Notas de Aula de
Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083–970 Campinas–SP
{fkm,tomasz}@ic.unicamp.br

1 Introdução à Computação

1.1 Organização do Computador

Um computador é uma coleção de componentes que realizam operações lógicas e aritméticas sobre um grande volume de dados. Na figura 1 apresentamos uma organização básica em um computador seqüêncial.

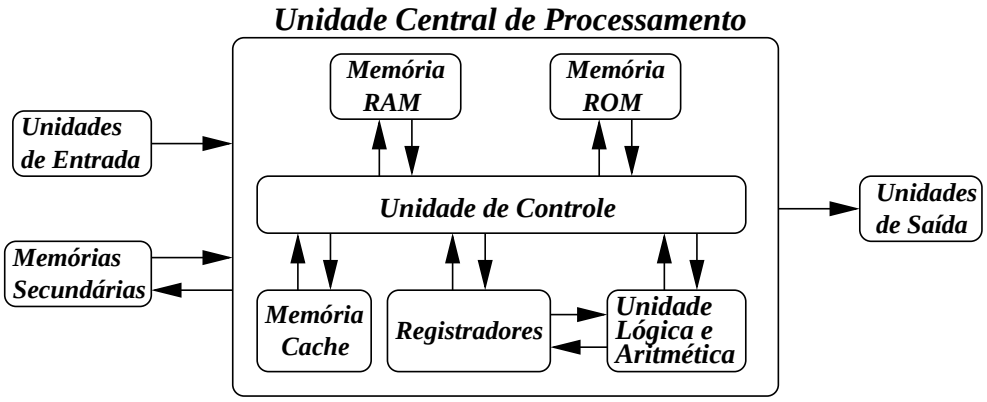


Figura 1: Organização Básica de um Computador Seqüêncial.

A seguir descreveremos cada uma destas partes.

Unidade de Entrada São os componentes que permitem a entrada de informações exteriores para serem processadas pelo computador. Exemplo: teclado, mouse, câmera de vídeo, etc.

Unidade de Saída São os componentes que permitem a apresentações de informações processadas para o meio externo. Exemplo: monitor, impressora, etc.

Unidade Central de Processamento Também conhecida como CPU *(Central Processing Unit)*. É responsável pela execução dos programas e pelo comportamento das outras unidades no sistema. É capaz de fazer contas matemáticas e fazer decisões simples. As principais partes da CPU são: a Unidade Lógica e Aritmética, Unidade de Controle e Memórias (Registradores, Memória Principal (ou Memória RAM), Memória ROM e Cache).

Unidade Lógica e Aritmética Parte da CPU que realiza operações aritméticas (soma, subtração, multiplicação, divisão, resto, troca de sinal, etc) e operações lógicas (*and, or, not, xor,* etc).

Memória Principal É usado na CPU para manter instruções e dados. Também conhecido como Memória RAM *(Random Access Memory)*. A recuperação dos dados é feita através de circuitos lógicos e por isso é rápida. Não é tão grande, já que depende muito da tecnologia de integração destes circuitos. É uma memória volátil, i.e., quando o computador é desligado, todos os dados nesta memória se perdem.

Memória ROM ROM *(Read Only Memory)* é uma memória que contém dados e códigos de execução que não podem ser alterados. Uma das aplicações desta memória é manter código de execução para a leitura e execução de um sistema operacional.

Memória Cache Memória rápida projetada para guardar dados que foram recentemente acessados. Para buscar um certo dado na memória RAM, é considerado se este pode estar na memória cache, e em caso positivo a busca na memória RAM é interrompido e este é recuperado diretamente da memória cache. Tem tempo de acesso mais rápido que a memória RAM.

Registradores Memórias de alta velocidade ligada a operações de cálculos lógicos e aritméticos. Em geral em quantidade e tamanhos pequenos.

Unidade de Controle Parte da CPU que busca na memória a próxima instrução e a decodifica para ser executada. Dependendo da instrução, pode-se ter uma transferência do controle para a unidade lógica e aritmética ou o envio de dados para os componentes externos à CPU.

Memória Secundária Memória para armazenamento a longo prazo. Os dados armazenados nesta memória não são perdidos quando se desliga o computador. Em geral de dimensões maiores que a Memória RAM mas de acesso mais lento, já que envolvem o uso de dispositivos mecânicos. Ex. Discos rígidos, disquetes, fitas magnéticas, etc.

