Eletrônica Digital

Sistema de Numeração e Conversão entre Sistemas.

Prof. Rômulo Calado Pantaleão Camara

Carga Horária: 60h

Representação de grandeza com sinal

- ✓ O bit mais significativo representa o sinal:
  - **0** (indica um **número positivo**)
  - **1** (indica um **número negativo**)
- ✓ Os demais bits representam a **grandeza (magnitude)**.

sinal	magnitude

- ✓ O valor dos bits usados para representar a magnitude independe do sinal (sendo o número positivo ou negativo, a representação binária da magnitude será a mesma).

Exemplos: (8 bits)

- ❑ $00101001_2 = +41_{10}$
- ❑ $10101001_2 = -41_{10}$

Exemplo

Valor decimal	Valor binário com 8 bits (7 + bit de sinal)
+9	00001001
-9	10001001
+127	01111111
-127	11111111

Assim, uma representação em binário com **n** bits teria disponível para a representação do número ***n-1*** bits (o bit mais significativo representa o sinal).

Representação de grandeza com sinal

- ✓ Apresenta uma grande **desvantagem**: ela exige um grande número de testes para se realizar uma simples soma de dois números inteiros.

- ✓ Requer que na ULA existam dois circuitos distintos para a adição e a subtração.

- ✓ Existem **duas representações** para o zero.

Representação Complemento de 2

- **Representação de números inteiros positivos**
  - igual à representação de grandeza com sinal.
- **Representação de números inteiros negativos**
  - mantém-se os bits menos significativos da direita para a esquerda até à ocorrência do primeiro bit igual a 1 (inclusive), sendo os bits restantes complementados de 1.
  - Esta operação equivale a realizar: complemento de 1 + 1.

Exemplo : (8 bits)

$00001100_2 = +12_{10}$

$11110100_{c2} = -12_{10}$

Exemplo : (8 bits)

$00101001_2 = +41_{10}$

$11010111_{c2} = -41_{10}$

Representação Complemento de 2

- ✓ **Exemplo:** Números inteiros codificados em **binário de 8 bits** em um sistema que utiliza complemento de 2:

(-128, -127, ..., -2. -1, 0, +1, +2,..., +127)

{10000000, 10000001, ..., 11111110, 11111111, 00000000, 00000001, 00000010, ..., 01111111}

- ✓ Bit mais significativo ⇨ informação de sinal

(0 = positivo e 1 = negativo)

Representação Complemento de 2

- Requer **um só circuito** (somador) para fazer a adição e a subtração.
- Há apenas uma representação para o valor **0** (disponibilidade para **mais uma representação)** - mais um número negativo pode ser representado (para 8 bits, pode-se representar o número $-128_{10} \Rightarrow 10000000_2$) .
- A quantidade de números positivos é **diferente** da quantidade de números negativos.

Outras formas de representação

- ✓ **Complemento de 1**: para negar o valor de um número deve-se inverter os bits do sinal (obsoleta) e **Excesso de 2^{m-1}**: representação do número é dada pela soma de seu valor absoluto com 2^{m-1}. Exemplo: Um sistema de 8 bits é chamado de excesso de 128 e um número é armazenado com seu valor real somado a 128. Ex.:-3=01111101_2 (-3+128=125)

- ❑ Exercício de fixação:

- ✓ Escreva os números decimais abaixo nas seguintes representações: sinal e magnitude; representação em complemento de 1; representação em complemento de 2 e excesso de 128 (utilizando 8 bits, se existir representação).
  a) -1
  b) -20
  c) -127
  d) -128

Exemplos

- ✓ **Números negativos de 8 bits expressos em 4 sistemas diferentes**

N (decimal)	N (binário)	-N (sinal-magnitude)	-N (comple-mento de 1)	-N (comple-mento de 2)	-N (excesso de 128)
1	00000001	10000001	11111110	11111111	01111111
2	00000010	10000010	11111101	11111110	01111110
3	00000011	10000011	11111100	11111101	01111101
4	00000100	10000100	11111011	11111100	01111100
10	00001010	10001010	11110101	11110110	01110110
20	00010100	10010100	11101011	11101100	01101100
100	01100100	11100100	10011011	10011100	00011100
127	01111111	11111111	10000000	10000001	00000001
128		Não existe represen-tação	Não existe represen-tação	10000000	00000000

Representação de Números Reais

- ✓ Em alguns tipos de cálculo, a faixa de variação dos números envolvidos é muito grande.
- ✓ **Exemplo**:
  - 1) Massa do elétron - da ordem de 9×10^{-28} gramas
  - 2) Massa do Sol - aproximadamente igual a 2×10^{33} gramas
  - Faixa de variação: $> 10^{60}$
  - Exemplo de representação (34 dígitos à esquerda do ponto decimal e 28 dígitos à direita do mesmo)

1) 0000000000000000000000000000000000.0000000000000000000000000009

2) 2000000000000000000000000000000000.0000000000000000000000000000

- ✓ Como representar esses números num computador?

