Princípios da Computação

Representação de dados num computador.

isep Instituto Superior de **Engenharia** do Porto

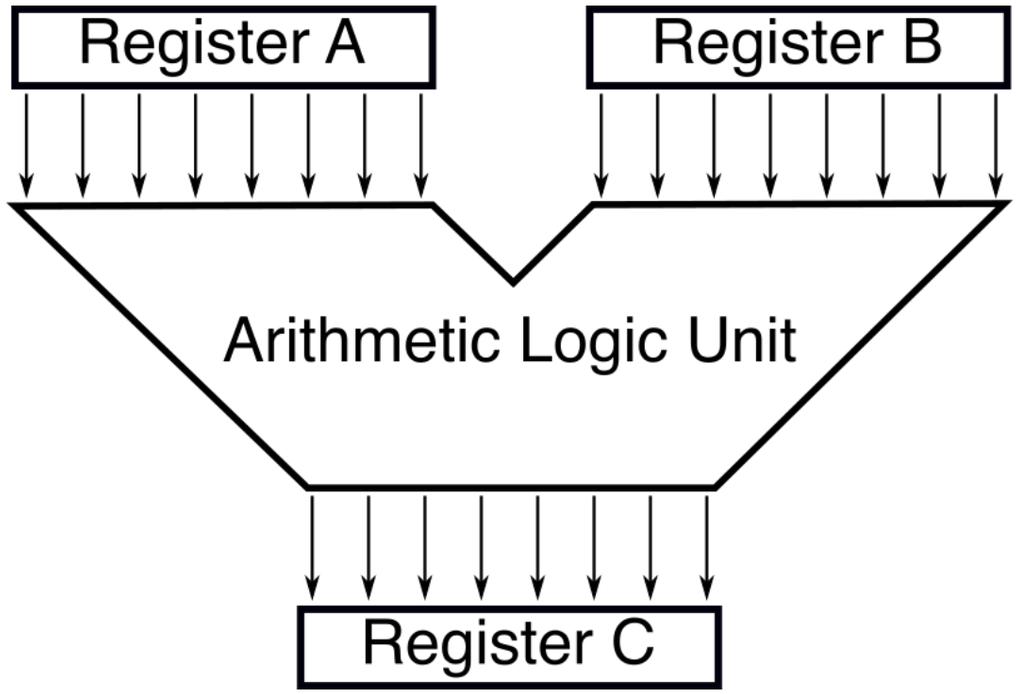
Processadores vs. dados

- Os processadores operam dados em blocos de bits de tamanho fixo.

- Um conjunto de bits desse tamanho é uma **palavra** (*word*).

- As **instruções** de um processador são implementadas através de circuitos que operam sobre **registos** (i.e. memórias internas do processador).

- Num processador, todos os registos de uso genérico têm habitualmente (mas nem sempre) o tamanho de uma palavra.



Processadores vs. dados

- O tamanho da palavra de um dado processador indica a largura de bits da sua **arquitectura**.

- As arquitecturas mais frequentes (neste momento), têm larguras de 8, 16, 32 e 64 bits.

Memória vs. dados

- Os computadores actuais utilizam um sistema de memória endereçável.
    - Composto por células de memória directamente acessíveis para leitura e/ou escrita através de um endereço.
- Os endereços são números inteiros sequenciais.
- Todas as células têm o mesmo tamanho (i.e. número de bits armazenados) - o ***byte***.
    - Desde a década de 1960, a norma *de facto* são 8 bits por célula:
        - 1 byte = 8 bits.

