

Métodos Computacionais em Física I (FIW234)

Turmas IFA e IFB

Noções Básicas de Linguagem C

Edivaldo M. Santos e João R. T. de Mello Neto

Aula 3

Tópicos de linguagem C a serem abordados

- Diretivas de préprocessamento
- Identificadores
- Tipos
- Variáveis e Aritmética
- Operadores e precedência
- Efeitos colaterais
- Laços: while and do, for, switch, break, continue
- controle de fluxo e expressões lógicas: if
- Funções
- Passagem de argumentos e recursividade
- Arrays
- Entrada e Saída
- Apontadores
- Alocação Dinâmica de Memória
- Bibliotecas e Funções Matemáticas

Tópicos de linguagem C a serem abordados

- Diretivas de préprocessamento
- Identificadores
- Tipos
- Variáveis e Aritmética
- Operadores e precedência
- Efeitos colaterais
- Laços: **while** and do, for, switch, break, continue
- controle de fluxo e expressões lógicas: **if**
- Funções
- Passagem de argumentos e recursividade
- Arrays
- Entrada e Saída
- Apontadores
- Alocação Dinâmica de Memória
- Bibliotecas e Funções Matemáticas

Na aula passada vimos:

- a estrutura de um programa em C;
- diretivas de pré-processamento;
- instruções de entrada e saída: **printf** e **scanf**;
- instrução de laço **while**;
- variáveis e expressões aritméticas;
- operadores binários.
- compilação e execução de um programa em C;
- regras de etiqueta do bom programador.

Na aula de hoje veremos:

- representação numérica em um computador;
- o conjunto de caracteres da linguagem C;
- identificadores e palavras-chaves;
- um pouco mais acerca de laços;
- operadores (binários e unários) e precedência.

Números inteiros e reais em C

Na aula passada, vimos 3 tipos de variáveis em C:

- **int**
- **float**
- **double**

A primeira delas foi utilizada para representar inteiros e as duas últimas para números reais. Para compreender um pouco mais sobre esses 3 tipos é preciso analisar a maneira como é feita a [representação numérica em um computador.](#)

Números inteiros e reais em C

- Agora que já sabemos algo sobre a maneira com que números inteiros e reais são representados nos computadores, é possível entender as diferenças entre as variáveis **int**, **float** e **double** do C. Num computador de 32-bits, uma **possível** definição dessas variáveis seria:
 - **int** representa variáveis inteiras em **precisão simples**, isto é, com **palavras de 32-bits**.
 - **float** representa variáveis reais em ponto flutuante e precisão simples.
 - **double** representa variáveis reais em ponto flutuante e **precisão dupla**, isto é, com **palavras de 64-bits**.

Números inteiros e reais em C

- Agora que já sabemos algo sobre a maneira com que números inteiros e reais são representados nos computadores, é possível entender as diferenças entre as variáveis **int**, **float** e **double** do C. Num computador de 32-bits, uma **possível** definição dessas variáveis seria:
 - **int** representa variáveis inteiras em **precisão simples**, isto é, com **palavras de 32-bits**.
 - **float** representa variáveis reais em ponto flutuante e precisão simples.
 - **double** representa variáveis reais em ponto flutuante e **precisão dupla**, isto é, com **palavras de 64-bits**.
- Note a presença da palavra possível acima e o fato de que não especificamos os intervalos das variáveis também. Tomamos esse cuidado, porque as definições e intervalos precisos de cada tipo em C é **dependente de cada máquina**. A arquitetura de cada computador, com o tamanho da palavra em bits e o número de bits alocados para o expoente e a mantissa é que definirão, em última caso, os intervalos de cada tipo.