Representação de Números Reais

- ✓ Forma usual de representação de números reais: **parte inteira, vírgula (ou ponto), parte fracionária.**

- ✓ Esta representação, embora cômoda para cálculos no papel, não é adequada para processamento no computador.

- ✓ **Exemplo: 45,724**
- ✓ O número **45,724** pode ser expresso como:
  - **$45{,}724 \times 10^{0}$**
  - **45724×10^{-3}**
  - **$0{,}45724 \times 10^{2}$**

Representação de Números Flutuante

- ✓ É necessário o uso de um sistema de representação de números no qual a faixa de variação dos números seja independente do número de dígitos significativos dos números representados.
- ✓ Uma maneira de separar a faixa de variação dos números de sua precisão consiste em representá-lo na notação científica.

$$n = f \times 10^{e}$$

f - fração ou significando (ou mantissa)
e - expoente (inteiro positivo ou negativo)

Representação de Números Flutuante

- ✓ Qualquer número (inteiro ou fracionário) pode ser expresso no formato **número x base^{expoente}**, podendo-se variar a posição da vírgula e o expoente.
- ✓ Denominação (computacional): **representação em ponto flutuante** (o ponto varia sua posição, modificando, em consequência, o valor representado).
- ✓ Representação pode variar ("**flutuar**") a posição da vírgula, ajustando a potência da base.

Representação de Números Flutuante

- ✓ **Exemplos**:
  - $3,14 = 0,314 \times 10^{-1} = 3,14 \times 10^{0}$
  - $0,000001 = 0,1 \times 10^{-5} = 1,0 \times 10^{-6}$
  - $1941 = 0,1941 \times 10^{4} = 1,941 \times 10^{3}$

- ✓ A **faixa de variação** dos números é determinada pela quantidade de dígitos do expoente e a **precisão** é determinada pela quantidade de dígitos do significando.

Representação de Números Flutuante

- ✓ **Forma normalizada:** usa um único dígito antes da vírgula, diferente de zero (*).
- ✓ Na representação computacional de números em ponto flutuante, a representação normalizada é, em geral, melhor que a não-normalizada.
  - **Forma normalizada**: só existe uma forma de representar um número.
  - **Forma não normalizada**: um mesmo número pode ser representado de diversas maneiras.

(*) Padrão IEEE 754 para números em ponto flutuante – **significando normalizado** – começa com um bit 1, seguido de um ponto (vírgula) binário e pelo resto do significando (número = ± 1,_ _ ... x 2^{exp})

Mantissa normalizada - começa com o ponto (vírgula) binário seguido por um bit 1 e pelo resto da mantissa (bit antes da vírgula igual a zero).

Representação de Números Flutuante

Ilustração:

- ✓ No sistema binário:
  - **110101** = **110,101x2^3** = **1,10101x2^5** = **0,0110101x2^7**
- ✓ Números armazenados em um computador - os expoentes serão também gravados na base dois
  - Como $3_{10} = 11_2$ e $7=111_2$
  - $110,101 \times (10)^{11} = 1,10101x(10)^{101} = 0,0110101x(10)^{111}$
- ✓ Representação normalizada - há apenas um “1” antes da vírgula
  - **Exemplo**: **$1,10101x(10)^{101}$**

Armazenamento em *floats*

- ✓ **Na organização/arquitetura do computador, deve-se definir**:
  - Número de bits do significando (precisão, ***p*** ou ***f***)
  - Número de bits do expoente (***e***)
  - Um bit ("**0**" para + e "**1**" para -) de sinal (tipicamente o primeiro, da esquerda)

Armazenamento em *floats*

- ✓ **Ilustração (8 bits)**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sinal	Expoente (+/-)			Significando			

- ✓ Sinal do número: 0 = + e 1 = -
- ✓ Expoentes: 8 combinações possíveis
  - **OBS**: Não seguem aritmética normal
    (p.ex.: Utiliza notação em excesso)

