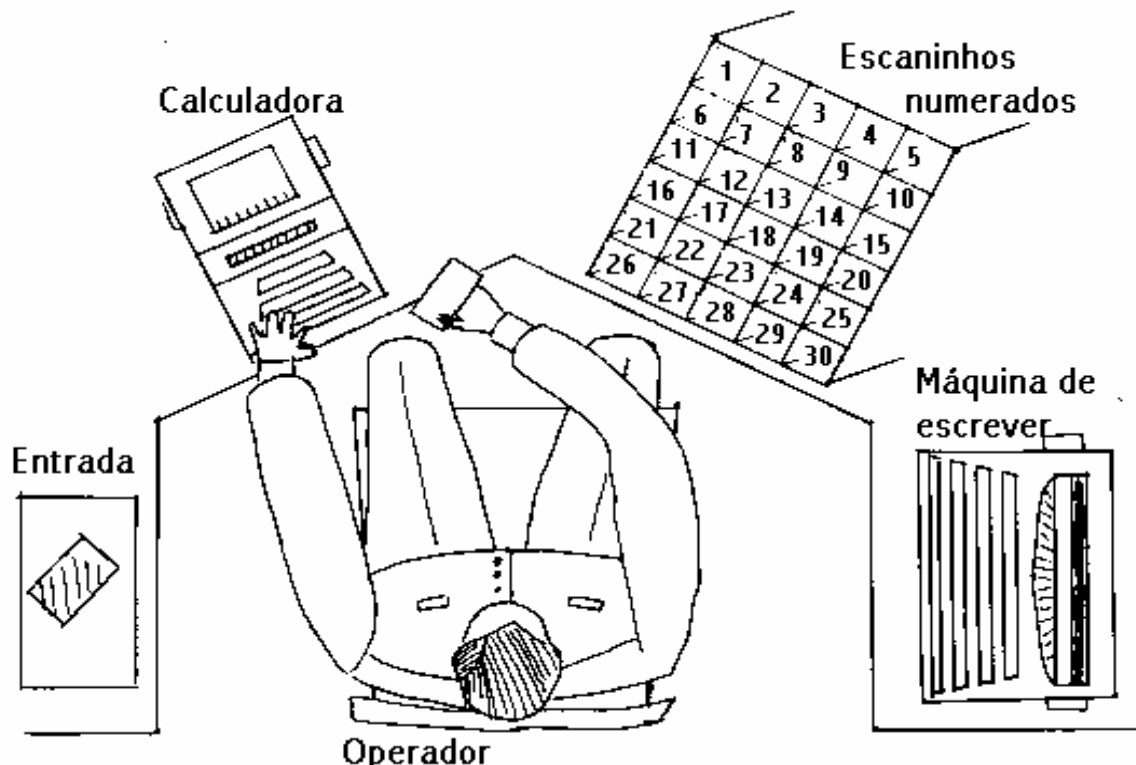


Cap. I- CONCEITOS BÁSICOS DE PROGRAMAÇÃO

1.1 - Computador

1.1.1 - Analogia de funcionamento



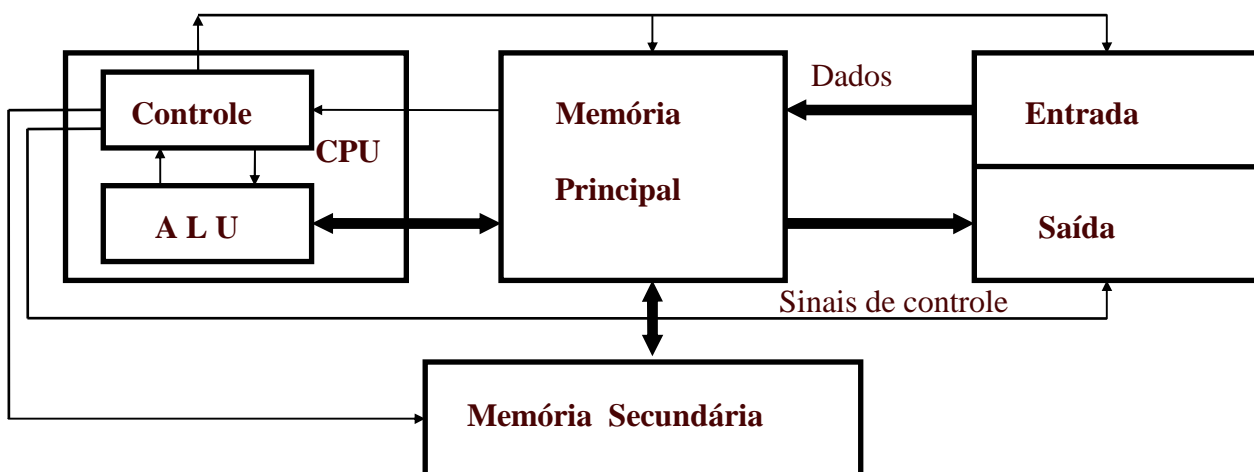
Conteúdo dos escaninhos:

- | | |
|--|---|
| 1- Zerar o conteúdo do Esc 14 | 8- Executar a instrução do Esc 3 |
| 2- Zerar o conteúdo do Esc 15 | 9- Se Esc 15 tiver zero, executar a instrução do Esc 13 |
| 3- Ler um número | 10- Dividir Esc 14 por Esc 15 |
| 4- Se o número for negativo, executar a instrução do Esc 9 | 11- Atualizar Esc 16 com tal divisão |
| 5- Somar o número lido com Esc 14 | 12- Imprimir Esc 16 |
| 6- Atualizar Esc 14 com tal soma | 13- Encerrar |
| 7- Somar 1 ao Esc 15 | |

- Tipos de instruções vistas:
Entrada de dados, aritméticas (e lógicas), controle do programa, movimentação de dados, saída de resultados.

1.1.2 - Unidades básicas de um computador

- Unidades de entrada: teclado, mouse, leitora de cartões, leitora ótica, sensores em geral, microfone, câmara ótica
- Unidade de saída: vídeo, impressoras em geral, plotter, auto-falante, controladores de processos em geral
- Memória principal
- Memória secundária: winchester, drive de disquetes, fita
- Unidade central de processamento (CPU):
 - Unidade lógica e aritmética (ALU)
 - Unidade de controle



1.1.3 - Memória

- **Memória:** conjunto de compartimentos numerados de igual capacidade, chamados de palavras.
- **Bit:** unidade de armazenamento que pode guardar um de dois valores possíveis: o **zero** (0) ou o **um** (1).
- **Byte:** conjunto de 8 bits; **Palavra:** conjunto de 1, 2, 4 ou 8 bytes.
- A memória armazena:
 - Números inteiros: 1, 2 ou 4 bytes
 - Números reais: 4, 8 ou 16 bytes
 - Caracteres alfanuméricos: 1 byte
 - Valores lógicos: 1 bit ou 1 byte
 - Instruções: 1, 2 ou mais bytes

Sistema binário de numeração

0	0010	1011	0000	1000
1	1011	0101	0111	0100
2	0000	0010	0011	1111
3	0100	1110	0110	1000
...				
1048575	1000	1111	1111	1111

Memória de
1 Megapalavras
de 16 bits ou
2 bytes

1.2 - Evolução das Linguagens de Programação

• Exemplo 1.1: Cálculo do fatorial do número 5

Linguagem C:		Linguagem Assembly e de Máquina:			
void main ()		End	Máquina		Assembly
{		0	5		c5: const 5
int n, fat, i;		1	1		c1: const 1
n = 5;		2	0		n: const 0
fat = 1;		3	0		fat: const 0
i = 1;		4	0		i: const 0
while (i <= n) {		5	1	0	inic: load c5
fat = fat * i;		6	2	2	store n
i = i + 1;		7	1	1	load c1
}		8	2	3	store fat
}		9	2	4	store i
		10	6	2	loop: sub n
		11	16	19	jp parar
		12	1	3	load fat
		13	7	4	mult i
		14	2	3	store fat
		15	1	4	load i
		16	5	1	add c1
		17	2	4	store i
		18	15	10	jump loop
		19	21	0	parar: stop
					end inic
Tabela das instruções:					
Nome	Número				
load	1				
store	2				
add	5				
sub	6				
mult	7				
jump	15				
jp	16				
stop	21				