Números inteiros e reais em C

- Além do tipo **int**, o C possui outros tipos para a representação de inteiros:
 - **unsigned short int**: inteiro sem sinal com pelo menos 16 bits;
 - **unsigned int**: inteiro sem sinal com pelo menos 16 bits;
 - **unsigned long int**: inteiro sem sinal com pelo menos 32 bits;
 - **short int**: inteiro com sinal com pelo menos 16 bits;
 - **long int**: inteiro com sinal com pelo menos 32 bits.
- um **short** nunca pode ser mais longo que um **int**, que nunca pode ser mais longo que um **long**.
- uma possível implementação dos tipos acima com 16-bits para short, 32-bits para int e 32-bits para longs, produziria então os seguintes intervalos:
 - **unsigned short int**: [0,65535]
 - **unsigned int**: [0,4294967295]
 - **unsigned long int**: [0,4294967295]
 - **short int**: [-32768,32767]
 - **long int**: [-2147483648,2147483647]

Um pouco mais sobre laços: do e while

- Uma outra forma de utilizar a instrução de laço **while** é juntamente com a instrução **do**. Essa forma alternativa é para garantir que as instruções que porventura estejam dentro do laço **sejam executadas ao menos uma vez**.
- Por exemplo:

```
...
do
{
    celsius = 5 * (fahr-32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + passo;
}
while (fahr <= sup);
...
```

executaria a conversão de Fahrenheit para Celsius pelo menos uma vez, mesmo se o valor inicial da temperatura em Fahrenheit fosse maior que sup, pois a condição entre parenteses é testada após o conjunto de instruções do laço.

Um pouco mais sobre laços: a instrução for

- Se você pensou um pouco acerca do laço empregado na tabela de conversão de temperaturas, percebeu que num laço há sempre 3 instruções presentes:
 - inicialização de variáveis
 - verificação da condição de parada
 - atualização de variáveis

Um pouco mais sobre laços: a instrução for

- Se você pensou um pouco acerca do laço empregado na tabela de conversão de temperaturas, percebeu que num laço há sempre 3 instruções presentes:
 - inicialização de variáveis
 - verificação da condição de parada
 - atualização de variáveis
- Em geral, as duas primeiras estão próximas uma da outra e a última pode estar em qualquer ponto no interior do laço. De fato, pode ser custoso identificar o ponto em que a atualização de variáveis está sendo. Essa tarefa, por exemplo, pode estar sendo delegada a um função sendo invocada num ponto obscuro no interior de um laço com centenas de instruções!

Um pouco mais sobre laços: a instrução for

- Se você pensou um pouco acerca do laço empregado na tabela de conversão de temperaturas, percebeu que num laço há sempre 3 instruções presentes:
 - inicialização de variáveis
 - verificação da condição de parada
 - atualização de variáveis
- Em geral, as duas primeiras estão próximas uma da outra e a última pode estar em qualquer ponto no interior do laço. De fato, pode ser custoso identificar o ponto em que a atualização de variáveis está sendo. Essa tarefa, por exemplo, pode estar sendo delegada a um função sendo invocada num ponto obscuro no interior de um laço com centenas de instruções!
- A instrução de laço **for** foi criada para deixar mais simples a tarefa de se executar um laço nos casos em que as variáveis de controlo desse são contadores, ou seja, inteiros. Essa é uma situação muito comum em programação: você tem um contador que deve varrer um intervalo de valores (em passos consecutivos ou não) e para cada valor do contador, um certo número de instruções devem ser executadas.

Um pouco mais sobre laços: a instrução for

- A instrução **for** faz é reunir as 3 tarefas listadas anteriormente para o mesmo ponto: o início do laço:

```
for (inicializacao; verificacao; atualizacao){  
    instrucao  
}
```

Um pouco mais sobre laços: a instrução for

- A instrução **for** faz é reunir as 3 tarefas listadas anteriormente para o mesmo ponto: o início do laço:

```
for (inicializacao; verificacao; atualizacao){  
    instrucao  
}
```

- Vejamos como ficam, por exemplo, os programas de conversão de temperatura e de impressão dos primeiros ímpares da aula passada, agora utilizando a instrução **for**...