Memória vs. dados

- Um valor é guardado em memória utilizando **uma ou mais células**.
    - Depende da amplitude de valores a representar.
        - Quantos mais bits, maior a quantidade de valores representáveis.
    - Seleccionado pelo programador.
- Tamanhos habituais para um inteiro:
    - 8, 16, 32 e 64 bits

Tamanho (bits)	Endereços ocupados	Valores representáveis
8	1	2^8 = 256
16	2	2^{16} = 65 536
32	4	2^{32} = 4 294 967 296
64	8	2^{64} = 1,84467 x 10^{19}

Números inteiros com sinal

isep Instituto Superior de Engenharia do Porto

Números inteiros com sinal

- Caso prático: com 8 bits podemos representar 256 números inteiros:
    - desde o ***0*** (00000000)
    - até ao ***255*** (11111111)
- E como se representam números negativos?
    - A solução passa por dividir o espaço de representação em dois:
        - um para os positivos e
        - outro para os negativos.

Sinal e grandeza

- Nesta representação, o bit mais à esquerda indica o sinal:
  - **0** - o número é positivo
  - **1** - o número é negativo
- Os restantes bits representam a grandeza do número.

Sinal e grandeza (codificar)

- A representação é construída juntando o bit do sinal aos bits da grandeza.

- Exemplos, com representações de 8 bits:

- +13 => 00001101
  - Sinal : 0 (+)
  - Grandeza : 0001101 (13)

- -13 => 10001101
  - Sinal : 1 (-)
  - Grandeza : 0001101 (13)

Sinal e grandeza (descodificar)

- Separa-se o bit do sinal dos bits da grandeza.
- Exemplos, com representações de 8 bits:

- 01010001 => +81
  - Sinal : 0 (+)
  - Grandeza : 1010001 (81)

- 11010001 => -81
  - Sinal : 1 (-)
  - Grandeza : 1010001 (81)

Sinal e grandeza… é útil?

- Esta codificação é de fácil compreensão para os humanos, mas…
- … não tem qualquer vantagem para o projecto e operação dos processadores!

Complemento para 1

- Nesta representação, **apenas os números negativos** são transformados!

- Codificar um número **negativo**:

    1. Tomar a representação do positivo correspondente.

    2. Complementar todos os bits para obter a representação do número negativo.

- Os números negativos são identificados pelo **1** mais à esquerda!

Complemento para 1 (codificar)

- Exemplos, com representações de 8 bits:

- +13 => 00001101
    - Positivo: não há transformação.

- -13 => 11110010
    - Negativo: há transformação!
        1. Escrever positivo correspondente => 00001101
        2. Complementar os bits e obter -13 codificado => 11110010

Complemento para 1 (descodificar)

- Exemplos, com representações de 8 bits:

- 01000011 => ???

- O **0** mais à esquerda indica que o número é positivo: não está transformado, a leitura é directa.

- **01000011 => +67**

Complemento para 1 (descodificar)

- Exemplos, com representações de 8 bits:
- 10110111 => ???
    - O **1** mais esquerda indica que o número é negativo: está transformado!
    - Complementam-se os bits para se obter o positivo correspondente:
        - 01001000 => +72
        - Então **10110111 => -72**

Complemento para 1: é útil?

- Esta codificação tem o inconveniente de ter duas representações para o número zero: uma positiva e outra negativa.
- Exemplo para codificação em complemento para 1 em 3 bits:

Valor	Representação
0	000
1	001
2	010
3	011
-3	100
-2	101
-1	110
-0	111

Complemento para 2

- Nesta representação, **apenas os números negativos** são transformados!

- Codificar um número **negativo**:

    1. Tomar a representação do positivo correspondente.

    2. Complementar todos os bits (obter o complemento para 1).

    3. Somar 1 ao resultado anterior.

- Os números negativos são identificados pelo **1** mais à esquerda!

Complemento para 2 (codificar)

- Exemplos, com representações de 8 bits:

- +13 => 00001101

  - Positivo: não há transformação.

Complemento para 2 (codificar)

- Exemplos, com representações de 8 bits:

- -13 => 11110011
  - Negativo: há transformação!
    1. Escrever positivo correspondente => 00001101
    2. Complementar os bits => 11110010
    3. Somar 1 e obter -13 codificado => 11110011

Complemento para 2 (descodificar)

- Exemplos, com representações de 8 bits:

- 01000011 => ???

