

**Instituto de
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



Organização Básica de computadores e linguagem de montagem

Representação de Informações no Computador

Prof. Edson Borin

<https://www.ic.unicamp.br/~edson>

Institute of Computing - UNICAMP

Representação de Informações

Como representar informações em um computador?

- Números inteiros?
- Texto?
- Registros?
- Vetores?

Representação de Informações

Como representar informações em um computador?

- Informações são representadas através de dígitos binários, ou *BITs* (**B**inary **digi**Ts).
- Dígitos 0 e 1
- Quantos estados (ou números) distintos podemos representar com 3 dígitos da base binária?

Representação de Informações

Como representar informações em um computador?

- Informações são representadas através de dígitos binários, ou *BITs* (**B**inary **digiT**s).
- Dígitos 0 e 1
- Quantos estados (ou números) distintos podemos representar com 3 dígitos da base binária?
 - 8 estados se utilizarmos notação posicional
 - 4 estados se utilizarmos notação não posicional
 - 1: 001, 010, 100 (um bit 1 e dois bits 0)
 - 2: 110, 101, 011 (um bit 0 e dois bits 1)
 - 3: 000 (três bits 0)
 - 4: 111 (três bits 1)

Representação de Informações

Notação posicional: valor do dígito depende da sua posição.

- Exemplo: Número decimal 132
 - Valor do dígito 2 = 2
 - Valor do dígito 3 = 30
 - Valor do dígito 1 = 100

Representação de Informações

Notação posicional: valor do dígito depende da sua posição.

- Exemplo: Número decimal 132
 - Valor do dígito 2 = 2
 - Valor do dígito 3 = 30
 - Valor do dígito 1 = 100

“Informações no computador são representadas através de números, codificados na base binária com notação posicional”

Representação de Informações

A quantidade de símbolos (ou valores) distintos que que cada dígito pode ter define a base numérica.

Exemplos

- Base 2, ou binária \Rightarrow 2 símbolos distintos: 0 e 1
- Base 8, ou octal \Rightarrow 8 símbolos distintos: 0, 1, ..., 7
- Base 10, ou decimal \Rightarrow 10 símbolos distintos: 0, ..., 9
- ...

Quais são os símbolos utilizados na base 16?

Representação de Informações

A quantidade de símbolos (ou valores) distintos que que cada dígito pode ter define a base numérica.

Exemplos

- Base 2, ou binária \Rightarrow 2 símbolos distintos: 0 e 1
- Base 8, ou octal \Rightarrow 8 símbolos distintos: 0, 1, ..., 7
- Base 10, ou decimal \Rightarrow 10 símbolos distintos: 0, ..., 9
- ...

Quais são os símbolos utilizados na base 16?

- Símbolos da base hexadecimal: 0, 1, ..., 9, A, B, C, D, E, F

Bases numéricas

Qual é a base dos números abaixo?

- FE03
- 8230
- 9210
- 1001

Bases numéricas

Qual é a base dos números abaixo?

- FE03
- 8230
- 9210
- 1001

Para distinguir temos que anotar o número com a base. Exemplos:

- $FE03_{16}$
- 1001_8
- 1001_2

Bases numéricas

Em linguagens de programação esta anotação é geralmente realizada com prefixos. Por exemplo, em 'C', números nas bases binária, octal e hexadecimal são representados com os prefixos 0b, 0 e 0x enquanto que números na base decimal são representados sem prefixos.

Para distinguir temos que anotar o número com a base. Exemplos:

- $FE03_{16}$
- 1001_8
- 1001_2

- 0xFE03
- 01001
- 0b1001

Bases numéricas

Qual é o valor de cada dígito nos números abaixo?

- 9210_{10}
- 1001_2

Bases numéricas

Qual é o valor de cada dígito nos números abaixo?

- 9210_{10}
- 1001_2

O valor de um dígito ***d*** em um número na base ***t*** é dado por:

- ***d*** × ***t***^{posição}

Onde a posição é dada pela seguinte convenção:

Dígitos	0	0	0	0	9	2	1	0
Posição	7	6	5	4	3	2	1	0

Bases numéricas

Qual é o valor de cada número abaixo em decimal?

- 1001_2
- FF_{16}

Bases numéricas

Qual é o valor de cada número abaixo em decimal?

- 1001_2
- FF_{16}

O valor de um número na base **t** com **n** dígitos é o somatório dos valores dos dígitos:

$$V = \sum_{i=0}^{n-1} d_i \times t^i$$

onde d_i é o dígito na posição i .