Armazenamento em *floats*

000	Caso especial
001	Expoente –2
010	Expoente -1
011	Expoente 0
100	Expoente 1
101	Expoente 2
110	Expoente 3
111	Caso especial

Abaixo de zero

(*bias* = polarização)

Acima de zero

Armazenamento em *floats*

Exemplo: Realize as conversões abaixo:

$6{,}75_{10}$ = (?)$_2$ (ponto flutuante, com 8 bits)

$1110\ 1001_2$ (ponto flutuante, com 8 bits) = (?)$_{10}$

Armazenamento em *floats*

Solução:

- ✓ $\mathbf{6,75_{10}} = 110,11_2 = 1,1011 \times 2^2$

  sinal: 0

  expoente: $2_{10} + 3_{10} = x_{10}$, $x_{10} = 5_{10} = 101_2$

  significando: 1011

  Número (ponto flutuante, com 8 bits):

  $$\mathbf{01011011_2}$$

Armazenamento em *floats*

Solução:

- ✓ **1110 1001_2** (ponto flutuante, 8 bits)

  sinal: 1

  expoente: $110_2 = 6_{10}$, $x_{10} + 3_{10} = 6_{10}$,

  $x_{10} = 3_{10}$

  significando: 1001_2 =

  Número:(negativo) $1{,}1001_2 \times 2^3 = 1100{,}1_2 = -12{,}5_{10}$

Armazenamento em *floats*

- ✓ **Observações:**

- ✓ **Maior número positivo** (lembre do bit escondido):
  0 110 1111 = + 2^3 x 1,1111 = 1111,1 = **15,5 decimal**

- ✓ **Menor número positivo** (lembre do bit escondido):
  0 001 0000 = + 2^{-2} x 1,0000 = 0,01 ou **0,25 decimal**

Armazenamento em *floats*

Combinações especiais dos expoentes na ilustração...

- ✓ **000** – representação NÃO normalizada
  - Significando passa a ser 0,_ _ _ ...
  - Expoente (000) = -2
  - **Menor número positivo passa a ser**
  - 0 000 0001 = 2^{-2} x 0,0001 = 2^{-2} x 2^{-4} = 2^{-6} = **0,015625 decimal**

A norma IEEE prevê o *underflow* gradual (a mantissa deixa de ser normalizada), permitindo obter números bem mais próximos de zero.

Armazenamento em *floats*

Ainda as combinações especiais...

- ✓ **Normalização não permite representar zero!**
- ✓ **000** – representação NÃO normalizada
  - 00000000 = + 0 decimal
  - 10000000 = - 0 decimal (iguais em comparações)
- ✓ **111** - representações de infinito
  - 01110000 = + infinito
  - 11110000 = - infinito
  - 11111000 = indeterminação
  - Outras combinações 11111_ _ _ = *Not A Number* (**NANs**)

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ A representação de números reais em ponto flutuante é perfeitamente adequada para fazer cálculos matemáticos, científicos, etc.
- ✓ Na representação em ponto flutuante pode-se ter perda de precisão do número representado ou mesmo haverá números que não podem ser representados por *overflow*.
- ✓ Para representação de números em que é necessário manter precisão até o último algarismo, não é admissível erro por aproximação.

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ Solução: usar a representação BCD ou Binary Coded Decimal (Decimal Representado em Binário).

- ✓ A idéia do BCD é representar, em binário, cada algarismo de forma que o número original seja integralmente preservado.

- ✓ A codificação BCD não possui extensão fixa, possibilitando representar números com precisão variável - quanto maior o número de bits, maior será a precisão.

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ Tabela de Representação dos Números Decimais em BCD

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

» Continua...

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ Tabela de Representação dos Números Decimais em BCD

Decimal	BCD
10	Inválido
11	Inválido
12	Inválido
13	Inválido
14	Inválido
15	Inválido

- ✓ Exemplo: 239_{10} = (?) BCD
  - 2 = 0010_2
  - 3 = 0011_2 e
  - 9 = 1001_2, logo: 239 = 001000111001 (BCD).

