Introdução aos Sistemas Digitais

Aula 2

Quantidades com sinal
Operações aritméticas
Códigos binários

ieeta

Operações aritméticas em bases ≠ 10

Os processos aritméticos em outros sistemas de numeração obedecem às mesmas regras básicas existentes no sistema decimal. Só as tabuadas respetivas (de soma, de multiplicação, etc.) é que mudam.

Sejam x_1x_0 e y_1y_0 dois valores de 2 dígitos em base *r*

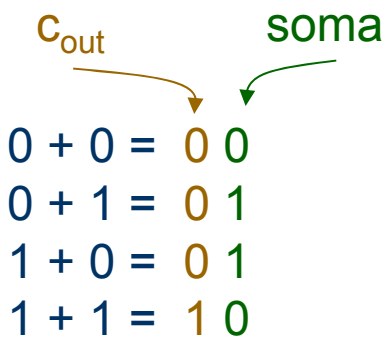
$$\begin{array}{r} x_1x_0 \\ +\ y_1y_0 \\ \hline s_2s_1s_0 \end{array}$$

Se $x_i+y_i \geq r$, $\mathbf{s_i = x_i+y_i - r}$ e há um transporte para a casa seguinte (i+1)

Se $x_i+y_i < r$, $\mathbf{s_i = x_i+y_i}$ e não há nenhum transporte para a casa seguinte

ieeta

Adição de números binários



c_{in}	x	y	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Exemplo:

	1	1	0	0	0	0	1	
	1	0	1	0	1	1	0	1
+	1	1	1	0	0	0	0	1
1	1	0	0	0	1	1	1	0

Adição de números em bases 8 e 16

Exemplos:

base 8

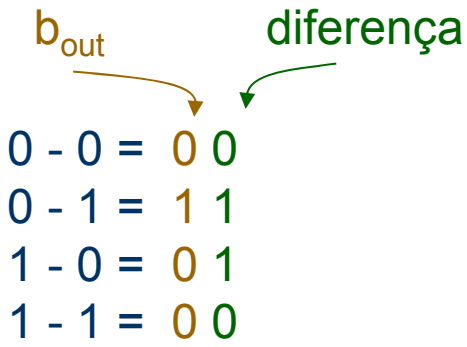
```
  0 0 0          1 1 1
  3 0 4 1        3 4 5 6
+ 1 7 3 2      + 1 7 3 4
-------------  -------------
  4 7 7 3        5 4 1 2
```

base 16

```
  1 0 0          1 1 0
  3 A 4 1        3 F A A
+ 1 7 8 2      + B 7 E 4
-------------  -------------
  5 1 C 3        F 7 8 E
```

ieeta

Subtração de números binários



b_{in}	x	y	b_{out}	d
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Exemplos:

```
  0 1 1 1 1 0 0          1 1 1
  1 1 1 0 0 0 0 1        1 0 0 0
- 1 0 1 0 1 1 0 1      - 0 0 1 1
-----------------      ---------
  0 0 1 1 0 1 0 0        0 1 0 1
```

ieeta

Subtração de números em bases 8 e 16

Exemplos:

base 8

```
    1 0 1          1 1 1
    3 0 4 1        6 0 0 0
  - 1 7 3 2      - 1 5 7 7
  ---------      ---------
    1 1 0 7        4 2 0 1
```

base 16

```
    0 1 1          1 1 1
    3 A 4 1        B 0 0 0
  - 1 7 8 2      - A 7 E 4
  ---------      ---------
    2 2 B F        0 8 1 C
```

Representação de números negativos

Sinal e módulo (sinal e magnitude)

Complemento para 1

Complemento para 2

Sinal e módulo

Sinal e módulo – a representação inclui o módulo e um símbolo adicional para indicar se o número é positivo ou negativo.

$00101_2 = +5_{10}$ $\quad$ $10101_2 = -5_{10}$

$00000_2 = +0_{10}$ $\quad$ $10000_2 = -0_{10}$

Um inteiro de *n* bits representado em sinal e módulo pode tomar valores situados na gama $[-(2^{n-1}-1), +(2^{n-1}-1)]$.