Bases numéricas

Qual é o valor de cada número abaixo em decimal?

- $1001_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9_{10}$
- $FF_{16} = F \times 16^1 + F \times 16^0 = 15 \times 16 + 15 \times 1 = 255_{10}$

O valor de um número na base **t** com **n** dígitos é o somatório dos valores dos dígitos:

$$V = \sum_{i=0}^{n-1} d_i \times t^i$$

onde d_i é o dígito na posição i .

Bases numéricas

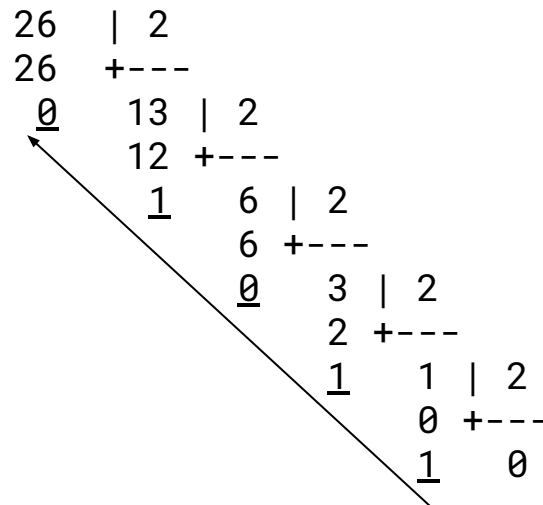
Como fazemos para encontrar a representação de um valor na base binária?

- Por exemplo: o valor 26_{10}

Bases numéricas

Como fazemos para encontrar a representação de um valor na base binária?

- Por exemplo: o valor $26_{10} = 11010_2$



Bases numéricas - Exercícios

Represente os seguintes números na base binária

- $151_6 =$
- $139_{10} =$

Represente os seguintes números na base hexadecimal

- $101001_2 =$
- $16_{10} =$
- $240_{10} =$
- $20_8 =$

Bases numéricas - Exercícios

Represente os seguintes números na base binária

- $151_6 = 1000011_2$
- $139_{10} = 10001011_2$

Represente os seguintes números na base hexadecimal

- $101001_2 = 29_{16}$
- $16_{10} = 10_{16}$
- $240_{10} = F0_{16}$
- $20_8 = 10_{16}$

Conversão de bases numéricas

Tipo de conversão	Procedimento
Decimal => Binário	Divisões sucessivas por 2 até se obter zero no quociente. Leitura dos dígitos binários no resto de baixo para cima.
Binário => Decimal	Soma de potências de 2 cujo expoente é a posição do bit e cujo coeficiente é o próprio bit.
Hexadecimal => Binário	Expandir cada dígito hexa em quatro dígitos binários segundo seu valor.
Binário => Hexadecimal	Compactar cada quatro dígitos binários em um único dígito hexa segundo seu valor.
Decimal => Hexadecimal	Divisões sucessivas por 16 até se obter zero no quociente. Converter restos p/ dígitos hexadecimais. Leitura dos dígitos de baixo para cima.
Hexadecimal => Decimal	Soma de potências de 16 cujo expoente é a posição do dígito e cujo coeficiente é o valor do próprio dígito hexa.

Números Sem Sinal

Na representação sem sinal, todos os *bits* são utilizados como dígitos do número.

- Exemplo: Registradores com 3 *bits* podem representar 8 números distintos: 0 a 7

$$000_2 = 0_{10}$$

$$001_2 = 1_{10}$$

$$010_2 = 2_{10}$$

$$011_2 = 3_{10}$$

$$100_2 = 4_{10}$$

$$101_2 = 5_{10}$$

$$110_2 = 6_{10}$$

$$111_2 = 7_{10}$$

Números Com Sinal

Três tipos de codificação mais conhecidas

- Sinal e magnitude
- Complemento de 1
- Complemento de 2

Sinal e Magnitude

Na representação “sinal e magnitude” o *bit* mais à esquerda (o mais significativo) representa o sinal do número e os outros *bits* representam a magnitude.

Qual é o valor dos números abaixo na representação “sinal e magnitude” e sem sinal?

Sinal e Magnitude

Na representação “sinal e magnitude” o *bit* mais à esquerda (o mais significativo) representa o sinal do número e os outros *bits* representam a magnitude.