- O **0** mais à esquerda indica que o número é positivo: não está transformado, a leitura é directa.

- **01000011 => +67**

Complemento para 2 (descodificar)

- 10110111 => ???
    - O **1** mais esquerda indica que o número é negativo: está transformado!
    1. Complementam-se os bits: 01001000
    2. Somar 1 e ler positivo correspondente: 01001001 => +73
- Então **10110111 => -73**

Complemento para 2: é útil?

- Esta codificação tem uma única representação do valor zero.
- Eliminar a representação negativa do zero permite representar mais um valor negativo.
- Exemplo para codificação em complemento para 2 em 3 bits:

Valor	Representação
0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111

Complemento para 2

- Este é o método habitualmente utilizado para representar números inteiros com sinal.

- Permite realizar a adição de dois números (positivos ou negativos) sem necessitar da descodificação dos números negativos.

  - Exercício: codificar com 8 bits em complemento para 2, os números 72 e -15, e realizar a soma das duas representações.

Números reais em vírgula flutuante

isep Instituto Superior de Engenharia do Porto

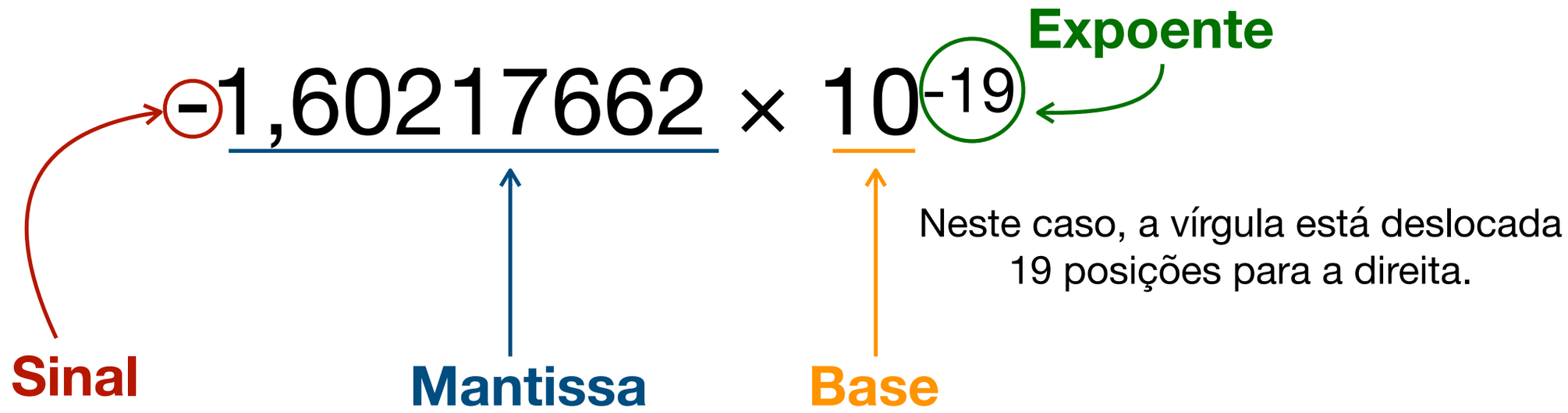
Números reais, números muito pequenos, números muito grandes…

- Como representamos, habitualmente números "exóticos"?

Pi	3,14159265359…
Massa da Terra	$5{,}972 \times 10^{24}$ kg
Carga eléctrica de um electrão	$-1{,}60217662 \times 10^{-19}$ C

Notação científica

- Um número pode ser representado de forma compacta, através da notação científica.



Em binário, é igual!

- Assumindo o número decimal 5,3125 (5 + 1/4 + 1/16).
- A sua representação é binário é:

$$101{,}0101_{(2)} \times 2^{0}$$

- Pode ser representado em notação científica assim:

$$1{,}010101_{(2)} \times 2^{2+0}$$

A vírgula deslocou-se 2 posições para a esquerda…

… incrementando duas vezes o expoente.

