FUNDAMENTOS DA COMPUTAÇÃO

PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

COMPUTAÇÃO X ÁLGEBRA DE BOOLE

- A representação 1/0 tem sua origem na teoria e álgebra publicada por George Boole (matemático, filósofo) em 1854 – a computação BINÁRIA e todo o desenvolvimento a partir daí utiliza esta base (simular circuito em http://www.falstad.com/circuit/)
- O sistema BCD (Binary Code Digit) foi criado inicialmente pela IBM, sendo de 06 bits, para representar no máximo 64 símbolos; como representava apenas letras maiúsculas, logo foi substituído pelo EBCDIC, de 08 bits, mas era restrito à máquinas IBM. Posteriormente foi realizada uma padronização no ASCII, de 08 bits.
- O método mais comum, portanto, pode ser estendido para 16 8 4 2 1; 128, 64, 32, 16, 8, 4, 2, 1 etc. Aos bits (LIGADOS, 1) somam-se os valores decimais equivalentes – isto caracteriza o processo de CODIFICARxDECODIFICAR. Ambos lados da comunicação devem usar o mesmo algoritmo/esquema/técnica. Mesma ideia de sistema posicional!

COMPUTAÇÃO X ÁLGEBRA DE BOOLE

- Ex: suponha que o número 15 (d) foi codificado em grupos de nibbles:

  00011001

  - Agora, desejamos que este número binário, ao chegar ao destino, seja decodificado de volta ao sistema decimal com o algoritmo das potências:

128	64	32	**16**	**8**	4	2	**1**	
0	0	0	**1**	**1**	0	0	**1**	= 25

Logo:

COMPUTAÇÃO X ÁLGEBRA DE BOOLE

- Os circuitos eletrônicos podem implementar algoritmo de conversão decimal para binário de **números inteiros** por meio de divisões sucessivas pela base 2
- Ex: 25 (d) -> 11001 (b)