Podemos ver que há diversos tipos de memórias em um computador. Cada uma destas memórias usa tecnologia que reflete no custo, na velocidade de acesso e na quantidade de armazenamento. A seguinte ordem apresenta algumas memórias ordenadas, de maneira crescente, pela quantidade de armazenamento:

Registrador – Memória Cache – Memória RAM – Discos Rígidos.

Esta mesma ordem apresenta o custo relativo e a velocidade de maneira decrescente.

1.2 Alguns Termos Técnicos

Hardware Componentes mecânicos e eletro- eletrônicos que compõem o computador. Parte *dura* do computador.

Software Seqüência de instruções e comandos que fazem o computador realizar determinada tarefa., também chamados de *programas de computador*. Devem estar armazenados em algum tipo de memória.

Periférico É qualquer componente do computador *(hardware)* que não seja a CPU. Exemplos: leitoras de disquete, monitores, teclados, vídeo, impressoras, etc.

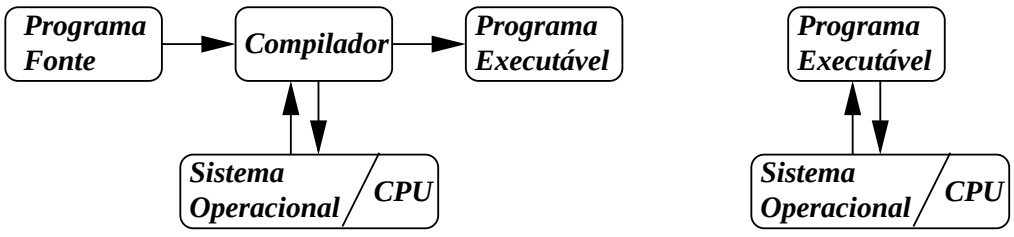
Sistema Operacional Coleção de programas que gerencia e aloca recursos de hardware e software. Exemplos de tarefas que um sistema operacional realiza são: leitura de dados pelo teclado, impressão de informações no vídeo, gerenciamento da execução de vários programas pela CPU, gerenciamento da memória principal e da memória secundária para uso dos programas em execução, etc. Exemplos: Linux, Unix, Windows98, OS2, MS-DOS, etc.

Linguagem de Máquina Conjunto de instruções que podem ser interpretados e executados diretamente pela CPU de um dado computador. É específica para cada computador.

Linguagem Assembler *(Linguagem de Baixo Nível)* Representação da linguagem de máguina através de códigos mnemônicos. Também é específica de cada máquina.

Linguagem de alto nível Linguagem que independe do conjunto de instruções da linguagem de máquina do computador. Cada instrução de alto nível equivale a várias instruções da linguagem de máquina, sendo assim mais produtiva. Ex.: Pascal, C, Algol, BASIC, Lisp, Prolog, etc.

Compilador Tradutor de programas escritos em uma linguagem de programação para programas em linguagem de máquina. Uma vez que o programa foi convertido para código de máquina, este pode ser executado independente do compilador e do programa original. Veja a figura 2.

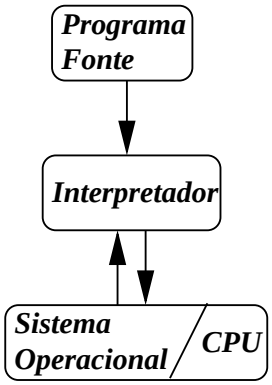


(a) Geração do Programa Executável ***(b) Execução do Programa***

Figura 2: Etapas para execução de um programa compilado.

Interpretador É um programa que executa outros programas escritos em alguma linguagem de programação. A execução de um programa interpretado é em geral mais lenta que o programa compilado. Por outro lado, o uso de programas interpretados permite que trechos de códigos possam ser trocados por novos facilmente, fazendo com

que o programa fonte possa mudar durante sua execução. Este é um dos grandes motivos de se usar programas interpretados em sistemas especialistas. Duas linguagens para as quais podemos encontrar interpretadores são Lisp e Prolog. Veja a figura 3.



Execução de programa interpretado

Figura 3: Execução de um programa interpretado.

Algoritmo É a descrição de uma seqüência de ações para realizar alguma tarefa. Em geral, estaremos interessados em algoritmos computacionais, que descrevem uma seqüência de ações que podem ser traduzidos para alguma linguagem de programação.

