Operações no Sistema Binário

- Adição e multiplicação binária: o conjunto de operações básicas de adição e multiplicação no sistema binário pode ser representado de forma resumida pelas seguintes tabelas:

Adição

+	**0**	**1**
0	0	1
1	1	$^{1}0$

Multiplicação

x	**0**	**1**
0	0	0
1	0	1

Leia-se ‘0’ e ‘vai 1’ para o dígito de ordem superior

Operações no Sistema Binário

- Obs. 1: As operações de adição e multiplicação são realizadas operando-se as colunas da direita para a esquerda, da mesma forma que nas operações decimais.

- Obs. 2: Todas as operações aritméticas podem ser realizadas através da soma:
  - a multiplicação pode ser feita através de sucessivas somas (um número N vezes *'b' é igual a soma de N com N 'b'* vezes);
  - a subtração pode ser feita através do método de complemento a base (que veremos a seguir);
  - Finalmente, a divisão pode ser feita através de sucessivas subtrações.

Exemplos de operações binárias

$3_{10} + 5_{10} = 11_2 + 101_2 \longrightarrow$

$$\begin{array}{r} 101 \\ +\quad 11 \\ \hline 1000 \end{array} \longrightarrow 1000_2 = 8_{10}$$

$3_{10} \times 5_{10} = 11_2 \times 101_2 \longrightarrow$

$$\begin{array}{r} 101 \\ \times\quad 11 \\ \hline 101 \\ +\quad 101 \\ \hline 1111 \end{array} \longrightarrow 1111_2 = 15_{10}$$

Exemplos de operações binárias

$37_{10} + 30_{10} = 100101 + 11110 \longrightarrow$

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1\,} \\ 100101 \\ +\ \underline{11110} \\ 1000011 \end{array} \longrightarrow 2^6+2^1+2^0 = 67_{10}$$

$7_{10} \times 7_{10} = 111 \times 111 \longrightarrow$

$$\begin{array}{r} 111 \\ \times\ \underline{111} \\ 111 \\ 111 \\ +\ \underline{111} \\ 110001 \end{array} \longrightarrow 2^5+2^4+2^0 = 49_{10}$$

Operações Binárias

- Subtração: o método mais simples de subtração entre dois valores binários é através do complemento a base, executado pela seguinte sequência de instruções (ei, é um algoritmo!):

  - Mantenha o minuendo na sua forma original;
  - Inverta o subtraendo (todo ‘1’ vira ‘0’ e todo ‘0’ vira ‘1’;
  - Some o minuendo e o subtraendo;
  - Some 1;
  - Ignore o algarismo mais significativo caso ele esteja numa posição notacional que os operandos não tenham um algarismo significativo.

- Obs.: não se esqueça de representar os zeros não-significativos (pois este serão importantes na inversão)!

Complemento a 2

- Esta forma de representação que vimos é chamada **complemento a 2**; pode ser usada para outras operações particulares, além da transformação de uma subtração em adição.

- Em qualquer situação, a conversão é feita da mesma forma: invertem-se os bits (0 → 1 e 1 → 0) e soma ‘1’.

- É importante sempre lembrar dos zeros não-significativos para realizar uma conversão, pois zeros à esquerda se tornarão ‘1’s.

Exemplo de subtração binária

$37_{10} - 12_{10} = 100101 - 001100$

A quantidade de casas foi igualada com zeros à esquerda!

O minuendo foi mantido!

$$
\begin{array}{r}
1\quad 111 \\
100101 \\
+\quad 110011 \\
\hline
1011000 \\
+\qquad\quad 1 \\
\hline
\underline{1}011001
\end{array}
$$

O subtraendo foi invertido!
12 = 001100 → 110011

Soma 1

Este algarismo é desprezado!

$= 011001 = 2^4+2^3+2^0 = 25_{10}$

Divisão de números binários

- Como nas demais operações aritméticas, a divisão binária é similar à divisão decimal, sendo:
  - 0 / 1 = 0
  - 1 / 1 = 1
  - Divisão por zero → erro

- Divisões binárias podem ser realizadas pelo método tradicional (dividendo / divisor = quociente e um resto) ou através de sucessivas subtrações

Passos para a divisão binária

- A partir da esquerda, avançam-se tantos algarismos quantos sejam necessários para obter-se um valor maior ou igual ao divisor;
- Encontrado este valor, registra-se 1 para o quociente;
- Subtrai-se do valor obtido no dividendo o valor do divisor;
- Ao resultado da subtração, acrescentam-se mais algarismos do dividendo (se ainda houver algum que não fora utilizado) até obter-se um valor maior ou igual ao divisor.
- Se não houverem mais algarismos a serem utilizados e o valor do dividendo for menor do que o divisor, encerra-se a operação e temos um resto não-nulo;

Passos para a divisão binária

- Se os algarismos ainda não utilizados do dividendo tiverem valor igual a zero, a cada um deles utilizado deve se acrescentar um zero ao quociente

- Repita o processo até que não existam mais algarismos do dividendo.