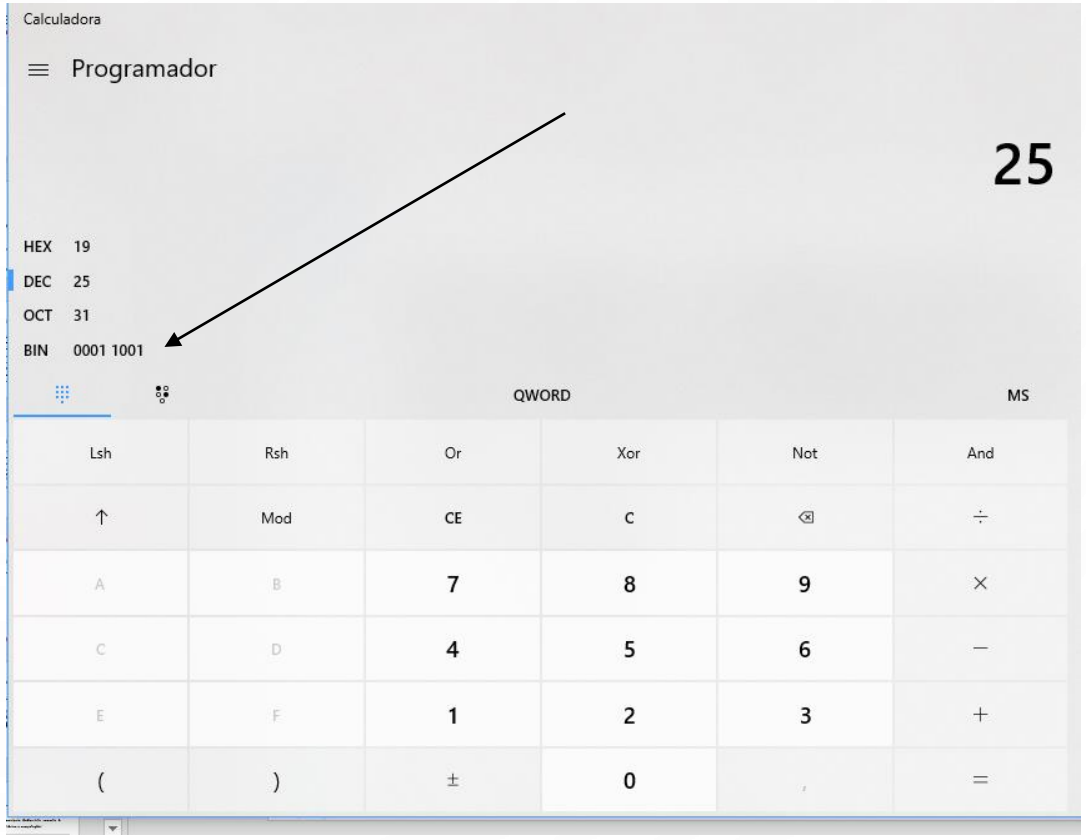
  25 / 2 = 12, RESTO **1**

  12 / 2 = 6, RESTO **0**

  6 / 2 = 3, RESTO **0**

  3 / 2 = **1**, RESTO **1**

Resposta com 8 bits (1 byte)



PALAVRA DA CPU

- Número de bytes/bits que a CPU processa como uma unidade de dados. Na verdade define o tamanho de uma memória volátil interna à CPU denominado REGISTRADOR

4004 e 4040: 4 bits
8008, 8080, 8085, Z-80: 8 bits
1982: 8086, 8088, 80186, 80188, INTEL 80286 – 16 bits, 134 kTransistores
1985: INTEL 80386 – 32 bits
2004 (19 anos depois): AMD ATHLON 64 bits (qword) compatível com Windows

  - **64 bits (preenche com zeros à esquerda)**

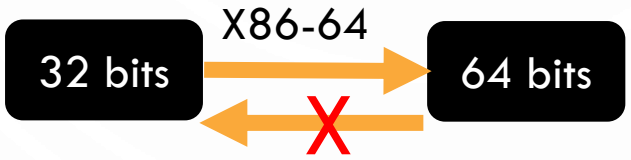
1001:

0000000000000000000...00001001 (barramento)

- Outros conceitos sobre dado binário
  - Nibble: 4 bits
  - Byte: 8 bits (octeto)
  - Word: 16 bits, dword: 32 bits, qword: 64 bits

ARQUITETURAS E SOFTWARE

- Software 32 bits (x86) são executados de modo compatível com processadores 64 bits através da extensão x86-64



Ao compilar um código fonte, define-se se o código alvo executável será de 32 ou 64 bits

- Os processadores possuem internamente blocos ou componentes digitais para armazenamento temporário chamados REGISTRADORES (registers). A palavra do processador define então o "tamanho" destes registradores. *Algumas arquiteturas, como PENTIUM, possuíam 64 bits de barramentos de dados (externo), mas os registradores internos eram de 32 bits. Normalmente, estes dois números são iguais.*

REPRESENTAÇÃO EM PONTO FLUTUANTE

- Binário – Decimal

$$10011{,}1101 = 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4} = 19{,}8125$$

- Decimal – Binário

$$19{,}8125 = 19 + 0{,}8125$$
$$19 = 10011\ (div.sucessiva\ por\ 2\ ou\ método\ da\ tabela)$$
$$0{,}8125\ x\ 2 = 1{,}625\ - 1 = 0{,}625 = 1$$
$$0{,}625\ x\ 2 = 1{,}250 - 1 = 0{,}250 = 1$$
$$0{,}250\ x\ 2 = 0{,}5 = 0$$
$$0{,}5\ x\ 2 = 1\ (FIM\ DO\ ALGORITMO)$$
$$19{,}8125 = 10011{,}1101$$