Uma maneira para calcular o MDC (Máximo Divisor Comum) de dois números inteiros positivos m e n é através do Algoritmo de Euclides apresentado no quadro seguinte.

Passo 1: Adote $x = m$ e $y = n$;
Passo 2: Adote $r =$(resto de x dividido por y);
Passo 3: Adote novos valores $x = y$ e $y = r$;
Passo 4: Se r é diferente de 0, volte ao passo 2; senão pare com a resposta x.

Algoritmo de Euclides

O seguinte programa apresenta uma versão mais estilizada:

Passo1:	Dados: m e n.
Passo2:	$x \leftarrow m$.
Passo3:	$y \leftarrow n$.
Passo4:	Repita
Passo4.1:	$r \leftarrow \text{resto}(x, y)$;
Passo4.2:	$x \leftarrow y$;
Passo4.3:	$y \leftarrow r$;
Passo4.4:	Até que $r = 0$.
Passo5:	Imprima o resultado x.

Algoritmo de Euclides Estilizado.

O seguinte programa apresenta uma versão na linguagem Pascal:

```
Program Euclides;
var x, y, r, m, n : integer;
begin
        Readln(m,n);
        x := m;
        y := n;
        repeat
                r := (x mod y);
                x := y;
                y := r;
        until r = 0;
        writeln(x);
end.
```

Implementação do Algoritmo de Euclides em Pascal

1.3 Bits e Bytes

A menor unidade de informação usada pelo computador é o *bit*. Este tem atribuições lógicas **0** ou **1**. Cada um destes estados pode, internamente, ser representado por meios eletro-magnéticos (negativo/positivo, ligado/desligado, etc). É por isso que é mais fácil para armazenar dados em formato binário. Assim, todos os dados do computador são representados de forma binária. Mesmo os números são comumente representados na base 2, em vez da base 10, e suas operações são feitas na base 2.

Um conjunto de 8 bits é chamado de *byte*. Um byte pode ter até $2^8 = 256$ configurações diferentes. O principal padrão usado para representar caracteres (*'a','b','c',...,'A','B','C',...,'!','@','#','$',...*) é o padrão ASCII (*American Standard Code for Information Interchange*), usada na maioria dos computadores. Cada um destes caracteres é representado por um byte. A tabela 1 apresenta o código de alguns caracteres no código ASCII: Observe que:

1. As codificações para letras em maiúsculas e minúsculas são diferentes.
2. A codificação de 'B' é a codificação de 'A' somado de 1; a codificação de 'C' é a codificação de 'B' somado de 1; assim por diante. Esta codificação permite poder comparar facilmente se um caracter vem antes do outro ou não. Internamente, verificar se o caracter 'a' vem antes do 'b', é verificar se o número binário correspondente a 'a' é menor que o número binário correspondente a 'b'.
3. As letras maiúsculas vem antes das minúsculas.

As seguintes denominações são comumente usadas na área de informática

nome	*memória*
bit	$\{0, 1\}$
byte	8 bits
kilobyte (kbyte)	2^{10} bytes (pouco mais de mil bytes ($2^{10} = 1024$))
megabyte	2^{20} bytes (pouco mais de um milhão de bytes)
gigabyte	2^{30} bytes (pouco mais de um bilhão de bytes)

Atualmente, configurações de computador com 64 megabytes de memória RAM, 4,2 gigabytes de disco rígido, disco flexivel de 1,44 megabytes são muito comuns no mercado. Certamente esta configuração já será considerada pequena dentro de um ou dois anos, devido ao contínuo avanço da tecnologia nesta área.

Vejamos alguns exemplos de quanto é esta memória. Uma página de um livro, armazenada em formato ASCII, tem em torno de 50 linhas e 80 caracteres por linha. Assim, um livro de 1000 páginas teria algo em torno de 4.000.000 de caracteres, que poderiam ser guardados em 4 megabytes. Assim, um disco rígido de 4,2 gigabytes poderia guardar

Caracter	Representação em ASCII	Valor na base decimal
$\vdots$	$\vdots$	$\vdots$
$($	00101000	40
$)$	00101001	41
$\star$	00101010	42
$+$	00101011	43
$\vdots$	$\vdots$	$\vdots$
0	00110000	48
1	00110001	49
2	00110010	50
3	00110011	51
$\vdots$	$\vdots$	$\vdots$
A	01000001	65
B	01000010	66
C	01000011	67
D	01000100	68
$\vdots$	$\vdots$	$\vdots$
a	01100001	97
b	01100010	98
c	01100011	99
d	01100100	100
$\vdots$	$\vdots$	$\vdots$