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ A codificação de um dígito em BCD requer 4 bits.
- ✓ Como a utilização de apenas 4 bits por byte não é eficiente, normalmente são armazenados 2 dígitos BCD em um só byte. Esta representação é chamada **BCD comprimido ou compactado** ("***packed BCD***").
- ✓ Exemplo: **14239_{10} = (?) BCD**

1	42	39	número decimal
xxxx0001	**01000010**	**00111001**	representação BCD comprimido
a+2	a+1	a	endereço

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ Entre os algarismos sem código válido em decimal (códigos representativos dos valores decimais de 10 a 15), é comum utilizar alguns deles para indicar o sinal do número.
- ✓ Há sistemas que adotam a seguinte convenção para o sinal dos números representados em BCD:
  - 1100 representa o sinal positivo ("+")
  - 1101 representa o sinal negativo ("-")
- ✓ Como nesta representação ainda há um desperdício de códigos; como BCD usa 4 bits (16 representações possíveis) para representar 10 algarismos, 6 (ou 4) códigos não são utilizados.

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ Portanto, essa representação é menos eficiente em relação à utilização dos recursos do computador que a **representação em ponto flutuante**.

- ✓ **Observações – Representação em Ponto Fixo**

1. Esse método consiste na determinação de uma posição fixa para a vírgula (ou ponto).
2. Todos os valores representados em ponto fixo para uma determinada operação possuem a mesma quantidade de algarismos inteiros, bem como a mesma quantidade de algarismos fracionários.
   - **Exemplo**: 1101,101 1110,001 0011,110

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ **Observações – Representação em Ponto Fixo**

- ✓ **As posições mais adotadas para a vírgula são**:
  - Na extremidade esquerda do número – nesse caso, o número é totalmente fracionário;
  - Na extremidade mais a direita do número – nesse caso, o número é inteiro.

- ✓ Em qualquer desses casos, no entanto, a vírgula fracionária não estará fisicamente representada na memória; sua posição é determinada na definição da variável, realizada pelo programador (ou pelo compilador), e o sistema memoriza essa posição, mas não a representa fisicamente.

Representação de Números Decimais Codificados em Binário (BCD)

- ✓ **Observações – Representação em Ponto Fixo**
- ✓ Na maioria das linguagens de programação e nos sistemas de computação (e os compiladores da maior parte das linguagens de programação) emprega-se a representação de números em ponto fixo para indicar apenas valores inteiros (a vírgula fracionária é assumida na posição mais à direita do número); **números fracionários são, nesses casos, representados apenas em ponto flutuante**.

- ✓ Exemplos de tipos de dados na linguagem Pascal:

Tipo de dado	Representação interna
INTEGER	Ponto fixo (inteiro)
REAL	Ponto flutuante (real)

Codificação Gray

- ✓ Os sistemas digitais operam em altas velocidades e reagem a variações que ocorrem nas entradas digitais. A fim de reduzir a probabilidade de um circuito digital interpretar mal uma entrada que está mudando, desenvolveu-se o **Código Gray**.

- ✓ Principal característica: Muda apenas um bit entre dois números sucessivos.

Codificação Gray

DECIMAL	BINARIO	GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Codificação Gray