Norma IEEE 754

- A norma IEEE 754 estabelece formas de codificar números binários em vírgula flutuante, com sinal.

- Um número em vírgula flutuante segue o seguinte formato:

- O número de bits utilizado depende da precisão seleccionada.

Norma IEEE 754

Precision	Size	Signal bits	Exponent bits	Significand bits	Exponent bias
Half precision	**16 bits**	**1**	**5**	**10**	*15*
Single precision	**32 bits**	**1**	**8**	**23**	*127*
Double precision	**64 bits**	**1**	**11**	**52**	*1023*
Quadruple precision	**128 bits**	**1**	**15**	**112**	*16383*
Octuple precision	**256 bits**	**1**	**19**	**236**	*262143*

Norma IEEE 754 : como se codifica?

- SINAL:
    - 0 se positivo; 1 se negativo.
- EXPOENTE:
    - O expoente é somado ao avanço do expoente para a precisão seleccionada (ver na tabela "*Exponent bias*").
- MANTISSA:
    - São representados somente todos os bits à direita da vírgula.
    - O bit à esquerda da vírgula é sempre 1, pelo que se ganha um bit na precisão.

IEEE 754
Exemplo de codificação para precisão simples

- Considerando o número do exemplo anterior…

$$+1{,}010101_{(2)} \times 2^{2}$$

Sinal (1 bit)	Positivo	0
Expoente (8 bits)	2 + 127 = 129	1000 0001
Mantissa (23 bits)	Bits à direita da vírgula	0101010 00000000 00000000

01000000 1**0101010 00000000 00000000**

Norma IEEE 754 : como se descodifica?

- O processo é exactamente o inverso ao de codificação.
- Não esquecer que…
  - é necessário subtrair o avanço do expoente para a precisão seleccionada (ver na tabela "*Exponent bias*").
  - não esquecer de representar o bit mais significativo da mantissa (oculto na codificação).

IEEE 754
Exemplo de descodificação para precisão simples

10111110 11100000 00000000 00000000

Sinal (1 bit)	1	- (negativo)
Expoente (8 bits)	1111101 (i.e.125)	125 - 127 = -2
Mantissa (23 bits)	Bits à direita da vírgula	1,1100000 00000000 00000000

$$-1{,}110000_{(2)} \times 2^{-2} = -0{,}01110000_{(2)} = -0{,}4375$$

Representação de texto

isep Instituto Superior de Engenharia do Porto

Codificação de texto

- Um computador digital binário apenas aceita dois símbolos: *0* e *1*.

- Como é que o texto é então representado num computador?

Codificação de texto

- Um texto é representado por uma sequência de números!
    - Cada **número** representa um **carácter**.
    - Um conjunto de associações número-carácter é um **código**.
    - Ou seja, o texto é **codificado** numa sequência de números.

ASCII (1963)

- *American Standard Code for Information Interchange*

- Padrão americano para teletipos, de 7 bits (128 códigos):
  - 33 códigos de controlo
  - 95 caracteres imprimíveis (caracteres latinos, sem acentos)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

ISO 8859 (1985)

- Conjunto de extensões ao ASCII, para suportar as diversas grafias mundiais.
- 8 bits (256 símbolos)
    - Acrescenta 128 caracteres imprensáveis ao ASCII.
- 16 partes, ou extensões.
- A grafia portuguesa é suportada pela parte 1: **ISO 8859-1**.
- http://www.open-std.org/JTC1/SC2/WG3/docs/n411.pdf

Unicode (1991)

- Norma que utiliza até 4 bytes para representar 144697 caracteres:
    - Cobre 159 grafias (actuais ou históricas), símbolos e emojis.
    - Os primeiros 256 símbolos do UTF-8 coincidem com a norma ISO 8859-1.
- O UTF-8 é a norma mais utilizada na WWW.