Um pouco mais sobre laços: a instrução for

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    float fahr, celsius;
    float inf, sup, passo;
    int i, n;
    inf = 0.;      /* limite inferior da escala de temperaturas */
    sup = 300.;    /* limite superior */
    passo = 20.;   /* tamanho do passo */
    n = (sup-inf)/passo + 1; /* calculo do numero de passos necessarios */

    for (i=1; i<=n; i++) {
        fahr = (i-1)*passo;
        celsius = 5 * (fahr-32) / 9;
        printf("%.3f\t%.3f\n", fahr, celsius);
    }

    return 0;
}
```

Um pouco mais sobre laços: a instrução for

Comparação linha a linha

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    float fahr, celsius;
    float inf, sup, passo;
    int i, n;
    inf = 0.; /* limite inferior da escala de temperaturas */
    sup = 300.; /* limite superior */
    passo = 20.; /* tamanho do passo */
    n = (sup-inf)/passo + 1; /* calculo do numero de passos necessarios */
    for (i=1; i<=n; i++) {
        fahr = (i-1)*passo;
        celsius = 5 * (fahr-32) / 9;
        printf("%6.3f\t%6.3f\n", fahr, celsius);
    }
    return 0;
}
```

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    float fahr, celsius;
    float inf, sup, passo;
    inf = 0; /* limite inferior da escala de temperaturas */
    sup = 300; /* limite superior */
    passo = 20; /* tamanho do passo */
    fahr = inf;
    while (fahr <= sup) {
        celsius = 5 * (fahr-32) / 9;
        printf("%6.3f\t%6.3f\n", fahr, celsius);
        fahr = fahr + passo;
    }
    return 0;
}
```

Um pouco mais sobre laços: a instrução for

```
#include <stdio.h>

int main() {
    int n,          /* guardara o numero dado          */
        i,          /* contador = quantidade de impares ja' impressos */
        impar;      /* usado para gerar a sequencia de numeros impares */

    /* Inicializacoes */
    printf("\nDigite o valor de n: ");
    scanf("%d", &n);
    impar = 1;

    printf("Os %d primeiros impares sao:\n", n);
    for (i=1; i<=n; i++) {
        printf ("%d\n", impar);
        impar = impar + 2; /* vai para o proximo impar */
    }

    return 0;
}
```

Operadores e precedência

- Até agora vimos os operadores:
 - **aritméticos:** $+$, $-$, $*$, $=$, $\%$
 - **de relação:** $>$, $>=$, $<$, $<=$, $==$ e $!=$

Operadores e precedência

- Até agora vimos os operadores:
 - **aritméticos:** $+$, $-$, $*$, $=$, $\%$
 - **de relação:** $>$, $>=$, $<$, $<=$, $==$ e $!=$
- Temos também os chamados operadores lógicos: $\&\&$ (e) e $||$ (ou)

Operadores e precedência

- Até agora vimos os operadores:
 - **aritméticos**: $+$, $-$, $*$, $=$, $\%$
 - **de relação**: $>$, $>=$, $<$, $<=$, $==$ e $!=$
- Temos também os chamados operadores lógicos: $\&\&$ (e) e $||$ (ou)
- Todos os operadores acima são chamados de **operadores binários**, pois para fazerem sentido, necessitam ser aplicados a um **par de operandos (variáveis ou condições lógicas)**.

Operadores e precedência

- Até agora vimos os operadores:
 - **aritméticos**: $+$, $-$, $*$, $=$, $\%$
 - **de relação**: $>$, $>=$, $<$, $<=$, $==$ e $!=$
- Temos também os chamados operadores lógicos: $\&\&$ (e) e $||$ (ou)
- Todos os operadores acima são chamados de **operadores binários**, pois para fazerem sentido, necessitam ser aplicados a um **par de operandos (variáveis ou condições lógicas)**.
- Outra classe de operadores em C é a dos **operadores unários** que são aplicados a um único operando como:
 - o operador de negação $!$ que converte um operando não-nulo em zero e um operando nulo em 1.
 - 1 $!5$ é igual a 0
 - 2 $!0$ é igual a 1
 - 3 $!(1 > 2)$ é igual a 1
 - 4 $!(\text{O Santos será campeão brasileiro de 2010})$ é igual a 0.

Operadores e precedência

- Outro operador unário visto na aula de hoje é o operador de incremento inteiro `++`: `i++` leva `i` em `i+1`;
- de maneira equivalente: `i--` leva `i` em `i-1`.

Operadores e precedência

- Voltemos uma vez mais ao programa de conversão de temperaturas. Em particular, preste atenção na linha seguinte:

```
...  
n = (sup-inf)/passo + 1; /* calculo do numero de passos necessarios *  
...
```

onde calculamos o número de passos a serem executados para construir a tabela de conversão.

Operadores e precedência

- Voltemos uma vez mais ao programa de conversão de temperaturas. Em particular, preste atenção na linha seguinte:

```
...  
n = (sup-inf)/passo + 1; /* calculo do numero de passos necessarios *  
...
```

onde calculamos o número de passos a serem executados para construir a tabela de conversão.

- ao escrevê-la, estamos supondo que o compilador será capaz de compreender que a ordem de cálculo de n é:

- 1 calcule $(\text{sup-inf})/\text{passo}$
- 2 some 1 ao resultado

e não:

- 1 calcule (sup-inf)
- 2 e divida por $\text{passo}+1$

Operadores e precedência

- A primeira ordem é a correta, porque em C, assim como em qualquer linguagem de programação os operadores possuem precedência uns sobre os outros. Em particular, dizemos que o operador `/` tem precedência sobre o operador `+`;

Operadores e precedência

- A primeira ordem é a correta, porque em C, assim como em qualquer linguagem de programação os operadores possuem precedência uns sobre os outros. Em particular, dizemos que o operador `/` tem precedência sobre o operador `+`;
- `*`, `/` e `%` têm precedência sobre `+` e `-`;
- os operadores de relação têm todos a mesma precedência entre si, mas perdem em precedência para os operadores aritméticos.

Operadores e precedência

- A primeira ordem é a correta, porque em C, assim como em qualquer linguagem de programação os operadores possuem precedência uns sobre os outros. Em particular, dizemos que o operador `/` tem precedência sobre o operador `+`;
- `*`, `/` e `%` têm precedência sobre `+` e `-`;
- os operadores de relação têm todos a mesma precedência entre si, mas perdem em precedência para os operadores aritméticos.
- Mas não se preocupe, você não precisará decorar a precedência de todos os operadores em C. **Você sempre poderá usar os parênteses!!!**

Palavras-chaves e identificadores

- Utilizamos identificadores desde a primeira aula, sem nos darmos conta.
Identificador é qualquer conjunto de variáveis (pertencente ao conjunto de caracteres reconhecido pelo C) utilizado para dar nome a algo no seu programa.
- Exemplos de identificadores que utilizamos até agora: fahr, celsius, passo, impar, etc...
- há duas restrições para a escolha de um identificador em C:
 - 1 todo identificador deve começar com um caractere do tipo alfabético: a, A, b, B, ... e se restringir aos 52 caracteres alfabéticos, os 10 dígitos e _.
 - 2 deve-se excluir nomes que façam parte do conjunto de **palavras-chaves** da linguagem.

Palavras-chaves e identificadores

- A linguagem C guarda um conjunto de palavras para uso próprio: as **palavras-chaves**
- São elas

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Para fazer em sala de aula

- 1 Ao se tentar armazenar um número maior do que o valor máximo permitido pela máquina, dizemos que ocorreu um *overflow*. Quando, pelo contrário, tentamos armazenar um número menor do que o mínimo permitido pela máquina, temos o chamado *underflow*.