Tabela 1:

em torno de 1.000 livros deste tipo. Isto aparenta uma quantidade bastante grande de dados. Por outro lado, a maioria das aplicações atuais está fazendo uso cada vez maior de imagens, gráficos e sons. Estas aplicações demandam muita memória. Por exemplo, se você quiser representar uma imagem de tamanho 1000×1000 pontos (10^6 pontos), cada ponto com uma cor entre 65000 cores possíveis (dois bytes por ponto), gastaremos algo como 2 megabytes para armazenar apenas uma imagem deste tipo. A quantidade de memória aumenta quando armazenamos filmes, que usam em torno de 30 imagens por segundo. Apesar do uso de métodos de compressão sobre estes tipos de dados a necessidade de grande quantidade de memória ainda é crucial para muitas aplicações.

1.4 Base Binária, Base Decimal, ...

Como vimos, é muito mais fácil armazenar os dados na base binária que na base decimal. Assim, muitas das operações usadas no computador são feitas na base binária.

Muito provavelmente, nós usamos a base decimal porque temos 10 dedos nas duas mãos. E se tivéssemos 8 dedos em vez de 10 ? Neste caso, provavelmente estaríamos usando a base *octal.* Bom, agora imagine que você tem apenas dois dedos. Neste raciocínio, usaremos o sistema binário !!

Primeiro, vamos lembrar o que representa o número 4027 na base decimal.

$$4 \cdot 10^3 + 0 * 10^2 + 2 * 10^1 + 7 * 10^0$$

Agora um número binário tem apenas os dígitos **0** e **1**. O número 100110_b no sistema binário representa o número:

$$1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

Isto nos dá o número

$$32 + 4 + 2 = 38$$

Assim o número 100110_b no sistema binário é igual ao número 38 no sistema decimal.

As operações aritméticas também podem ser feitas em binário. Por exemplo: Vamos somar o número acima (100110_b) com (111_b).

$$\begin{array}{r} 100110_b \\ +\ 111_b \\ \hline 101101_b \end{array}$$

Agora vamos conferir o resultado: $111_b = 4 + 2 + 1 = 7$. E $101101_b = 32 + 8 + 4 + 1 = 45$. De fato, este número está correto. Em decimal seria $38 + 7 = 45$.

Exercício 1.1 *Dado um número no sistema decimal, encontre uma maneira de escrevê-lo no sistema binário.*

Assim, em um byte (8 bits), é possível representar os números de 0 até 255 ($255 = 2^8 - 1$).

binário	decimal
00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
⋮	⋮
11111110	254
11111111	255

Da mesma forma, em dois bytes (16 bits) é possível representar os números de 0 até $65535 = 2^{16} - 1$.

Muitas vezes um compilador (ou mesmo o próprio computador) tem uma estrutura para definir sinais (se negativo ou positivo).

Um exemplo disso é usar um bit para representar o sinal do número. Por exemplo, um número de 16 bits pode ter a representação interna com um bit para sinal e os outros 15 bits para o número propriamente dito. Neste exemplo, poderíamos representar números de $-(2^{15} - 1)$ até $2^{15} - 1$ (i.e., de -32767 até 32767). Note que neste exemplo o número 0 é representado duas vezes ($+0$ e -0). Na prática, as representações internas levam isto em consideração de forma que cada número é representado apenas uma vez, neste caso podendo ir de -32768 até 32767.

É interessante observar que números positivos não nulos que são potência de 2 são números que têm todos os bits iguais a 0, exceto um bit. Assim, a multiplicação de um número inteiro x por um inteiro positivo não nulo que é potência de 2 faz apenas um deslocamento dos bits de x de algumas casas. Por exemplo, a multiplicação de x por 8 (2^3) faz o deslocamento dos bits de x de 3 casas para a esquerda.