Por exemplo, com 4 bits pode-se representar em sinal e módulo valores [-7,7], i.e. 7 positivos, 7 negativos e 2 zeros.

Adição de números em sinal e módulo

É bastante difícil construir um circuito digital que some dois números representados em sinal e módulo dado que é necessário comparar as magnitudes dos operandos para determinar o sinal do resultado.

Exemplos:

1+3 = 4

0001 + 0011 =

```
  1 1
  0 0 1
+ 0 1 1
-------
0 1 0 0
```

-1+(-3) = -4

1001 + 1011 =

```
  1 1
  0 0 1
+ 0 1 1
-------
1 1 0 0
```

1+(-3) = -2

0001 - 0011 =

1 < 3 =>

```
  0 0
  0 1 1
- 0 0 1
-------
1 0 1 0
```

3+(-1) = 2

0011 - 0001 =

3 > 1 =>

```
  0 0
  0 1 1
- 0 0 1
-------
0 0 1 0
```

ieeta

Adição de números em sinal e módulo (cont.)

Algoritmo:

```
if sinal(A) = sinal(B) then
      begin
            |soma| = |A| + |B|
            sinal(soma) = sinal(A)
      end
else
      if |A| > |B| then
            begin
                  |soma| = |A| - |B|
                  sinal(soma) = sinal(A)
            end
      else
            begin
                  |soma| = |B| - |A|
                  sinal(soma) = sinal(B)
            end
```

Complemento para 1

Códigos de complemento – o número é negado calculando o seu complemento de acordo com as regras estabelecidas (complemento para a base, complemento para a base diminuído, etc.).

Complemento para a base diminuído – o complemento de um número de *n* dígitos na base *r* obtém-se subtraindo o número de r^n-1.

Exemplos: Complemento para 9: $1234 \rightarrow 10^4 - 1 - 1234 = 8765$

Complemento para 1: $1001 \rightarrow 2^4 - 1 - 1001 = 1111 - 1001 = 0110$

Para obter o complemento para a base diminuído de um número D = dd...d de n bits:

$$r^n - 1 - D$$

$$r^n - 1 = mm..m, (m = r - 1)$$

$$r^n - 1 - D = (r - 1 - d)(r - 1 - d)..(r - 1 - d)$$

$1234 \rightarrow (9-1)(9-2)(9-3)(9-4) = 8765 = 8765$

$1001 \rightarrow (1-1)(1-0)(1-0)(1-1) = 0110 = 0110$

Adição de números em complemento para 1

Números em complemento para 1 podem ser adicionados aplicando regras habituais de adição binária. Transportes para além do bit mais significativo devem ser somados ao resultado (para evitar que o zero seja contado duas vezes).

Exemplos:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-7	1	0	0	0
-6	1	0	0	1
-5	1	0	1	0
-4	1	0	1	1
-3	1	1	0	0
-2	1	1	0	1
-1	1	1	1	0
-0	1	1	1	1

2+3 = 5

```
  0 1 0
  0 0 1 0
+ 0 0 1 1
---------
  0 1 0 1
```

2+(-5) = -3

```
  0 1 0
  0 0 1 0
+ 1 0 1 0
---------
  1 1 0 0
```

4+(-3) = 1

```
1 1 0 0
  0 1 0 0
+ 1 1 0 0
---------
  0 0 0 0
+       1
---------
  0 0 0 1
```

-4+(-3) = -7

```
1 0 0 0
  1 0 1 1
+ 1 1 0 0
---------
  0 1 1 1
+       1
---------
  1 0 0 0
```

Overflow

Overflow ocorre se a soma de dois números positivos produzir um resultado negativo, ou se a soma de dois números negativos produzir um resultado positivo.

Exemplos:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-7	1	0	0	0
-6	1	0	0	1
-5	1	0	1	0
-4	1	0	1	1
-3	1	1	0	0
-2	1	1	0	1
-1	1	1	1	0
-0	1	1	1	1

5+3 = 8 (-7 ?)

```
  0 1 1 1
    0 1 0 1
+   0 0 1 1
  ---------
    1 0 0 0
```

-7+(-2) = -9 (6 ?)

```
  1 0 0 0
    1 0 0 0
+   1 1 0 1
  ---------
    0 1 0 1
+           1
  ---------
    0 1 1 0
```

Complemento para 2

Complemento para a base – o complemento de um número de *n* dígitos na base *r* obtém-se subtraindo o número de r^n.