- Ex.: 100_2 / 10_2

10’0	**10**
-10	**10**
00	

Exemplo de divisão binária

- $18_{10} / 3_{10} \rightarrow 10010_2 / 11_2$

100’1’0	**11**
- 11	**110**
011	
00	

Deslocamento de bits

- Operação *shift*: faz o deslocamento de bits dentro de um número, equivalendo às operações de multiplicação ou divisão por potência de 2
  - Multiplicação: deslocamento à esquerda;
  - Divisão: deslocamento à direita;
- Exemplos:
  - 2 x 2 = 4 (10_2 x 10_2 = 100_2) → bit 1 se deslocou 1 casa para à esquerda
  - 4 (2^2) x 2= 8 (100_2 x 100_2 = 1000_2) → bit 1 se deslocou 2 casas para à esquerda
  - 16 (2^4) / 4 (2^2) = 4 (10000_2 / 10_2 = 100_2) → bit 1 se deslocou 2 casas para à direita

Números binários fracionários

- É possível trabalhar com números binários com representação fracionária, que da mesma maneira que os valores decimais, utiliza uma vírgula para separar a parte inteira da parte fracionária
  - Ex: $101{,}11_2$
- Como é feita a conversão?
  - A parte inteira é convertida da maneira tradicional, enquanto a parte fracionária utiliza expoentes negativos

Números binários fracionários

- Para o exemplo apresentado, temos:

  - $101{,}11_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 4 + 0 + 1 + 0{,}5 + 0{,}25 = 5{,}75_{10}$

- Recordando potências negativas:

  - $2^{-1} = (1/\ 2^1) = 0{,}5$
  - $2^{-2} = (1/\ 2^2) = 0{,}25$
  - $2^{-3} = (1/\ 2^3) = 0{,}125$
  - $2^{-4} = (1/\ 2^4) = 0{,}0625$
  - e assim por diante

Números binários fracionários

- Como é a conversão de um decimal fracionário para binário?
  - Para a parte inteira, segue a regra de conversão já estudada;
  - Para a parte fracionária, inicialmente multiplica-se o seu valor pela base que se deseja converter, neste caso, 2;
  - Em seguida, o valor da parte inteira gerada representa o primeiro bit da parte fracionária;
  - Repete-se o processo com a parte fracionária até a precisão desejada ou até que a parte fracionária seja igual a zero.

Números binários fracionários

- Para o exemplo apresentado anteriormente, temos:
  - $5{,}75_{10} = 5\ (101_2) + 0{,}75$
  - $0{,}75 \times 2 = 1{,}5 = 1 + 0{,}5$
  - $0{,}5 \times 2 = 1{,}0 = 1 + 0{,}0$
- Logo, a representação binária é igual a $101{,}11_2$
- Em alguns casos, dificilmente encontramos a parte fracionária igual a zero. Isto é causado por diferenças de precisão entre as bases.

Números binários fracionários

- Quando encontramos uma dízima periódica em binário com uma quantidade ilimitada de bits, o número fica da forma:

  - 90,8 = 1011010,1100110011001100110011001100110
  - 0110011001100110011001100110011001100110011001100...

- Se a variável utilizada for do tipo *float*, são reservados 23 bits para representar a parte fracionária, ou seja, só teremos espaço para armazenar uma parte da resposta, e será feito um arredondamento, acarretando um pequeno erro.

Números binários fracionários

- Como resolver este problema?
  - Toda linguagem de programação dispõe de um tipo de variável que dispõe de mais casas binárias para a representação da parte fracionária. Em ‘C’, é possível declarar o tipo da variável como *double*, que reserva 52 bits para armazenar os bits da fração

Representação de Dados

- Antes de prosseguirmos, precisamos entender os seguintes conceitos:
  - BIT (*BInary digiT*): é a representação do menor item de dado possível;
  - Byte: um conjunto de bits (é adotado como padrão que 1 byte é formado por 8 bits);
  - Palavra: um conjunto de bytes; um computador com palavra de 32 bits tem 4 bytes por palavra. A maioria das instruções de um computador opera sobre palavras, por exemplo, cada operação de um computador de 32 bits opera sobre palavras de 32 bits, deve ter registradores de 32 bits, instruções para 32 bits, etc.