- **Linguagem de máquina:** tediosa; sujeita a erros; difícil compreensão; exige conhecimento da estrutura interna do computador.
- **Linguagem Assembly:** programas de difícil interpretação; exige conhecimento da estrutura interna do computador; exige tradutor (**Assembler**).
- **Linguagens de alto nível:** próximas das linguagens dos seres humanos; formais; sem ambigüidades; não exigem conhecimento da estrutura interna do computador; exigem tradutor bem mais complexo (**Compilador**).
- **Linguagens aplicativas:** implementadas por programas escritos em linguagens de alto nível; Exemplos: linguagens para manipulação de banco de dados, de fórmulas matemáticas, de figuras geométricas, para gerar analisadores léxicos e sintáticos de compiladores, para simulação, etc..
- **Ambientes de trabalho:** usando mouse e recursos gráficos tais como manipuladores de janelas muito mais que teclado e cursor seqüencial de vídeo, proporcionam maior comodidade ao usuário.

1.3 - Sistemas de Numeração

1.3.1 - Números inteiros

- Alfabeto de sistemas de numeração:

Sistema	Base	Alfabeto
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binário	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- Equivalência entre os sistemas:

$$(29)_{\text{base } 10} = (11101)_{\text{base } 2} = (35)_{\text{base } 8} = (1D)_{\text{base } 16}$$

- Converter das bases 2, 8 e 16 para a base 10:

$$(11100101)_2 = 2^7 + 2^6 + 2^5 + 2^2 + 2^0 = (229)_{10}$$

$$(271)_8 = 2 * 8^2 + 7 * 8^1 + 1 * 8^0 = (185)_{10}$$

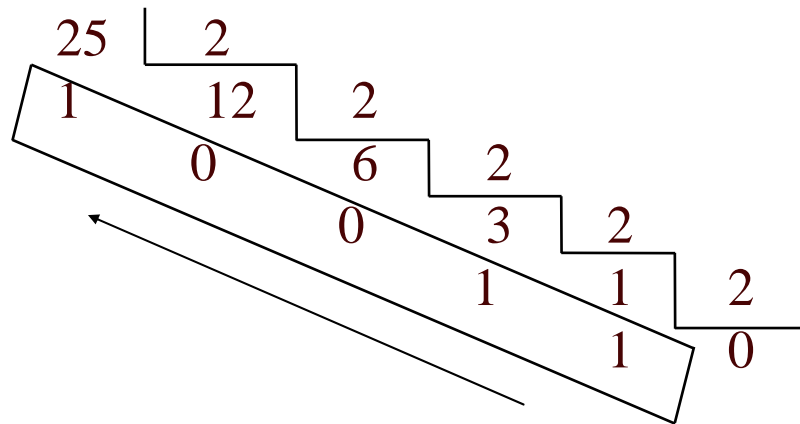
$$(2AB3)_{16} = 2 * 16^3 + 10 * 16^2 + 11 * 16^1 + 3 * 16^0 = (10931)_{10}$$

- Converter da base 10 para as bases 2, 8 e 16:

Divisões sucessivas do número na base 10 pela **base alvo**, colhendo-se os restos e invertendo a ordem de sua produção.

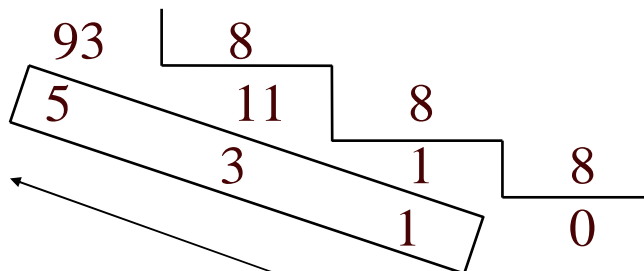
• **Exemplo 1.2:** conversões da base 10 para outras bases:

- $(25)_{10}$ para a base 2:



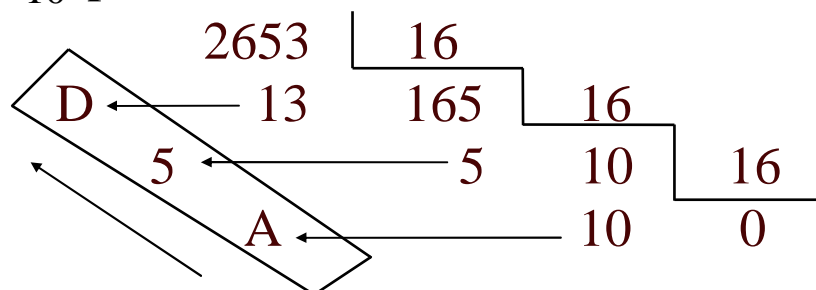
então $(25)_{10} = (11001)_2$

- $(93)_{10}$ para a base 8:



então $(93)_{10} = (135)_8$

- $(2653)_{10}$ para a base 16:



então $(2653)_{10} = (A5D)_{16}$

- Representação de números inteiros com sinal: representação mais usada: **Complemento de 2** (Comp_2): Por exemplo, sejam os números inteiros guardados em 5 bits; o bit mais a esquerda representa o **sinal** (**zero** para números não negativos e **um** para números negativos).

Tabela dos números decimais e seus códigos em Comp_2:

N.o	Comp_2	N.o	Comp_2
+15	01111	-1	11111
+14	01110	-2	11110
+13	01101	-3	11101
+12	01100	-4	11100
+11	01011	-5	11011
+10	01010	-6	11010
+9	01001	-7	11001
+8	01000	-8	11000
+7	00111	-9	10111
+6	00110	-10	10110
+5	00101	-11	10101
+4	00100	-12	10100
+3	00011	-13	10011
+2	00010	-14	10010
+1	00001	-15	10001
0	00000	-16	10000

- **Exemplo 1.3:** dado um número decimal em $[-16, +15]$, achar sua representação em Comp_2 em 5 bits

a) se o número for não negativo: é só achar o correspondente binário em 5 bits

Ex: Comp_2 (9) = 01001

b) se o número for negativo: é só achar seu módulo complementado somado com 1

Ex: Comp_2 (-6) = 11010 - ou seja,
Módulo de 6 em 5 bits: 00110
Módulo complementado: 11001
Somado com 1: 11010

- **Exemplo 1.4:** dada uma representação em Comp_2, achar o decimal correspondente

a) Se o bit sinal for '0', é só converter para a base 10

Ex: 01101 é Comp_2 de 13

b) Se o bit sinal for '1', complementa-se a representação, soma-se 1 e converte-se para a base 10.

Ex: 11001 complementado fica 00110;
somado com 1: 00111; isto vale 7 na base 10;
então 11001 é o Comp_2 de -7 na base 10.

1.3.2 - Números reais

- Nas conversões de base, converte-se a parte inteira separada da parte fracionária.
- Como já foi vista a conversão de inteiros, será vista agora a conversão da parte fracionária de números reais

a) Conversão das bases 2, 8 e 16 para a base 10

- $(0.1011)_2 = 1 * 2^{-1} + 1 * 2^{-3} + 1 * 2^{-4} = (0.6875)_{10}$
- $(0.307)_8 = 3 * 8^{-1} + 7 * 8^{-3} = (0.388671875)_{10}$
- $(0.2B)_{16} = 2 * 16^{-1} + 11 * 16^{-2} = (0.16796875)_{10}$

b) Conversão da base 10 para as bases 2, 8 e 16

- $(0.5625)_{10}$ para a base 8:

$$\begin{array}{rcl} 8 * 0.5625 & = & \boxed{4} + 0.5 \\ 8 * 0.5 & = & \boxed{4} + 0.0 \end{array}$$

$$(0.5625)_{10} = (0.44)_8$$

- $(0.169189453125)_{10}$ para a base 16:

$$\begin{array}{rcl} 16 * 0.169189453125 & = & \boxed{2} + 0.70703125 \\ 16 * 0.70703125 & = & \boxed{11} + 0.3125 \\ 16 * 0.3125 & = & \boxed{5} + 0.0 \end{array}$$

$$(0.169189453125)_{10} = (0.2B5)_{16}$$

- $(0.3)_{10}$ para a base 2:

$$\begin{array}{rcl}
 2 * 0.3 & = & 0 + 0.6 \\
 2 * 0.6 & = & 1 + 0.2 \\
 2 * 0.2 & = & 0 + 0.4 \\
 2 * 0.4 & = & 0 + 0.8 \\
 2 * 0.8 & = & 1 + 0.6 \\
 2 * 0.6 & = & 1 + 0.2
 \end{array}
 \quad \text{Dízima periódica}$$

$$(0.3)_{10} = (0.0 \underline{1001} \underline{1001} \cdots)_2$$

c) Sistemas de ponto-flutuante

- Um número real sempre pode ser colocado na forma:

$$X = 0.< \text{mantissa} > * < \text{base} >^{< \text{expoente} >}$$

- No computador, armazena-se $< \text{mantissa} >$ e $< \text{expoente} >$, além do sinal do número.
- $< \text{base} >$ é implícita, varia de plataforma para plataforma (circuitos ou programas) e geralmente é potência de 2.
- **Exemplo 1.5:** um computador utiliza 32 bits (4 bytes) para armazenar um número real; 1bit para o sinal, 8 bits para o expoente e 23 bits para a mantissa; a base é 2.

Sinal	Mantissa	Expoente
1 bit	23 bits	8 bits

O expoente não deve ter bit-sinal; para armazená-lo soma-se a ele 128:

$$\langle \text{característica} \rangle = \langle \text{expoente} \rangle + 128$$

Como ficam armazenados nesse computador os números 0.5625, 2.3 e 0.0625 ?

- $0.5625 = (0.1001)_2 = (0.1001)_2 * 2^0$
 característica = $0 + 128 = 128 = (10000000)_2$
 mantissa = 100100000000000000000000

0	100100000000000000000000	10000000
---	--------------------------	----------

- $2.3 = (10.0\ 1001\ 1001\ 1001\ \dots)_2 * 2^0$

É necessário **normalizar** (colocar o primeiro bit 1 da mantissa à direita do ponto, o mais próximo dele possível).

$$2.3 = (0.100\ 1001\ 1001\ 1001\ \dots) * 2^2$$

característica = $2 + 128 = 130 = (10000010)_2$
 mantissa = 100 1001 1001 1001 1001 1001

0	100 1001 1001 1001 1001 1001	10000010
---	------------------------------	----------

Ocorreu um truncamento da mantissa: imprecisão.

- $0.0625 = (0.0001)_2 * 2^0$ (pede normalização)
 $= (0.1)_2 * 2^{-3}$

$$\text{característica} = -3 + 128 = 125 = (01111101)_2$$

0	10000000000000000000000000000000	01111101
---	----------------------------------	----------