Exemplos: Complemento para 10: $1234 \rightarrow 10^4 - 1234 = 8766$

Complemento para 2: $1001 \rightarrow 2^4 - 1001 = 10000 - 1001 = 0111$

Para obter o complemento para a base de um número D = dd...d de n bits:

$$r^n - D = (r^n - 1) - D + 1$$

$$r^n - 1 = mm..m, (m = r - 1)$$

$$r^n - D = (r - 1 - d)(r - 1 - d)..(r - 1 - d) + 1$$

$$1234 \rightarrow (9-1)(9-2)(9-3)(9-4) + 1 = 8765 + 1 = 8766$$

$$1001 \rightarrow (1-1)(1-0)(1-0)(1-1) + 1 = 0110 + 1 = 0111$$

Para obter o complemento para 2 de um número binário pode-se, começando do lado menos significativo copiar todos os bits até (e inclusive) encontrar o primeiro 1, a partir daí inverter todos os bits:

$$100_{10} = 01100100_2 \rightarrow 10011100_2 = -100_{10}$$

Adição de números em complemento para 2

Números em complemento para 2 podem ser adicionados aplicando regras habituais de adição binária ignorando transportes para além do bit mais significativo.

Exemplos:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

2+3 = 5

```
  0 1 0
  0 0 1 0
+ 0 0 1 1
---------
  0 1 0 1
```

2+(-5) = -3

```
  0 1 0
  0 0 1 0
+ 1 0 1 1
---------
  1 1 0 1
```

4+(-3) = 1

```
  1 1 0 0
    0 1 0 0
+   1 1 0 1
-----------
    0 0 0 1
```

-4+(-3) = -7

```
  1 1 0 0
    1 1 0 0
+   1 1 0 1
-----------
    1 0 0 1
```

Adição é mais simples do que em complemento para 1.

Overflow

Overflow ocorre se a soma de dois números positivos produzir um resultado negativo, ou se a soma de dois números negativos produzir um resultado positivo.

Exemplos:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

5+3 = 8 (-8 ?)

$$\begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ + \; 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 \end{array}$$

-7+(-2) = -9 (7 ?)

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ + \; 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \end{array}$$

Overflow ocorre se no bit mais significativo $c_{in} \neq c_{out}$.

Subtração de números em complemento para 2

Números em complemento para 2 podem ser subtraídos complementando o segundo operando e realizando a operação de soma:

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

Exemplos:

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

2-3 = 2+(-3) = -1

```
  0 0 0
  0 0 1 0
+ 1 1 0 1
---------
  1 1 1 1
```

-5 - 6 = -5+(-6) = -11

```
  1 0 1 0
    1 0 1 1
+   1 0 1 0
-----------
  0 1 0 1
```

overflow

Para somar e subtrair números em complemento para 2 precisamos de apenas um circuito somador.

Códigos de complemento binários

Um inteiro de *n* bits representado em complemento para 1 pode tomar valores situados na gama $[-(2^{n-1}-1), +(2^{n-1}-1)]$.

Por exemplo, com 4 bits pode-se representar em complemento para 1 valores [-7,7], i.e. 7 positivos, 7 negativos e 2 zeros.

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-7	1	0	0	0
-6	1	0	0	1
-5	1	0	1	0
-4	1	0	1	1
-3	1	1	0	0
-2	1	1	0	1
-1	1	1	1	0
-0	1	1	1	1

Um inteiro de *n* bits representado em complemento para 2 pode tomar valores situados na gama $[-(2^{n-1}), +(2^{n-1}-1)]$.

Por exemplo, com 4 bits pode-se representar em complemento para 2 valores [-8,7], i.e. 7 positivos, 8 negativos e 1 zero.

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
-8	1	0	0	0
-7	1	0	0	1
-6	1	0	1	0
-5	1	0	1	1
-4	1	1	0	0
-3	1	1	0	1
-2	1	1	1	0
-1	1	1	1	1

Códigos de complemento binários (cont.)