- A limitação do número de bits é necessária para que possamos representar na forma binária diferentes tipos de dados (números, instruções, etc.)

Representação de Dados

- Da forma que vimos até agora todas as representações numéricas são de valores inteiros e sem um limite máximo, isto é, iríamos de 0 a ∞. Devemos estabelecer um <u>domínio de valores</u> dentro do qual o sistema irá operar. Será necessário estabelecermos uma forma de representar valores negativos

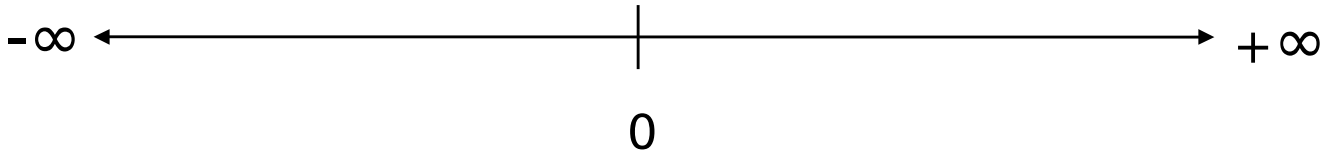
- Na prática utilizamos o bit na posição mais significativa (isto é, o bit mais a esquerda) para a representação do sinal, com a seguinte convenção:
  - Bit 0 → sinal positivo.
  - Bit 1 → sinal negativo.

Representação de Dados

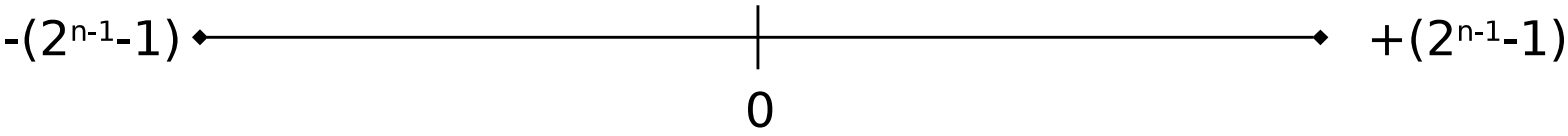
- Desta forma, uma representação em binário com *n* bits teria disponíveis para a representação do número *n-1* bits (o bit mais significativo representa o sinal). Este modelo de representação é chamado de <u>representação em sinal e magnitude</u>

Representação de Dados

- Teoria:



- Prática (limitado a *n* bits):



Representação em Sinal e Magnitude

- A magnitude é o valor absoluto de um número, que independe do sinal, representada em binário;

- O sinal é representado pelo bit mais significativo, sendo positivo se representado por ‘0’ e negativo se representado por ‘1’;

- O valor dos bits usados para representar a magnitude independe do sinal, ou seja, seja o número positivo ou negativo, a representação da magnitude será exatamente a mesma, variando somente o bit de sinal.

Representação em Sinal e Magnitude

Sinal	Magnitude

Ex.: $0110110 = +\ 54_{10}$

$1110110 = -\ 54_{10}$

o binário 110110 corresponde ao valor absoluto 54.

A faixa de representação de um valor binário em sinal e magnitude com *n* bits possui 2^n representações, representando os valores de **$-\ (2^{n-1}-1)$** (menor valor negativo) a **$+\ (2^{n-1}-1)$** (maior valor positivo).

Como é a representação do zero em sinal e magnitude?

Exemplo de Representação de Dados

Valor decimal	Valor binário com 8 bits (7 + bit de sinal)
+9	00001001 (bit inicial 0 significa positivo)
-9	10001001 (bit inicial 1 significa negativo)
+127	01111111 (bit inicial 0 significa positivo)
-127	11111111 (bit inicial 1 significa negativo)

Como podemos observar, o maior e menor valores que podemos representar com 8 bits são, respectivamente, +127 e -127.

Exercício

- Complete o quadro a seguir com os valores correspondentes para uma arquitetura de 32 bits, representados como uma potência de base 2, quando conveniente:

32 bits	Sem Sinal	Sinal e Magnitude
Menor Valor		
Maior Valor		
Quantidade de valores distintos		

Aritmética em Sinal e Magnitude

<u>Soma em sinal e magnitude:</u>

- Verificar o sinal das parcelas a serem somadas;
  - Se forem iguais, repetir o sinal e somar as magnitudes;
  - Se forem diferentes:
    - verificar qual parcela tem a maior magnitude;
    - repetir o sinal da maior magnitude;
    - subtrair a menor magnitude da maior magnitude.

- Os bits referentes ao sinal dos operandos, positivo ou negativo, não devem ser operados aritmeticamente!