Qual é o valor dos números abaixo na representação “sinal e magnitude” e sem sinal?

- $0001\ 0101_2$
- $1000\ 1010_2$

E estes números?

- $0000\ 0000_2$
- $1000\ 0000_2$

Sinal e Magnitude

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-0
101	5	-1
110	6	-2
111	7	-3

Complemento de 1

Na representação “complemento de 1” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

- Primeiro *bit* 0 \Rightarrow o número é positivo e o valor pode ser obtido da mesma maneira que na representação sem sinal
- Primeiro *bit* 1 \Rightarrow o número é negativo. Para descobrir a magnitude, basta inverter todos os *bits* e computar o valor na representação sem sinal

Qual é o valor de 10010_2 ?

Complemento de 1

Na representação “complemento de 1” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

- Primeiro *bit* indica o sinal e o resto representa a magnitude. O valor pode ser descoberto descobrindo o valor da magnitude e descobrindo o sinal. Logo: $10010_2 = -13_{10}$
- Primeiro *bit* indica o sinal e o resto representa a magnitude. O valor pode ser descoberto descobrindo o valor da magnitude e descobrindo o sinal. Logo: $10010_2 = -13_{10}$

Qual é o valor de $10010_2 = -13_{10}$

Complemento de 1

Na representação “complemento de 1” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	-0	
101	5	-1	
110	6	-2	
111	7	-3	

Complemento de 1

Na representação “complemento de 1” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	-0	-3
101	5	-1	-2
110	6	-2	-1
111	7	-3	-0

Complemento de 2

Na representação “complemento de 2” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

- Primeiro *bit* 0 \Rightarrow o número é positivo e o valor pode ser obtido da mesma maneira que na representação sem sinal
- Primeiro *bit* 1 \Rightarrow o número é negativo. Para descobrir a magnitude, **devemos inverter todos os *bits*, somar 1, e então computar o valor na representação sem sinal.**

Qual é o valor de 10010_2 ?

Complemento de 2

Na representação “complemento de 2” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

$$\text{Magnitude}(10010_2) = 01101_2 + 1_2 = 01110_2 = 14_{10}$$

$$\text{Logo: } 10010_2 = -14_{10}$$

os *bits*, somar 1, e então obter o valor na representação sem sinal.

Qual é o valor de $10010_2 = -14_{10}$

Complemento de 2

Na representação “complemento de 2” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1	Valor na rep. Comp. de 2
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	
101	5	-1	-2	
110	6	-2	-1	
111	7	-3	-0	

Complemento de 2

Na representação “complemento de 2” o *bit* mais à esquerda também indica o sinal, entretanto a magnitude é representada de maneira diferente.

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1	Valor na rep. Comp. de 2
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Representação de Números

Representações “Sinal e Mag.” e “Comp. de 1” possuem duas representações para o zero: 0 e -0

=> A representação “complemento de 2” é a mais utilizada.

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1	Valor na rep. Comp. de 2
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Representação de Números

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1	Valor na rep. Comp. de 2
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Maior	7	3	3	3
Menor	0	-3	-3	-4

Representação de Números

Número binário	Valor na rep. sem sinal	Valor na rep. Sinal e Mag.	Valor na rep. Comp. de 1	Valor na rep. Comp. de 2
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Maior	7	3	3	3
Menor	0	-3	-3	-4

Maior	$2^n - 1$	$2^{n-1} - 1$	$2^{n-1} - 1$	$2^{n-1} - 1$
Menor	0	$-(2^{n-1} - 1)$	$-(2^{n-1} - 1)$	$-(2^{n-1})$

Números no Computador

“Informações no computador são representadas através de números, codificados na base binária com notação posicional”

Números no Computador

“Informações no computador são representadas através de números, codificados na base binária com notação posicional”

- Quantos *bits* o computador usa para codificar cada número?

Números no Computador

“Informações no computador são representadas através de números, codificados na base binária com notação posicional”

- Quantos *bits* o computador usa para codificar cada número?

O IAS utiliza 40 *bits*!

- Palavras da memória possuem 40 *bits*
- Registradores da ULA possuem 40 *bits*.

Números no Computador

Computadores modernos codificam números com palavras de 8, 16, 32, 64 ou mais *bits*.

- Geralmente é uma potência de 2.

Uma arquitetura de 32 *bits* é uma arquitetura que é capaz de armazenar e realizar operações aritméticas em números com até 32 *bits*.

