Software Básico

Representação de Dados: Inteiros não Negativos

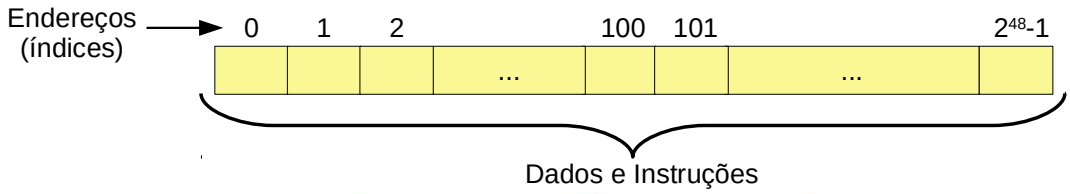
INSTITUTO DE INFORMÁTICA - UFG

Reconhecimento

- Material produzido por:
  - Noemi Rodriguez – PUC-Rio
  - Ana Lúcia de Moura – PUC-Rio
- Adaptação
  - Bruno Silvestre – UFG

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

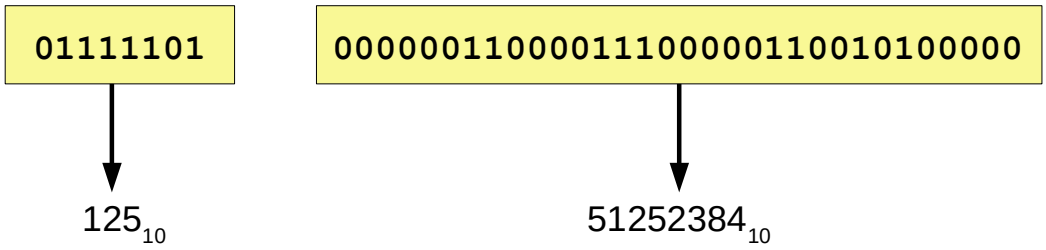


Representação da Informação

- 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)

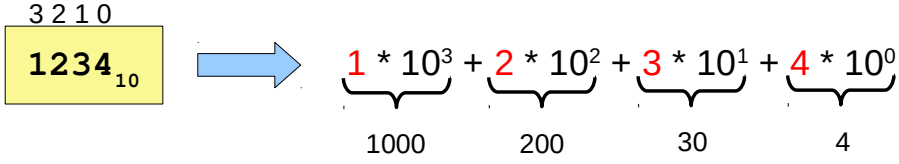
Representação da Informação

- Exemplo



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



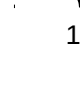
Notação Binária

- Base 2 → dígitos 0 e 1

7 6 5 4 3 2 1 0

$01111101_2 = 125_{10}$

$$\underbrace{0 * 2^7}_{0} + \underbrace{1 * 2^6}_{64} + \underbrace{1 * 2^5}_{32} + \underbrace{1 * 2^4}_{16} + \underbrace{1 * 2^3}_{8} + \underbrace{1 * 2^2}_{4} + \underbrace{0 * 2^1}_{0} + \underbrace{1 * 2^0}_{1}$$



Notação Hexadecimal

- Base 16 → dígitos 0 a 9,
  letras de A a F

3 2 1 0

$2ABC_{16} = 10940_{10}$

$$\underbrace{2 * 16^3}_{8192} + \underbrace{10 * 16^2}_{2560} + \underbrace{11 * 16^1}_{176} + \underbrace{12 * 16^0}_{12}$$

DECIMAL	HEXA
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Notação Hexadecimal

- Base 16 → dígitos 0 a 9, letras de A a F
- A notação hexadecimal é mais usada para descrever padrão de bits ou endereços de memória
- Em C (e Assembly) as constantes que começam com “0x” estão em notação hexadecimal: 0x10, 0xFF, 0x55aA

Conversão:

Binário ↔ Hexadecimal

Conversão: Hexa → Binário

- Cada dígito em hexa é expandido para 4 bits
  - Ex: $3A4C_{16}$ → $????_{2}$

3	A	4	C

DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão: Hexa → Binário

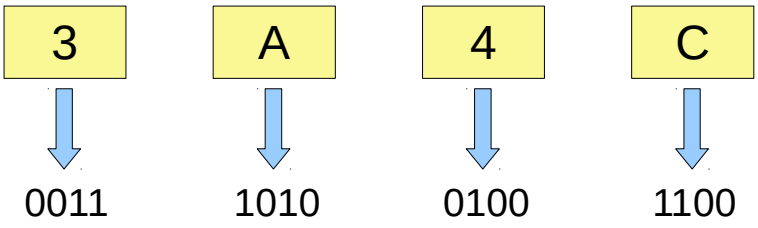
- Cada dígito em hexa é expandido para 4 bits
  - Ex: $3A4C_{16}$ → $????_{2}$