Escreva um programa para determinar os limites de *overflow* e *underflow* da sua máquina.

- (a) Verifique quando e como *overflows* e *underflows* ocorrem para reais representados em ponto flutuante e precisão simples na sua máquina.
- (b) Verifique quando e como *overflows* e *underflows* ocorrem para reais representados em ponto flutuante e precisão dupla na sua máquina.
- (c) Verifique quando e como *overflows* e *underflows* ocorrem para inteiros na sua máquina.

Dica: dado que o sistema numérico de representação num computador é o binário, use fatores 2 multiplicativos para aumentar ou diminuir o valor de um número inicial.

- 2 A precisão ϵ_m de uma máquina é definida como o máximo número positivo que, no computador, pode ser somado à representação 1_c do número 1 sem que essa representação mude, isto é

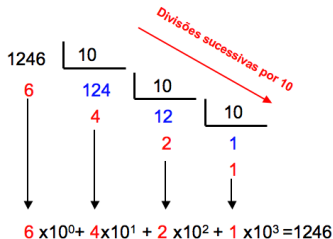
$$1_c + \epsilon = 1_c, \quad \text{sempre que} \quad \epsilon \leq \epsilon_m \quad (1)$$

Escreva um programa para determinar a precisão da sua máquina.

- (a) verifique a precisão para números reais em ponto flutuante e precisão simples.
- (b) verifique a precisão para números reais em ponto flutuante e precisão dupla.

Exercício Programa 2 - EP2 - (Entrega: Até 12/04/2010)

- 1 O algoritmo de obtenção dos dígitos decimais de um número já deve ser familiar para vocês. Tomemos, como exemplo, o número 1246_{10} , isto é, 1246 na base decimal.



O algoritmo acima deve funcionar também para a conversão da base 10 (decimal) para a base 2 (binária), simplesmente trocando-se as divisões por 10 por divisões por 2. Escreva um programa em C que dado um número natural na base decimal digitado pelo usuário, imprima na tela a representação binária desse número. Por exemplo, para o caso do número 13: $13_{10} = 1101_2$.

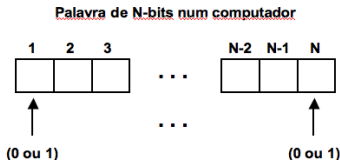
Dica: Lembre-se que em C a função `printf` possui instruções de formato para flutuantes decimais, mas não para binários. Logo, para imprimir a sequência de 0s e 1s na tela, construa um número decimal cujos dígitos decimais sejam os 0s e 1s obtidos pelo seu algoritmo.

Representação numérica em um computador

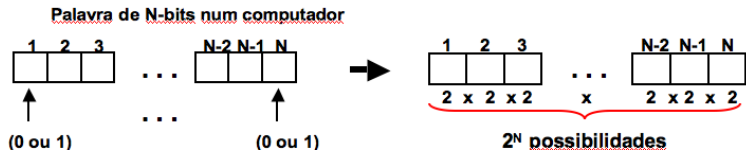
- Computadores têm se tornado cada vez mais poderosos, tanto na capacidade de armazenamento quanto na velocidade de processamento. Todavia, são e sempre serão **sistemas finitos (limitados)**.
- Assim sendo, a representação de um número qualquer (com um número arbitrário de dígitos), seja ele real ou inteiro, num computador será uma aproximação do verdadeiro valor do número.
- Se você começou a utilizar computadores no início dessa década ou no final da passada, talvez jamais tenha pensado sobre essa limitação. Mas acredite, até 30 anos atrás essa era uma das primeiras limitações que um usuário encontrava ao entrar em contato com um computador.

Representação numérica em um computador

- A representação de qualquer informação por um computador é feita no sistema binário em que a unidade fundamental é o bit: 0 ou 1.
- No final das contas, qualquer informação manipulada por uma máquina (números, palavras, sons, cores, etc) resume-se a uma sequência de zeros e uns:
Exemplo: 01001100000010111001100001