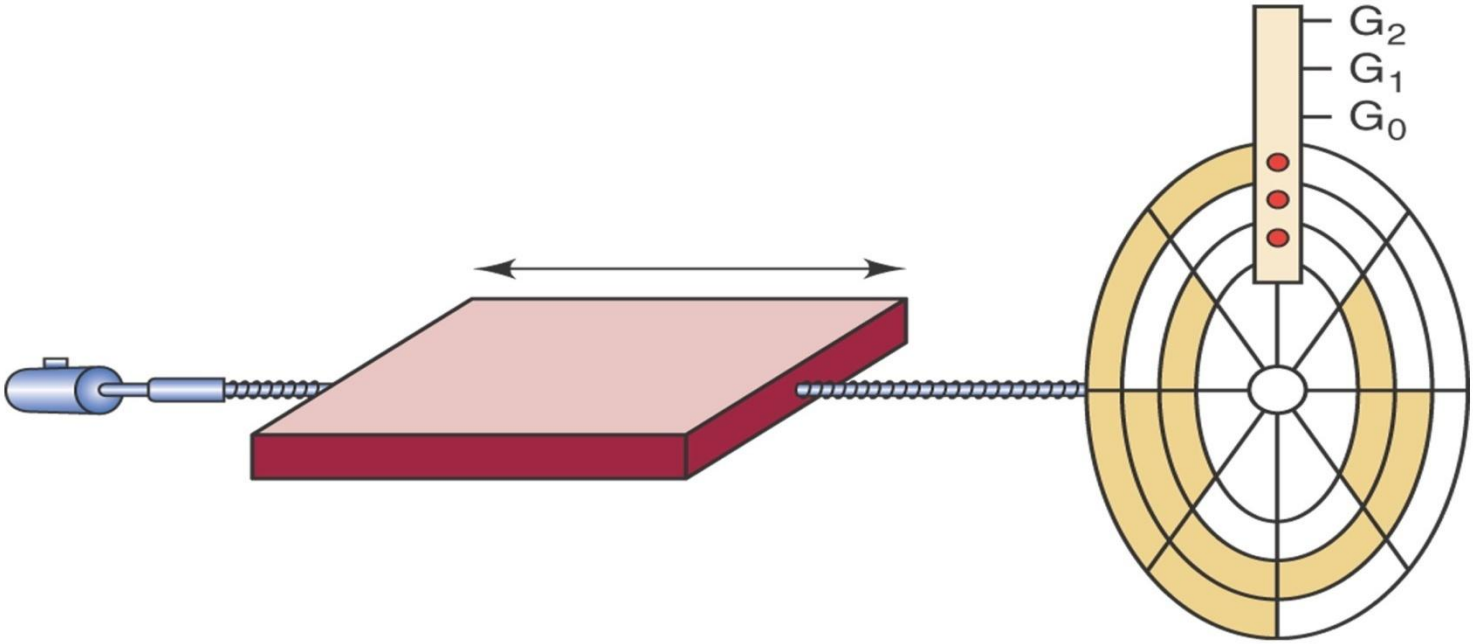
Lista de 2-bit:	00, 01, 11, 10	
Reflete:		10, 11, 01, 00
Concatena:	00, 01, 11, 10,	10, 11, 01, 00
Coloca 0 nos números iniciais:	000, 001, 011, 010,	10, 11, 01, 00
Coloca 1 nos números espelhados:	000, 001, 011, 010,	110, 111, 101, 100

Codificação Gray

Valor Binario	Valor Desplazado	Valor Código Gray
A	B	A (XOR) B
0000	0000	0000
0001	0000	0001
0010	0001	0011
0011	0001	0010
0100	0010	0110
0101	0010	0111
0110	0011	0101
0111	0011	0100
1000	0100	1100
1001	0100	1101
1010	0101	1111
1011	0101	1110
1100	0110	1010
1101	0110	1011
1110	0111	1001
1111	0111	1000

Codificação Gray

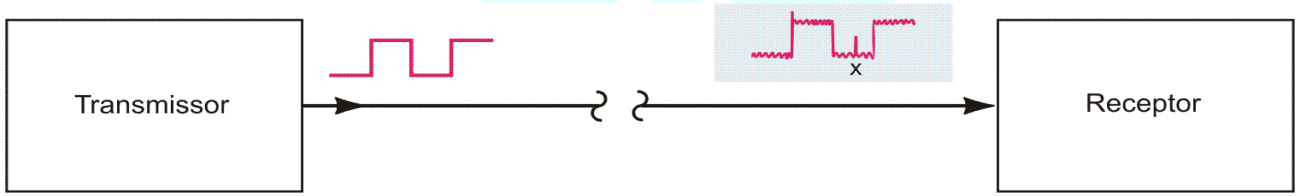
- ✓ A aplicação mais comum do código Gray é nos codificadores de rotação de eixo. **Esses dispositivos produzem um valor binário que representa a posição de um eixo mecânico em rotação.**



Detecção de Erro pelo Método da Paridade

- ✓ Transmissão de dados binários: operação mais comum em sistemas de comunicação. Ex.:
  - Sistema celular
  - Rede de computadores

- ✓ Tráfego de informação transmissor/receptor sujeito à ocorrência de erros provocados por ruído (flutuações no nível do sinal)
  - Interpretação errônea do sinal recebido.



Detecção de Erro pelo Método da Paridade

- ✓ Bit de paridade: bit extra anexado ao conjunto de bits para indicar o tipo de paridade:
  - Par: número de 1's deve ser par (contando com o bit de paridade)
  - Ímpar: número de 1's deve ser ímpar (contando com o bit de paridade)
  - Ex.: Código ASCII para 'C' – 1000011. Tem 3 bits 1s. Para que tenha paridade par, 1 é acrescentado.

Detecção de Erro pelo Método da Paridade

- ✓ Usado para detectar erros de um bit, que são mais prováveis de ocorrer.
- ✓ Deve haver concordância entre TX e RX pelo tipo usado (par ou ímpar).
- ✓ A informação é transmitida em formato binário e, geralmente, é representada por tensões na saída de um transmissor que está conectado à entrada de um circuito receptor.