- Representação MANTISSA, EXPOENTE padrão geral apenas como exemplo didático

$10011{,}1101 = 1{,}00111101\ x\ 2^4$ ← **NORMALIZAÇÃO**

$Sinal: (0: +), Mantissa: 00111101\ 00\ Expoente\ (e): (4)$

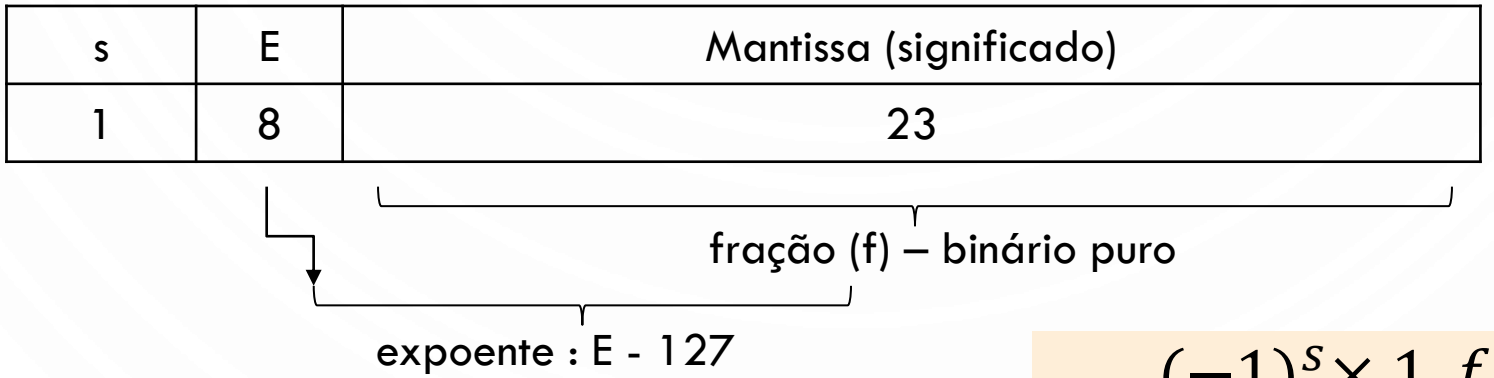
$Logo, 0\ 10000011\ 001111010\ 00$

8 bits 10 bits

A mantissa possui forma 1.F (1 implícito)
Na representação, usar E = 127+e = 131

REPRESENTAÇÃO EM PONTO FLUTUANTE (DECIMAL)

Precisão simples (float), 32 bits (4 bytes): norma define convenções como 0 (+), 1 (-), o formato, campos e tamanho da representação, além de fórmula ou algoritmo para codificação e decodificação. Define também formatos para NaN e Infinito



Na história...
Konrad Zuze
Z1: 1937
Z3: 1941

$$(-1)^s \times 1.f \times 2^{E-127}$$

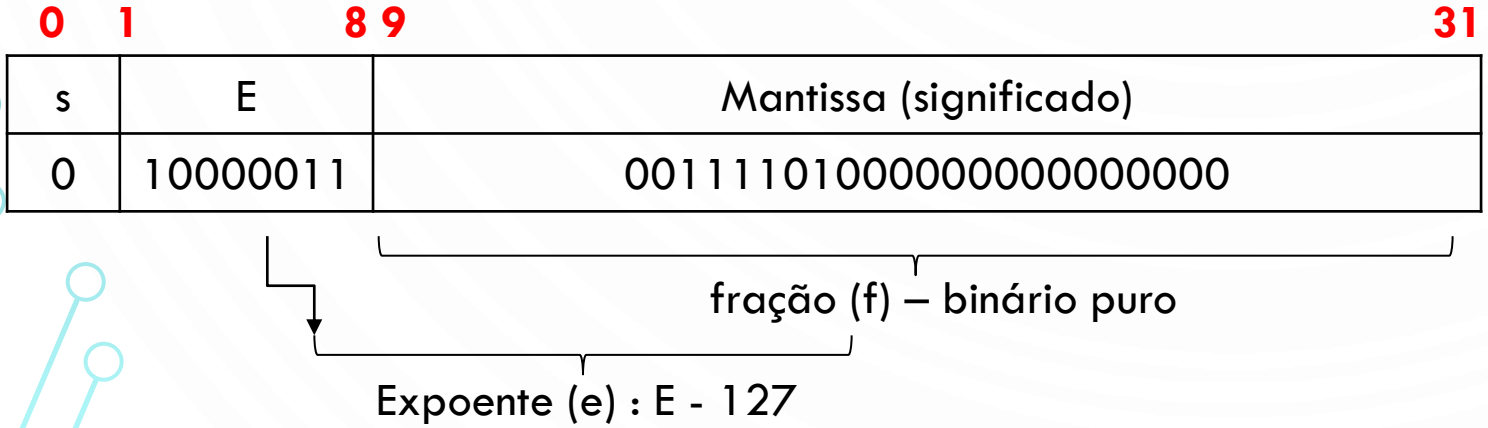
Normalizado: primeiro bit da mantissa sempre 1
Com mantissa entre 1.0 e 2.0
Desnormaliado: 0

PADRÃO: IEEE 754-1985

REPRESENTAÇÃO EM PONTO FLUTUANTE (DECIMAL)

Precisão simples (float), 32 bits (4 bytes): de 2^(-126) a 2^(127)