3	A	4	C
0011	1010	0100	1100

DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão: Hexa → Binário

- Cada dígito em hexa é expandido para 4 bits
  - Ex: $3A4C_{16}$ → 0011101001001100_2



DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

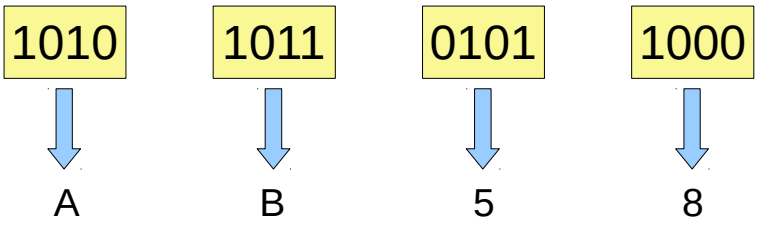
Conversão: Binário → Hexa

- Cada grupo de 4 bits dá origem a um dígito hexa
  - Ex: 1010101101011000_2 → $???_{16}$

DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão: Binário → Hexa

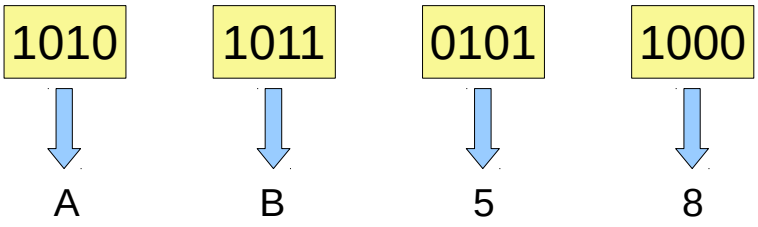
- Cada grupo de 4 bits dá origem a um dígito hexa
  - Ex: 1010101101011000_2 → $???_{16}$



DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão: Binário → Hexa

- Cada grupo de 4 bits dá origem a um dígito hexa
  - Ex: 1010101101011000_2 → $AB58_{16}$



DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão: Binário → Hexa

- Cada grupo de 4 bits dá origem a um dígito hexa
- Caso o número de bits não seja múltiplos de 4, deve-se completar com zero à esquerda antes de fazer a conversão
  - Ex: 11000_2 → $0001\ 1000_2$

DEC	HEX	BIN
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
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversão:

Decimal → Binário

Conversão: Decimal → Binário

- Divisões sucessivas por 2 (base)
  - Relação com a notação posicional
  - As “parcelas” são multiplicações de 0 ou 1 (restos) pela base elevada à posição

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_{2}$

$13_{10} = 6 * 2^1 + 1 * 2^0$

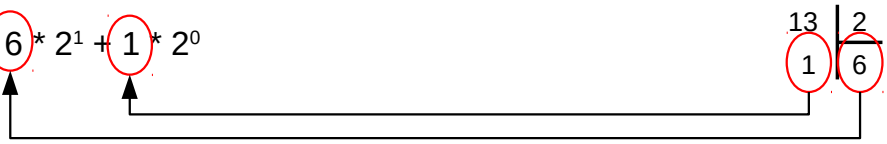
13	2
1	6

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$

13	2
1	6



Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$

13	2
1	6

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$

13	2	
1	6	2
	0	3

$6_{10} = 3 * 2^1 + 0 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$

13	2	
1	6	2
	0	3

$6_{10} = 3 * 2^1 + 0 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$

13	2	
1	6	2
0	3	

$6_{10} = 3 * 2^1 + 0 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$

13	2	
1	6	2
0	3	

$6_{10} = 3 * 2^1 + 0 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$

13	2		
1	6	2	
	0	3	

$6_{10} = 3 * 2^1 + 0 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$

13	2			
1	6	2		
	0	3	2	
		1	1	

$6_{10} = 3 * 2^1 + 0 * 2^0$

$3_{10} = 1 * 2^1 + 1 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$

13	2			
1	6	2		
	0	3	2	
			1	1

$6_{10} = 3 * 2^1 + 0 * 2^0$

$3_{10} = 1 * 2^1 + 1 * 2^0$

Conversão: Decimal → Binário

- Exemplo: 13_{10} → $????_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$
$13_{10} = (1 * 2^1 + 1 * 2^0) * 2^2 + 0 * 2^1 + 1 * 2^0$
$13_{10} = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