- A natureza finita da máquina para representar números está no fato de que os blocos de memória para seu armazenamento tem um tamanho finito, ou seja, um número N fixo de bits. Esse bloco de memória recebe um nome especial: **palavra** (word em inglês)
- com um bloco de memória desse tamanho, a quantidade total de números que podem ser representados é 2^N :

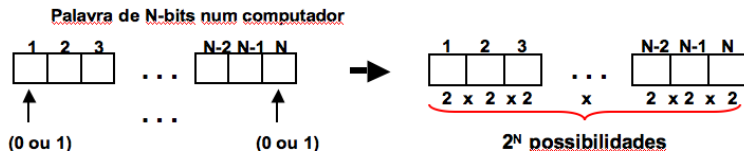


Representação numérica em um computador

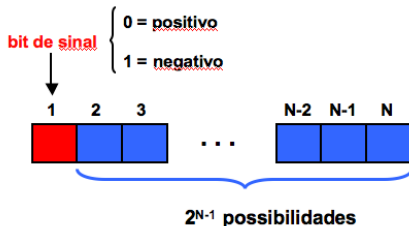


- Num computador com palavras de 8-bits, por exemplo, é possível representar 256 números diferentes (**pouca coisa, não?!).** Se só existissem números inteiros positivos, seria então possível representar os números de 0 a 255 numa máquina assim.

Representação numérica em um computador

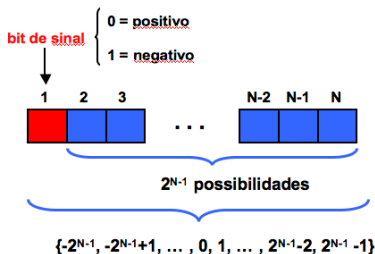


- Num computador com palavras de 8-bits, por exemplo, é possível representar 256 números diferentes (**pouca coisa, não?!).** Se só existissem números inteiros positivos, seria então possível representar os números de 0 a 255 numa máquina assim.
- Na realidade, um dos bits é sempre utilizado para armazenar o sinal dos números:



Representação numérica em um computador

- Para números inteiros temos então:



- Inteiros em máquinas com palavras de **8-bits**:
 $\{-128, \dots, 0, 1, \dots, 127\}$
- Inteiros em máquinas com palavras de **32-bits**:
 $\{-2\,147\,483\,648, \dots, 0, 1, \dots, 2\,147\,483\,647\}$
- Inteiros em máquinas com palavras de **64-bits**:
 $\{-9\,223\,372\,036\,854\,775\,808, \dots, 9\,223\,372\,036\,854\,775\,807\}$

Representação numérica em um computador

- E o que dizer sobre a representação de números reais?

Representação numérica em um computador

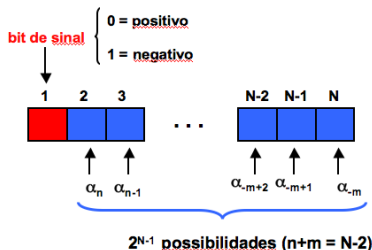
- E o que dizer sobre a representação de números reais?
- Há duas formas de representar um número real num computador: **ponto fixo** e **ponto flutuante**

Representação numérica em um computador

- E o que dizer sobre a representação de números reais?
- Há duas formas de representar um número real num computador: **ponto fixo** e **ponto flutuante**
- Em ponto fixo, a representação x_{fixo} feita por uma máquina de um número real x se escreve:

$$x_{fixo} = \text{ sinal } \times \left(\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_0 2^0 + \cdots + \alpha_{-m} 2^{-m} \right),$$

onde $\alpha_i = \{0, 1\}$.

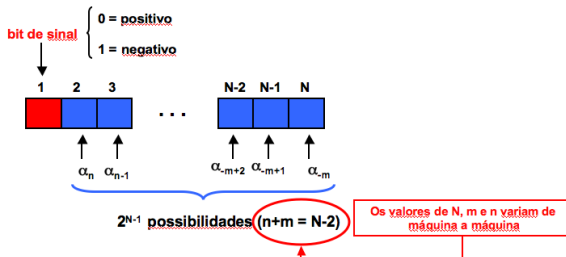


Representação numérica em um computador

- E o que dizer sobre a representação de números reais?
- Há duas formas de representar um número real num computador: **ponto fixo** e **ponto flutuante**
- Em ponto fixo, a representação x_{fixo} feita por uma máquina de um número real x se escreve:

$$x_{fixo} = \text{sinal} \times \left(\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_0 2^0 + \cdots + \alpha_{-m} 2^{-m} \right),$$

onde $\alpha_i \in \{0, 1\}$.



Representação numérica em um computador

- Reais representados em ponto fixo possuem todos a mesma **precisão absoluta**: 2^{-m-1} .

Representação numérica em um computador

- Reais representados em ponto fixo possuem todos a mesma **precisão absoluta**: 2^{-m-1} .