O bit mais significativo indica o sinal do número: para números negativos é igual a 1, para positivos – a 0.

Em complemento para 1, o peso do bit mais significativo é $-(2^{n-1}-1)$.

Exemplo: $1010 = -(2^3-1) + 2^1 = -7 + 2 = -5$

Em complemento para 2, o peso do bit mais significativo é -2^{n-1}.

Exemplo: $1010 = -2^3 + 2^1 = -8 + 2 = -6$

Conversão de códigos de complemento

Conversão de um número de n bits (que está em complemento para 2 ou complemento para 1) para um número de m bits:

Se n < m – realiza-se a extensão de sinal, i.e. são adicionadas m-n cópias do bit de sinal à esquerda do número:

n = 5, m = 8 — 00101 = 00000101
11110 = 11111110

Se n > m – removem-se n-m bits mais significativos – o resultado só será válido se todos estes bits são iguais ao bit de sinal do resultado:

n = 5, m = 3 — 00101 = 101 – resultado não válido
11110 = 110 – resultado válido

Códigos

Código – conjunto de sequências de *n* bits, em que cada sequência representa um determinado valor

Palavra do código – uma dada sequência de *n* bit

n – comprimento do código

m – número de valores a codificar

$$n \geq \lceil log_2(m) \rceil$$

andar	codificação	codificação	codificação
cave	000	000	000001
r/c	001	001	000010
1º andar	010	011	000100
2º andar	011	010	001000
3º andar	100	110	010000
4º andar	101	111	100000

Códigos BCD

BCD (*Binary Coded Decimal*) – serve para codificar algarismos decimais

algarismo decimal	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

código binário natural

- regular – comprimento de palavra é fixo
- ponderado – cada um dos bits de palavras de código tem um determinado peso
- descontínuo – palavras consecutivas diferem em mais do que 1 bit
- não cíclico – 1ª e última palavras do código não são adjacentes
- redundante – nem todas as combinações possíveis de bits são usadas

Exemplo:

$25_{10} = 11001_2$

$25_{10} = 00100101_{BCD}$

Códigos BCD (cont.)

algarismo decimal	BCD (8421)	2421 (AIKEN)	Excess-3 (XS3)	1-out-of-10
0	0000	0000	0011	0000000001
1	0001	0001	0100	0000000010
2	0010	0010	0101	0000000100
3	0011	0011	0110	0000001000
4	0100	0100	0111	0000010000
5	0101	1011	1000	0000100000
6	0110	1100	1001	0001000000
7	0111	1101	1010	0010000000
8	1000	1110	1011	0100000000
9	1001	1111	1100	1000000000
	palavras não usadas:			
	1010	0101	0000	0000000000
	1011	0110	0001	0000000011
	1100	0111	0010	0000000101
	1101	1000	1101	0000000110
	1110	1001	1110	0000000111
	1111	1010	1111	...

Autocomplementares (complemento para 9 de cada dígito decimal obtém-se subtraindo-o de 9)

$$x = (b_3 b_2 b_1 b_0)$$

$$\overline{x} = 9 - x = (\overline{b}_3 \overline{b}_2 \overline{b}_1 \overline{b}_0)$$

Exemplos:

AIKEN:

$\overline{x}$ = 3 = (0011)

$\overline{x}$ = 9 – 3 = 6 = (1100)

Excess-3:

$\overline{x}$ = 3 = (0110)

$\overline{x}$ = 9 – 3 = 6 = (1001)

Códigos autocomplementares

Num código decimal binário autocomplementar ponderado o somatório dos pesos dos seus bits deve ser 9.