$x =$				b_1	b_2	b_3	b_4	$\ldots$	b_{n-4}	b_{n-3}	b_{n-2}	b_{n-1}	b_n
$2^3 =$										1	0	0	0
				0	0	0	0	$\ldots$	0	0	0	0	0
			0	0	0	0	$\ldots$	0	0	0	0	0	
$+$		0	0	0	0	$\ldots$	0	0	0	0	0		
	b_1	b_2	b_3	b_4	$\ldots$	b_{n-4}	b_{n-3}	b_{n-2}	b_{n-1}	b_n			
$x \cdot 2^3 =$	b_1	b_2	b_3	b_4	$\ldots$	b_{n-4}	b_{n-3}	b_{n-2}	b_{n-1}	b_n	0	0	0

Assim, muitos compiladores, ao encontrar a multiplicação de um inteiro por uma potência de 2, trocam esta multiplicação por um deslocamento de bits.

Quando a operação é para obter a parte inteira da divisão de um inteiro x por uma potência de dois, digamos 2^k, basta deslocar os bits de x de k casas para a direita, perdendo k bits que estão mais a direita.

Também é sabido que a multiplicação de inteiros em geral leva mais tempo que somas e deslocamento de bits. Assim, uma possível otimização feita por compiladores é trocar a multiplicação de um inteiro por somas e deslocamento de bits. Por exemplo, digamos que desejamos obter $x \cdot 10$. O inteiro 10 não é potência de 2, mas em vez de fazermos

a multiplicação por 10, podemos reescrever esta multiplicação por $x \cdot (8+2)$. I.e., podemos fazer $x \cdot 2^3 + x \cdot 2^1$. Desta maneira trocamos uma multiplicação de um inteiro por dois deslocamentos e uma soma, o que em muitos computadores é feito de forma mais rápida que uma multiplicação direta. Obs.: É possível mostrar que podemos fazer a multiplicação $x \cdot c$, onde x é inteiro e c é uma constante inteira aplicando este método fazendo no máximo $\log_2(c)$ somas, onde c é a constante inteira a multiplicar.

Outro sistema muito usado na literatura é a base 16 (*hexadecimal*). Neste sistema temos 16 dígitos usados na seguinte ordem: 0_h, 1_h, 2_h, 3_h, 4_h, 5_h, 6_h, 7_h, 8_h, 9_h, A_h, B_h, C_h, D_h, E_h, F_h.

Assim, o número $(F+1)_h$ é igual a 10_h (10_h em hexadecimal é igual a 16 no sistema decimal).

Exercício 1.2 *Quanto é $A9B_h$ em decimal ?*

1.5 Álgebra Booleana

Alguns comandos de programação estão estreitamente relacionados com um sistema de álgebra, chamado *álgebra de boole*, desenvolvido por George Boole. Neste tipo de álgebra podemos operar sobre proposições que podem ser verdadeiras ou falsas, resultando em um resultado que também é verdadeiro ou falso. Em 1930, Turing mostrou que três funções lógicas (e *(and)*, ou *(or)* e não *(not)*) são suficientes para representar estas proposições lógicas. Uma das principais vantagens deste tipo de álgebra é que ela pode ser implementada eficientemente através de componentes eletrônicos.

Usando as letras F como falso e V como verdadeiro, apresentamos na tabela 2 os valores para as funções *(and)*, *or* e *not*.

x	y	$(x\ and\ y)$
V	V	V
V	F	F
F	V	F
F	F	F

x	y	$(x\ or\ y)$
V	V	V
V	F	V
F	V	V
F	F	F

x	$(\ not\ x)$
V	F
F	V

Tabela 2: Funções booleanas **and**, **or** e **not**.

Com estas três funções podemos construir funções mais complexas. Por exemplo, considere variáveis booleanas x e y, e uma função booleana $f(x,y)$ que assume os valores conforme a tabela a seguir.

x	y	$f(x,y)$
V	V	F
V	F	V
F	V	V
F	F	F

Para construir a função $f(x,y)$, podemos considerar a tabela acima, com todas as entradas possíveis de x e y, e construir $f(x,y)$ como uma seqüência de cláusulas ligadas pela função *or*. Cada cláusula corresponde a uma entrada verdadeira para a função $f(x,y)$, feita com as funções *and* e *not*. No exemplo acima, a função $f(x,y)$ pode ser escrita como:

$$f(x,y) = ((x\ and\ (\ not\ y))\ or\ ((\ not\ x)\ and\ y))$$

Exercício 1.3 *Construa uma fórmula booleana para a seguinte função $g(x,y,z)$ dada pela seguinte tabela:*

x	y	z	$f(x,y,z)$
V	V	V	F
V	V	F	V
V	F	V	F
V	F	F	V
F	V	V	V
F	V	F	V
F	F	V	F
F	F	F	F