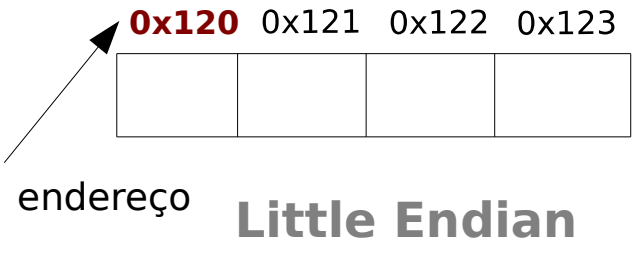



Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo




**Big Endian**

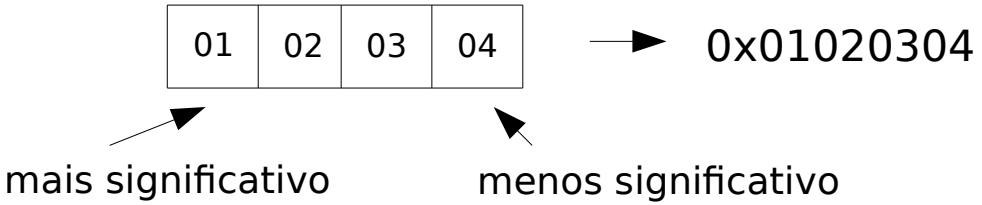



**Little Endian**

# Ordenação de Bytes

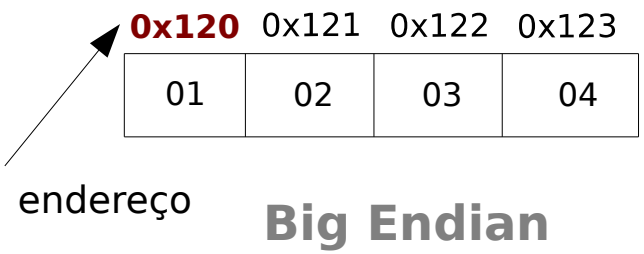
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

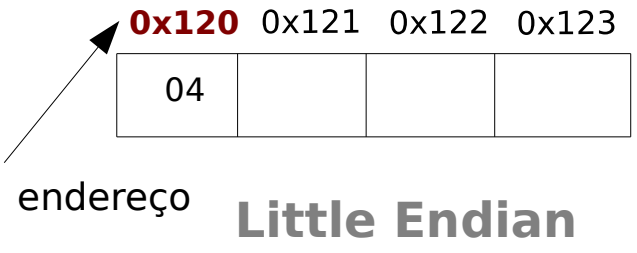



Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo




**Big Endian**



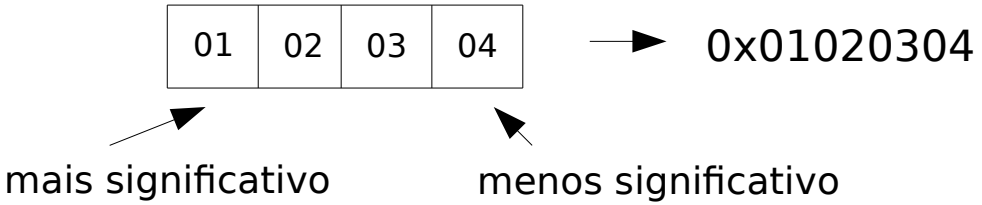

**Little Endian**

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

# Ordenação de Bytes

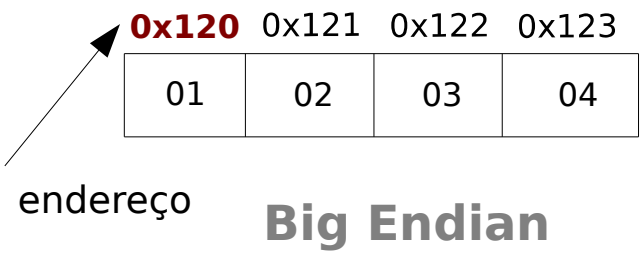
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