13	2			
1	6	2		
	0	3	2	
		1	1	

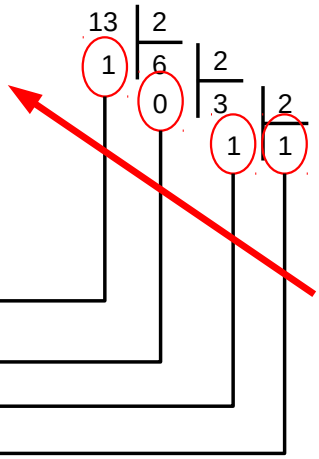
$6_{10} = 3 * 2^1 + 0 * 2^0$

$3_{10} = 1 * 2^1 + 1 * 2^0$

Conversão: Decimal → Binário

- Exemplo: $13_{10} \rightarrow 1101_2$

$13_{10} = 6 * 2^1 + 1 * 2^0$
$13_{10} = (3 * 2^1 + 0 * 2^0) * 2^1 + 1 * 2^0$
$13_{10} = 3 * 2^2 + 0 * 2^1 + 1 * 2^0$
$13_{10} = (1 * 2^1 + 1 * 2^0) * 2^2 + 0 * 2^1 + 1 * 2^0$
$13_{10} = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$



Conversão:

Decimal → Hexadecimal

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

1005_{10}

1005	16
13	62

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = 62 * 16^1 + 13 * 16^0$

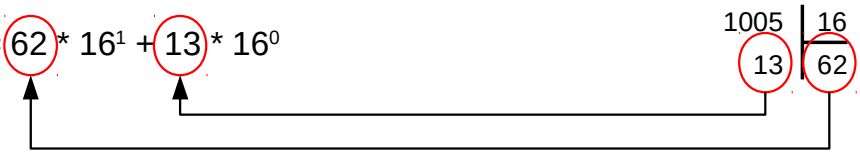
1005	16
13	62

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = (62) * 16^1 + (13) * 16^0$

1005	16
(13)	(62)



Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = (62) * 16^1 + 13 * 16^0$

1005	16
13	62

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = 62 * 16^1 + 13 * 16^0$

1005	16	
13	62	16
	14	3

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

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = 62 * 16^1 + 13 * 16^0$

1005	16	
13	62	16
	14	3

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

Conversão: Decimal → Hexa

- Exemplo: 1005_{10} → $????_{16}$

$1005_{10} = 62 * 16^1 + 13 * 16^0$

$1005_{10} = (3 * 16^1 + 14 * 16^0) * 16^1 + 13 * 16^0$

$1005_{10} = 3 * 16^2 + 14 * 16^1 + 13 * 16^0$

1005	16	
13	62	16
	14	3

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

Conversão: Decimal → Hexa

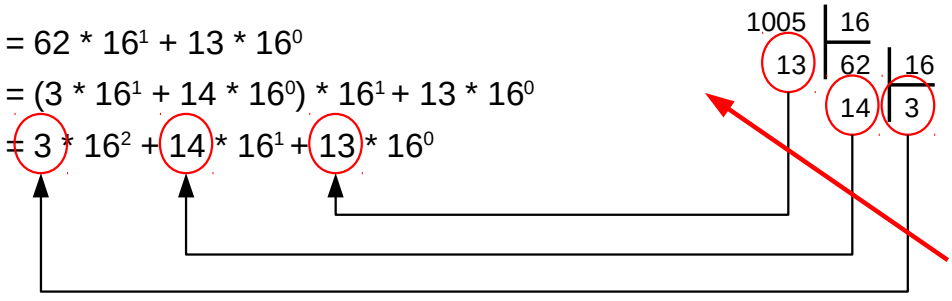
- Exemplo: 1005_{10} → $3ED_{16}$

$1005_{10} = 62 * 16^1 + 13 * 16^0$

$1005_{10} = (3 * 16^1 + 14 * 16^0) * 16^1 + 13 * 16^0$

$1005_{10} = 3 * 16^2 + 14 * 16^1 + 13 * 16^0$

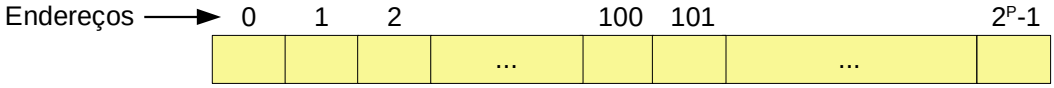
1005	16	
13	62	16
	14	3



Tamanhos dos Dados

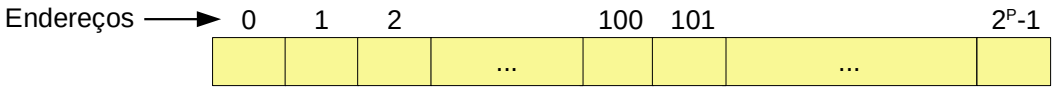
Palavra (*Word*)

- Cada computador tem seu tamanho de Palavra
  - Número de bits transferidos por vez (*chunk* de dados) entre memória e CPU
  - Número de bits de endereços (tamanho de um ponteiro)



Palavra (*Word*)

- Nas arquiteturas 32-bits, palavra de 32-bits (4 bytes)
- Nas arquiteturas 64-bits, palavra 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

Tamanhos de Tipos Numéricos de C

- Tamanhos geralmente adotados (em bytes)

Tipo C		Arquitetura	
Com Sinal	**Sem Sinal**	**32-bits**	**64-bits**
char [1]	unsigned char	1	1
short	unsinged short	2	2
int	unsigned int	4	4
long	unsigned long	4	8
T *	-	4	8
float	-	4	4
double	-	8	8

[1] Alguns compiladores adotam “char” como sendo sem sinal. Neste caso, tem que usar “signed char” para ser com sinal.

Tamanhos de Tipos Numéricos de C

- Para evitar problemas de tamanho, o padrão C99 introduziu tipos com tamanho explícito
  - int8_t, uint8_t
  - int16_t, uint16_t
  - int32_t, uint32_t
  - int64_t, uint64_t
- Devemos incluir o cabeçalho “stdint.h”

Faixa de Valores

Tipo	Tamanho	Faixa de Valores
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255 [$0,2^8-1$]
short	2 bytes	-32.768 a 32.767
unsigned short	2 bytes	0 a 65.535 [$0,2^{16}-1$]
int	4 bytes	-2.147.483.648 a 2.147.483.647
unsigned int	4 bytes	0 a 4.294.967.295 [$0,2^{32}-1$]
long	4 bytes	-2.147.483.648 a 2.147.483.647
	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
unsigned long	4 bytes	0 a 4.294.967.295
	8 bytes	0 a 18.446.744.073.709.551.615 [$0,2^{64}-1$]
float	4 byte	1.2E-38 a 3.4E+38
double	8 byte	2.3E-308 a 1.7E+308

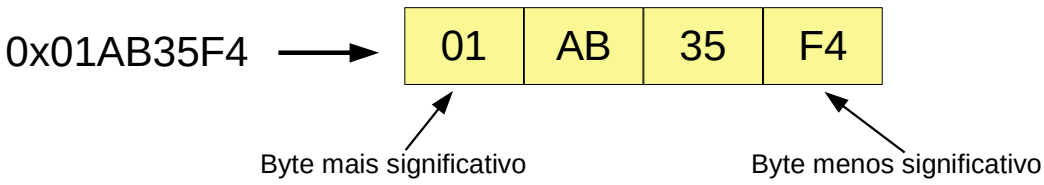
Ordenação de Bytes

Ordenação de Bytes

- Dados representados na memória como sequência de bytes
- Para os tipos que ocupam mais de um byte existem duas convenções para ordem de armazenamento dos bytes na memória que os processadores seguem:
  - Big Endian
  - Little Endian
- Deixando claro: Isso não se aplica a tipos de tamanho de 1 byte (por exemplo, char)