N_1 e N_2 – palavras complementares dum código decimal binário ponderado de comprimento *n*.

$$N_1 = \sum_{i=0}^{n-1} d_i * p_i, \quad N_2 = \sum_{i=0}^{n-1} \bar{d}_i * p_i, \quad d_i \in \{0,1\}$$

$$N_2 = 9 - N_1$$

$$9 = N_1 + N_2 = \sum_{i=0}^{n-1} (d_i + \bar{d}_i) * p_i$$

$$\sum_{i=0}^{n-1} p_i = 9$$

algarismo decimal	4221	84-2-1	75-41
0	0000	0000	0000
1	0001	0111	0001
2	0010	0110	0111
3	0011	0101	1010
4	0110	0100	1011
5	1001	1011	0100
6	1100	1010	0101
7	1101	1001	1000
8	1110	1000	1110
9	1111	1111	1111

Código de Gray

1 bit	2 bits	3 bits	4 bits
0	00	000	0000
1	01	001	0001
	11	011	0011
	10	010	0010
		110	0110
		111	0111
		101	0101
		100	0100
			1100
			1101
			1111
			1110
			1010
			1011
			1001
			1000

- regular
- não ponderado
- contínuo
- cíclico
- não redundante
- refletido
- distância de Hamming entre palavras consecutivas = 1

Distância de Hamming – número de bits em que diferem duas palavras de código

ieeta

Construção do código de Gray

- O código de Gray de 1 bit é composto por duas palavras: 0 e 1.

- No código de Gray de *n* bits as primeiras 2^{n-1} palavras são iguais às do código de Gray de *n*-1 bits precedidas de um ‘0’ colocado na posição do bit mais significativo.

- As últimas 2^{n-1} palavras são iguais às palavras do código de Gray de *n*-1 bits, escritas pela ordem inversa e precedidas de um ‘1’ colocado na posição do bit mais significativo.

Construção do código de Gray

Obtenção de uma palavra do código de Gray de *n* bits a partir da palavra correspondente do código binário de *n* bits:

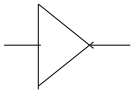
- Adicionar à palavra do código binário um bit à esquerda e atribuir-lhe o valor '0'.
- Numerar todos os bits do código binário da direita para a esquerda.
- Atribuir valor '1' ao bit *i* do código de Gray se os bits *i* e *i*+1 da palavra binária são diferentes.
- Atribuir valor '0' ao bit *i* do código de Gray se os bits *i* e *i*+1 da palavra binária são iguais.

Obtenção de uma palavra do código binário de *n* bits a partir da palavra correspondente do código de Gray de *n* bits:

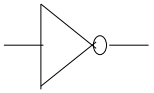
- Numerar todos os bits do código de Gray da esquerda para a direita.
- Atribuir o valor do bit 1 do código de Gray ao bit 1 do código binário.
- Bit *i* (*i*=2,3,…,*n*) do código binário é igual à soma exclusiva (XOR) do bit *i*-1 do código binário e do bit *i* do código de Gray.

Portas lógicas

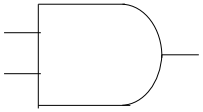
buffer



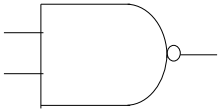
NOT



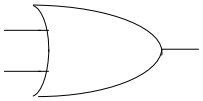
AND



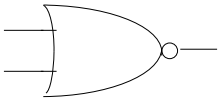
NAND



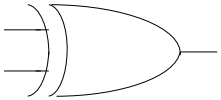
OR



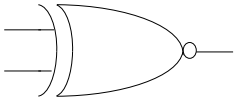
NOR



XOR



XNOR



$$x \oplus y$$

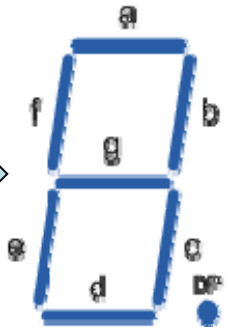
x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

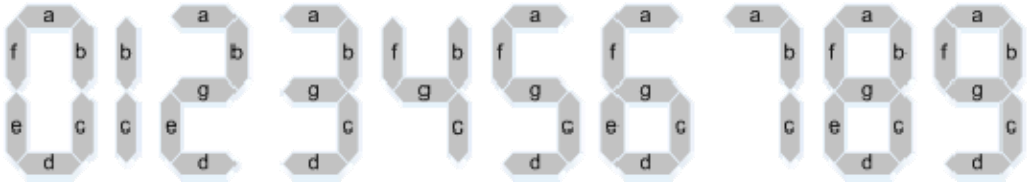
$$\overline{x \oplus y}$$

x	y	x XNOR y
0	0	1
0	1	0
1	0	0
1	1	1

Código para *display* de 7 segmentos

0,1,…,9 ?