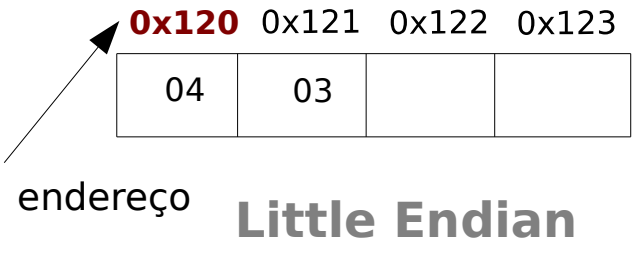



Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo




**Big Endian**



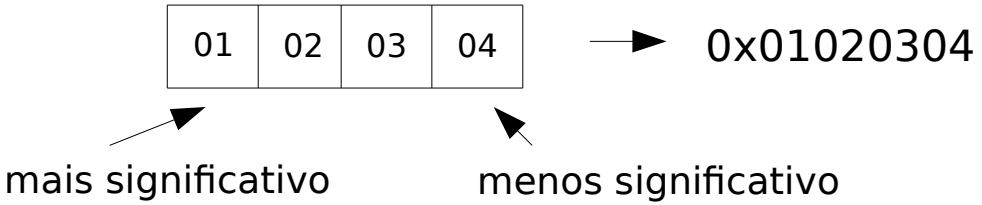

**Little Endian**

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

# Ordenação de Bytes

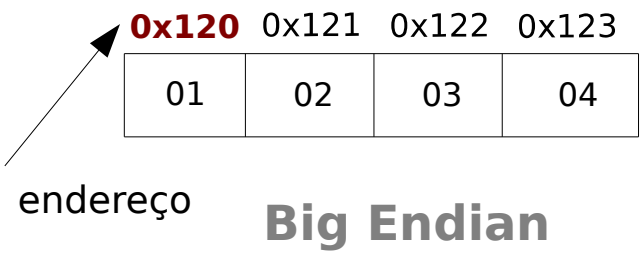
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

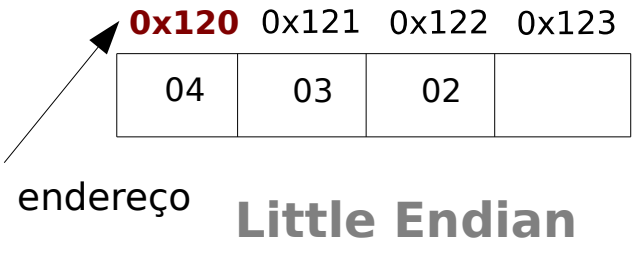



Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo




**Big Endian**



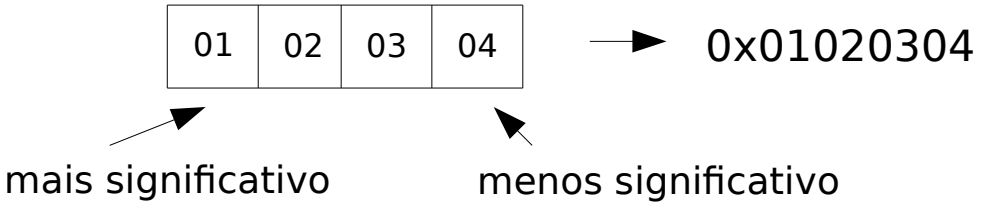

**Little Endian**

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

# Ordenação de Bytes

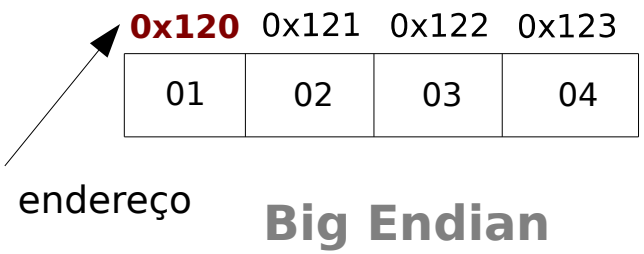
Dados representados na memória como **sequência de bytes**

inteiro (32 bits): 4 bytes

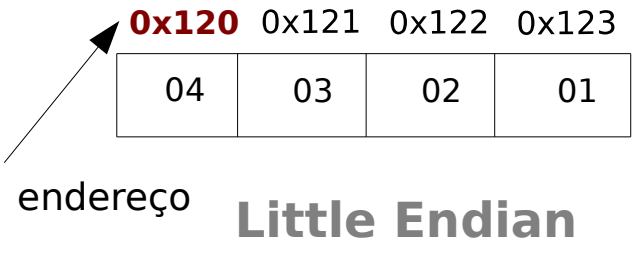



Duas convenções para **ordem de armazenamento dos bytes na memória**

- Big Endian (PowerPC): do byte mais significativo para o menos significativo
- Little Endian (Intel): do byte menos significativo para o mais significativo




**Big Endian**




**Little Endian**

DEPARTAMENTO DE INFORMÁTICA
PUC·RIO

# Verificando a ordenação

Do ponto de vista de um programa C, para verificar a ordenação da memória é necessário "quebrar" o sistema de tipos

inteiro ⟶ sequência de bytes

```
#include <stdio.h>
void dump (void *p, int n) {
  unsigned char *p1 = p;
  while (n--) {
    printf("%p - %02x\n", p1, *p1);
    p1++;
  }
}
int main() {
  int num = 0x01020304;
  dump(&num, sizeof(int));
  return 0;
}
```