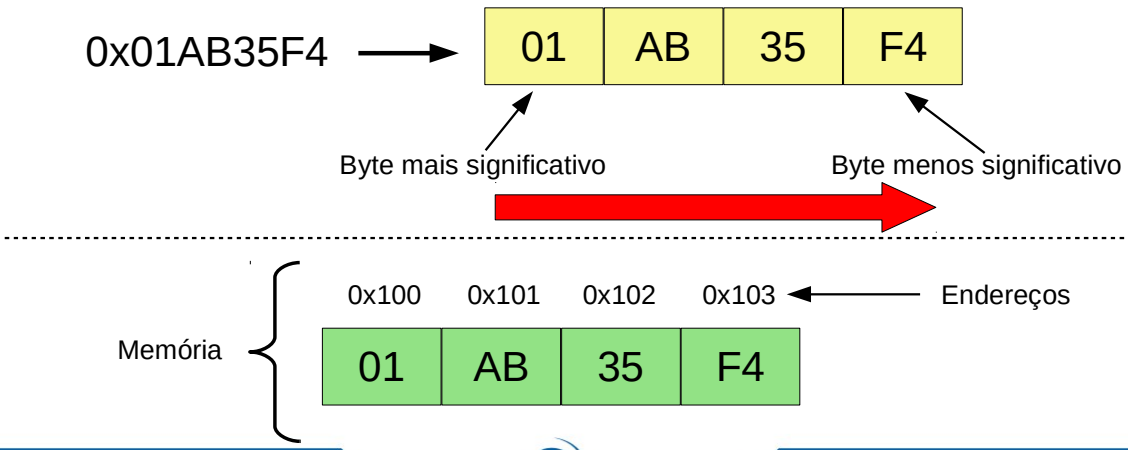
Ordenação de Bytes

- Primeiro, separar o valor em bytes



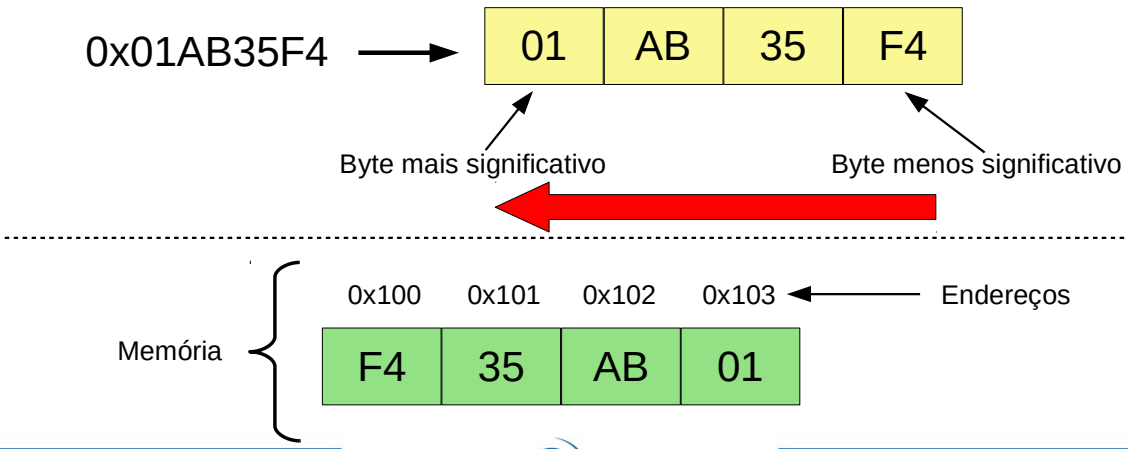
Ordenação de Bytes

- Big Endian (PowerPC da IBM)
  - Do byte mais significativo para o menos



Ordenação de Bytes

- Little Endian (Intel)
  - Do byte menos significativo para o mais



Ordenação de Bytes

- Não há uma definição clara de qual das abordagens é a melhor
- Para a grande maioria dos programadores, a ordenação de bytes é transparente na hora de desenvolver as aplicações
- Mas as vezes precisamos lidar com isso em alguns casos como Sistemas Operacionais, ou outros sistemas de computação

Ordenação de Bytes

```
#include <stdlib.h>
#include <stdio.h>

void mostra(unsigned char* p, int tam) {
   for (int i=0; i < tam; i++)
      printf("%02x ", p[i]);
   printf("\n");
}

int main() {
   int num = 0x01020304;
   mostra((unsigned char*) &num, sizeof(int));
   return 0;
}
```