- No entanto, há uma grande desvantagem para cálculos numéricos nessa representação: cálculos com números pequenos são sempre menos precisos do que com números grandes.

Representação numérica em um computador

- Reais representados em ponto fixo possuem todos a mesma **precisão absoluta**: 2^{-m-1} .
- No entanto, há uma grande desvantagem para cálculos numéricos nessa representação: cálculos com números pequenos são sempre menos precisos do que com números grandes.
- Seria mais desejável ter precisão relativa constante. Por isso, a maioria absoluta dos computadores faz cálculos em ponto flutuante e não em ponto fixo.

Representação numérica em um computador

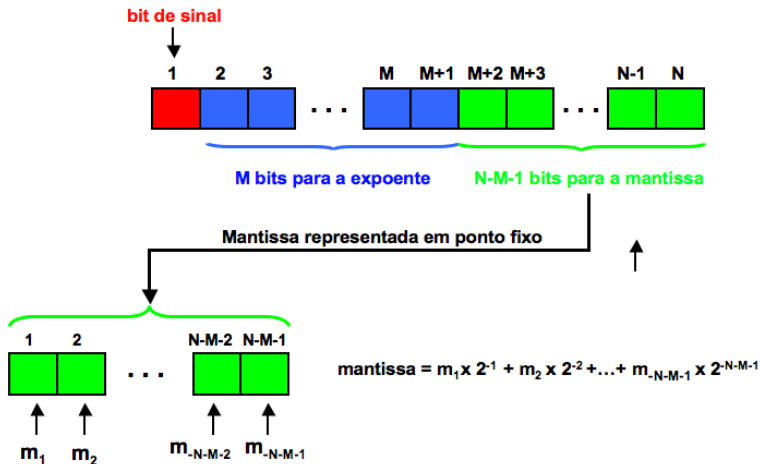
- Em ponto flutuante, a representação x_{float} feita por uma máquina de um número real x se escreve:

$$x_{float} = (-1)^{sinal} \times mantissa \times 2^{expo - bias}$$

Representação numérica em um computador

- Em ponto flutuante, a representação x_{float} feita por uma máquina de um número real x se escreve:

$$x_{float} = (-1)^{sinal} \times mantissa \times 2^{expo - bias}$$



Representação numérica em um computador

- Em ponto flutuante, a **mantissa** contém os dígitos significativos de um número;
- o termo chamado de **bias** é um inteiro fixo, enquanto o **expoente** varia e é sempre um inteiro positivo;
- Vejamos um exemplo:

Representação numérica em um computador

- Em ponto flutuante, a **mantissa** contém os dígitos significativos de um número;
- o termo chamado de **bias** é um inteiro fixo, enquanto o **expoente** varia e é sempre um inteiro positivo;
- Vejamos um exemplo:
- Tome uma máquina com palavras de 32-bits com $\text{bias}=127$. Se nessa máquina, 8-bits são usados para o expoente, ou seja, expo é um inteiro no intervalo $[0,255]$, de forma que $\text{expo}-\text{bias}$ está no intervalo $[-127,128]$.

Representação numérica em um computador

- Em ponto flutuante, a **mantissa** contém os dígitos significativos de um número;
- o termo chamado de **bias** é um inteiro fixo, enquanto o **expoente** varia e é sempre um inteiro positivo;
- Vejamos um exemplo:
- Tome uma máquina com palavras de 32-bits com $\text{bias}=127$. Se nessa máquina, 8-bits são usados para o expoente, ou seja, expo é um inteiro no intervalo $[0,255]$, de forma que $\text{expo}-\text{bias}$ está no intervalo $[-127,128]$.
- Os 23 bits restantes são usados para a mantissa, de forma que números reais nessa máquina terão de 6 a 7 dígitos de precisão ($1/2^{23}$).

Representação numérica em um computador

- Em ponto flutuante, a **mantissa** contém os dígitos significativos de um número;
- o termo chamado de **bias** é um inteiro fixo, enquanto o **expoente** expo varia e é sempre um inteiro positivo;
- Vejamos um exemplo:
- Tome uma máquina com palavras de 32-bits com bias=127. Se nessa máquina, 8-bits são usados para o expoente, ou seja, expo é um inteiro no intervalo $[0,255]$, de forma que expo-bias está no intervalo $[-127,128]$.
- Os 23 bits restantes são usados para a mantissa, de forma que números reais nessa máquina terão de 6 a 7 dígitos de precisão ($1/2^{23}$).
- Nessa máquina, o número 0.5 é representado por:

