## Capítulo 4

# Linguagens de Programação

## 4.0 Índice

## 4.1 Programação de Computadores

Embora o equipamento básico para a realização das tarefas associadas à Ciência da Computação seja, evidentemente, o computador, nós utilizaremos, ao longo deste curso, o conceito de *Sistema Computacional*, pelo seu significado mais abrangente, tanto quanto ao tipo de hardware envolvido quanto pela sua extensão aos demais componentes envolvidos nas atividades computacionais, particularmente os programas, métodos, regras e documentação.

Um Sistema Computacional pode ser visto como uma associação entre dois conceitos cada vez mais utilizados na terminologia de informática:

- o **hardware**, que está associado à parte física do sistema (os circuitos e dispositivos) que suporta o processamento da informação;
- o **software**, que corresponde ao conjunto de programas responsáveis pela pilotagem do sistema para a execução das tarefas consideradas.

No que diz respeito a este segundo conceito, pode-se estabelecer uma classificação segundo o tipo de serviço realizado pelo software. Assim, têm-se as seguintes definições:

- **software de sistema** (ou **sistema operacional**) capaz de oferecer ao usuário, ou a outros softwares, facilidades de acesso aos recursos do computador, seja através de comandos, seja através de serviços especiais ativados a nível de um programa. O sistema operacional administra os arquivos, controla periféricos e executa utilitários.
- **software utilitário**, que são programas desenvolvidos por especialistas ou mesmo por usuários experimentados e que tem por objetivo facilitar a realização de determinadas atividades correntes no uso dos computadores (detecção e eliminação de vírus, programas de comunicação em redes de computadores, compressão de arquivos, etc...);
- **software aplicativo**, que são os programas desenvolvidos ou adquiridos pelos usuários para algum fim específico, seja ele de natureza profissional, educacional ou mesmo de lazer (jogos).

## 4.2 Níveis de Linguagens de Programação

Informalmente, uma linguagem de programação pode ser definida como sendo um conjunto limitado de instruções (vocabulário), associado a um conjunto de regras (sintaxe) que define como as instruções podem ser associadas, ou seja, como se pode compor os programas para a resolução de um determinado problema.

Ao longo dos anos, uma grande quantidade de linguagens de programação foi desenvolvida (e continua sendo), algumas de uso mais geral e outras concebidas para áreas de aplicação específicas.

As linguagens de programação podem ser classificadas em níveis de linguagens, sendo que as linguagens de nível mais baixo são mais próximas da linguagem interpretada pelo processador e mais distante das linguagens naturais.

### 4.2.1 Linguagem de Máquina

Lembrando que o computador corresponde basicamente a um conjunto de circuitos, a sua operação é controlada através de programas escritos numa forma bastante primitiva, baseada no sistema binário de numeração. O sistema binário é usado tanto para a representação dos dados quanto das operações (instruções). A esta forma de representação dos programas, é dado o nome de **linguagem de máquina**, em razão de ser a forma compreendida e executada pelo hardware do sistema.

As instruções de linguagem de máquina são representadas por **códigos** que correspondem a números binários cuja extensão pode variar de 8 a 64 bits (Figura 1). Dependendo da operação considerada, o código de uma instrução pode simbolizar a operação a ser executada e os dados envolvidos na operação.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

**Figura 1.** Ilustração de um programa em linguagem de máquina

Por uma questão de custo de hardware, as operações representadas pelas instruções de linguagem de máquina são bastante elementares, como por exemplo, a transferência de dados entre memória e registro da CPU, a adição de dois valores, o teste de igualdade entre dois valores, etc...

A linguagem de máquina é impraticável para escrita ou leitura. É inviável escrever ou ler um programa codificado na forma de uma *string* de bits.

### 4.2.2 Linguagem Hexadecimal

Para simplificar a compreensão e a programação de computadores, num primeiro tempo foi adotado a notação hexadecimal para representar programas em linguagem de máquina, onde a seqüência de bits é representada por números hexadecimais, conforme ilustrado na Figura 2.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 1A | FB | AB | 7F | 43 | 27 | 5B | 6C | D5 | 6F | 99 | FF | 10 | 11 | 20 |
| 39 | 03 | 30 | 39 | 73 | 63 | F4 | 3A | B4 | 74 | 84 | AB | 7D | 6B | 54 | 35 |
| 84 | 47 | F3 | 37 | 84 | 50 | 83 | BC | 5F | 6C | 10 | 39 | 85 | 85 | 94 | 47 |
| 84 | 03 | 83 | 03 | 83 | 78 | 5F | FF | FF | 00 | 00 | 00 | 00 | 00 | 00 | 74 |

**Figura 2.** Ilustração de um programa em linguagem hexadecimal

A linguagem hexadecimal é portanto apenas uma simplificação de notação da linguagem de máquina. Apesar disto, a programação e leitura usando a linguagem hexadecimal continua impraticável.

### 4.2.3 Linguagem Assembly

Embora seja a linguagem diretamente executável pelos processadores, a programação de aplicações diretamente em linguagem de máquina é impraticável, mesmo representada na notação hexadecimal. Por esta razão, a linguagem de máquina de cada processador é acompanhada de uma versão "legível" da linguagem de máquina que é a chamada linguagem simbólica Assembly. Simbólica pois esta linguagem não é composta de números binários ou hexadecimais como nas duas linguagens anteriores. A linguagem Assembly é na realidade uma versão legível da linguagem de máquina. Ela utiliza palavras abreviadas, chamadas de **mnemônicos**, indicando a operação a ser realizada pelo processador. Abaixo são apresentados dois exemplos de instruções Assembly:

- MOV R1, R2 – nesta instrução identifica-se o mnemônico MOV (abreviação de MOVE) e dois registradores como parâmetros: R1 e R2. Quando o processador executa esta instrução, ele comanda o movimento do conteúdo de R2 para R1 (equivalente a instrução Pascal R1:=R2, sendo R1 e R2 equivalente a duas variáveis);
- ADD R1, R2 – nesta instrução identifica-se o mnemônico ADD (abreviação de ADDITION) e dois registradores como parâmetros: R1 e R2. Quando o processador executa esta instrução, ele comanda a adição do conteúdo de R1 ao conteúdo de R2 e o resultado é armazenado em R1 (equivalente a instrução Pascal R1:=R1+R2).

Escolhendo nomes descritivos para as variáveis do programa, e usando mnemônicos para representar códigos de operação, a linguagem assembly facilitou significativamente

a leitura de seqüências de instrução de máquina. Como exemplo, supomos a operação de dois números inteiros: A:=B+C. Esta operação, em um PC, em notação hexadecimal ficaria: A1000203060202A30402. Se associarmos o nome B à posição de memória $200_h$, C à posição $202_h$ e A à posição $204_h$, usando a técnica mnemônica, a mesma rotina poderá ser expressa da seguinte forma:

```
MOV AX,B      ; registro AX recebe o valor de memória contida na variável B
ADD AX,C      ; AX recebe a soma de AX (valor de B) com o valor de C
MOV A,AX      ; variável A recebe valor de AX
```

A maioria concorda que a segunda forma, embora ainda incompleta, é melhor que a primeira para representar a rotina. Apesar de oferecer uma representação mais próxima do que o programador está acostumado a manipular, a linguagem Assembly apresenta certas dificuldades para a realização dos programas, tais como a necessidade de definição de um conjunto relativamente grande de instruções para a realização de tarefas que seriam relativamente simples (se representadas através de outras linguagens) e a exigência do conhecimento de detalhes do hardware do sistema (arquitetura interna do processador, endereços e modos de operação de dispositivos de hardware, etc...).

Por outro lado, a utilização da linguagem Assembly proporciona um maior controle sobre os recursos do computador, permitindo também se obter bons resultados em termos de otimização de código.

Como a linguagem Assembly é apenas uma versão legível da linguagem de máquina, a passagem de um programa escrito em Assembly para a linguagem de máquina é quase sempre direta, não envolvendo muito processamento. Esta passagem de um programa Assembly para linguagem de máquina é chamada de Montagem, e o programa que realiza esta operação é chamado de **montador** (*Assembler*).

A linguagem Assembly é orientada para máquina (ou melhor, para processador), é necessário conhecer a estrutura do processador para poder programar em Assembly. A linguagem Assembly utiliza instruções de baixo nível que operam com registros e memórias diretamente. Assim ela é muito orientada às instruções que são diretamente executadas pelo processador. Na seqüência da evolução das linguagens de programação, procurou-se aproximar mais a linguagem de programação à linguagem natural que utilizamos no dia-a-dia: surgiram então as linguagens de alto nível, tipo Pascal, C, C++, etc.

### *Vantagens e Desvantagens da Linguagem Assembly*

Mas se nós temos as linguagens de alto nível para quê precisamos utilizar a linguagem Assembly? Para responder esta pergunta é necessário conhecer as vantagens e desvantagens da linguagem Assembly e a sua utilização.

Desvantagens com relação as linguagens de alto nível:

- A linguagem Assembly apresenta um número muito reduzido de instruções, do tipo operações de movimentação de dados em memória, para registros e para memórias, e operações lógicas e aritméticas bem simples. Estas instruções são de baixa expressividade, isto é, elas são de baixo nível. O programador deve programar num nível de detalhamento muito maior para fazer a mesma coisa que em um programa escrito em linguagem de alto nível.
- Como o programador utiliza diretamente os recursos do processador e memória, ele deve conhecer, e muito bem, a máquina onde ele está programando.
- Um programa escrito em linguagem Assembly não é muito legível, por isso ele deve ser muito bem documentado.
- Um programa Assembly não é muito portável (pode ser usado apenas em um tipo de computador). Ela é portável apenas dentro de uma família de processadores. Por exemplo, diferente de um programa C, ele não pode ser executado em PCs e estações de trabalho.
- Devido a sua baixa expressividade, ilegibilidade e exigência do conhecimento sobre a máquina faz a programação Assembly ter um custo de desenvolvimento

maior, requerendo um maior número de homens/hora comparado com a programação utilizando linguagens de alto nível.

Apesar das desvantagens acima citadas, a utilização da linguagem Assembly tem algumas vantagens que são listados abaixo:

- Ela permite o acesso direto ao programa de máquina. Utilizando uma linguagem de alto nível, não se tem o controle do código de máquina gerado pelo compilador (alguns compiladores permitem a otimização de tamanho e de velocidade do programa). Devido a este acesso, o programador pode gerar um programa mais compacto e eficiente que o código gerado pelo compilador. Um programa escrito em linguagem Assembly pode ser 0 ou 300 % menor e mais rápido que um programa compilado.
- Além disso, esta linguagem permite o controle total do hardware, por exemplo, permitindo a programação de portas serial e paralela de um PC.

### *Aplicações da Linguagem Assembly*

A linguagem Assembly é utilizada em vários tipos de aplicações:

- **Controle de processos com resposta em tempo real**, devido à possibilidade de gerar programas mais eficientes. Neste tipo de aplicação, geralmente o processador deve executar um conjunto de instruções em um tempo limitado. Por exemplo, a cada 10 milisegundos o processador deve ler um dado, processá-lo e emitir um resultado.
- **Comunicação e transferência de dados**, devido à possibilidade de acessar diretamente o hardware, a linguagem Assembly é utilizada para a implementação de programas de comunicação ou transferência de dados.
- **Otimização de sub-tarefas da programação de alto nível**: um programa não precisa somente ser escrito em linguagem Assembly ou linguagem de alto nível. Nós podemos ter programas de alto nível com sub-tarefas escritas em linguagem Assembly. Sendo assim, nós podemos otimizar partes de programas, no caso de tarefas tempo-real ou para a programação do hardware do computador.

## 4.2.4 Linguagem de Alto Nível

As linguagens de alto nível são assim denominadas por apresentarem uma sintaxe mais próxima da linguagem natural, fazendo uso de palavras reservadas extraídas do vocabulário corrente (como READ, WRITE, TYPE, etc...) e permitirem a manipulação dos dados nas mais diversas formas (números inteiros, reais, vetores, listas, etc...); enquanto a linguagem Assembly trabalha com bits, bytes, palavras, armazenados em memória.

As linguagens de alto nível ou de segunda geração surgiram entre o final da década de 50 e início dos anos 60. Linguagens como Fortran, Cobol, Algol e Basic, com todas as deficiências que se pode apontar atualmente, foram linguagens que marcaram presença no desenvolvimento de programas, sendo que algumas delas têm resistido ao tempo e às críticas, como por exemplo Fortran, que ainda é visto como uma linguagem de implementação para muitas aplicações de engenharia. Cobol é um outro exemplo de linguagem bastante utilizada no desenvolvimento de aplicações comerciais.

Em comparação com a linguagem Assembly, a passagem de um programa escrito em linguagem de alto nível para o programa em linguagem de máquina é bem mais complexa. Para esta passagem são utilizados compiladores e linkadores.

Com o desenvolvimento das linguagens de alto nível, o objetivo da independência de máquina foi amplamente alcançada. Dado que os comandos das linguagens de alto nível não referenciam os atributos de uma dada máquina, eles podem ser facilmente compilados tanto em uma máquina como em outra. Assim, um programa escrito em linguagem de alto nível poderia, teoricamente, ser usado em qualquer máquina, bastando escolher o compilador correspondente.

Em realidade, no entanto, esta independência de máquina não é tão simples. Quando um compilador é projetado, certas restrições impostas pela máquina subjacente são, em última instância, refletidas como características da linguagem a ser traduzida. Por

exemplo, o tamanho do registrador e as células de memória de uma máquina limitam o tamanho máximo dos inteiros que nela podem ser convenientemente manipulados. Disso resulta o fato de que, em diferentes máquinas, uma "mesma" linguagem pode apresentar diferentes características, ou dialetos. Conseqüentemente, em geral é necessário fazer ao menos pequenas modificações no programa antes de movê-lo de uma máquina para outra.

A causa deste problema de portabilidade é, em alguns casos, a falta de concordância em relação à correta composição da definição de uma linguagem em particular. Para auxiliar nesta questão, o *American National Standards Institute* (ANSI) e a *International Organization for Standardization* (ISO) adotaram e publicaram padrões para muitas das linguagens mais populares. Em outros casos, surgiram padrões informais, devido à popularidade de um dados dialeto de uma linguagem e ao desejo, por parte de alguns autores de compiladores, de oferecerem produtos compatíveis.

### 4.2.5 Linguagens estruturadas

Nesta classe, encaixam-se as chamadas linguagens de programação de alto nível surgidas em meados dos anos 60. As linguagens concebidas neste período foram resultado da necessidade da produção de código de programa de forma clara, aparecendo o conceito de estruturação do código (endentação, utilização de letras maiúsculas e minúsculas nos identificadores, eliminação de instruções "problemáticas" como o "go to", etc...).

O período compreendido entre a década de 60 e a de 80 foi bastante produtivo no que diz respeito ao surgimento de linguagens de programação, o que permitiu o aparecimento de uma grande quantidade de linguagens as quais podem ser organizadas da seguinte forma:

- as ***linguagens de uso geral***, as quais podem ser utilizadas para implementação de programas com as mais diversas características e independente da área de aplicação considerada; encaixam-se nesta categoria linguagens como Pascal, Modula-2 e C;
- as ***linguagens especializadas***, as quais são orientadas ao desenvolvimento de aplicações específicas; algumas das linguagens que ilustram esta categoria são Prolog, Lisp e Forth;
- as ***linguagens orientadas a objeto***, que oferecem mecanismos sintáticos e semânticos de suporte aos conceitos da programação orientada a objetos; alguns exemplos destas linguagens são Smalltalk, Eiffel, C++ e Delphi.

## 4.3 Desenvolvimento de Programas

O desenvolvimento de programas é associado ao uso de ferramentas ou ambientes de desenvolvimento que acompanham o programador desde a etapa de codificação propriamente dita até a geração e teste do código executável. Serão apresentadas a seguir as principais etapas de geração de um programa, além das ferramentas utilizadas. Mais adiante serão apresentadas metodologias mais completas que definem os passos para o desenvolvimento de programas. Esta área da informática é chamada de engenharia de software.

### 4.3.1 Geração do código fonte (codificação)

A codificação é a escrita, utilizando uma linguagem de programação, das instruções que o computador deve realizar para alcançar um resultado. Para a realização desta tarefa são utilizados os chamados editores. Os editores são a primeira ferramenta à qual o programador recorre na etapa de codificação, pois é através dela que será gerado o arquivo (ou o conjunto de arquivos) que vai conter o **código-fonte** do programa a ser desenvolvido.

Apesar de que é possível utilizar qualquer editor de linha (como por exemplo o EDIT do DOS) para gerar o arquivo de programa, alguns ambientes oferecem ferramentas de

edição mais poderosas (orientadas à sintaxe ou de coloração de sintaxe). Alguns ambientes mais recentes oferecem a possibilidade de projeto de interfaces gráficas e gerenciadores de eventos, com geração automatizada do código-fonte.

### 4.3.2 Tradução do Código Fonte (código objeto)

Independente da linguagem utilizada e da arquitetura do sistema computacional, o código-fonte não é executável diretamente pelo processador. Ele permite apenas que o programador consiga definir o programa em uma forma legível aos humanos. Para que se possa obter o programa executável, é necessário que o código-fonte seja traduzido para o código de máquina do processador que compõe a arquitetura do sistema. Felizmente, isto é realizado de forma automática graças à existência de ferramentas como os Montadores (ou Assemblers que, como o nome indica, são orientados para traduzir programas escritos na linguagem Assembly) e os Compiladores, construídos para gerar o código de programas originalmente escritos em linguagens de alto nível como Pascal, C, Fortran e outras. O código gerado por estas ferramentas é representado segundo o sistema de numeração binária e é denominado **código-objeto**.

O código-objeto é o código produzido pelo compilador. Ele se trata de uma forma intermediária similar a linguagem de máquina do computador. O código-objeto, apesar de estar representado em binário, não é auto-contido, ou seja, não é executável diretamente pelos processadores. A principal razão disto é que o código-objeto é caracterizado normalmente por referências a partes de programa que não estão necessariamente definidas no mesmo arquivo que gerou o aquele arquivo objeto. Estas outras partes do código do programa podem ter sido geradas a partir de outros arquivos de código-fonte concebidos pelo mesmo programador ou existirem sob a forma de arquivos de bibliotecas de sub-rotinas.

### 4.3.3 Editores de ligação

A tarefa realizada pelo editor de ligações, ou linker como é mais conhecido é rearranjar o código do programa, incorporando a ele todas as partes referenciadas no código original, resultando num código executável pelo processador. Esta tarefa pode ser feita também pelos chamados carregadores.

A figura abaixo resume as três etapas anteriores.

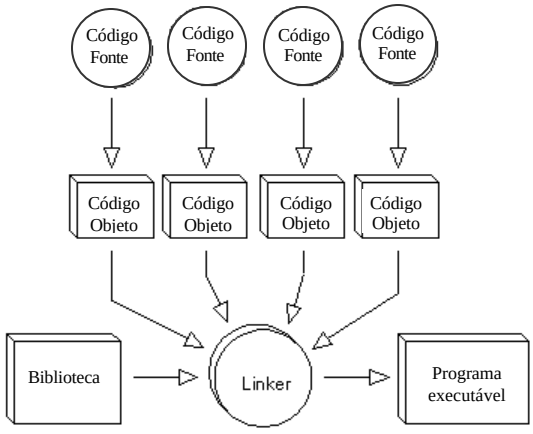


**Figura 3.** Desenvolvimento de um programa

### 4.3.4 Depuradores ou debuggers

Os debuggers são assim chamados devido à sua função essencial que é de auxiliar o programador a eliminar (ou reduzir) a quantidade de "bugs" (erros) de execução no seu programa. Eles executam o programa gerado através de uma interface apropriada que possibilita uma análise efetiva do código do programa graças à:

- execução passo-a-passo (ou instrução por instrução) de partes do programa;

- visualização do “estado” do programa através das variáveis e eventualmente dos conteúdos dos registros internos do processador;
- alteração em tempo de execução de conteúdos de memória ou de variáveis ou de instruções do programa;
- etc...

## 4.4 Execução de Programas

Para que um programa possa ser executado, é preciso que seja transferido para a memória principal. A maioria dos programas fica armazenada em disco (disco rígido, disquetes, etc.), mas a CPU não pode executar nenhum programa diretamente a partir do disco. O programa precisa ser antes lido do disco e carregado na memória principal. Por exemplo, para executar o programa FORMAT (usado para formatar disquetes), é preciso que você forneça pelo teclado um comando como:

```
FORMAT A:
```

Uma vez que você digita este comando, o programa FORMAT.COM é lido do disco rígido e carregado na memória principal. O “carregador” (loader) é o utilitário do sistema operacional responsável pela cópia do programa do dispositivo de armazenamento para a memória principal. A CPU pode então executar o programa, que fará a formatação de um disquete. A Figura 4 simboliza a leitura do programa FORMAT.COM a partir do disco para a memória principal (essa operação é chamada de CARGA), e seu processamento pela CPU (essa operação é chamada de EXECUÇÃO).

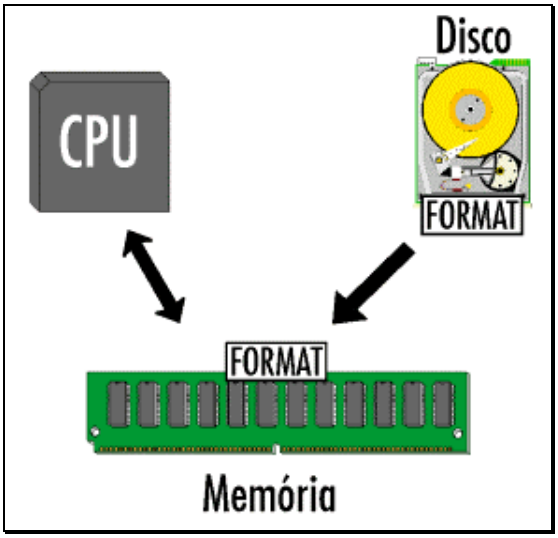


**Figura 4.** Carga e execução do programa FORMAT.COM

O sistema operacional é responsável pela leitura do arquivo FORMAT.COM e a execução. O MS-DOS é um exemplo de sistema operacional. O WINDOWS também pode ser considerado uma espécie de sistema operacional. Uma das várias funções do sistema operacional é permanecer o tempo todo ativo na memória principal, esperando que o usuário comande a execução de algum programa. Portanto, quando se usa um comando como "FORMAT A:", o que ocorrer na verdade é o seguinte:

- Inicialmente o sistema operacional checa se você fornece algum comando.
- Você digita o comando "FORMAT A:".
- O sistema operacional procura no disco o arquivo FORMAT.COM e carrega-o na memória RAM.
- O sistema operacional momentaneamente transfere o controle da CPU para o programa FORMAT.COM, que a essa altura já está carregado na memória principal.

- A CPU executa o programa FORMAT.COM
- Ao terminar a execução do FORMAT.COM, o sistema operacional volta a ter o controle da CPU. Fica então aguardando que você envie um novo comando.

Podemos entender então que nenhum programa chega até a memória por mágica, e sim, através do controle feito pelo sistema operacional. Alguém mais observador pode então ficar com a seguinte dúvida: "Se é o sistema operacional quem lê para a memória principal todos os programas a serem executados, como é então que o próprio sistema operacional chegou nesta memória?". No instante em que ligamos o computador, a memória principal não contém programa algum. Nesse instante, o sistema operacional está armazenado no disco (normalmente no disco rígido, no caso dos PC's), e precisa ser carregado na memória. Quem faz a carga do sistema operacional para a memória é um programa chamado BIOS, que fica gravado em memória ROM. Lembre-se que a memória ROM não perde seus dados quando o computador é desligado. Portanto, no instante em que ligamos o computador, o BIOS já está na memória, e é imediatamente processado pela CPU. O processamento do BIOS começa com uma contagem de memória, seguido de alguns testes rápidos no hardware, e finalmente a leitura do sistema operacional do disco para a memória principal. Esse processo, ou seja, a carga do sistema operacional na memória RAM, é chamado de BOOT. A Figura 5 mostra o processo de BOOT para a carga do sistema operacional DOS:

1) No instante em que o computador é ligado, o sistema operacional está armazenado em disco, a RAM está "vazia", e a CPU executa o BIOS.
2) Mostra o instante em que termina a operação de BOOT. O sistema operacional já está carregado na memória e já está sendo executado pela CPU.
3) Mostra o que ocorre imediatamente antes da execução do programa FORMAT.COM. O sistema operacional recebe um comando do usuário para que leia o arquivo FORMAT.COM do disco para a memória RAM.
4) O programa FORMAT.COM está sendo executado pela CPU.

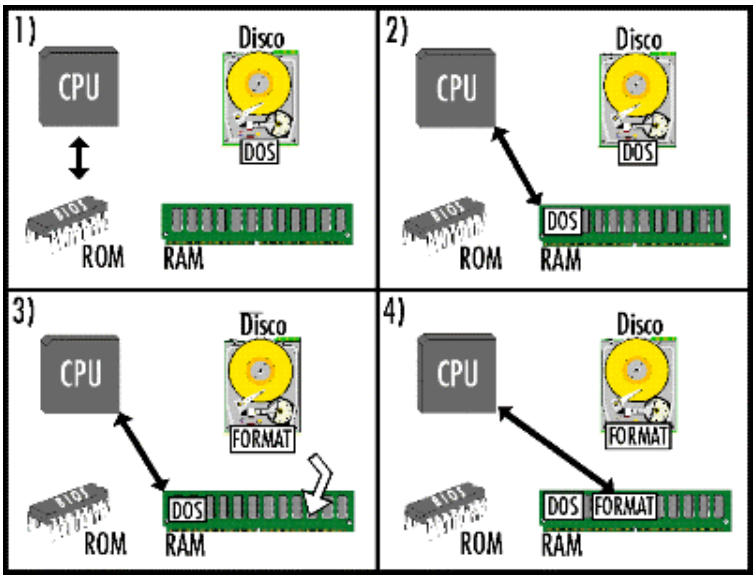


**Figura 5.** BOOT e carga de um programa