1.4 - Passos para a Elaboração de um Programa

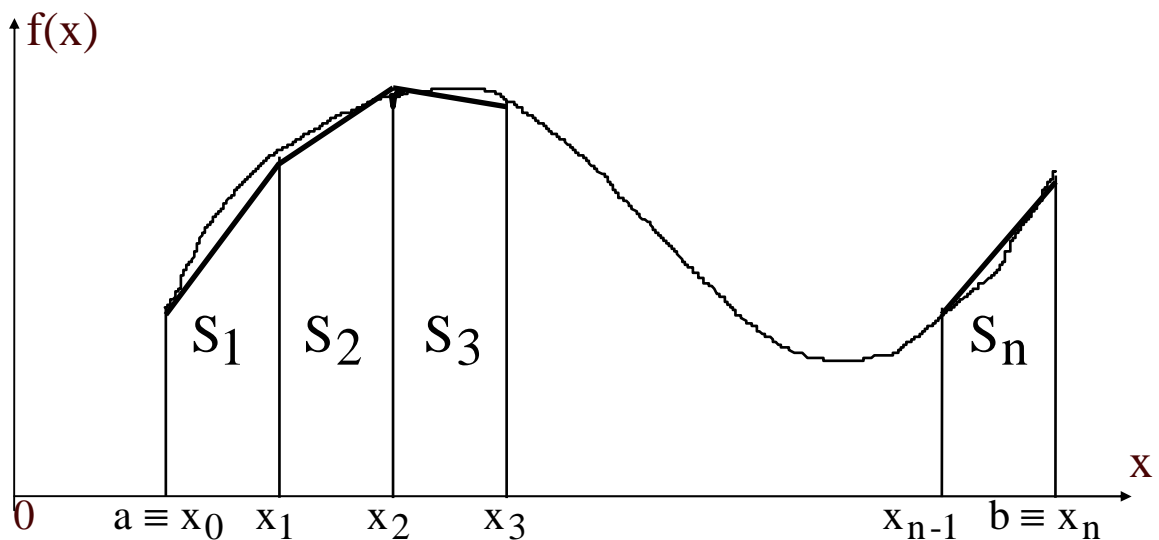
- Programar é muito mais do que fazer uma lista de comandos numa linguagem de programação.
- Passos envolvidos:
 - 1) Definição do problema
 - 2) Determinação do método de solução
 - 3) Desenvolvimento de um algoritmo
 - 4) Codificação do algoritmo na linguagem escolhida
 - 5) Compilação do programa
 - 6) Correção dos erros de sintaxe
 - 7) Correção dos erros de lógica
 - 8) Execução correta
 - 9) Utilização do programa
 - 10) Otimização e manutenção
- **Exemplo 1.6:** Cálculo de integrais definidas

1) Definição do problema

- Calcular a integral de uma função $f(x)$ sempre positiva, no intervalo $[a, b]$ com uma precisão p .

2) Determinação do método de solução

- Regra do Trapézio com refinamento:



- $S = \sum_{i=1}^n S_i$ E as subáreas são aproximadas para trapézios
- $S_i = \frac{\Delta x * (f(x_{i-1}) + f(x_i))}{2}$
- Calcula-se S para $n = 10, 20, 40, 80, \dots$; quando a diferença de duas S 's consecutivas for menor que p , supõe-se que a precisão foi alcançada

3) Desenvolvimento de um algoritmo

Integral_f_trapézio {

Ler (a, b, p);

S2 \leftarrow 0;

n \leftarrow 5;

Repetir {

a, b, p: reais de precisão simples;

i, n: inteiros de tamanho duplo;

S1, S2, Sq, Δx : reais de precisão dupla;

f: função real de precisão dupla;

S1 \leftarrow S2;

n \leftarrow 2 * n;

$\Delta x \leftarrow (b - a) / n$;

S2 \leftarrow 0;

Para (i \leftarrow 1; i \leq n; i \leftarrow i + 1) {

Sq $\leftarrow \Delta x * (f(a + (i-1)*\Delta x) + f(a + i*\Delta x)) / 2$;

S2 \leftarrow S2 + Sq;

}

} **Enquanto** | S2 - S1 | \geq p;

Escrever (“A área de f(x) no intervalo [”, a, “ , ”,
b, “] é ”, S2);

}

4) Codificação do algoritmo na Linguagem C

```

#include<stdio.h>
#include<math.h>

double f (double x)
{
    return (log10(x)+5);
}

void main ( )
{
    long n, i; float a, b, p;
    double s1, s2, sq, dx;

    scanf ("%f%f%f", &a, &b, &p);
    s2 = 0; n = 5;
    do {
        s1 = s2; n = 2*n;
        dx = (b-a)/n; s2 = 0;
        for (i = 1 ; i <= n; i = i+1) {
            sq = dx * (f(a+(i-1)*dx)+f(a+i*dx)) / 2;
            s2 = s2 + sq;
        }
    } while (fabs(s1-s2) >= p);
    printf ("Integral de f(x), intervalo [%f, %f]: %g\n", a, b,
        s2);
    printf ("Precisao: %f, numero de sub-intervalos: %ld", p,
        n);
}

```