Complemento de 2

Números de 32 *bits* em Complemento de 2:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{10}$$

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = +1_{10}$$

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = +2_{10}$$

...

$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = +2,147,483,646_{10}$$

$$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = +2,147,483,647_{10}$$

$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2,147,483,648_{10}$$

$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2,147,483,647_{10}$$

$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2,147,483,646_{10}$$

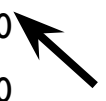
...

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{10}$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{10}$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{10}$$

maxint



minint

Aritmética Binária: Soma e Subtração

Como no ensino fundamental: (vai-um/vem-um)

$$\begin{array}{r} 0111 \quad (7) \\ + 0110 \quad (6) \\ \hline \end{array} \quad \begin{array}{r} 0111 \quad (7) \\ - 0110 \quad (6) \\ \hline \end{array} \quad \begin{array}{r} 0110 \quad (6) \\ - 0101 \quad (5) \\ \hline \end{array}$$

Aritmética Binária: Soma e Subtração

Como no ensino fundamental: (vai-um/vem-um)

11			*
0111 (7)	0111 (7)	0110 (6)	
+ 0110 (6)	- 0110 (6)	- 0101 (5)	
<hr/>	<hr/>	<hr/>	
1101 (13)	0001 (1)	0001 (1)	

Adição e subtração em complemento de 2 podem ser realizados da mesma forma que na representação sem sinal!

Aritmética Binária: Soma e Subtração

Subtração pode ser realizada com o *hardware* de soma!

- $(A - B = A + (-B))$.

Ex: $7 - 6 = 7 + (-6)$

$$\begin{array}{r} 0111 \ (+7) \\ + \underline{1010} \ (-6) \end{array}$$

Aritmética Binária: Soma e Subtração

Subtração pode ser realizada com o *hardware* de soma!

- $(A - B = A + (-B))$.

Ex: $7 - 6 = 7 + (-6)$

$$\begin{array}{r} 111 \\ 0111 \quad (+7) \\ + \underline{1010} \quad (-6) \\ \hline 0001 \quad (=1) \end{array}$$

Aritmética Binária: *Overflow*

Overflow: quando o resultado é maior (menor) do que a palavra do computador pode representar.

- Exemplo: Ocorre *overflow* na operação abaixo?

$$\begin{array}{r} 0111 \quad (7) \\ + 0001 \quad (1) \\ \hline \end{array}$$

Aritmética Binária: *Overflow*

Overflow: quando o resultado é maior (menor) do que a palavra do computador pode representar.

- Exemplo: Ocorre *overflow* na operação abaixo?

$$\begin{array}{r} 0111 \quad (7) \\ + 0001 \quad (1) \\ \hline 1000 \end{array}$$

Na representação **sem sinal** não ocorre *overflow*.
Note que $7 + 1 = 8$

Na representação **complemento de 2** ocorre *overflow*.
Não é possível representar o valor 8 com 4 *bits* nesta representação!

Aritmética Binária: Detecção de *Overflow*

Exercício: Compute o resultado da operação abaixo e verifique se houve *overflow*

4 + 5 em uma representação com **números sinalizados de 8 *bits***

$$\begin{array}{r} (4) \\ \pm(5) \end{array}$$

Aritmética Binária: Detecção de *Overflow*

Exercício: Compute o resultado da operação abaixo e verifique se houve *overflow*

4 + 5 em uma representação com **números sinalizados de 8 bits**

$$\begin{array}{r} 1 \\ 0000 0100 (4) \\ + 0000 0101 (5) \\ \hline 0000 1001 (9) \end{array}$$

Aritmética Binária: Detecção de *Overflow*

Exercício: Compute o resultado da operação abaixo e verifique se houve *overflow*

4 + 5 em uma representação com **números sinalizados de 4 *bits***

$$\begin{array}{r} (4) \\ \pm(5) \end{array}$$

Aritmética Binária: Detecção de *Overflow*

Exercício: Compute o resultado da operação abaixo e verifique se houve *overflow*

4 + 5 em uma representação com **números sinalizados de 4 *bits***

$$\begin{array}{r} 1 \\ 0100(4) \\ + \underline{0101(5)} \\ 1001(-7) \end{array}$$

Representação de Caracteres

Cada caractere é associado a um número distinto.

Existem diversos padrões.