BCD	número	segmentos individuais						
		a	b	c	d	e	f	g
0000	0	1	1	1	1	1	1	0
0001	1	0	1	1	0	0	0	0
0010	2	1	1	0	1	1	0	1
0011	3	1	1	1	1	0	0	1
0100	4	0	1	1	0	0	1	1
0101	5	1	0	1	1	0	1	1
0110	6	1	0	1	1	1	1	1
0111	7	1	1	1	0	0	0	0
1000	8	1	1	1	1	1	1	1
1001	9	1	1	1	1	0	1	1
101x	x	x	x	x	x	x	x	x
11xx	x	x	x	x	x	x	x	x

Códigos de carateres

ASCII – *American Standard Code for Information Interchange*

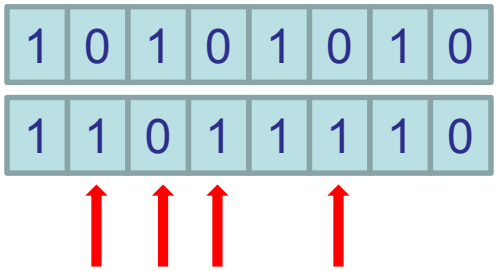
		$b_6b_5b_4$ (column)							
$b_3b_2b_1b_0$	Row (hex)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	‘	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	’	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

ieeta

Distância de Hamming

Dadas duas palavras de um código de comprimento *n* bits, a distância de *Hamming* entre essas palavras é igual ao número de bits do mesmo índice que mudam de valor entre essas duas palavras.

Exemplo:



Distância de Hamming = 4

Erros em sistemas digitais

Um erro em sistemas digitais é a corrupção de dados do valor correto para um valor diferente. Erros podem ocorrer tanto em sistemas de transmissão de informação digital (ruído) como em sistemas de armazenamento (memória, discos rígidos).

Erro singular – só um bit de dados é corrompido

Erros múltiplos – 2 ou mais bits de dados são corrompidos

Erros múltiplos são normalmente menos prováveis que erros singulares.

Deteção de erros

Um código permite deteção de erros se a corrupção de uma palavra resulta numa nova palavra que não faz parte do código.

Um sistema que permite deteção de erros só gera, transmite e guarda palavras de código válidas.

Se uma sequência de bits é uma palavra válida de código, é assumido que está correta.

Se uma sequência de bits é uma palavra não válida de código, é assumido que está errada.

Deteção de erros (cont.)

É possível detetar todos os erros singulares se a distância mínima entre todos os possíveis pares de palavras de código ≥ 2. Tal código chama-se código de distância 2.

Exemplo: para construir um código de 2^n palavras que deteta erros singulares são precisos pelo menos n+1 bits

bit de paridade	bits de informação
0	000
1	001
1	010
0	011
1	100
0	101
0	110
1	111

Uma palavra válida contém sempre número par de 1s.

1001 -> 1011 -> erro detetado

Não deteta erros de 2 bits!

1001 -> 1010 -> erro não detetado

Deteta todos os erros em número ímpar de bits.

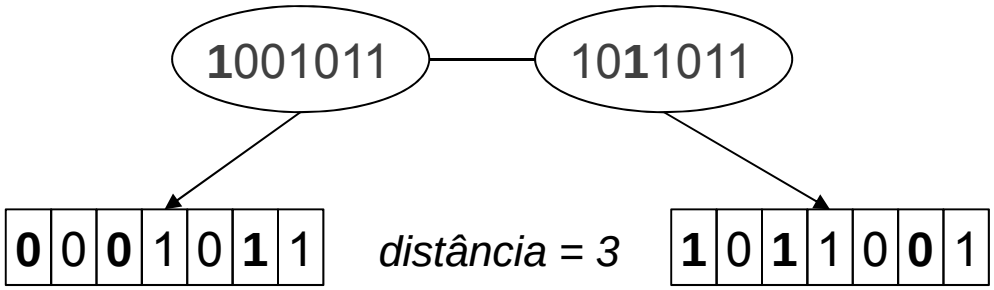
1001 -> 1110 -> erro detetado

Deteção de erros (cont.)