Seja a representação 0 10000011 00111101000...0, obter o DEC fracionário equivalente



$$n = (-1)^s \times 1.f \times 2^e$$

e = E - 127

$$(-1)^0 \times (1.00111101) \times 2^4$$

$$(-1)^0 \times (1 + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{256}) \times 2^4$$

$$1 \times (1 + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{256}) \times 16$$

19,8125

REPRESENTAÇÃO EM PONTO FLUTUANTE (DECIMAL)

Regra geral

Sinal (S)	Expoente (E)	Mantissa (M ou F)
Normalizado		
+ ou -	0<E<255	Qualquer padrão de bits
Valor = 0 (Zero)		
+ ou -	0	0
Valor = InF		
+ ou -	255	0
	NaN (not a number)	
+ ou -	255	Qualquer padrão de bits menos o 0...0

Um NaN é gerado quando o resultado de uma operação de ponto flutuante não pode ser representado no formato de ponto flutuante do IEEE-754 para o tipo especificado.

REPRESENTAÇÃO EM PONTO FLUTUANTE (DECIMAL)

Testes em C++

testnan.cpp

```
#include <iostream>
using namespace std;
#include <cmath>

int main() {
    float a = NAN;
    if (isnan(a)) cout << a << ": a is nan" << endl;
    float b = INFINITY;
    if (isinf(b)) cout << b << ": b is infinite" << endl;
    float c = 0; // also -0
    if (iszero(c)) cout << c << ": c is zero" << endl;
    //here isnan does not work
    if (isinf(5.0/0)) cout << " division error " << endl;

    bool testNaN = (a != a);
    if (testNaN) cout << "a is indeed NaN" << endl;
    return 0;
}
```

Um NaN é gerado quando o resultado de uma operação de ponto flutuante não pode ser representado no formato de ponto flutuante do IEEE-754 para o tipo especificado.

REPRESENTAÇÃO DE NÚMEROS NEGATIVOS

- Uma representação típica é denominada COMPLEMENTO DE 2
  - Exemplo: -10 – vamos supor palavra de 8 bits:

O sinal negativo pode ser o primeiro bit, ou seja, 0 para positivo, 1 para negativo

Converte-se o número para binário pelo método das potências: 10 = 0001010

A este número binário, inverte-se os bits (COMPLEMENTO DE 1)

0001010 -> 1110101

Soma-se 1 ao bit menos significativo (de menor peso, mais à direita), LOGO:

```
  1110101
+         1
-------------
```

11110110 NO CALC WINDOWS: 64 BITS..., logo, 1111111....1110110

64 bits

CPU: CÁLCULOS ARITMÉTICOS, RELACIONAIS, LÓGICOS

- Unidade Lógico Aritmética: componente da CPU com esta tarefa

  Um comando condicional do tipo IF ou um comando de repetição como WHILE ou FOR envolve o TESTE lógico de condições, que retornam 0 (F) ou 1 (V): GEORGE BOOLE!

```
#include <iostream>
using namespace std;

int main() {
    int a, b, c;
    bool y;
    cin >> a;
    cin >> b;
    cin >> c;
    y = (((a < b) && (b > c)) || (a>b));
    // se usar and e or fora de condicional dá erro
    if (y) cout << "y=V " << a << " " << b << " " << c << endl;
    else cout << "y=F" << endl;
    return 0;
}
```

APLICAÇÕES/OUTROS SISTEMAS DE NUMERAÇÃO

- Um exemplo típico de uso do sistema binário é utilizado para identificar endereços de rede IP. Por exemplo, na versão IPv4, tem-se 4 OCTETOS (4 bytes), separados por ponto:

  192.168.0.1 = 11000000.10101000.00000000.00000001

- A computação também utiliza sistema base 8 (OCTAL) e sistema base 16 (HEXADECIMAL). A base 16 é particularmente útil para representar endereços na memória e endereços IPv6, de 128 bits. Cada dígito hexadecimal equivale a 4 bits (0...9, A, B, C, D, E, F), e portanto simplifica a representação: 32 dígitos hexa ao invés de 128 dígitos binários

TABELA HEXADECIMAL

1 – 0001

2 – 0010

3 – 0011

4 – 0100

5 – 0101

6 – 0110

7 – 0111

8 – 1000

9 – 1001

A – 1010

B – 1011

C – 1100

D – 1101

E – 1110

F - 1111