- Exemplo: Padrão ASCII - American Standard Code for Information -- Usa *7 bits*, (128 caracteres distintos)

64	@	96	'	48	0
65	A	97	a	49	1
66	B	98	b	50	2
67	C	99	c	51	3
68	D	100	d	52	4
69	E	101	e	53	5
70	F	102	f	54	6
71	G	103	g	55	7
72	H	104	h	56	8
73	I	105	i	57	9

Representação de Caracteres

Cada caractere é associado a um número distinto.

- ASCII usa *7 bits*
- Um texto é armazenado como uma cadeia de caracteres!
 - Posições consecutivas da memória!

Representação de Caracteres

Cada caractere é associado a um número distinto.

- ASCII usa *7 bits*
- Um texto é armazenado como uma cadeia de caracteres!
 - Posições consecutivas da memória!

	A	s	s	e	m	b	l	y		r	o	c	k	s	
...	41	73	73	65	6D	62	6C	79	20	72	6F	63	6B	73	...
codificação ASCII															

Representação de Caracteres

ASCII não possui letras acentuadas (á, é, ç, ...)

Diversos padrões alternativos foram inventados para adicionar este tipo de letra e outros caracteres usados em outras línguas

- Exemplo: ISO-LATIN-1, MacOSRoman, UTF-8

Atualmente o padrão UTF-8 é o padrão mais utilizado em páginas na web

- Este padrão é compatível com o padrão ASCII
 - Caracteres que podem ser representados no padrão ASCII são representados no padrão UTF-8 com o mesmo valor numérico!

Representação de Caracteres

Representação de Cadeias de Caracteres (*strings*) na memória do computador:

Exemplo: “Maças Assadas”

	I	2		M	a	ç	ã	s		A	s	s	a	d	a	s		
...	31	32	20	4D	61	E7	E3	73	20	41	73	73	61	64	61	73	00	...
codificação ISO-LATIN-I																		

	I	2		M	a	ç	ã	s		A	s	s	a	d	a	s		
...	31	32	20	4D	61	8D	8B	73	20	41	73	73	61	64	61	73	00	...
codificação MacOSRoman																		

	I	2		M	a	ç		ã		s		A	s	s	a	d	a	s		
...	31	32	20	4D	61	C3	A7	C3	A3	73	20	41	73	73	61	64	61	73	00	...
codificação UTF-8																				

Organização de dados na memória

Caracteres na memória do computador

- No IAS, as palavras da memória possuíam 40 *bits*!
- A grande maioria das memórias de computadores atuais possuem palavras (unidades de armazenamento endereçáveis) de 1 *byte* (8 *bits*).
 - No endereço 0 cabe um dado de 1 *byte*, no endereço 1 cabe um dado de 1 *byte* e assim por diante.

Organização de dados na memória

Caracteres na memória do computador

- No IAS, as palavras da memória possuíam 40 *bits*!
- A grande maioria das memórias de computadores atuais possuem palavras (unidades de armazenamento endereçáveis) de 1 *byte* (8 *bits*).
 - No endereço 0 cabe um dado de 1 *byte*, no endereço 1 cabe um dado de 1 *byte* e assim por diante.
- Quando armazenamos números de 7 *bits* em 1 *byte* nós desperdiçamos *bits* da memória. Por outro lado, esta abordagem facilita a leitura dos dados pois cada palavra de memória possui um único caractere e cada caractere está armazenado em uma única palavra de memória.

Organização de dados na memória

Números na memória do computador

Como fazemos para armazenar um número de 32 *bits* em uma memória endereçada a *byte*? Ou seja, em uma memória onde as unidades de armazenamento possuem 1 *byte*.

Exemplo: Número de 32 *bits* (4 *bytes*)

$$1025_{10} = 00000000 \ 00000000 \ 00000100 \ 00000001_2$$

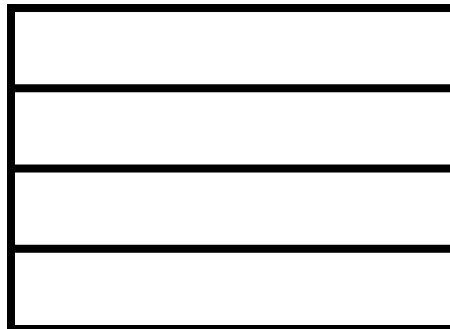
Endereço

00

01

02

03



Organização de dados na memória

Números na memória do computador

Como fazemos para armazenar um número de 32 *bits* em uma memória endereçada a *byte*? Ou seja, em uma memória onde as unidades de armazenamento possuem 1 *byte*.