Para detetar erros múltiplos são necessários códigos de distância maior que 2.

Para criar um código de distância maior que 2 é necessário utilizar mais que um bit de paridade.

Exemplo: código de distância 3 permite deteção de 2 erros



A palavra inválida 1001011 (com um erro) está mais próxima da palavra 0001011 do que da qualquer outra palavra de código.

Torna-se possível corrigir o erro modificando a palavra recebida para a palavra mais próxima no código.

Quantificação de informação

Bit	– *b*	– 0 ou 1
Byte	– *B*	– 8 bits
Nibble		– 4 bits
Word		– 8, 16, 32, 64 ... bits (depende do contexto)

1 K/k	10^3	≈	2^{10}	(*kilo*)
1 M	10^6	≈	2^{20}	(*mega*)
1 G	10^9	≈	2^{30}	(*giga*)
1 T	10^{12}	≈	2^{40}	(*tera*)

IEEE 1541-2002:

Ki	2^{10} = 1 024	(*kibi*)
Mi	2^{20} = 1 048 576	(*mebi*)
Gi	2^{30} = 1 073 741 824	(*gibi*)
Ti	2^{40} = 1 099 511 627 776	(*tebi*)
Pi	2^{50} = 1 125 899 906 842 624	(*pebi*)
Ei	2^{60} = 1 152 921 504 606 846 976	(*exbi*)

ieeta

Exercícios

Represente números seguintes em sinal e módulo, complemento para 1 e complemento para 2 com 8 bits:

39_{10} -22_{10}

Calcule o resultado das operações seguintes em complemento para 2 com 8 bits de representação. Identifique os casos em que ocorre *overflow*.

```
  1 0 1 0 1 0 1 0
+ 0 1 0 1 0 1 0 1
-----------------
```

```
  0 0 0 0 0 0 0
  1 0 1 0 1 0 1 0
+ 0 1 0 1 0 1 0 1
-----------------
  1 1 1 1 1 1 1 1
```

```
  0 1 1 1 1 1 1 1
+ 0 0 1 1 1 1 1 1
-----------------
```

```
  0 1 1 1 1 1 1 1
  0 1 1 1 1 1 1 1
+ 0 0 1 1 1 1 1 1
-----------------
  1 0 1 1 1 1 1 0
```

```
  1 1 0 1 0 0 1 0
- 0 1 1 0 1 1 0 1
-----------------
```

```
  1 1 0 1 1 0 1
  1 1 0 1 0 0 1 0
- 0 1 1 0 1 1 0 1
-----------------
  0 1 1 0 0 1 0 1
```

ieeta

Exercícios (cont.)

Quantos bits de informação se pode armazenar numa *pen* de 16 GB?

Quantas fotos digitais seria possível quardar numa *pen* de 8 GB se cada foto ocupasse 4000 x 3000 *píxeis* e cada *píxel* fosse representado por 24 bits?

Os números seguintes são escritos em complemento para 2. Represente-os em complemento para 1 com 8 bits.

001011 110111

Exprima nos sistemas decimal, binário e hexadecimal o valor da maior quantidade inteira não negativa que pode representar num registo com capacidade de armazenamento de 2 algarismos octais.

Assumindo que a quantidade seguinte está codificada em complemento para 2 indique o seu equivalente decimal:

111111111111111111111001

Exercícios (cont.)

Quantos bits são necessários para codificar em BCD_{8421} o valor 123456_{10}? Este código é ponderado? É autocomplementar?

Represente os seguintes números nos códigos BCD natural, AIKEN, BCD de excesso 3 e no código de Gray.

108_{10} $= 000100001000_{BCD}$
$= 000100001110_{AIKEN}$
$= 010000111011_{XS3}$
$= 1101100_2$
$= 1011010_{GRAY}$

33_8 $=27_{10}$
$= 00100111_{BCD}$
$= 00101101_{AIKEN}$
$= 01011010_{XS3}$
$= 011011_2$
$= 010110_{GRAY}$

Prove que um número em complemento para 2 pode ser representado com mais bits através de extensão do bit de sinal.

Determine a distância de Hamming entre palavras seguintes:

011010101011
000010101011 $= 2$

ieeta