Representação de operações

- Lembre-se que as instruções são interpretadas em linguagem de montagem (que será estudada com detalhes mais adiante), utilizando *mnemônicos* para representar as operações, desta forma, apenas os operandos possuem sinais, por exemplo:

  - +5 – (– 7) → SUB (5, -7)
  - +12 + (– 3) → ADD (12, -3)

Aritmética em Sinal e Magnitude

- Subtração em sinal e magnitude:
  - É calculada exatamente como uma soma entre duas parcelas de sinais diferentes
  - É importante lembrar que as operações aritméticas são realizadas somente com as magnitudes, então sempre o menor valor é subtraído do maior valor!
  - Quanto ao sinal, basta fazer uma análise lógica se o resultado será positivo ou negativo, da mesma forma que é feito com operações decimais.

Exemplos de operações em Sinal e Magnitude

$+54_{10} + (+43_{10})$ (em uma representação com palavras de 8 bits):

$= 00110110 + 00101011$

$$\begin{array}{r} {\scriptstyle 11111} \\ 0110110 \\ +\quad 0101011 \\ \hline 1100001 \end{array} \longrightarrow 2^6 + 2^5 + 2^0 = +\,97_{10}$$

$+54_{10} - (+43_{10})$ (em uma representação com palavras de 8 bits):

$$\begin{array}{r} 0110110 \\ -\quad 0101011 \\ \hline \end{array} \longrightarrow \begin{array}{r} {\scriptstyle 111\ 1} \\ 0110110 \\ +\quad 1010100 \\ \hline 10001010 \\ + \qquad\quad 1 \\ \hline 10001011 \end{array} \longrightarrow$$

A operação é feita sem o bit de sinal!

$10001011 = 2^3+2^1+2^0 = 11_{10}$

$\boxed{00001011 = +\,11_{10}}$

Exemplos de operações em Sinal e Magnitude

$+43_{10} - (54_{10})$ (em uma representação com palavras de 8 bits):

```
                    11 1  1
    0101011          0110110
-   0110110     +    1010100
                    10001010
                +           1
                    10001011
```

A operação é feita sem o bit de sinal!

$0001011 = 2^3+2^1+2^0 = 11_{10}$

$\boxed{10001011 = -11_{10}}$

Por que os operandos foram invertidos? Lembre-se que as operações são feitas somente com a magnitude, que são valores positivos, portanto, o menor valor deve ser subtraído do maior valor.

Exemplo de Multiplicação

<u>Multiplicação</u>:

Na multiplicação são utilizadas as regras já conhecidas para a magnitude, e o sinal é manipulado da mesma forma que na aritmética tradicional.

$+19_{10} \times (-19_{10}) = 010011 \times 110011 = -361_{10}$

```
          10011
     x    10011
     ----------
          10011
         10011
   +    10011
   ----------
     101101001
```

Operamos sem o bit de sinal (o mais significativo)

$2^8+2^6+2^5+2^3+2^0 =$

$256 + 64 + 32 + 8 + 1 = 361_{10}$

Limites de memória

- Em sistemas reais, tanto os valores operados quanto os resultados produzidos podem exceder os limites de armazenamentos impostos pela arquitetura, ou seja, o número de bits que compõem a palavra (principalmente em multiplicações).

- Para contornar esta limitação, no caso de o número de bits da solução exceder o limite da palavra, podem ser utilizadas duas palavras para armazenar o resultado

- Uma palavra de n bits contém os *n*-1 bits do valor, precedidos pelo bit de sinal. A outra palavra conterá os bits mais significativos, ‘0’s complementares, se necessários, e o sinal do resultado.

Exemplo de uso de mais de uma palavra para expressar um valor

$+19_{10} \times (-19_{10}) = 010011 \times 110011$ **(Agora limitado a 6 bits)**

```
        10011
   x    10011
        10011
       10011
+     10011
      101101001
```

Operamos sem o bit de sinal (o mais significativo)

O bit de sinal será negativo

1	0	1	0	1	1		1	0	1	0	0	1

$= 1101101001_2 = -361_{10}$

bits mais significativos **bits menos significativos**

Limites de memória

- No caso de não ser possível armazenar um valor mesmo usando o limite de palavras do sistema para a representação de um valor, ocorre um erro chamado *OVERFLOW*, que pode ser traduzido livremente como ‘estouro de memória’. Isto significa que tentamos armazenar mais bits do que uma capacidade pré-estabelecida para uma variável.
- Ex.:

Tipo	Tamanho (em bytes)	Faixa Mínima
unsigned char	1	0 a 255
signed char	1	-128 a 127
long int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295