- **Resposta: Depende do *Endianness*.**

Exemplo: Número de 32 *bits* (4 *bytes*)

$$1025_{10} = 00000000 \ 00000000 \ 00000100 \ 00000001_2$$

Endereço	<i>Big-Endian</i>
00	
01	
02	
03	

Organização de dados na memória

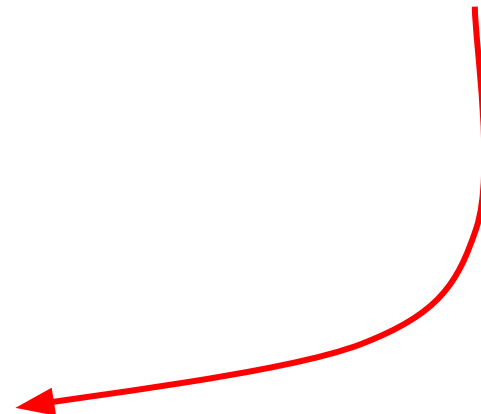
Números na memória do computador

Big-Endian: Byte menos significativo é armazenado no **maior** endereço

Exemplo: Número de 32 *bits* (4 *bytes*)

$1025_{10} = 00000000 \ 00000000 \ 00000100 \ 00000001_2$

Endereço	<i>Big-Endian</i>
00	00000000
01	00000000
02	00000100
03	00000001



Organização de dados na memória

Números na memória do computador

Little-Endian: Byte menos significativo é armazenado no menor endereço

Exemplo: Número de 32 *bits* (4 *bytes*)

$1025_{10} = 00000000 \ 00000000 \ 00000100 \ 00000001_2$

Endereço

Big-Endian

Little-Endian

00

00000000

00000001

01

00000000

00000100

02

00000100

00000000

03

00000001

00000000

Organização de dados na memória

Vetores na memória

Como fazemos para armazenar um vetor de dados em uma memória endereçada a *byte*?

Resposta:

- Os elementos do vetor são armazenados de forma consecutiva na memória.

Organização de dados na memória

Vetores na memória

Os elementos de um vetor são armazenados de forma consecutiva na memória.

Supondo que cada elemento ocupe TAM bytes, e o vetor se inicie no endereço $BASE$, então o i -ésimo elemento é armazenado nos bytes associados aos endereços

$BASE + i \times TAM$ a $BASE + (i+1) \times TAM - 1$.

- O primeiro elemento ($i=0$) será armazenado nos bytes associados aos endereços $BASE$ a $BASE + TAM - 1$
- O décimo elemento ($i=9$) será armazenado nos bytes associados aos endereços

$BASE + 9 \times TAM$ a $BASE + 10 \times TAM - 1$

Organização de dados na memória

Vetores na memória

Ex: `int v[3] = {9, 8, 1};`

- Supondo que o vetor `v` seja alocado no endereço 0

Endereço	<i>Little-Endian</i>	
00	00001001	v[0] = 9
01	00000000	
02	00000000	
03	00000000	
04	00001000	v[1] = 8
05	00000000	
06	00000000	
07	00000000	
08	00000001	v[2] = 1
09	00000000	
10	00000000	
11	00000000	

Organização de dados na memória

Registros na memória

Como fazemos para armazenar registros (*structs*) em uma memória endereçada a *byte*?

Resposta:

- Os campos do registros são armazenados de forma consecutiva na memória.

Organização de dados na memória

Registros na memória

Exemplo:

```
struct id {  
    int cpf;  
    char nome[256];  
    short idade;  
} fulano;
```

Supondo que o registro fulano
seja armazenado no (a partir do)
endereço zero de memória.

Endereço	<i>Little-Endian</i>	
00	00000000	cpf
01	00000000	
02	00000000	
03	00000000	
04	00000000	nome[0]
05	00000000	nome[1]
...	...	
259	00000000	nome[255]
260	00000000	idade
261	00000000	

Organização de dados na memória

Matrizes na memória

Como fazemos para armazenar uma matriz de dados em uma memória endereçada a *byte*?

Resposta:

- Depende da linguagem de programação:
 - Em 'C': As linhas da matriz são armazenadas de forma consecutiva na memória (uma linha por vez)
 - Organização conhecida como "*row-major order*"
 - Em Fortran: As colunas da matriz são armazenadas de forma consecutiva na memória
 - Organização conhecida como "*column-major order*"